DISTRIBUTED AND MULTI-AGENT PLANNING

Proceedings of the 1st Workshop on
Distributed and Multi-Agent Planning

DMAP 2013

Rome, Italy - June 11, 2013

Edited By:

Raz Nissim, Daniel L. Kovacs, Ronen Brafman

Organizing Commitee

Raz Nissim
Ben-Gurion University of the Negev, Israel

Daniel L. Kovacs
Budapest University of Technology and Economics, Hungary

Ronen Brafman
Ben-Gurion University of the Negev, Israel

Program Committee

Bradley J. Clement, NASA Jet Propulsion Laboratory
Amanda Coles, King's College London, UK
Andrew Coles, King's College London, UK
Carmel Domshlak, Technion, Israel Institute of Technology
Naoki Fukuta, Shizuoka University, Japan
Antonin Komenda, Czech Technical University
Roman van der Krogt, University College Cork
Alejandro Torreno Lerma, Universidad Politecnica de Valencia
Scott Sanner, National ICT Australia (NICTA)
Matthijs Spaan, Delft University of Technology
Roni Stern, Harvard University
Mathijs de Weerdt, Delft University of Technology
Shlomo Zilberstein, University of Massachusetts, Amherst

The workshop is organized in cooperation with COST Action IC1205 on Computational Social Choice.

Foreword

This volume contains the papers accepted for presentation at DMAP 2013, the First Distributed and Multi-Agent Planning workshop, held in Rome, Italy, June 11, 2013. It continues the tradition of the "Multiagent Planning and Scheduling" workshop series held at ICAPS 2005 and 2008, and the joint AAMAS-ICAPS session on multi-agent planning in 2010.

There is growing interest in multi-agent planning in recent years with much progress in formalizing general models, algorithmic techniques, and solution concepts for the multi-agent planning problem. The majority of this work has been published in the Autonomous Agents and Multi-Agent Systems (AAMAS) conferences and in general AI conferences such as AAAI and IJCAI. However, this line of work often builds upon planning and scheduling models, techniques and applications that are studied and developed in the ICAPS community.

The goal of this workshop is to bring researchers working in the different subfields together in order to bridge the gap between these communities.

The papers accepted to the workshop cover many of these subfileds and include work on classical multi-agent planning, partial observability, multi-agent replanning, heuristics and planning by selfish agents. We thus hope to offer attendees a view of the diverse work done on multi-agent planning, and the progress made in this area, fostering works that will farther progress the field forward.

For DMAP 2013, we received 14 submissions from 8 countries. From these submissions, 13 were full papers, and 1 was short. These papers were all reviewed by a Program Committee made up of 13 members, coordinated by the three PC Chairs. As a result of this evaluation, 11 papers were accepted: 10 full papers and 1 short paper. All these papers will be presented orally at the workshop.

We thank all members of the Program Committee for their effort in the review process that is critical for maintaining the level of the workshop. We also thank the ICAPS workshop chairs for their help in organization, and most importantly, the multi-agent planning and scheduling community who supported this event by submitting their work and participating actively in it.

– Raz Nissim, Daniel L. Kovacs,
Ronen Brafman
DMAP-2013 Chairs

Table of Contents

Cost-Optimal Planning by Self-Interested Agents

Raz Nissim and **Ronen I. Brafman**
Ben-Gurion University of the Negev
Be'er Sheva, Israel
raznis,brafman@cs.bgu.ac.il

Abstract

As our world becomes better connected and autonomous agents no longer appear to be science fiction, a natural need arises for enabling groups of selfish agents to cooperate in generating plans for diverse tasks that none of them can perform alone in a cost-effective manner. While most work on planning for/by selfish agents revolves around finding stable solutions (e.g., Nash Equilibrium), this work combines techniques from mechanism design with a recently introduced method for distributed planning, in order to find cost optimal (and, thus, social welfare maximizing) solutions. Based on the Vickrey-Clarke-Groves mechanisms, we present both a centralized, and a privacy-preserving distributed mechanism.

Introduction

As our world becomes better connected and more open ended, production becomes more customized, and autonomous agents no longer appear to be science fiction, a natural need arises for enabling groups of selfish agents to cooperate in generating plans for diverse tasks that none of them can perform alone in a cost-effective manner. To build a house, one requires workers with diverse skills, such as architects, lawyers, construction workers, plumbers, electricians, and more. Similarly, to organize a conference, one requires caterers, speakers, AV technicians, publicity, hotel negotiations, program committees, etc. These are just two examples of tasks that require the combined actions of diverse agents and careful planning, as one agent cannot act before other agents have achieved the preconditions for its actions. Agents involved in such tasks realize the need to collaborate with other agents both in planning and execution in order to succeed, but when doing so, they are usually motivated by the desire to maximize their own profit/utility.

Fully cooperative agents can address this problem easily by sending all information to a trusted party that will generate a plan, or by using a distributed planning algorithm, and then employing some agreed upon scheme to share any profit. Self-interested agents, on the other hand, are unlikely to agree to such a scheme for a number of reasons. First, it is easily manipulable – i.e., an agent may report incorrect information about its abilities or its costs in order to increase its share of the profit, thereby decreasing the share of the profit of agents that truthfully report their information. This can lead to a globally sub-optimal plan, discouraging agents from taking part in such a process. Second, many commercial entities are reluctant to reveal certain information to third parties. Whereas an agent must advertise in some way its public capabilities, so others will ask for its services, it prefers *not* to reveal its inner workings – its private state, how it manipulates it, and the cost of such private actions.

Conceptually, the issue of manipulation can be addressed using existing ideas. Semantically, optimal planning algorithms seek a shortest path from the initial state to a goal state in a graph that describes a transition system. In our context, edges in this graph – which correspond to actions – are controlled by different agents. A similar situation arises in network routing, where we seek to route a packet from its source to its destination using a network controlled by different agents, via a shortest path. There, Vickrey-Clarke-Groves (VCG) type mechanisms using distributed implementations of classical shortest path algorithms, such as Dijkstra's (1959), have been developed (Nisan and Ronen 2001; Feigenbaum et al. 2002). These methods provide an attractive distributed solution to this problem, which ideally, we could apply in planning problems. There is a fundamental difference, however: In planning, the transition graph is implicitly defined because the explicit graph is exponential, and thus never constructed explicitly by planning algorithms. Thus, despite their attractiveness, these methods are inapplicable in planning problems.

In this paper we present a mechanism that offers similar properties, thus addressing the problem of manipulation, maintains the privacy of information pertaining to the agents' private state, and is practically efficient. More specifically, our mechanism is distributed, strategy-proof, socially efficient, and privacy preserving. To the best of our knowledge, it is the first mechanism for planning to offer even just the first three properties. Moreover, it is based on state-of-the-art planning technology– making it relatively practical: while there is a price to selfishness, we are still able to optimally solve most solvable benchmark planning problems.

Background

We begin by describing the model used for multi-agent (MA) planning and an overview of mechanism design and

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

VCG mechanisms.

Planning for Self-Interested Agents

The framework we chose for our work is the MA-STRIPS model (Brafman and Domshlak 2008). This framework minimally extends the classical STRIPS problem to MA planning for *cooperative* agents. The main benefits of using this model are its simplicity, and the fact that it is easily extended to handle the case of *non-cooperative* agents.

Definition 1. *A MA-STRIPS planning task for a set of fully cooperative agents $\Phi = \{\varphi_i\}_{i=1}^k$ is given by a 4-tuple $\Pi = (P, \{A\}_{i=1}^k, I, G)$ with the following components:*

- *P is a finite set of* atomic propositions*, $I \subseteq P$ encodes the* initial state *and $G \subseteq P$ encodes the* goal conditions*.*
- *For $1 \leq i \leq k$, A_i is the set of actions that the agent φ_i is capable of performing, and each action $a \in A = \bigcup A_i$ is given by its preconditions and effects, as well as its cost.*

A *solution* to a planning task is a legal plan $\pi = (a_1, \ldots, a_k)$ which transforms the initial state into a state satisfying the goal conditions. A solution is *optimal* if it has minimal cost (sum of action costs) over all solutions.

Given the partitioning of the actions, we can now distinguish between *private* and *public* atoms and actions. A *private* atom of agent φ is required and affected only by the actions of φ. An action is *private* if all its preconditions and effects are private. All other actions are classified as *public*. That is, φ's private actions affect and are affected only by φ, while its public actions may require or affect the actions of other agents. Note that this implies the public action of an agent may have private preconditions and/or effects. For ease of the presentation of our algorithms and their proofs, we assume all goal conditions are *public*, thus all actions that achieve a goal condition are considered *public* as well. Our methods are easily modified to remove this assumption.

Extending MA-STRIPS from fully-cooperative agents to self-interested agents does not require changing the model itself, but rather changing how it is used, and how information is distributed to different agents. We think of the public part of public actions (i.e., public preconditions and effects) as the *interface* of the agent to the external world – the information it advertises to the external world. The private aspects of public actions, the private variables, and the private actions and their cost is information that agents do not advertise, e.g., because it reveals information about their internal operation that may be useful to competitors, or because of other privacy issues.

Since the self-interest of agents requires defining their utility functions, we make the standard assumption of *quasi-linear* utility functions, where agent φ_i's net utility, given plan π and payments $\mathcal{P}$, is

$$u_i(\pi, \mathcal{P}) = \mathcal{P}_i - cost_i(\pi)$$

where $\mathcal{P}_i$ is the payment given to agent φ_i for its participation and $cost_i(\pi)$ denotes the sum of the costs of all actions by agent φ_i in π. Finally, we assume that no collusion between agents occurs. This also implies that an agent cannot pretend to be several independent agents.

Mechanism Design and the VCG Mechanism

Mechanism design deals with the problem of optimizing some criteria (e.g. social welfare), when self-interested agents having private information are involved. In the standard (centralized) setting, agents report their private information to a "center", which solves the optimization problem and enforces the outcome.

The Vickery-Clarke-Groves (VCG) mechanism (Vickrey 1961; Clarke 1971; Groves 1973) is one of the most celebrated results of mechanism design. Very generally, given the agents' report of their private information, the center computes an optimal solution and the payments to be made to each of the agents. These payments reflect the impact each agent's participation has on other agents, and are computed based on the solutions to the *marginal* problems $\Pi_{-\varphi_i}$, in which agent φ_i is completely ignored. VCG is *strategyproof*, meaning that each agent's utility-maximizing strategy regardless of other agents' strategies and private information is to *truthfully* reveal its private information to the center. Strategyproofness implies that the agents don't need to model the behavior of others in order to compute their equilibrium strategy – *truth telling is a weakly dominant strategy*. VCG is also *efficient*, meaning that the outputted solution maximizes the total utility to agents over all possible solutions to Π. Both these properties prove useful when applying VCG to our setting of cost-optimal MA planning.

Related Work

Our work relates to two research areas: Multi-agent/ distributed planning and mechanism design for decision making in systems of self-interested agents. We now survey relevant results in both areas.

Recently, several methods for planning in the *cooperative* setting of MA-STRIPS have been presented. *Planning-First* (Nissim, Brafman, and Domshlak 2010), a distributed MA planner based on the *Planning as CSP+Planning* methodology (Brafman and Domshlak 2008), showed scalability in problems having loosely coupled agents. More recently, an approach which uses plan refinement (Torreño, Onaindia, and Sapena 2012), showed state-of-the-art performance in more tightly coupled problems. We note that both methods are privacy preserving, but do not guarantee cost-optimal solutions (Planning-First minimizes the maximal number of public actions in an agent's plan).

Tackling the strategic side of MA planning, work has been done on achieving equilibria by a strategic diagnosis of the planning scenario by the agents (Bowling, Jensen, and Veloso 2003; Ben Larbi, Konieczny, and Marquis 2007). This work attempts to find strategies (plans) for the agents which constitute a Nash Equilibrium, by performing strategic analysis of all possible agent plans. This analysis, where every strategy of every agent must be evaluated against all other strategies, makes these methods ineffective for planning, since even if plan length is bounded polynomially (not the case for many planning benchmarks), the number of available strategies is exponential. This problem pertains to all such "pure" game-theoretic approaches requiring the agents to perform some strategic evaluation of other agents.

In light of this, further work has been done on feasibility conditions of MA planning for selfish agents, aiming to find stable solutions in problems exhibiting certain structure – acyclic agent interaction graph (Brafman et al. 2009; 2010), and safe interaction, where no agent benefits from invalidating another agent's plan (Crosby and Rovatsos 2011). Furthermore, work has been done on using single-agent planners in order to perform plan improvement using best-response, in scenarios where an initial MA plan is available (Jonsson and Rovatsos 2011). We note that all the mentioned approaches do not aim to find an optimal solution.

In the field of mechanism design, much work has been done on using *centralized* incentive mechanisms for distributed systems (Ephrati and Rosenschein 1991; Parkes, Kalagnanam, and Eso 2001). The problem of lowest-cost routing on the Internet provided the motivation for work on *distributed algorithmic mechanism design* (Feigenbaum et al. 2002; Feigenbaum and Shenker 2002). This work presents distributed protocols for computing all-pairs least-cost routes, where the links are associated to strategic agents. The assumption here is that the graphs are given, and are small (one node per agent), since the algorithms require memory polynomial in graph size. This assumption means these methods cannot be used in the planning setting, where the graphs are implicit and exponentially large. The main advantage of these works was that the agents could not benefit by misreporting their costs, so no strategic analysis is required by the agents. Later work (Parkes and Shneidman 2004) discussed the robustness of the algorithm itself to manipulation, and introduced the principles required for achieving an incentive compatible distributed mechanism. These principles are discussed in further detail later on.

Based on these results, the first faithful distributed implementation of efficient social choice problems (M-DPOP) was presented (Petcu, Faltings, and Parkes 2008). M-DPOP is a distributed constraint optimization algorithm implementing the VCG mechanism. It ensures that no agent can benefit by deviating from any part of the distributed protocol. While M-DPOP can be used to solve planning problems, which can be cast as DCOPs, previous work (Nissim, Brafman, and Domshlak 2010; Nissim and Brafman 2012a) shows that distributed *planning* algorithms, especially ones using heuristic forward search, have strong computational benefits, whereas CSP-based methods can barely handle problems that require 3 or more actions per agent.

Centralized VCG Mechanism for Planning

To set the stage for our distributed mechanism, we now describe how to apply the VCG mechanism to our setting of optimal MA planning, when a trusted center exists:

Definition 2 (VCG mechanism for optimal MA planning)**.** *Given a MA planning problem* Π *in which agents are self-interested and have private information, the mechanism is as follows:*

1. *Agents report their private actions and all action costs to the center.*
2. *Given the agents' reports, the center finds an optimal solution* $\pi^\star$ *for* Π*, as well as* $\pi^*_{-\varphi_i}$ *for each* $\Pi_{-\varphi_i}$.
3. *Each agent receives payment*

$$\mathcal{P}_i = \sum_{j \neq i} (cost_j(\pi^\star_{-\varphi_i}) - cost_j(\pi^\star))$$

In this setting, each agent reports its actions and their costs to the center, which then computes the payment rule, i.e. the payments made to each of the agents for participating in the mechanism. The payment of each of the agents is determined by the center according to the agent's *social cost* – the aggregate impact its participation has on the utilities of other agents. To compute the payments, the center needs to solve Π, as well as the marginal problems $\Pi_{-\varphi_i}$ for each $\varphi_i \in \Phi$. Note that the payments are always non-negative, since an agent's presence may only have a positive effect on solution cost. Therefore, a *pivotal* agent φ_i, i.e. one which improves solution cost by participating, will have $\mathcal{P}_i > 0$ and will be paid for its participation, whereas non-pivotal agents will pay exactly 0. This means the center will run a budget-deficit (i.e. will never make a profit), but no agent will lose by participating. This deficit can be viewed as the agents' profit from taking part in the plan. In a market where numerous agents can perform similar actions at different costs, e.g., due to efficiency differences, proximity, availability of resources, etc., the profit is related to such advantages that one agent may have over others. Note that in this schema, an agent that is essential (i.e., no solution exists without it) will receive a payment of ∞, so the underlying assumption is that no such agent exists.

As an example, consider the well-known *logistics* planning domain, which involves vehicles transporting packages to their goal location. Each of the three agents $\varphi_1 \ldots \varphi_3$ can *pickup/drop* one of two packages p_1, p_2, and can *drive* between two locations A, B. Initially, both packages and the three agents are at location A, and the goal is to move both packages to location B. Action costs differ between agents: φ_1's *pickup/drop* actions have cost 1 for p_1, and cost 2 for p_2. φ_2's *pickup/drop* actions have cost 2 for p_1, and cost 1 for p_2. φ_3's *pickup/drop* actions have cost 2 for both p_1, p_2. All drive actions have cost 1.

Given the report of actions costs, the center now finds an optimal plan having cost 6, in which φ_1, φ_2 *pickup, drive and drop* packages p_1, p_2, respectively. For the marginal problems $\Pi_{-\varphi_1}$ and $\Pi_{-\varphi_2}$, the optimal solution has cost 8. Therefore, $\mathcal{P}_1 = \mathcal{P}_2 = 8 - 3 = 5$. This gives the two pivotal agents the positive utility $u_1 = u_2 = 5 - 3 = 2$. For the non-pivotal agent φ_3, $\Pi_{-\varphi_3}$ has an optimal solution cost of 6, therefore its payment (and its utility) will be 0.

Distributed Implementation of VCG

While the centralized approach finds the optimal solution, maximizes social welfare, and ensures truthfulness of the agents, in many cases it will be impossible to find a trusted central authority. Moreover, even if one exists, this centralized approach is not *privacy preserving*, as agents must reveal their private information to the center.

A desirable alternative is a *distributed* implementation of the VCG mechanism that computes the payments, solves Π, while maintaining some sense of VCG's strategyproofness.

However, such a distributed implementation introduces new opportunities for agent manipulation, as in addition to reporting its private information untruthfully, an agent may deviate from the distributed protocol, unless the right incentives for following the protocol are provided. We present such an implementation next.

In describing our distributed implementation, we make the following assumptions[1], in addition to the ones made in the background section:

1. There is a trusted *bank* which can communicate with the agents and distribute payments.
2. The agents are rational but helpful, i.e. they are self-interested, but will deviate from a protocol only if this makes them *strictly* better off.
3. Every agent can communicate directly with all other agents.
4. The bank may rescind payment if a legal, goal-achieving plan is not found and executed.

Work by Parkes and Shneidman (2004) describes principles that guide the distribution of computation, focusing in particular on the VCG mechanism. A distributed algorithm (protocol) is said to be *ex post faithful* if it is in the best interest of every agent to truthfully follow all aspects of the algorithm (information revelation, message passing, computation, etc.) regardless of other agents' private information, given that all other agents follow the algorithm[2]. An algorithm is said to satisfy the *Partition Principle* if (1) optimal solutions are always obtained for Π and $\Pi_{-\varphi_i}$, given that all agents fully comply with the suggested distributed protocol; (2) agent φ_i cannot affect the distributed computation of the marginal problem $\Pi_{-\varphi_i}$, nor its utility from the outcome; (3) the solution found for Π is correctly executed and the payments are made to the agents. Parkes and Shneidman show that *a distributed algorithm that satisfies the partition principle is ex post faithful.*

The MAD-A* Algorithm

We now describe the distributed protocol which we use for the VCG computations. At its heart is a distributed planning algorithm: MAD-A* (Nissim and Brafman 2012a; 2012b) is a distributed, privacy-preserving variation of A*, which maintains a separate search space for each agent. Each agent maintains an *open list* of states that are candidates for expansion and a *closed list* of already expanded states. It expands the state with the minimal $f = g + h$ value in its open list. When an agent expands state s, it uses its own actions only. This means that two agents expanding the same state will generate *different* successor states.

The messages sent between agents contain the full state s, i.e., including both public and private variable values, as well as the cost of the best plan from the initial state to s found so far, the sending agent's heuristic estimate of s and the last action performed leading up to s[3]. When agent φ receives a state via a message, it checks whether this state exists in its open or closed lists. If it does not appear in these lists, it is inserted into the open list. If a copy of this state with higher g value exists, its g value is updated, and if it is in the closed list, it is reopened. Otherwise, it is discarded. Whenever a received state is (re)inserted into the open list, the agent computes its local h_φ value for this state, and assigns the maximum of its h_φ value and the h value in the received message.

Once an agent expands a solution state s, it sends s to all agents and initiates the process of verifying its optimality. When the solution is verified as optimal, the agent initiates the trace-back of the solution plan. This is also a distributed process, which involves all agents that perform some action in the optimal plan. When the trace-back phase is done, a terminating message is broadcasted.

Algorithms 1-3 depict the MAD-A* algorithm for agent φ_i. Unlike in A*, expansion of a goal state in MAD-A* does not necessarily mean an optimal solution has been found. Here, a solution is known to be optimal only if all agents prove it so. Intuitively, a solution state s having solution cost f^* is known to be optimal if there exists no state s' in the open list or the input channel of some agent, such that $f(s') < f^*$. In other words, solution state s is known to be optimal if $f(s) \leq f_{\text{lower-bound}}$, where $f_{\text{lower-bound}}$ is a lower bound on the f-value of the entire system, including all states in open lists and states in unprocessed messages.

Algorithm 1 MAD-A* for Agent φ_i

```
1: while did not receive true from a solution verification procedure do
2:     for all messages m in message queue do
3:         process-message(m)
4:     s ← extract-min(open list)
5:     expand(s)
```

Algorithm 2 process-message($m = \langle s, g_{\varphi_j}(s), h_{\varphi_j}(s) \rangle$)

```
1: if s is not in open or closed list or g_{φi}(s) > g_{φj}(s) then
2:     add s to open list and calculate h_{φi}(s)
3:     g_{φi}(s) ← g_{φj}(s)
4:     h_{φi}(s) ← max(h_{φi}(s), h_{φj}(s))
```

To detect this situation, MAD-A* uses Chandy and Lamport's *snapshot algorithm* (Chandy and Lamport 1985), which enables a process to create an approximation of the global state of the system, without "freezing" the distributed computation. Although there is no guarantee that the computed global state actually occurred, the approximation is

[1] similar to the ones made by Petcu et al. (2008)

[2] The weakening of VCG's strategyproofness in the centralized case, to ex post faithful in the distributed case is often referred to as the *cost of decentralization*.

[3] It may appear that agents are revealing their private data because they transmit their private state in their messages. However, as will be apparent in the algorithm, other agents do not use this information in any way, nor alter it. They simply copy it to future states. Only the agent itself can change the value of its private state. Consequently, this data can be encrypted arbitrarily – it is merely used as an ID by other agents.

Algorithm 3 expand(s)

```
move s to closed list
if s is a goal state then
    broadcast s to all agents
    initiate verification of stable property f_{lower-bound} ≥ g_{φ_i}(s)
    return
for all agents φ_j ∈ Φ do
    if the last action leading to s was public and φ_j has a public action for which all public preconditions hold in s then
        send s to φ_j
apply φ_i's successor operator to s
for all successors s' do
    update g_{φ_i}(s') and calculate h_{φ_i}(s')
    if s' is not in closed list or f_{φ_i}(s') is now smaller than it was when s' was moved to closed list then
        move s' to open list
```

good enough to determine whether a stable property currently holds in the system. A property of the system is *stable* if it is a global predicate which remains true once it becomes true. Specifically, properties of the form $f_{\text{lower}-\text{bound}} \geq c$ for some fixed value c, are stable when h is a *globally consistent* heuristic function. That is, when f values cannot decrease along a path. In our case, this path may involve a number of agents, each with its h values. If each of the local functions h_φ are consistent, and agents apply the $\max$ operator when receiving a message, this property holds.

Distributed Implementation

Consider now our distributed implementation of VCG which uses the MAD-A* algorithm, presented as Algorithm 4. In this setting, the agents participate in $|\Phi|+1$ sequential MAD-A* searches, beginning with the original problem Π, and continuing with the marginal problems $\Pi_{-\varphi_i}$ for each i. When solving the marginal problem $\Pi_{-\varphi_i}$, messages from φ_i are ignored by all agents, and no messages are sent to it. Once each of the marginal problems $\Pi_{-\varphi_i}$ are solved, each agent $\varphi_j \neq \varphi_i$ knows its local cost in π^* and in $\pi^*_{-\varphi_i}$. This is sufficient in order to send the value $\mathcal{P}_{ij}$, that is, the cost removing φ_i incurs on φ_j, to the bank. Upon receiving all these messages, the bank is able to compute $\mathcal{P}$. Since MAD-A* does not require the agents to reveal their private information, Algorithm 4 is automatically privacy preserving.

Algorithm 4 Selfish-MAD-A*

```
Run MAD-A* on Π with all agents in Φ to find π*
for all φ_i ∈ Φ do
    Run MAD-A* on Π_{-φ_i} to find π*_{-φ_i}
    Agents φ_j ≠ φ_i compute P_ij = cost_j(π*_{-φ_i}) − cost_j(π*) and send it to the bank.
    The bank computes P_i = Σ_{j≠i} P_ij
Bank pays P_i to every agent φ_i, and broadcasts π* to all agents.
```

Proof of Correctness

In order to prove the correctness of algorithm 4, we must prove that it satisfies the partition principle (Parkes and Shneidman 2004). Therefore, we must show that (1) MAD-A* computes optimal solutions to Π and $\Pi_{-\varphi_i}$ for all $\varphi_i \in \Phi$, given that all agents comply with the protocol; (2) No agent can affect the solution of its respective marginal problem, nor its utility from the outcome; (3) the optimal solution of Π is correctly executed and all payments are made.

Condition (1) follows immediately from the correctness and optimality of MAD-A*, which requires only that the heuristic used by each of the agents is consistent. The first part of condition (2) holds since agent φ_i does not even participate in the solving of $\pi^*_{-\varphi_i}$, and therefore, cannot influence it in any way. For the second part, agent φ_i's utility $u_i(\Pi)$ is given by

$$\mathcal{P}_i - \text{cost}_i(\pi^\star) = \sum_{j \neq i} (\text{cost}_j(\pi^\star_{-\varphi_i}) - \text{cost}_j(\pi^\star)) - \text{cost}_i(\pi^\star)$$

$$= \text{cost}(\pi^\star_{-\varphi_i}) - \text{cost}(\pi^\star)$$

Since φ_i cannot influence $\text{cost}(\pi^\star_{-\varphi_i})$, it can have a positive influence on its utility only by decreasing $\text{cost}(\pi^\star)$. As $\pi^\star$ is already optimal, any such decrease means either that other agents' costs have been decreased, which will cause the plan to fail (some agent will not agree to such a solution) and the payments to be rescinded, or that φ_i misreported its own costs, which does not change its actual utility. Condition (3) follows directly from our assumption that there exists a trusted bank, which can rescind payments if a legal plan is not executed.

We have shown that Selfish-MAD-A* satisfies the partition principle, and therefore, it is *ex post faithful*.

Optimizing Selfish-MAD-A* by Multigoal Search

In the distributed implementation presented in the previous section, each of the $|\Phi| + 1$ problems is solved in isolation. It is clear that when these problems are solved, the generated search spaces overlap, causing some states to be generated (and evaluated) multiple times. For example, in $\Pi_{-\varphi_i}$, all reachable states s having $f(s) \leq \text{cost}(\pi^\star_{-\varphi_i})$ must be expanded. In Π, all these states are reachable, and will be expanded as well.

In order to prevent the duplication of effort Selfish-MAD-A* entails, we propose a *multigoal* variation of MAD-A*, which finds optimal solutions to all $|\Phi| + 1$ problems in a single run and on a single (distributed) search space. The main idea is to additionally identify each state s with the set of agents participating in the (currently) best plan leading up to s. If s can be reached via two paths having different participating agents sets, s is duplicated. When a goal state $s^\star$ is reached, it is treated as a candidate solution for $\Pi_{-\varphi_i}$ if φ_i does not participate in the path leading to $s^\star$. Termination detection remains unchanged, except that it is performed for each marginal problem, as well as for Π. $s^\star$ is known to be optimal for marginal problem $\Pi_{-\varphi_i}$, if $f(s^\star) \leq f^i_{\text{lower}-\text{bound}}$, where $f^i_{\text{lower}-\text{bound}}$ is a lower bound on the f-value of all states for which φ_i does not participate

Table 1: Comparison of centralized A*, centralized VCG, and two versions of distributed VCG. Running time (in sec.) and the number of generated states are shown.

	Time (sec.)				Generated states			
Problem	A*	Cen	Dis	OptDis	A*	Cen	Dis	OptDis
rovers6 (2)	278	282.5	324	290	34M	35M	114M	102M
rovers7 (3)	0.7	2.78	6.62	5.1	62271	316553	2262129	2104573
rovers12 (4)	1	16.7	102.1	67.2	55783	1306970	29412049	24506767
satellite5 (3)	0.06	0.06	2.78	1.1	2817	2961	1928245	932233
satellite6 (3)	0.4	0.4	4.67	2.16	39182	39384	3311637	1763072
satellite7 (4)	2.96	2.98	31.9	11.04	246762	246902	25675038	10208671
transport2 (2)	0.01	0.02	0.2	0.12	166	290	10626	8874
transport3 (2)	3.68	12.1	42.7	33.2	27354	120256	2583063	2207614
transport4 (2)	40.4	52.2	1036	712	112824	154469	24543906	22787598
zenotravel8 (3)	0.06	0.08	0.66	0.32	725	1009	46282	26560
zenotravel9 (3)	20.8	45.4	835.7	611	227670	538352	44768124	35520708
zenotravel10 (3)	56.8	375	X	X	539895	4287493	X	X
zenotravel11 (3)	2.22	2.4	22.4	14.8	24094	26332	1490640	1126475

in their currently best plan. Once all $|\Phi| + 1$ solutions are verified, the algorithm terminates. To further optimize the algorithm, states that are irrelevant for remaining marginal searches can be pruned, reducing computational effort.

Since all $|\Phi| + 1$ are now solved on a single, albeit distributed, search space, this presents a new possibility of manipulation, having φ_i influencing the solution of $\Pi_{-\varphi_i}$. To avoid this, and to retain property (2) of the partition principle, a state arriving via message from φ_i is automatically considered to have φ_i in its participating agents, and thus never to be considered as a candidate solution for $\Pi_{-\varphi_i}$.

Empirical Evaluation

We performed an empirical evaluation on several MA planning benchmark domains taken from the International Planning Competition . We compare the computational effort required by centralized A* and our two mechanisms (centralized and distributed). We refer to the overhead of computing the centralized mechanism (compared to centralized A*) as the *cost of selfishness*, and to the computational overhead of distributing the mechanism as the *cost of privacy*. We show that empirically, these costs are not high, providing evidence of the feasibility of our methods.

Table 1 depicts the running time and the number of generated states of A* (solving the underlying classical planning problems with full knowledge) and the 3 VCG mechanisms – Centralized (Cen), Selfish-MAD-A* (Dis), and optimized multigoal Selfish-MAD-A* (OptDis). All planners were implemented on top of the Fast-Downward planning system (Helmert 2006) and use the state-of-the-art LM-cut heuristic (Helmert and Domshlak 2009). The number of agents is given in parentheses in the Problem column. We note that the two centralized configurations use the pruning method described by Nissim et al. (2012), which simulates the computational benefits of MAD-A* in centralized search. This is done so MAD-A* wouldn't have an advantage, giving an accurate view of the costs of selfishness and privacy. Running time was limited to 30 minutes and memory to 4GB.

In most cases, the cost of selfishness (overhead incurred by the centralized computation of the VCG payments) is not high – most problems are solved in roughly the same time, with a few exceptions solved 4 and 16 times slower. This is because solving the marginal problems is fairly easy once an optimal solution has been found. However, the cost of privacy is much higher – most problems are solved 4 times slower than centralized VCG, with one problem solved 18 times slower. This is mainly because the privacy of information hurts the heuristic quality, an important factor in the effectiveness of forward search (heuristic) algorithms. However, given that the distributed configurations solve a more difficult problem than the centralized ones, this slowdown is expected and acceptable. Comparing the two distributed approaches, we see that the optimized multigoal version dominates Selfish-MA-A* w.r.t. both runtime and generated nodes. It is clear that elimination of the duplicate effort has a positive effect computation-wise.

Discussion

We described a distributed, strategy-proof, socially efficient, and privacy preserving mechanism for planning by a group of self-interested agents. This result contributes to a small, but growing body of research on this topic. One important advantage of our mechanism is its efficiency. Although there is a clear computational cost to selfishness, we can optimally solve the same order of problems as state-of-the-art optimal planning algorithms. This is in stark contrast to most alternatives: work on stable planning relies on CSP-based algorithms discussed by Nissim, Brafman and Domshlak (2010), that have difficulty handling plans that require more than 3 public actions per agent and do not generate optimal plans, whereas work that seeks equilibria either does not consider the issue of privacy, assumes truthfulness ignoring the issue of manipulation, or requires costly computations.

The VCG mechanism is not without faults. Its main weakness is the the fact that the side payments that the plan beneficiary has to pay the agents can be very substantial in theory. This makes the mechanism suitable in two contexts: 1) Competitive environments where there are a number of different agents with similar capabilities and similar costs, in which case the payments agents receive are not significantly beyond their true cost. Thus, in the example of building construction, there is a large number of providers for every possible task, some more efficient or appropriate (e.g., because their proximity to the site lowers their costs), but all in the same ball park. 2) Plan beneficiary for which the benefit of optimality far outweighs the cost paid. For example, road construction by local government, where the time to complete the project outweighs other considerations. How to address the issue of overcharging is the subject of much current work on related problems (Elkind, Sahai, and Steiglitz 2004; Karlin, Kempe, and Tamir 2005; Singer 2010), although none of the current results are applicable to the planning settings. We hope to examine this question in future work.

Acknowledgments

The authors were partly supported by the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.

References

Ben Larbi, R.; Konieczny, S.; and Marquis, P. 2007. Extending classical planning to the multi-agent case: A game-theoretic approach. In *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, ECSQARU, 731–742.

Bowling, M. H.; Jensen, R. M.; and Veloso, M. M. 2003. A formalization of equilibria for multiagent planning. In *IJCAI*, 1460–1462.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, 28–35.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *IJCAI*, 73–78.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2010. Transferable utility planning games. In *AAAI*.

Chandy, K. M., and Lamport, L. 1985. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.* 3(1):63–75.

Clarke, E. H. 1971. Multipart pricing of public goods. *Public Choice* 2:19–33.

Crosby, M., and Rovatsos, M. 2011. Heuristic multiagent planning with self-interested agents. In *AAMAS*, 1213–1214.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Elkind, E.; Sahai, A.; and Steiglitz, K. 2004. Frugality in path auctions. In *SODA*, 701–709.

Ephrati, E., and Rosenschein, J. S. 1991. The Clarke Tax as a consensus mechanism among automated agents. In *AAAI*, 173–178.

Feigenbaum, J., and Shenker, S. 2002. Distributed algorithmic mechanism design: recent results and future directions. In *DIAL-M*, 1–13. ACM.

Feigenbaum, J.; Papadimitriou, C. H.; Sami, R.; and Shenker, S. 2002. A BGP-based mechanism for lowest-cost routing. In *PODC*, 173–182. ACM.

Groves, T. 1973. Incentives in Teams. *Econometrica* 41:617–631.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*.

Helmert, M. 2006. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

ICAPS. The International Planning Competition. http://www.plg.inf.uc3m.es/ipc2011-deterministic/.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *ICAPS*.

Karlin, A. R.; Kempe, D.; and Tamir, T. 2005. Beyond VCG: Frugality of truthful mechanisms. In *FOCS*, 615–626.

Nisan, N., and Ronen, A. 2001. Algorithmic mechanism design. *Games and Economic Behavior* 35(1-2):166–196.

Nissim, R., and Brafman, R. I. 2012a. Multi-agent A* for parallel and distributed systems. In *AAMAS*, 1265–1266.

Nissim, R., and Brafman, R. I. 2012b. Multi-agent A* for parallel and distributed systems. In ICAPS *Workshop on Heuristics and Search for Domain-Independent Planning*, 43–51.

Nissim, R.; Apsel, U.; and Brafman, R. I. 2012. Tunneling and decomposition-based state reduction for optimal planning. In *ECAI*, 624–629.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *AAMAS*, 1323–1330.

Parkes, D. C., and Shneidman, J. 2004. Distributed implementations of Vickrey-Clarke-Groves mechanism. In *AAMAS*, 261–268. IEEE Computer Society.

Parkes, D. C.; Kalagnanam, J.; and Eso, M. 2001. Achieving budget-balance with Vickrey-based payment schemes in exchanges. In *IJCAI*, 1161–1168. Morgan Kaufmann.

Petcu, A.; Faltings, B.; and Parkes, D. C. 2008. M-DPOP: Faithful distributed implementation of efficient social choice problems. *J. Artif. Intell. Res. (JAIR)* 32:705–755.

Singer, Y. 2010. Budget feasible mechanisms. In *FOCS*, 765–774.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *ECAI*, volume 242, 762–767. IOS Press.

Vickrey, W. 1961. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance* 8–37.

Optimizing distributed resource exchanges in multiagent systems under uncertainty

Aurélie Beynier
aurelie.beynier@lip6.fr
LIP6, University Pierre and Marie Curie
4 place Jussieu
75005 Paris, France

Sylvia Estivie
sylvia.estivie@univ-valenciennes.fr
Univ Lille Nord de France, F-59000 Lille, France
UVHC, LAMIH, F-59313 Valenciennes, France
CNRS, UMR 8201, F-59313 Valenciennes, France

Abstract

A resource allocation problem is a problem in which a set of agents - cooperative or not - must find an assignment of a set of resources. This allocation must match, as best as possible, with the agents' preferences, which are often antagonist. In most allocation problems, the solution space has a combinatorial structure that creates difficulties with the preference formal representation and the optimal choice calculation. Furthermore, agents must frequently make a decision with an incomplete knowledge of the system state while exogenous factors may affect this state after the decision. There is thus a need for the agents to handle uncertainty about the system in order to maximize their satisfaction. However, the allocation problem gets more complicated in case of uncertainty and usual models of collective resource allocation are not appropriate for this context. Whereas resource allocation models for combinatorial domains are the subject of studies, none of these models consider uncertainty aspects in a distributed context. In this paper, we propose a decision-theoretic approach that allows a set of agents to solve, in a distributed way, resource allocation problems under uncertainty. We represent possible interactions and limited observability as an interaction graph and we propose an MDP based approach to compute exchange strategies taking into account future interaction opportunities.

Introduction

In multiagent systems, the problem of the allocation of a set of resources over a set of agents is major issue. This problem consists in finding an assignment of a finite set of resources over a finite set of agents where the agents' welfare depends on the resources they own. Many application frameworks deal with the problematic of resource sharing such as networks, robotics, logistic, ambient intelligence, security,... For example, in networks (Internet, sensor networks, social networks) agents (users, softwares) must share a set of limited amounts of resources (sensors, computational resources, bandwidth, services) across the network.

In this paper, we focus on the problem of resource sharing under uncertainty. It must be recognized that very little work deals with this problematic despite the fact that it is very fundamental in many multiagent systems. Indeed, due to the topology of the system, each agent has limited observability of its environment and of the others agents. In the resource allocation framework, the agents often have a partial observability over resources owned by the others agents and they have incomplete knowledge of the valuation of the resources owned by the other agents. This limited observability leads to uncertainty over the state of the system and over its evolution, that must be taken into account while deciding how to act.

When it is not possible to have a central entity which decides of the resource allocation between the agents, the process of resource allocation must be fulfilled in a distributed way. For the agents, this process consists in computing individual policies for resource exchanges considering the other agents' states and strategies which are partially observable.

In this paper, we propose an approach that allows resource exchanges between the agents with limited observability. Given an initial resource allocation, agents will try to exchange resources in order to maximize their personal welfare considering future possible opportunities of resource exchanges. We propose to model the observation capacities of each agent as an interaction graph defining for each agent ag_i, the set of agents observed by ag_i and with whom ag_i can interact. In applicative contexts like sensor networks or social networks, this graph formalizes direct connections between the agents. In order to take into account uncertainty about the system state and evolution, we formalize each individual decision problem as a Markov Decision Process (MDP). This mathematical model has been successfully used in many domains to solve sequential decision making problems under uncertainty. In our approach, MDPs are used to compute resource exchange strategies taking into account future resource opportunities.

The remainder of the paper is organized as follows. We first introduce the MultiAgent Resource Allocation (MARA) framework. Next, we discuss issues dealing with uncertainty and MARA problems and we present Markov Decision Processes. We then propose an MDP based approach to model the problem of distributed resource allocation under uncertainty. We address the problem of policy computation and detail how each agent can anticipate resource moves to determine its best proposal at each time step. Finally, we present experiments and conclude the paper.

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Problem statement

Let $\mathcal{A}g = \{ag_0 \cdots ag_{n-1}\}$ be a finite set of (at least 2) agents and let $\mathcal{R}$ be a finite set of indivisible resources (which we also refer to as goods). An allocation A is a partitioning of the items in $\mathcal{R}$ amongst the agents in $\mathcal{A}g$ (i.e. each good must be owned by exactly one agent). For instance, an allocation A, defined via $\mathcal{R}(i) = \{r_1\}$ and $\mathcal{R}(j) = \{r_2, r_3\}$, would allocate r_1 to agent ag_i, and r_2 and r_3 to agent ag_j. Each agent $ag_i \in \mathcal{A}g$ is equipped with a valuation function v_i mapping bundles of resources (subsets of $\mathcal{R}$) to rational numbers. These valuation functions denote the interests of individual agents $ag_i \in \mathcal{A}g$.

Agents' valuation functions may be represented in different ways. In this paper, we choose to use an intuitive representation of valuation functions which is the *k-additive* functions. They are frequently used because they are simple, compact and very expressive. These functions have been studied in the context of fuzzy measure theory (Grabisch 1997). A valuation function is called k-additive iff the utility assigned to a bundle of resources $\mathcal{R}(i)$ hold by ag_i can be represented as the sum of basic utilities associated to subsets T of $\mathcal{R}(i)$ with cardinality $\leqslant k$, $k \in \mathbb{N}$. The valuation function v_i of an agent ag_i is thus defined as:

$$v_i(\mathcal{R}(i)) = \sum_{T \in \mathcal{R}(i)} \alpha_i^T$$

The agent ag_i enjoys an increase in utility of α_i^T when it owns all the items in T together. We say that bundle T becomes active for agent ag_i. For example, let's assume ag_i's valuation function is $v_i = 5r_1 + 10r_1r_2$. If ag_i only owns r_2, its utility is 0. But, if it owns r_1 and r_2, it activates both bundles r_1 and r_1r_2 and its utility is $5 + 10 = 15$. We shall make the assumption that all valuations are normalised in the sense that $v_i(\{\}) = 0$.

Deals and monetary payments

Given a particular allocation of resources, agents may agree on deals to exchange some of the resources they currently hold in order to improve their own welfare. A deal $\delta = (A, A')$ is a pair of allocations (with $A \neq A'$), specifying the situation before and afterwards. It transforms an allocation of resources A into a new allocation A'. In general, a single deal δ may involve the reassignment of any number of goods amongst any number of agents (Sandholm 1998). In this work, we focus on one-deals which involve only a single resource and hence only two agents.

We assume that agents are rational in the sense of aiming to maximise their individual welfare. According with (Estivie et al. 2006), we call a deal rational iff it results in a gain in utility (or money) that strictly outweighs a possible loss in money (or utility) for each of the agents involved in that deal. So, agents are assumed to only negotiate individually rational deals. Deals may be accompanied by monetary side payments to allow agents to compensate others for otherwise disadvantageous deals. If an agent receives several propositions for a same resource, it chooses the most advantageous rational deal for it, i.e. the rational deal with the highest value for this resource.

Interaction graph

In this paper, we consider problems where agents exchange resources in a distributed way: there is no central entity to coordinate resource allocation amongst the agents. Each agent is assumed to have limited observations about its environment, especially concerning the other agents. We consider that the agents are organized as an interaction graph $\mathcal{G} = \{\mathcal{A}g, \mathcal{E}\}$. Vertices represent the agents and edges formalized possible interactions between the agents. An edge $e(i,j) \in \mathcal{E}$ links two agents ag_i and ag_j, if agent ag_i (respectively ag_j) observes agent ag_j (respectively ag_i) and can interact with ag_j. It means that ag_i and ag_j both observe the resources they own and can exchange some resources (we do not consider asymmetric situations where ag_i observes ag_j but ag_j does not observe ag_i). All the agents ag_j such as $\exists\, e(i,j) \in \mathcal{E}$ are called the neighbours of ag_i and denoted $\mathcal{N}(i)$. This interaction graph enables to represent situations where agents have limited sensing capabilities i.e. we assume that an agent only observes and interacts with the agents that are in its range of perception. However, each agent is assumed to know the whole topology of the interaction graph (or at least its neighbourhood and the number of neighbours of its neighbours). This assumption holds in many applicative frameworks where spatial organization is common knowledge. For instance, in robotics system for multi-robot exploration, each robot is able to know which robots are closed to it and can obtain the information about the neighbours of its neighbours via wifi communication. In Web Services, each service interacts with a closed list of preferred and well-known services but it knows the whole list of services (through a yellow pages service for example). In these examples, agents only interact with a subset of favourite and well-known agents.

In our framework, it is assumed that each agent observes the resources owned by its neighbours and their valuation functions. In the case where each agent also knows the policies of its neighbours and of the neighbours of its neighbours, it can infer possible competing proposals. Otherwise, heuristics can be used to make up for the lack of knowledge about other agents' preferences and strategies. Possible heuristics are discussed in the section dealing with policy computation.

Figure 1 represents a system involving 10 agents. In this graph, agent ag_0 can only observe the resources of ag_1, ag_2 and ag_3 and only interacts with these three agents. So, agent ag_0 could see and could propose a bid for the resources r_1, r_3 and r_5, but not for r_7, r_8 and r_9.

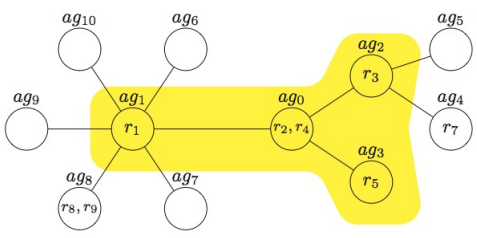

Figure 1: Graph and knowledge of agent ag_0

MARA and uncertainty

Since an agent ag_i has limited observations about the other agents (it does not observe agents ag_j such that $ag_j \notin \mathcal{N}(i)$), it is uncertain about the proposals of the other agents. In fact, the agent does not know exactly which resources are owned and wanted by the other agents. When it makes a proposal for a deal, agent ag_i is not sure that it will get the resource: it obtains the resource if it makes the highest proposal but it has uncertainty about the other agents' proposals. In this paper, we propose an approach that allows each agent to deal with the uncertainty about the other agents' proposals and to compute a policy that dictates which proposal to make and when. Due to k-additive utility functions it may be interesting to anticipate future opportunities. Indeed, an agent may not benefit immediately from getting a resource r_j but, it may get a high reward in steps further ahead once it will activate a bundle with r_j. Let's consider the graph of Figure 1 and let's assume agent ag_0 has the valuation function $v_0 = 5r_2 + 10r_8r_3r_4$. When it owns $\{r_2, r_4\}$, ag_0 has no immediate interest to propose a deal in order to obtain r_3. Indeed, ag_0 cannot activate a new bundle when it only obtains r_3: its valuation remains the same whether it owns r_3 or not (valuation equals to 5). However, if the agent anticipates that it could use r_3 later to activate on more bundle once it will obtain r_8, the agent has interest to make a proposal for r_3. Its interest has to be balanced with the probability r_8 becomes reachable in the next steps. Our approach allows each agent to anticipate future resource exchanges and to make proposals at the best price and time. In order to model the uncertainty about the other agents' proposals, we propose to formalise each agent's decision problem as a Markov Decision Process.

MDP and resource allocation

Markov Decision Processes (Puterman 2005) propose a rich mathematical framework to formalize and solve sequential decision problems in stochastic environments. A Markov Decision Process (MDP) is defined as a tuple $\langle S, \mathcal{A}, P, R\rangle$ where: S denotes the finite set of possible agent states ; $\mathcal{A}$ is the finite set of actions which can be executed ; $P : S \times \mathcal{A} \times S \to [0, 1]$ is a transition function which describes the dynamics of the system: $P(s, a, s')$ gives the probability the system moves to state s' when action a is executed from s ; $R : S \times \mathcal{A} \to \mathbb{R}$ is a reward function: $R(s, a)$ is the reward obtained by the system while executing a from s.
Optimally solving an MDP consists in finding a policy function π which maps each possible system state s to an action a and which maximizes the expected cumulative sum of reward.

Markov Decision Processes have been used to solve centralized multiagent task allocation problems where a set of tasks or resources must be allocated over a set of agents (Hanna and Mouaddib 2002; Dibangoye, Mouaddib, and Chaib-Draa 2007; Beynier and Mouaddib 2009): at each decision step, a decision maker must decide to which agents a task or a resource must be allocated. Plamondon and Chaib-Draa (Plamondon and Chaib-Draa 2006) describe an MDP-based approach for allocating a set of resources among a set of tasks: an MDP is defined for each resource and a central agent coordinates the MDPs considering resource interactions.

Dolgov and Durfee (Dolgov and Durfee 2004) tackle the problem of computing an optimal strategy for a set of agents that have to share limited resources to execute their actions under uncertainty. The agents are assumed to be independent except for the limited resources they share. The optimization problem is reduced to a mixed integer linear program which computes policies that ensure the limited shared resources are not over-consumed. In (Dolgov and Durfee 2006), Dolgov and Durfee tackle the problem of decentralized multiagent resource allocation using combinatorial auctions where bids are defined using MDPs: instead of sending a bid for each possible bundle, each agent sends to the auctioneer a resource-parametrized MDP as bids.

However, to our knowledge, there is no existing approach dealing with distributed system under uncertainty where agents must exchange resources through one-deal to optimize their satisfaction. In fact, very few approaches have been proposed to solve decentralized resource allocation under uncertainty. Recently, Lumet et al. (Lumet, Bouveret, and Lematre 2011) studied the problem of fair allocation of indivisible goods under risk. The preferences of the agents are assumed to be additive i.e. the utility of an agent is the sum of the utility of the resource it owns. Although their approach allows for considering uncertainty on the value of each resource, allocation is centralized.

Distributed MDPs for resource allocation

In the last decade, many attention has been paied to Decentralized Markov Decision Processes (DEC-MDPs) which extends MDPs to distributed multiagent decision making (Goldman and Zilberstein 2004). DEC-MDPs describe a framework to model mutiagent sequential decision problems where each agent has to make decisions based on its own partial view of the system such as to maximize a common reward function. DEC-MDPs could provide a useful tool to formalize distributed allocation problems. However, they suffer from a high complexity (Bernstein, Zilberstein, and Immerman 2002). Approximate approaches have been proposed to overcome this complexity (Amato, Dibangoye, and Zilberstein 2009; Wu, Zilberstein, and Chen 2010). However, most of them propose centralized policy computation (while we focus on decentralized policy generation) and do no exploit locality of interactions - except Networked Distributed POMDPs (ND-POMDP). Although, our work share many characteristics with the ND-POMDP framework (Nair et al. 2005), our approach considers more general problems since we do not make the assumption of observation and transition independence. Indeed, in MARA framework, there are strong dependences between neighbour agents' transition functions.
Partially Observable Stochastic Games (POISG) (Hansen 2004) generalize DEC-POMDPs to represent problems where the agents may not share the same reward function like our context. Indeed, in the problems we consider, each agent tries to maximize its own welfare.

In our approach, we propose to decompose the problem as a set of MDPs. It has been proved that such an approach allows to solve larger sizes of problems (Beynier and Mouaddib 2009; Scott and Prasad 2009) than the ones which represent the decision problem as a single multiagent MDP. An MDP is defined to formalize each agent's individual decision problem: it models the problem facing the agent about deal proposals and deal acceptances. Note that a valuation function v_i is assumed to be known by each agent ag_i to value the set of resources $\mathcal{R}(i)$ it owns. Unlike (Dolgov and Durfee 2006), MDPs are not used to compute the value of each resource or set of resources.

At each decision step, an agent must decide whether to make a proposal for a resource or not, which proposal to make and to which agent. A proposal consists of an agent (the receiver of the proposal), a resource and a price for this resource. We develop the following protocol to structure resource exchange between the agents: Agents are assumed to be synchronized, i.e. at each decision step, each agent can send a proposal to another agent. At the end of the step, each agent examines the proposals it has received and decides, for each proposal, to accept it or not. The senders of the proposals are then notify of the acceptance or not of their proposals. When a proposal is accepted, the agents proceed to the exchange. It is assumed that an agent cannot retract a proposal once it has sent it.

The decision problem of an agent ag_i can be formalized as an MDP where the components of the tuple $\langle S_i, \mathcal{A}_i, P_i, R_i \rangle$ are defined as follows:

States Each state s_i of the agent ag_i describes the observations of the agent about resource owners. For each resource owned by an agent $ag_j \in \{\mathcal{N}(i) \cup ag_i\}$, the state gives the agent which owns the resource. Obviously, some resources are not in the observation range of the agent, so the agent may have no information for some of the resources. Since the set of resources and neighbours of each agent is finite, the state space is finite.

Actions At each decision step, the agent ag_i must choose between two kinds of actions: *do nothing* (denoted by $\emptyset$) or *make a proposal* to an agent $ag_k \in \mathcal{N}(i)$ for a resource r_m at price φ (the price the agent is willing to pay r_m) . Such a proposal is denoted $\rho(r_m, \varphi, ag_k)$. An agent makes proposals for the resources owned by its neighbours that could increase its welfare.

Several prices can be considered for a resource r_m. Possible prices φ, are computing using valuation functions v_l of the agents $ag_l \in \mathcal{A}g$ (agent may have different valuation functions). In fact, the coefficients of the bundles, in the valuation functions, which contain r_m are considered as possible prices φ. Beside, for each price φ (except the maximum price), we add an ϵ which is a very small value that ensures the owner of r_m will accept the deal if it values r_m at φ. Price $0 + \epsilon$ is also considered as a possible price for dealing with situations where the owner of r_m has no interest towards r_m.

Let's consider the possible actions of agent ag_0 on Figure 2. The system involves 5 agents and 3 resources r_1, r_2, r_3. Resource r_3 is hold by ag_4 or ag_3 (ag_0 does not observe the owner of r_3 which is represented by the character ? on the Figure). The valuation function of all the agents is assumed to be $v_i = 5r_1 + 10r_1r_2$ except for ag_4 whose valuation function is $v_4 = 6r_1r_3$. Possible prices for r_1 are deduced from these functions. For each bundle containing r_1, agent ag_0 will propose for r_1 the coefficient values associated to these bundles and also 0. So in this example, agent ag_0 may propose for r_1: $0 + \epsilon$, $5 + \epsilon$, $6 + \epsilon$, $10 + \epsilon$. These different prices imply different probabilities for agent ag_0 to obtain r_1. Note that coefficients of other agents' valuation functions allow agent ag_0 to deduce competing price. Nevertheless, to deduce possible prices, the agent only needs to know the valuation functions of its neighbours and of the neighbours of its neighbours (i.e. all the valuation functions on Figure 2, a subset of the valuation functions in the general case).

Transition function The transition function gives the probability agent ag_i moves from a state s_i to a state s'_i . Next state s'_i depends of the deals fulfilled by agent ag_i and by its neighbours $ag_j \in \mathcal{N}(i)$. Since agent ag_i only observes the resources and utility functions of its neighbours, it cannot predict with certainty which proposals will be made to its neighbours by the agents $ag_k \notin \mathcal{N}(i)$ to the agents $ag_j \in \mathcal{N}(i)$. For instance, on Figure 2, agent ag_0 cannot predict with certainty the proposals agents ag_4 will make to agent ag_1. Given a state s_i and an action a_i, it thus difficult to predict which resources will be owned by agent ag_i and by its neighbours at the next decision step. However, when probabilities on resource moves and locations can be estimated, transition probabilities can be deduced. Next section presents heuristic approaches to estimate these probabilities.

Reward function The reward function is defined over states using the utility function v_i of the agent ag_i. The reward for being in a final state s_i is thus the sum of valuations of the bundles that ag_i owns in s_i. A final state is a state where there is no more possible exchange. Other states have a reward of 0.

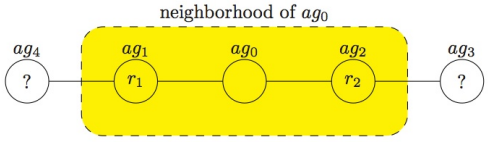

Figure 2: Graph and knowledge of ag_0

Distributed policies for resource exchanges

Two kinds of information are crucial for an agent ag_i to compute its strategy:

- which proposals made by the other agents would compete with its proposals i.e. which is the best price to offer for a resource r_j in order to obtain this resource,
- how resources would move in the interaction graph i.e. which resources could not be available any more at the next decision step and which resources could appear in the neighbourhood in the next decision steps.

Therefore, an agent must decide which proposals (an agent, resource and price) it should make and in which order to

maximize its performance criterion. Although an agent ag_i knows which resources are owned by its neighbours $\mathcal{N}(i)$, it does not know which resources are owned by each agent $j \notin \{\mathcal{N}(i) \cup ag_i\}$ and which resources are desired by these agents. In Figure 1, agent ag_0 knows that agent ag_1 has r_1, agent ag_2 has r_3 and agent ag_3 has r_5. Nonetheless, agent ag_0 does not know which resources are owned by agents ag_k with $k \in [4, 10]$ and it cannot easily predict which proposals could be made by these agents to its neighbours. Moreover, it is uncertain about the resources that would be owned by its neighbours at the next steps. In this section, we propose heuristic approaches that allow an agent to compute probabilities on resource moves and other agents' proposals. These probabilities are then used to define the transition function of the previously presented MDP.

Heuristics for estimating transition probabilities

From its observations, an agent ag_i knows which resources are not owned by its neighbours and can deduce which resources r_k are owned by the agents $ag_j \notin \{\mathcal{N}(i) \cup ag_i\}$. For each resource r_k whose position is unknown, agent ag_i assumes that r_k has uniform probability to be owned by each agent $ag_j \notin \{\mathcal{N}(i) \cup ag_i\}$, that is for each ag_j, the probability agent ag_j has r_k is $\frac{1}{|\mathcal{A}g - \{\mathcal{N}(i) \cup ag_i\}|}$. Let's consider the interaction graph of Figure 1. From its observations, agent ag_0 knows the positions of r_1, r_2, r_3, r_4 and r_5. The position of r_7 is unknown and ag_0 deduces that r_7 has a probability of $\frac{1}{7}$ to be owned by agent ag_4.

Once probabilities on resource positions are estimated, it is possible to deduce probabilities on resource moves. Since an agent only makes deals with its neighbours, a resource can only move on an edge in the interaction graph. Moves depend of the agents' valuation functions (i.e. the preferences of the agents) and their individual policy. Indeed, even if an agent knows the valuation functions of the other agents, it may not be able to determine whether its proposals will be accepted or not. For instance a neighbour agent which applies a myopic strategy will accept different proposals from an agent which uses an MDP-based policy (myopic agents only consider immediate gain whereas MDP-based policies consider future opportunities).

If each agent knows the other agents' utility functions and policies, it will be able to deduce all the other agents' proposals. However, an agent only needs to anticipate competing proposals, i.e. proposals made by the neighbours of its neighbours. Thus, knowing the utility functions and policies of these agents is sufficient[1].

When an agent ag_i has not this knowledge, the following heuristics can be used:

h1) agent ag_i assumes that each resource moves to a neighbour agent with uniform probability (probabilities independent of valuation functions and policies). The probability a resource r_i owned by ag_i moves to an agent ag_k is $\frac{1}{|\mathcal{N}(i)|}$ if $ag_k \in \mathcal{N}(i)$, 0 otherwise. However, probabilities on resource moves computed using this method are poor quality and do not allow the agents to compute efficient policies of actions (see experimental section).

h2) agent ag_i assumes all the other agents have the same valuation function as its valuation and the other agents are myopic i.e. they only expect immediate gain and do not consider long term effect of their actions (and possibly future higher gain). Thus, a myopic agent ag_j would accept a proposal $\rho(r_m, \varphi, ag_j)$ from an agent ag_k if its loss of utility (difference between its utility when it has r_m and its utility when it has not r_m) is less than φ.

h3) When ag_i knows the valuation functions but does not know the policies of the other agents, it can be assumed that all the other agents are myopic.

Let's consider the point of view of agent ag_0 in the interaction graph of Figure 1. Agents are assumed to have the valuation function $50r_1r_2r_3r_4 + 25r_1r_8 + 30r_1r_2r_4 + 10r_1r_3 + 40r_3r_7$. We exemplify how probabilities can be inferred from the point of view of agent ag_0 when all the agents are assumed to be myopic (except ag_0) and to have the same valuation function (heuristic h2). In order to maximize its performance, agent ag_0 desires to get r_1 and r_3. Agent ag_0 assumes that all the other agents have the same utility function and deduces that r_1 could be desired by the agents which own r_8. If r_8 is owned by agents ag_5 or ag_4, these agents won't try to obtain r_1 since r_1 is not in their neighbourhoods. If r_8 is owned by one of the agents ag_6 to ag_{10}, one of these agents will certainly make a bid to agent 1 for r_1. Since having r_1 and r_8 allows an agent to get a satisfaction of 10 and r_8 does not belong to another bundle, agent ag_0 knows that agents ag_6 to ag_{10} highest proposal for r_8 would be 10. Moreover, agent ag_0 can deduce, from the common valuation function, that agent ag_1 will not receive other proposals for r_1 (there is no other agent that needs r_1 to achieve a bundle). Given a state s, agent ag_0 can therefore deduce, for each possible proposal (action), the probability to obtain the resource and to move to a next state s'.

These heuristics allow us to compute probabilities on resource moves and other agents' proposals. We can therefore deduce transition probabilities of the previously presented MDP model. This individual MDP can then be solved using classical methods for finite-horizon MDPs (Puterman 2005). An optimal solution is obtained if the others agents are myopic and the range-2 (neighbours and neighbours of neighbours) valuation functions are well known, i.e. heuristics accurately formalize the strategies of competing agents. Otherwise, an approximation of the optimal solution is computed. The quality of this solution depend of the distance between the heuristics that are used and the other agent's policies.

Elimination of dominated strategies

Note that time performance of optimal policy computation can be improved by eliminating dominated strategies. Let's consider an agent ag_i that can propose 3 different prices φ^1, φ^2 and φ^3 (such as $\varphi^1 < \varphi^2 < \varphi^3$) for a resource r_j. If the agent is sure to get the resource for φ^1 (probability equal to 1 to obtain the resource), it is useless to develop strategies that propose to pay φ^2 or φ^3 for the resource: these strategies get worse results than the one which proposes r_j for φ^1. Elimination of dominated strategies and policy computation is exemplified below.

[1] It is realistic to know valuation functions but policies are hardly ever known

Illustrative example

In this section, we detail on a example how an agent computes its MDP-based strategy. Let's consider the interaction graph presented in Figure 2. The system involves 5 agents where ag_0 to ag_3 have the same valuation function $v_{i\in[0,3]} = 5r_1 + 10r_1r_2$ and ag_4 has the valuation function $v_4 = 6r_1r_3$. We consider the point of view of agent ag_0. Agent ag_0 only observes the resources of agent ag_1 and agent ag_2, it does not know which resources are owned by agent ag_3 and agent ag_4. Initially, we assume that agent ag_0 is in a state where it has no resource and it observes that agent ag_2 has r_2 and agent ag_1 has r_1. Agent ag_0 has not information on the owner of r_3, so it does not know whether the initial state of the system is H_1 or H_2. To develop our example, we consider the case where ag_0 knows all the valuation functions of the other agents and assumes that other agents are myopic (heuristic **h3**).

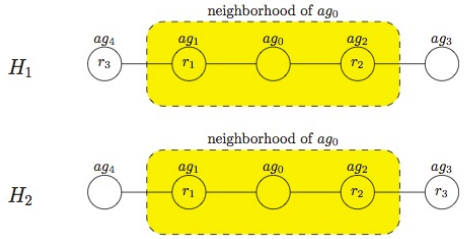

Figure 3: Feasible configurations

Figure 4 shows the possible executions of the agent from its initial state considering the possible system states H_1 and H_2. Heuristics on resource locations presented previously make the assumption that these configurations have uniform probability. While computing its optimal policy, ag_0 selects the policy with the highest expected utility. Expected utility is defined by the gain of the final states that can be reached weighted by the probability to reach these states.

The agent ag_0 can try first to obtain r_1 and r_2. From the observations of the value functions, the agent deduces that r_1 and r_2 would allow it to activate 2 bundles: $5r_1$ and $10r_1r_2$. Agent ag_1 would therefore be interested in getting these resources. Note that agent ag_2 does not know where is r_1 so it won't try to obtain it for the moment and it cannot activate any bundle having just r_2. Agent ag_1 activates only one bundle with r_1 (bundle $5r_1$), so agent ag_0 should propose more than 5 to obtain r_1.

Agent ag_0 must decide whether to first make a proposal for r_1 or r_2 and it must decide which price to propose. As explained previously, possible prices for a resource r_i are deduced from the values of the bundles where r_i appears (valuation functions of ag_0, of neighbours of ag_0 and of neighbours of neighbours of ag_0 are considered in this case). Thus, agent ag_0 can propose $0+\epsilon$, $5+\epsilon$, $6+\epsilon$ or $10+\epsilon$ for r_1, and it can propose $0+\epsilon$ or $10+\epsilon$ for r_2. The action set of the agent is thus: $\{\emptyset, \rho(r_1, 0+\epsilon, ag_1), \rho(r_1, 5+\epsilon, ag_1), \rho(r_1, 6+\epsilon, ag_1), \rho(r_1, 10+\epsilon, ag_1), \rho(r_2, 0+\epsilon, ag_2), \rho(r_2, 10+\epsilon, ag_2)\}$. Figure 4 describes the different possible actions of the agent and their possible outcomes:

- If agent ag_0 decides to do nothing, it neither obtains r_1 nor r_2.
- If it proposes $0+\epsilon$ for r_1 it does not obtain r_1. Indeed, agent ag_1 obtains a value of 5 with the bundle $5r_1$ so such a deal would lead to a loss in value for agent ag_1.
- If agent ag_0 proposes $5+\epsilon$ for r_1 (❶ on Figure 4) it is sure to obtain r_1 if there is no competing offers sent to ag_1 (agent ag_1 is assumed to be myopic and obtains a gain of ϵ with this exchange). Nonetheless, if r_3 is hold by ag_4, ag_4 will make a proposal for r_1 at 6 and ag_0 will not obtain the resource. If r_3 is hold by ag_3, no competing offers will be made. The probability ag_4 has r_3 is 0.5 so ag_0 has probability 0.5 to obtain r_1 with an offer at $5+\epsilon$.
- If agent ag_0 proposes $6+\epsilon$ for r_1 (❷ on Figure 4), it is sure to obtain the resource even if ag_4 holds r_3.
- Proposing $10+\epsilon$ for r_1 (❸ on Figure 4) also allows agent ag_0 to obtain r_1 (but this strategy is dominated by ❷).
- Agent ag_0 can also propose $0+\epsilon$ for r_2 (❹) or $10+\epsilon$ (❺). When it proposes $0+\epsilon$, it is sure to obtain r_2 since agent ag_2 does not have r_1 and cannot obtain it from its neighbourhood. Exchanging r_2 for $0+\epsilon$ thus increases the satisfaction of agent ag_2 by ϵ.
- If agent ag_0 proposes $10+\epsilon$ for r_2, it is sure to obtain the resource (dominated strategy).

Once agent ag_0 has executed its first action, it can move to different system configurations which are illustrated in Figure 5. If agent ag_0 chooses action $\rho(r_1, 5+\epsilon, ag_1)$, it moves to configurations C_1 or C_5 (depending of the owner of r_3). If it proposes $6+\epsilon$ for r_1, it always obtains the resource (it moves to configurations C_1 or C_6 whose observations are similar for ag_0). If it chooses $\rho(r_2, 0+\epsilon, ag_2)$, two cases are possible: C_3 and C_4. The agent moves to C_4 if ag_4 holds r_3 (ag_4 has made an offer for r_1 at the first decision step and obtained r_1). The agent ag_0 moves to C_3 if ag_4 does not hold r_3 and thus no offer was made at the first decision step for r_1. Cases C_3 and C_4 are assumed to have uniform probability (due to heuristic on resource locations and heuristic $h3$). Note that observing all value functions allows agent ag_0 to deduce that case C_2 described on Figure 5 is impossible since agent ag_3 has no interest to get r_2. For each possible next configuration, a second action is performed.
Once it has obtained r_1, agent ag_0 can try to obtain r_2 (for $0+\epsilon$ or $10+\epsilon$). If it has r_2, it can try to obtain r_1 if it is still reachable (case C_3). In case C_4, r_1 is no more reachable and the only possible action for the agent is "do nothing".

Then, ag_0 moves to terminal states, i.e. states where it has no more offers to make. From terminal states, the value of each strategy can be computed by applying Bellman Equation.The best strategy therefore consists in first proposing $6+\epsilon$ for r_1 and then proposing $0+\epsilon$ for r_2. In fact, it is more valuable to first try to obtain r_1 since this resource could be unreachable at the next step. Then, the agent can obtain r_2 which will still be reachable (no neighbour of ag_1 -except ag_0- has interest to make a proposal for r_2 at the first step[2]).

[2]This proposition holds since the agents are not competitive. In competitive settings, one agent can try to obtain a resource r_j to prevent a competitive agent from owning r_j .

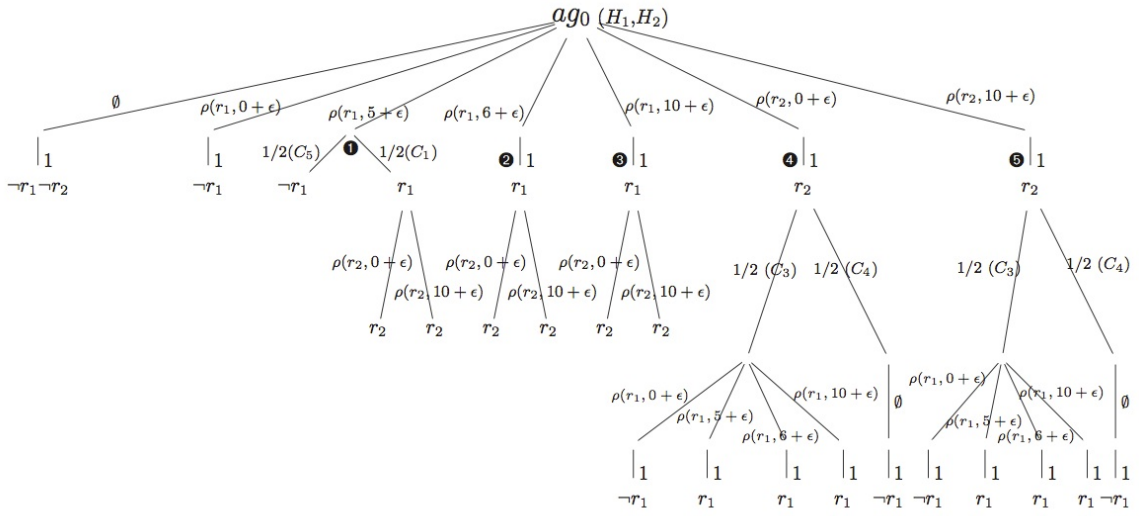

Figure 4: Possible policies of agent ag_0

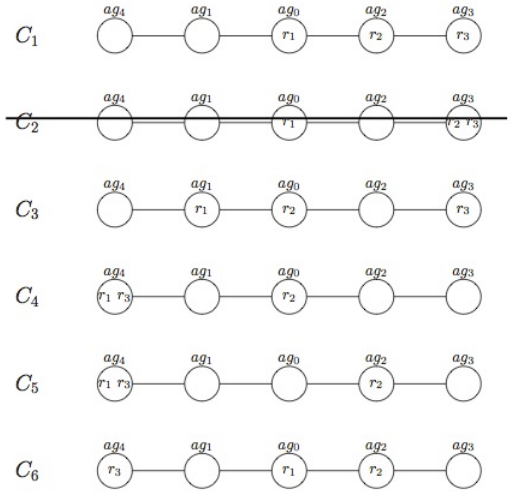

Figure 5: Possible system configurations at the second time step

Policy computation and execution

In this section, we briefly describe policy computation algorithms and action execution protocol used by the agents while exchanging resources. Before execution, each agent computes its exchange policy. For each agent using our approach, its decision problem from an initial state s_0 is automatically formalized with an MDP. This MDP is solved using *value iteration* algorithm (Puterman 2005). The state and action spaces are defined by listing all reachable states from s_0 and computing the set of available actions from each state. The transition function is computed using heuristics on resource locations and moves. The reward function is defined from the valuation function of the agent ag_0.

Once each agent has computed its policy, policies are executed (Algorithm 1). During the execution, agents are assumed to be synchronized. At each step of the decision process, each agent chooses, from its policy, an action to execute i.e. an offer make or "do nothing". Each offer $\rho(r_m, \varphi, ag_k)$ is sent to the receiver (ag_k) of ρ. For the agent using our approach, this action is selected from the optimal policy given by the MDP of the agent. Other kinds of agents follow their offer policy (for instance, myopic strategy).
Each agent has a mailbox where it receives the offers from the others agents. Once all the offers are made, each agent selects the best rational offer received i.e. the best offer resulting in an increase of utility. Deals for best offers are concluded and resources are exchanged. Then, the agents move to the next decision step and make new offers,... This process repeats until there is no more exchange of resources.

Algorithm 1 AgentBehaviour

```
strategy ← ComputeStrategy();
end ← 0
while (end ≠ null) do
    currentState ← Observation(system)
    Execute(action(currentState, strategy))
    ρ* ← Select Best rational offer from the mailbox
    Exchange resources following ρ*
    end ← ValueEnd()
end while
```

Experiments

We describe experiments on the problem of Figure 2. We developed algorithms to deduce transition probabilities from observations and valuation functions. MDPs were solved using *value iteration* algorithm (Puterman 2005) with $\gamma = 0.8$. ϵ was fixed to 0.02 (this value does not influence the results if it is strictly positive). We tested our approach in the following settings:

- **E1**: Agent ag_0 uses our MDP-based approach to compute its strategy assuming uniform probabilities on resource moves. During the execution, ag_0 follows its MDP strategy and all the other agents are myopic.
- **E2**: Agent ag_0 uses our MDP-based approach to compute its strategy assuming the other agents are myopic. ag_0 knows all the other agents valuation functions. During the execution, ag_0 follows its MDP strategy and all the other agents are myopic.
- **E3**: Agent ag_0 uses our MDP-based approach to compute its strategy assuming the other agents are myopic. ag_0 knows the valuation function of its neighbours and assumes all the other agents have the same valuation functions as itself. During the execution, ag_0 follows its MDP strategy and all the other agents are myopic.
- **E4**: Agent ag_0 and all the other agents are myopic.
- **E5**: All the agents follow an MDP-based strategy.

For each case of experiments, we registered the average gain of ag_0 (valuation of its final state) and its average profit (final gain minus the amount of money spent during exchanges to obtain resources) (see Table 1).

	Gain	Profit
E1	0	0
E2	15	9-ϵ
E3	0	0
E4	0	0
E5	5	-1 -ϵ

Table 1: Gain and profit on example of Figure 2

While assuming uniform probability on resources moves (experiments **E1**), agent ag_0 never gets r_1. Since probabilities on resource moves are assumed to be uniform, transition probabilities are uniform whatever the price proposed for a resource. ag_0 thus chooses to propose $0 + \epsilon$ for r_1 (higher prices are assumed to lead to the same result) and never obtains the resource. This case of experiments represents situations where the agent has no knowledge on the other agents valuation function. It can thus deduce nothing on other agents preferences towards resources.

When the agent has no knowledge about valuation functions of the others, it could assume that all the other agents have the same valuation function as its function. This is the case in experiments **E3**. If the other agents do have the same valuation functions, gain and profit are the same as the ones obtained in **E2**. Otherwise, performances are related to the difference between assumed valuation functions and real valuation functions. On our example, if ag_4 has the valuation function $v_4 = 5r_1 + 10r_1r2$, ag_0 gains 15 and its profit is $10 - \epsilon$. If ag_4 has the valuation function $v_4 = 6r_1r_3$, the gain and the profit of ag_0 are 0.

Best results are obtained when ag_0 applies the policy computed by our approach (experiments **E2**). Its optimal policy dictates to first propose $6+\epsilon$ for r_1 (it is sure to obtain the resource whatever the location of r_3), and then propose $0+\epsilon$ for r_2. This policy guarantees ag_0 to obtain both r_1 and r_2 whatever the initial configuration ($H1$ or $H2$). Thus, the agent always obtains a final gain of 15 and always spends $(6 + \epsilon)$. Note that the optimal policy of ag_0 dictates to first make a proposal for r_1 (not r_2). Indeed, policy computation takes into account the fact that ag_4 may make a proposal for r_1 at the first decision step and this resource will become unreachable if ag_0 does not make a competing proposal. Since, there is no risk r_2 moves, ag_0 starts with an offer for r_1.

Finally, we experimented our approach assuming all the agents compute an MDP-based strategy. While computing its strategy, each agent assumes that the others follow a myopic strategy (which is not the case during policy execution). Agent ag_0 starts with an offer for r_1 at $6 + \epsilon$ to ag_1 and always get the resource since ag_4 has no interest in making a higher proposal and ag_1 values r_1 at 5. Then, ag_0 proposes $0+\epsilon$ to ag_2. While making this proposal, ag_0 assumes ag_2 is myopic. But, ag_2 follows a similar MDP-based strategy which dictates it to make a proposal to ag_0 for r_1. That leads to a deadlock between the two selfish agents ag_0 and ag_1 which both want the resource owned by the other. One way to solve this problem is to use negotiation during exchange execution while detecting deadlocks from the offers received and the offer made. However, negotiation phases would not be taken into account during off-line policy optimization. Another solution we are currently considering, consists in coordinating the agents during policy computation and allowing the agents to exchange *value of interest*.

Conclusion

In this paper, we tackled the problem of distributed resource allocation under uncertainty. We first formalized possible agent interactions as a graph. This graph is generic and allows us to represent problems where each agent can exchange with all the other agents, as well as problems with restricted interactions. We proposed to formalize the decision problem of an agent as an MDP and we defined each component of the model. We formalized possible interactions between the agents in the transition function using some heuristics on possible resource locations and moves. These heuristics are used to tackle the lack of observability about resources and valuation functions of the other agents.

We described experimental results obtained on the problem of Figure 2. When an agent ag_i applies our method and all the other agents follow myopic strategies, ag_i is able to obtain optimal gain. Experiments show that our approach allows to anticipate resource moves and to decide which is the best proposal to make at the best time. Future works will consist in solving deadlock situations and coordinating the agents when all of them follow an MDP strategy. We expect that off-line communication of *values of interest* would allow the agents to avoid such situations and to obtain higher profits. We also plan to study connections with combinatorial auctions thus allowing an agent to make an offer for a set of resources instead of making an offer for a single resource at a time.

References

Amato, C.; Dibangoye, J. S.; and Zilberstein, S. 2009. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2–9.

Bernstein, D.; Zilberstein, S.; and Immerman, N. 2002. The complexity of decentralized control of mdps. In *Mathematics of Operations Research*, 27(4):819–840.

Beynier, A., and Mouaddib, A. 2009. Decentralized decision making process for document server networks. In *Proceedings of the International Conference on Game Theory and Networks*.

Dibangoye, J. S.; Mouaddib, A.; and Chaib-Draa, B. 2007. Periodic real-time resource allocation for teams of progressive processing agents. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 07)*, 115.

Dolgov, D. A., and Durfee, E. H. 2004. Optimal resource allocation and policy formulation in loosely-coupled Markov decision processes. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 04)*, 315–324.

Dolgov, D. A., and Durfee, E. H. 2006. Resource allocation among agents with mdp-induced preferences. *Journal of Artificial Intelligence in Research (JAIR)* 27:505–549.

Estivie, S.; Chevaleyre, Y.; Endriss, U.; and Maudet, N. 2006. How equitable is rational negotiation? In *Proc. AAMAS-2006*. ACM Press.

Goldman, C., and Zilberstein, S. 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research* 22:143–174.

Grabisch, M. 1997. k-order additive discrete fuzzy measures and their representation. *Fuzzy Sets and Systems* 92:167–189.

Hanna, H., and Mouaddib, A. 2002. Task selection as decision making in multiagent system. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 02)*, 616–623.

Hansen, E. A. 2004. Dynamic programming for partially observable stochastic games. In *Proceedings of the ninetheenth national conference on Artificial Intelligence (AAAI)*, 709–715.

Lumet, C.; Bouveret, S.; and Lematre, M. 2011. Fair division of indivisible goods under risk. In *Proceedings of the IJCAI-2011 Workshop on Social Choice and Artificial Intelligence*.

Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed pomdps: a synthesis of distributed constraint optimization and pomdps. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 1*, AAAI'05, 133–139.

Plamondon, P., and Chaib-Draa, B. 2006. A multiagent task associated mdp (mtamdp) approach to resource allocation. In *Proceedings of AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*.

Puterman, M. L. 2005. *Markov Decision processes : discrete stochastic dynamic programming*. New York: Wiley-Interscience.

Sandholm, T. W. 1998. Contract types for satisficing task allocation: I Theoretical results. In *Proceedings of the AAAI Spring Symposium: Satisficing Models*.

Scott, P., and Prasad, T. 2009. Solving multiagent assignment markov decision processes. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, 681–688.

Wu, F.; Zilberstein, S.; and Chen, X. 2010. Point-based policy generation for decentralized POMDPs. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, 1307–1314.

Coordinating Stochastic Multi-Agent Planning in a Private Values Setting*

Joris Scharpff and **Matthijs T.J. Spaan** and **Leentje Volker** and **Mathijs de Weerdt**
{j.c.d.scharpff, m.t.j.spaan, l.volker, m.m.deweerdt}@tudelft.nl
Delft University of Technology, The Netherlands

Abstract

Coordinating multi-agent planning in contingent environments, in which a center seeks to optimise the joint plan while considering the (contradicting) preferences of autonomous agents, poses a difficult challenge. A typical approach in such a setting models the planning problem of each agent as a Markov Decision Process and solve the joint problem to obtain a socially efficient policy, in which no agent can be better off without harming others. Nevertheless this approach relies on the assumption that the agent preferences (rewards) are public knowledge. We consider settings in which the agents are selfish and their preferences are private knowledge.

Illustrated by the real-world use-case of infrastructural maintenance planning, we propose a combination of stochastic planning and dynamic mechanism design to find efficient joint policies when agent preferences are private knowledge. Through the use of a payment mechanism we motivate agents to report their preferences truthfully such that we can develop socially optimal policies.

The main contributions of this work are: 1) multi-agent coordination on a network level through a novel combination of planning under uncertainty and dynamic mechanism design, applied to real-world problems, 2) accurate modelling and solving of maintenance-planning problems and 3) empirical exploration of the complexities that arise in these problems. We introduce a formal model of the problem domain, present experimental insights and identify open challenges for both the planning and scheduling as well as the mechanism design communities.

1 Introduction

The planning of maintenance activities on large infrastructural networks, such as a national highway network, is a challenging real-world problem. While improving the quality of the infrastructure, maintenance causes temporary capacity reductions of the network. Given the huge impact of time lost in traffic on the economic output of a society, planning maintenance activities in a way that minimises the disruption of traffic flows (commonly referred to as *social cost*) is an important challenge for the planning and scheduling field. In this paper, we address this challenge by a novel combination of stochastic multi-agent planning, captured in Markov Decision Processes (MDPs), and dynamic mechanism design.

A powerful real-world example of the benefits of careful maintenance planning is the summer 2012 closure of the A40 highway in Essen, Germany. Instead of choosing for the default option of restricting traffic to fewer lanes for 2 years, authorities fully closed off a road segment for 3 months and diverted traffic to parallel highways. Traffic conditions on the other highways hardly worsened, while an estimated €3.5M in social costs due to traffic jams were avoided (besides lowering building costs) (Der Spiegel 2012).

As maintenance activities often have an uncertain duration due to delays in construction, it is important to take uncertainty into account while planning. Also, there may be multiple ways to perform a certain maintenance action by varying the amount of resources dedicated to it, leading to options that have different duration, cost, risk and quality impact. Furthermore, long-term planning is required to ensure overall network quality. Assuming these uncertainties are known beforehand, as in this work, Markov Decision Processes (MDP) provide a suitable framework to model and solve these types of planning-under-uncertainty problems (Puterman 1994). A complicating factor, however, is that while a single public road authority is responsible for the quality, throughput and costs of the network, the actual maintenance is performed by autonomous agents (the contractors), typically third-party companies interested primarily in maximising their profits. Road authorities face the problem of aligning objectives; we introduce monetary incentives for the contractors to consider global objectives. Nonetheless, an agent servicing one part of the network also influences agents in other parts as its work has a negative impact on the traffic flow. As a consequence, such congestion based payments may lead to very high throughput penalties for all agents if their maintenance plans are not coordinated on a network level.

In this work we focus on socially optimal joint maintenance planning that maximises the sum of contractor utilities, in the presence of such monetary incentives, and therefore we have chosen a centralised coordination approach. The authority is given the responsibility to develop socially optimal plans, while considering the individual interests of

*This paper is a slightly adapted workshop version of the one to be presented at the Novel Applications track of the ICAPS conference.

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

all contractors expressed through cost functions. However, as these cost functions are private information, optimal coordination and hence outcomes can only be achieved if the contractors report these costs *truthfully*. Ensuring this truthfulness is the key motivation to combine stochastic planning with mechanism design.

Our main contribution is the application of a combination of stochastic planning and dynamic mechanism design to realise truthful coordination of autonomous contractors in a contingent, private-values setting. We focus on *dynamic mechanisms* that define payments over all expected outcomes such that in expectation it is in the agent's best interest to be truthful during the entire plan period. Applying dynamic mechanism design to (real-world) settings is relatively unexplored territory (Cavallo 2008).

Related Work Other approaches towards solving the problems discussed here have been considered, although they can not be applied to our setting for various reasons. Multi-agent MDP (Boutilier 1996) assumes cooperative agents that are willing to disclose private information and share the same utility function. In decentralised MDPs (Bernstein et al. 2002), although execution is decentralised, agents are still assumed to be cooperative and solving Dec-MDPs requires knowledge of all utility functions. Both methods are not suitable when agents misreport their private information to 'cheat' the center into different outcomes. Non-cooperative settings have been studied in the classical planning literature (Brafman et al. 2009; Jonsson and Rovatsos 2011; van der Krogt, de Weerdt, and Zhang 2008), but uncertainty is not addressed.

Multi-machine scheduling has also been considered for the planning of maintenance activities, but we found this infeasible for our contingent setting. The only work we are aware of in this area is by (Detienne, Dauzère-Pérès, and Yugma 2009), in which only non-decreasing regular step functions are considered. In our problem agents could both profit as well as suffer from concurrent maintenance, therefore cost functions do not have the non-decreasing property.

Another interesting related approach is that of reinforcement learning (Kok et al. 2005; Melo and Veloso 2009) and in particular Collective Intelligence (Wolpert, Tumer, and Frank 1999). In this approach agents learn how and when to coordinate and, in the case of collective intelligence, strive to optimise a global goal, without substantial knowledge of the domain model. Nevertheless, as these methods cannot provide theoretical guarantees concerning the incentives, they are not adequate in the presence of strategic behaviour, i.e., agents deliberately trying to manipulating the system.

Although stochastic planning has been well studied, only a handful of papers address dynamic mechanism design and/or a combination of the two. Bergemann and Valimaki (2006) proposed a dynamic variant of the VCG mechanism for repeated allocation, implementing the mechanism desiderata in a within-period, ex-post Nash equilibrium. Athey and Segal (2007) studied a dynamic variant of the AGV mechanism (d'Aspremont and Gérard-Varet 1979) that is budget-balanced in the weaker Bayes-Nash equilibrium solution concept. Highly related is the work by Cavallo, Parkes, and Singh (2006), in which the authors also study dynamic mechanism design to obtain desirable outcomes in multi-agent planning with private valuations. However, the focus is on allocation problems that can be modelled as multi-armed bandit problems, instead of the richer problem domains with dynamic states that we consider. Considering the complexity of the stochastic planning problem we study here, approximation of the planning also seems a viable approach. When resorting to approximate solutions, however, standard theory for strategy-proof mechanisms does not immediately apply (Procaccia and Tennenholtz 2009).

Outline In the next section, we present a theoretical framework for maintenance planning obtained and refined through interviews and discussions with public road and rail network authorities, as well as several of the larger contractors. We then introduce the theoretical background of both stochastic planning and mechanism design (Section 3), and show how to combine work on planning with uncertainty and dynamic mechanism design to solve two example applications, derived from practice (in Section 4). We present experimental insights where we compare this approach with uncoordinated agents and best-response playing agents (Section 5). We conclude with a summary of our findings and we present open challenges for both the planning and scheduling as well as the mechanism design communities (Section 6).

2 Maintenance Planning

Commonly in infrastructural maintenance planning there is one (public) institution responsible for the network on behalf of the network users. This road authority is given the task to maintain a high (i) network quality and (ii) throughput (iii) at low costs (although other objectives are also possible, e.g., environmental concerns, robustness). To this end, network maintenance has to be performed with minimal nuisance. However, the actual maintenance is performed by several autonomous, independent contractors and therefore some coordination of maintenance activities is required.

In the infrastructural maintenance planning problem (Volker et al. 2012) we are given a network of roads E. On this network we have a set N of agents (the contractors), with each agent $i \in N$ responsible for the maintenance of a disjoint subset $E_i \subseteq E$ of roads over a set of discrete periods T. An edge $e_k \in E$ has a quality level $q_{e_k} \in [0, 1]$ and a function $\hat{q} : q \times T \to q$ that models the quality degradation of a road given the current state and time (new roads degrade less quickly, seasons influence degradation, etc.).

For each edge $e_k \in E_i$, an agent i has a set of possible maintenance activities A_k that have been identified and assigned in the aforementioned procurement phase. We write A_i to denote all possible activities by an agent i, i.e., $A_i = \cup_{\{k|e_k \in E_i\}} A_k$. Each of the activities $k \in A_k$ has a duration $d_k \in \mathbb{Z}^+$, a quality impact function $\Delta q_k : q_{e_k} \times T \to q_{e_k}$ that depends on the current road quality and time, and a constant revenue $w_k \in \mathbb{R}$ that is obtained upon completion of the activity. Moreover, the agent has a (private) cost function $c_i : A_i \times T \to \mathbb{R}$ that represents the cost of performing an activity $k \in A_i$ at time $t \in T$. The dependency on time

enables modelling of different costs for example for different seasons, or for periods in which the agent has fewer resources available. We model the limited resources (machinery, employees, etc.) available to an agent by allowing at most one activity at a time. This restriction does not have much impact on the model we propose here but does greatly simplify resource reasoning and therefore the complexity of finding optimal maintenance plans.

Each agent strives to plan their maintenance activities in such a way that its profits are maximised, but plan execution is unlikely to be perfect. Uncertainties in various forms – for example delays, unknown asset states, failures – may be encountered during execution and and hence fixed plans might lead to rather poor results. To this end we focus on contingent plans, or *policies*, that dictate the best action to take *in expectation* for all possible agent states. Note that actions here are operations available to the contractors (e.g., start activity, do nothing) and states contain all relevant information for its planning problem. We formalise these concepts in Section 3.1, for now it is sufficient to know that we can always observe what activity has been performed by each contractor. We denote the observed activities by P_i and use $P_i(t) = k$ to denote that activity k was performed at time t. Each activity has to be completed before another can be started, therefore there must be exactly d_k time steps for which P_i returns k. Note that an agent can also choose to perform no activity during a time step, which we denote by $P(t) = \circ$ and we assume $\forall t \in T : c_i(\circ, t) = 0$.

Given performed activities P_i, the total revenue W_i agent i will receive is the sum of all w_k for all completed activities k. The total maintenance cost for agent i is given by $C_i(P_i) = \sum_{t \in T} c_i(P_i(t), t)$. Note that we do not explicitly require all activities of an agent to be planned or that they can be completed within the period T, but because agents will not receive revenue w_k for each uncompleted activity k they will be stimulated to complete them.

For the agents to also consider the global objectives, we introduce payments such that their profits depend on the delivered quality and additional congestion caused by their presence. The quality payment Q_i for each agent i can be both a reward as well as a penalty, depending on the final quality state of its roads (e.g., based on contracted demands). Again given performed activities P_i, we can determine the resulting quality state q_e^T at the end of the period T using the recursive formula

$$q_{e_k}^{t+1} = \begin{cases} \Delta q_k(q_{e_k}^t, t) & \text{if } P_i(t) = k \\ \hat{q}_{e_k}(q_{e_k}^t, t) & \text{otherwise} \end{cases} \quad (1)$$

with (given) initial quality $q_{e_k}^0$. We define the quality payment for agent i after performing activities P_i by $Q_i(P_i) = \sum_{e \in E_i} Q_i(q_e^{|P_i|})$ where $|P_i| = T$ if all performed activities have been observed.

Congestion payments, i.e., social costs, cannot be considered from just the single-agent perspective because network throughput depends on the planning choices of all agents. Let P^t denote the set of activities performed by all agents at time t, then the social cost of this combination is captured by $\ell(P^t)$. The impact of an individual agent, given the choices made by others, can be determined by $\ell_i(P^t) = \ell(P^t) - \ell(P^t_{-i})$ in which P^t_{-i} denotes the set of activities performed at time t minus any activity by agent i. The social cost function can for example capture the costs of traffic jams due to maintenance activities, possibly based on empirical data.

Recapitulating the above, each agent i is trivially interested in maximising its revenue and minimising its maintenance costs. In order to stimulate agents to plan maintenance in favour of global objectives, we introduce quality and throughput payments such that their profit u_i, given the performed activities P by all players, is given by:

$$u_i(P) = W_i(P_i) - \Big(C_i(P_i) + Q_i(P_i) + \ell_i(P)\Big) \quad (2)$$

in which $\ell_i(P) = \sum_{t \in T} \ell_i(P^t)$. As activity revenues follow directly from the procurement, we assume that agents in expectation are always able to achieve a positive profit for completing their activities, otherwise they would not have bid on the activity during procurement.

Recall from the introduction that we are interested in finding socially optimal solutions, but given the individual agent utility of Eq. 2, how should we define these payments such that the right balance is made between these costs and the agents' private costs, which are not known to the road authority? Moreover, how can we ensure truthful reporting of these private costs? We tackle these questions using dynamic mechanism design.

In the next section we start by discussing how to compute optimal solutions, required to guarantee mechanism truthfulness, to the problem variants introduced in this section, followed by a summary of how this can be combined with a dynamic mechanism.

3 Background

We briefly introduce the two concepts our work builds on, planning under uncertainty and dynamic mechanism design.

3.1 Planning under Uncertainty

To deal with uncertainties we model the planning problem using Markov Decision Processes (MDPs), which capture this type of uncertainty rather naturally (Puterman 1994). For each agent $i \in N$ we have an MDP $M_i = \langle S_i, \mathcal{A}_i, \tau_i, r_i \rangle$ that defines its local planning problem. In this definition, S_i is the set of states and $\mathcal{A}_i$ a set of available actions. The current state of an agent contains all activities that still remain to be performed and its actions are operations to start or continue an activity (explained in detail in Sections 4.3 and 4.4). Important to keep in mind is that the MDP actions $\mathcal{A}_i$ are not equivalent to the agent activities A_i (although in the case of unit-time actions these sets are almost similar).

The function $\tau_i : S_i \times A_i \to \Delta(S_i)$ describes the transition probabilities where $\tau_i(s_i, \mathcal{A}_i, s_i')$ denotes the probability of transitioning to state s' if the current state is s_i and action $\mathcal{A}_i$ is taken. Finally, $r_i : S_i \times \mathcal{A}_i \to \mathbb{R}$ is the reward function where $r_i(s_i, a)$ denotes the reward that the agent will receive when action $a \in \mathcal{A}_i$ is taken in state s_i (e.g., the utility of Eq. 2). We formalise the rewards and actions for

the agents in Section 4, as they depend on the encoding used to solve the MDP.

Solutions to MDPs are policies $\pi : S \rightarrow \mathcal{A}$ that dictate the best action to take in expectation, given the current state it is in. Formally, the optimal policy π^* is defined such that for all start states $s \in S$: $\pi^*(s) = \arg\max_{\pi \in \Pi} V^0(\pi, s)$ with

$$V^{t_0}(\pi, s) = \mathbb{E}\Big[\sum_{t=t_0}^{\infty} \gamma^t r(s^t, \pi(s^t))) \mid s^{t_0} = s\Big] \quad (3)$$

in which s^t is the state at time t and $\gamma \in [0, 1)$ is a shared discount factor commonly used to solve problems with infinite horizons.

We can obtain the individual policies π_i for each agent by solving its MDP M_i. However, in order to develop an (optimal) joint policy π^*, required to consider throughput payments, we need to solve the multi-agent MDP that results from combining all individual MDPs. Formally, the joint MDP is defined by $M = \langle S, \mathcal{A}, r, \tau \rangle$ where $S = \times_{i \in N} S_i$ is the joint state space containing in each state $s \in S$ a local state s_i for all agents $i \in N$, $\mathcal{A}$ is the set of combined actions, r the reward function defined as $\forall s \in S, a \in \mathcal{A}$: $r(s, a) = \sum_{i \in N} r_i(s_i, a_i)$ and τ the combined transition probability function. The joint action set can always be obtained by including an action for each element of the Cartesian product set of all individual action spaces but smarter construction can greatly reduce the joint action set. For planning problems (at least) we have developed a two-stage MDP encoding that effectively reduces the joint action set size from exponential to linear in the number of players and their action sets. This is discussed in detail in Section 4.2.

3.2 Dynamic Mechanism Design

Although MDPs facilitate optimal planning under uncertainty, they assume global knowledge of all costs and rewards. As the maintenance activities are performed by different, usually competing companies, we cannot assume that this knowledge is globally available. We therefore aim to design a game such that utility-maximising companies behave in a way that (also) maximises the global reward. This is exactly the field of mechanism design, sometimes referred to as inverse game theory.

Formally, in a static or one-shot game, each agent $i \in N$ has some private information θ_i known as its *type*. In so-called direct mechanisms, players are asked for their type, and then a decision is made based on this elicited information. Groves mechanisms (Groves 1973) take the optimal decision (π^*) and define payments $\mathcal{T}$ such that each player's utility is maximised when it declares its type truthfully.

Dynamic mechanisms extend 'static' mechanisms to deal with games in which the outcome of actions is uncertain and private information of players may evolve over time. In each time step t, players need to determine the best action to take (in expectation) while considering current private information and possible future outcomes. Private rewards are therefore defined depending on the state and the policy, given by $r_i(s^t, \pi(s^t)))$, in which the state contains the player's type. This type is denoted by θ_i^t to express the possibility of this changing over time. With θ^t we denote the type of all players at time t which are encoded in the state s^t.

An extension of Groves mechanisms for such a dynamic and uncertain setting is dynamic-VCG (Bergemann and Valimaki 2006; Cavallo 2008). For dynamic-VCG the decision policy is required to be optimal, i.e., the one maximising the reward of all players, when the types θ^t are encoded into the state s^t. We denote this optimal policy for time step t given the reported types θ^t encoded in state s^t by $\pi^*(s^t)$. A policy optimised for the game with all players except i is denoted by $\pi^*_{-i}(s^t)$ and we define $r_i(s_i^t, \pi^*_{-i}(s_i^t)) = 0$.

In every time step each player i pays the expected marginal cost it incurs to other players j for the current time step. This is defined as the difference between the reward of the other players for the socially optimal decision for the current time step t, i.e., $\sum_{j \neq i} r_j(s^t, \pi^*(s^t))$ and their expected reward optimised for just them in future time steps, i.e., $V^{t+1}(\pi^*_{-i}, s^{t+1})$ (Eq. 3) minus the expected reward of the other players for a policy optimised for them for all time steps including the current one, i.e., $V^t(\pi^*_{-i}, s^t)$. Summarising, the payment $T_i(\theta^t)$ for an agent i at time step t given that reports θ^t are encoded in state s^t is thus is defined as

$$\sum_{j \neq i} r_j(s^t, \pi^*(s^t)) + V^{t+1}(\pi^*_{-i}, s^{t+1}) - V^t(\pi^*_{-i}, s^t) \quad (4)$$

The dynamic-VCG mechanism yields maximum revenue among all mechanisms that satisfy efficiency, incentive compatibility and individual rationality in within-period, ex-post Nash equilibrium. This means that at all times for each player the sum of its expected reward and its expected payments is never more than when declaring its true type.

4 Coordinating Maintenance Planning

In this work we combine existing work on planning under uncertainty and dynamic mechanism design to solve the complex problem of maintenance planning where agents are selfish and execution is uncertain. Using the dynamic-VCG mechanism we ensure that agents are truthful in reporting their costs. Then, using these reports to model agent rewards, we apply planning-under-uncertainty techniques to find optimal policies and finally we determine the payments of the mechanism, as discussed in the previous section.

An important condition for the dynamic VCG mechanism is that the chosen policy is optimal. If it is not, the payments are not guaranteed to achieve truthful cost reports and agents may want to deviate. Therefore we focus on exact solving methods in our approach.

We implemented our mechanism using the SPUDD solver (Hoey et al. 1999) to determine optimal policies. The SPUDD solver allows for a very compact but expressive formulation of MDPs in terms of algebraic decision diagrams (ADDs) and uses a structured policy iteration algorithm to maximally exploit this structure. This allows it to find optimal solutions to moderately sized problems. We note, however, that our mechanism is independent of the particular MDP solver used, as long as it returns optimal solutions.

4.1 MDP Models for Maintenance Planning

Finding an efficient joint policy π^* that maximises the sum of all agent utilities u_i (Eq. 2) cannot be directly translated into an equivalent MDP encoding. Although in our model C, Q and ℓ can be general functions, encoding general functions in the MDP formulation potentially requires exponential space. Hence to be able to use the SPUDD solver in our experiments, we necessarily restricted ourselves to only linear functions.

The current state of the network, i.e., the quality levels q_e, are modelled using a 5 star classification (from (0) very bad to (5) excellent) are encoded as discrete variables $[0, 5]$. Road degradation functions $\hat{q}$ are modelled using decision diagrams that probabilistically decrease the road quality in each time slot by one state. Completing an activity k' increases the corresponding road quality q'_k by a specified number of states (additive), corresponding to its effect $\Delta q'_k$.

Encoding the social cost ℓ can be cumbersome, depending on the complexity of the chosen cost model. Again, general cost models could result in exponential MDP encoding sizes. Using only unary and binary rules to express social cost, we can overcome this exponential growth (at the cost of losing some expressiveness). The unary rules $l : A \to \mathbb{R}$ express the marginal latency introduced by each activity independently. Dependencies between activities are expressed using binary relations $l : A_i \times A_j \to \mathbb{R}$ that specify the additional social cost when both activities are planned concurrently. The costs incurred by the set of chosen activities A^t can then be computed using $\ell(A^t) = \sum_{k \in A^t} l(k) + \sum_{k_1 \in A^t} \sum_{k_2 \neq k_1 \in A^t} l(k_1, k_2)$.

4.2 Avoiding Exponentially-Sized Action Spaces

Factored MDP solvers are typically geared towards exploiting structure in transition and reward models, but scale linearly with the number of actions. In multi-agent problem domains such as ours, however, a naive construction of the joint action set – such as enumerating all elements of the Cartesian product of individual action sets – can be exponential in the number of agents. To overcome this issue, we model each time step in the real world by two stages in the multi-agent MDP, resulting in a larger number of backups due to additional variables, but crucially avoiding exponentially-sized action spaces. Note that the encoding technique we discuss in this section is not restricted to our problem; they can be applied to any multi-agent decision problem MDP formulation in which agent actions are dependent only through their rewards.

In our MDP encoding we have used a two-stage approach for each time step in the plan problem length T. In the first step agents decide on the activity to perform (or continue) and this activity is then 'executed' in the second stage (illustrated in Sections 4.3 and 4.4 for two example scenarios). We implement this separation through the use of additional variables that for each agent state the activity to perform in the current time step. Crucial is that these variables can be set independently from the actions available to other players (unlike the Cartesian product action space). The second stage then encodes the 'execution' of their choices using one additional action. Still there are multiple ways in which this first-stage activity selection can be implemented. Again enumeration is possible (although obliterating the purpose of the two-stage approach) but we have developed two smarter encodings: action chains and activity chains.

The action chain encoding exploits the fact that we can decide on an action for each player sequentially, instead of having to decide on them all at once (as with enumeration). Through the use of a player token, each agent gets a 'turn' to determine its action within a single time step. Therefore we require only $|A_i|$ actions for each agent i, one for each activity it can choose, and hence a total of $\sum_{i \in N} |A_i|$ states (and one additional variable), instead of the $\prod_{i \in N} |A_i|$ actions needed for enumerating the Cartesian product.

For activity chains we exploit a similar idea. We group the activities of agents into activity sets to obtain an even smaller set of joint MDP actions. Let $D = \max_{i \in N} |A_i|$ be the size of the largest activity set of any player, then the activity chains are defined as $AC_m = \bigcup_{i \in N} k_m \in A_i$ for $m = 1, 2, \ldots, D$. Hence we group all m-th activities of each player into set AC_m. If a player i has no m-th activity, i.e., $m > |A_i|$, we exclude the player from this activity chain using a high penalty. Through the player token we enforce that each player sequentially chooses an activity from one of these chains. This encoding requires exactly D actions in the joint MDP for the first stage and is therefore often more compact than action chains.

In the second stage we model the execution of these choices, i.e., apply maintenance effects, and compute the sum of utilities (Eq. 2) for this time step as the reward. Note that we only proceed in time after the second stage, hence both stages are effectively within one time slot $t \in T$.

So far we have introduced a general encoding for maintenance scheduling problems. Now we will go into the specifics for two real-world application we have chosen to study in this paper: one with unit-time activities that may fail, and one where activities always succeed, but possibly have a much longer duration. A summary of the main differences can be found in Table 1.

	repeat	duration d_k	success prob. α_k	delay duration h_k	delay prob. β_k
1	yes	1	$[0, 1]$	0	0
2	no	$\mathbb{Z}^+$	1	$\mathbb{Z}^+$	$[0, 1]$

Table 1: The differences between scenario 1 and 2. These parameters are explained in Section 4.3 and 4.4.

4.3 Scenario 1: Activities with Failures

As a step towards network maintenance, we first focus on scheduling repeatable unit-time activities with possible failures. Although this problem is conceptually rather simple, it captures essential parts of real-world applications such as factory scheduling and supply chain planning problems. In this scenario, activities $k \in A_i$ are repeatable, of unit-time ($d_k = 1$) and succeed with probability $\alpha_k \in [0, 1]$. It is possible for any activity $k \in A_i$ to fail with probability

$1 - \alpha_k$. Whether an activity fails will become apparent at its actual execution time. When an activity fails, it has no positive effect on the quality but its associated maintenance and throughput costs are still charged. If the agent still wants to perform the maintenance it has to include the activity in its plan again at a later time.

Because activities in this scenario are unit-time and repeatable, we can directly translate these into actions of the single-agent MDPs. For each activity $k \in A_i$ of agent i we create an action a_k with reward $c(k, t, 1)$. This action improves the quality level q_k by the number of levels corresponding to Δq_k with probability α_k. Thus with probability $1-\alpha_k$ the maintenance fails and the quality level remains unchanged.

4.4 Scenario 2: Portfolio Management

Portfolio management is a second variant of our model. Inspired by real-world consequences of signing a maintenance contract, in this setting agents have to perform each activity exactly once, although multiple alternatives exists for the activity, and instead of activity failure we consider delays. More formally, for each activity k we now additionally have a delay duration h_k and delay probability β_k.

Encoding the portfolio management planning in an MDP requires a substantially greater effort as we can no longer translate activities directly to actions. This problem is more complex because of (1) possible non-unit activity durations, (2) activities can be delayed, (3) for each road we can only choose one activity to perform, and (4) each road can be serviced only once. The latter two are easily resolved by introducing a variable that flags whether a road has been serviced and using corresponding penalties to prohibit planning of these activities later; the first two require more work.

From the single-agent MDP perspective, non-unit activity durations (including possible delay) do not pose any difficulties. We could use actions that update the time variable t according to the activity duration. For the joint MDP however, this time variable is shared by all the agents. Increasing the time by the activity duration makes it impossible for other agents to start their activities in this time period. Our solution is to decompose each activity k into unit-time MDP actions $\{start_k, do_k, delay_k, done_k\}$ and use a timer variable to keep track of the remaining activity duration and its delay status ($pending$, no or yes). The $start_k$ action marks the beginning of the activity. This action sets the delay status to $pending$ and the activity timer to the duration d_k. In subsequent time steps, the agent has to perform a do_k action until the activity timer reaches zero. At this point, the activity delay status is pending and the activity is delayed with probability β_k (also updating the delay status).

If the activity is not delayed, the $done_k$ action is executed and the associated road e_k is flagged as serviced. When an activity is delayed however, we set the activity timer to the delay duration h_k and continue with do_k actions until again the timer reaches zero, at which point the $stop_k$ action is executed (not $delay_k$ again because of the delay status value).

Important to keep in mind is that during the search for optimal policies, a solver might decide on any order of these actions. Hence we need to constrain the actions such that only feasible action sequences are considered. For example, the do_k action can only be chosen if the activity timer is greater than zero, otherwise a high penalty results.

Rewards are encoded using the two-stage approach as before. In the first stage, each agent chooses a $start$, do, $delay$ or $stop$ action. Then the second stage implements these actions and incurs maintenance, quality and social costs for the current time step t.

4.5 Planning Methods

Using the encodings we discussed, we can find the optimal policy π^* that minimises costs over all three objectives. In the experiments, we then compare this centralised computation that relies on truthful reporting to (1) the approach where each agent plans its own actions optimally individually, i.e., disregarding other agents, and (2) a best-response approach (Jonsson and Rovatsos 2011).

In the best-response approach, agents alternatingly compute their best plan (in expectation) in response to the current (joint) plan of the others. This approach allows us to solve much easier single agent problems but still consider agent dependencies (e.g., social cost). Of course, the downsides of this approach are that we will have to settle for Nash equilibria (if they exist) and the ordering of agents matters.

5 Evaluation

We have performed a substantial number of experiments to gain insight into this previously uncharted area. For both problem scenarios we have generated large benchmark sets on which we tested the various planning approaches and their encodings discussed in the previous section. These experiments are mainly of an exploratory nature in which we study the effect of each of the problem variables. The solver used in these experiments has been implemented in Java, using SPUDD as its internal MDP solver. All experiments have been run on a system with an 1.60 Ghz Intel i7 processor with a time limit of 3 hours per instance, except for the experiments of Section 5.2 which had a time limit of one day.[1]

5.1 Activities with Failures

In the first series of experiments we have been mainly interested in exploring the computational limits of solving the problem centrally using an exact algorithm. To this end we generated a set of simple instances that vary in both the number of players N (2-5) and activity set A_i sizes (1-15). We solved these instances using different planning period lengths T (1-46). From these experiments we identify the parameters that contribute the most to the difficulty of the problem.

Activity sets are generated using random, linear, time-dependent cost functions and always increase the quality level of the associated road by one. Quality cost functions are also generated for each road. Road quality is decreasing linearly in the quality with a random factor from [1, 3],

[1]The testset is available at http://www.alg.ewi.tudelft.nl/fileadmin/alg/homepages/scharpff/icaps-testset.rar

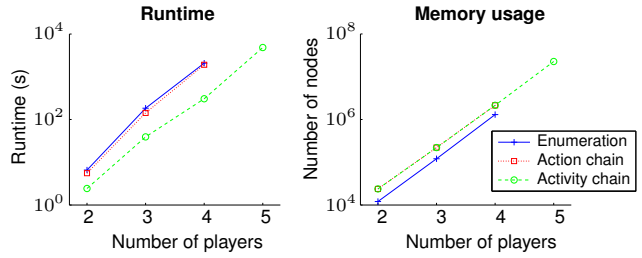

Figure 1: Comparison of runtime (left) and memory use (right) for different encoding methods and player set sizes, $|A_i| = 3$, $|T| = 46$, $|Q| = 6$ (both log scale).

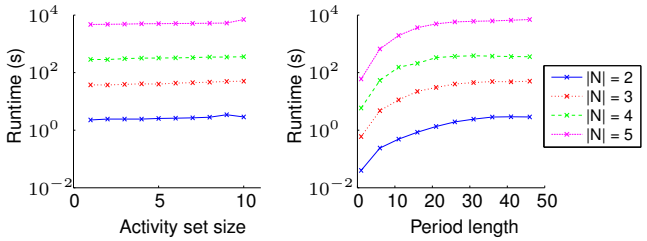

Figure 2: Runtimes for different activity set sizes $|A_i|$ with plan period length $|T| = 46$ (left), and different $|T|$ with $|A_i| = 10$ (right) using activity chains (both log scale).

which is fixed per road. Recall from Section 4.1 that linearity of this and other cost functions is a restriction not imposed by our model but is required to combat a potential exponential MDP encoding size. For the social costs ℓ we study the worst-case where all activities always interfere and define these costs using randomly chosen (marginal) cost $l(k1, k2) \in [1, 10]$ for each $k_1 \in A_i$ and $k_2 \in A_j$ where $i \neq j$. We do not consider the marginal cost for individual actions, i.e., $l(k) = 0$.

In Figure 1 we have depicted both the runtime (left) and the memory (right) required to solve each of these instances, under different encoding methods. The memory required is expressed in the number of nodes SPUDD generates. Not surprisingly this figure illustrates that the performance of the solver is exponential in both time and memory, and greatly depends on the structure of its input. By exploiting the problem structure, the activity chain encoding is able to greatly reduce the required runtime. With it we have been able to solve instances with 5 players and 3 activities per player within the time limit of 3 hours, whereas the other two failed on such instances. Observe that activity chain encoding requires slightly more memory. For the reasons stated above, we have illustrated the results of the remaining experiments only using the activity chain encoding (which indeed outperformed the others in all tests).

In Figure 2 we have plotted the required runtime for solving instances using activity chains for various activity set sizes and period lengths. From the figure we can conclude that the runtime is only linearly affected by the number of activities each player has. The plan period length shows almost the same: although the required runtime increases

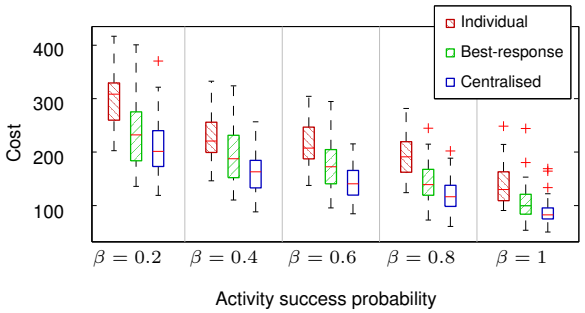

Figure 3: Total cost using different planning approaches for the activities with failure problems (lower is better).

rapidly at first, for larger plan horizons the increase is again almost linear. It is expected that instances with small plan lengths are easily solvable because only a small number of plans is possible. Increasing the plan length introduces an exponential number of new possible plans and therefore the computation time increases rapidly, up to the point where the roads reach maximum quality. From this time on, agents have to consider planning an activity only when the quality degrades.

Having identified the computational boundaries of the centralised problem, we compared the performance of different planning approaches discussed in Section 4.5 in terms of total reward obtained. For these experiments we have used 60 generated two-player instances in which each player is responsible for one road. The activity set of each player contains the no-operation and 1, 2 or 3 available maintenance operations that improve the quality of the road by 1, 2 or 3 levels respectively. The cost of each action $k \in A_i$ is drawn randomly from $[1, 3{*}\Delta q_k]$ and is therefore independent from its execution time. In each instance, the activities share the same success rate $\alpha = [0.2, 0.4, 0.6, 0.8, 1]$ for all activities. For the best-response algorithm we have used 3 iterations with random agent orderings. Smaller experiments support our choice for 3 iterations: less iterations result in far worse results while more iterations only slightly improve the quality but increase the runtime substantially. Note that we have no guarantee that the best-response approach will converge to an equilibrium at this point, however early experiments have shown that best-response almost always improves the initial solution.

Figure 3 illustrates the total cost obtained for each of the methods under different levels of uncertainty with a box plot. In the plot, the box contains the upper and lower quartile of the result values with the mean shown by the horizontal line. The whiskers show the smallest and largest values and outliers are plotted as crosses.

The centralised algorithm always computes the social optimal solution in which the total cost is minimal. As to be expected, the individual planning method perform much worse on these instances. Because in this approach the dependencies between agents are ignored, the resulting plan may suffer from high social cost. Indeed this figure shows that the

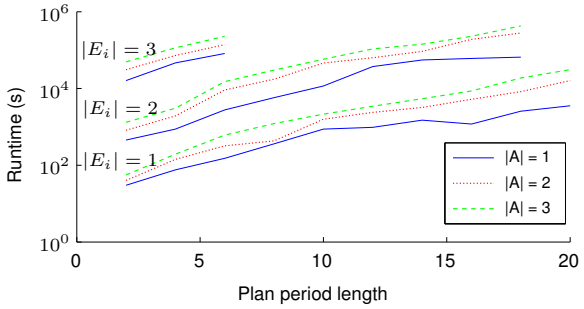

Figure 4: Runtimes of best-response planning for portfolio management for various road set sizes $|E_i|$, activities per road $|A|$ and plan length $|T|$ (log scale). The cut-off for $|E_i| = 3$ at $|T| = 6$ is due to the time limit of 1 day.

total costs are much higher on average, compared to the central solution. Using only 3 iterations, the best-response algorithm produces fairly acceptable plans. As we have mentioned before, best-response can been seen as a compromise between individual and central planning. Indeed our experiments show that the total cost is lower on average than when using individual planning, but higher than the centralised method.

5.2 Portfolio Management

For portfolio management we have performed similar experiments. We have generated a set of 5 games for each combination of $|N| \in [2, 5]$, $|E_i| \in [1, 5]$, $|A_i| \in [1, 3]$ and $\beta \in [0.2, 0.4, 0.6, 0.8, 1.0]$ (delay risk is the same for all activities in these instances). We ran our solver on these instances for different values of T. Again we study the worst case in which players are tightly coupled (all activities interfere with at least one of another agent), and we strive to gain insight in the factors contributing to the complexity.

Although exact solving for multiple agents poses a difficult challenge at this point, we have been able to develop joint plans for several non-trivial instances using the best-response approach. Figure 4 illustrates the runtime required for finding an optimal response, given the planning choices made by others, for various road set, activity and plan period sizes. These early experiments show that best-responses can be computed in the order of a few minutes for problems where agents are responsible for multiple roads with several activities to choose from, but also that it quickly becomes intractable for larger plan horizons and road set sizes.

6 Conclusions and Challenges

This paper introduces the practically very relevant problem of infrastructural maintenance planning under uncertainty for selfish agents in a private-values setting. With the help of experts in the field of maintenance planning we developed a model that captures the essence of this coordination problem. Dynamic mechanism design combined with optimally solving MDPs theoretically solves this modelled problem but might be difficult in practical scenarios. Through experimental analysis with different encodings in an existing solver, we found that we can solve practical examples of scenario 1 within reasonable time. For scenario 2, run times for best-response can be computed for multiple agents in a small network. We have thus made an important step towards this practical planning problem, and identified challenges for our community.

In this paper, we used scalar weighting to balance the different objectives in the system. However, asset maintenance planning for infrastructures is inherently a multi-objective problem, even though this has not been acknowledged in procurements until recently. The weighting model has two difficulties. Firstly, it requires accurate and exhaustive operationalisation of objectives in terms of monetary rewards schemes. Secondly, in any practical application, human decision makers are more likely to prefer insight into possible solutions trade-offs over a single black-box solution. In this context, the work by Grandoni et al. (2010) is relevant, in which the authors study approximation techniques for mechanism design on multi-objective problems. Nevertheless, their work has only been applied to static mechanisms. Developing methods combining multi-objective planning under uncertainty with dynamic mechanism design is a hard challenge for the community, but with high potential payoffs in terms of real-world relevance.

Scaling MDP solvers in terms of number of actions has received relatively little attention, but is crucial for solving multi-agent problems that suffer from exponential blow up of their action space. Furthermore, the best-response approach that we employed is not guaranteed to converge to the optimal solution, except for special cases such as potential games (Jonsson and Rovatsos 2011). Bounding the loss, e.g., by building on those special cases, will provide benefits to the adoption of best-response methods. Finally, as mentioned in the related work section, approximate solutions often preclude many of the theoretical mechanism-design results to apply. A major challenge here is to identify mechanisms that are more robust to such approximations.

With respect to the implications of our work, it is clear that the planning and coordination of (maintenance) activities in the presence of uncertainty is a complex problem. However, applications exist in several other domains such as bandwidth allocation or smart power grids, and hence the need for a practical solution is high.

The concept of traffic time loss can also be used to stimulate market parties in rethinking current working methods. By adjusting tendering criteria to specific needs on certain areas of the network, bidders can distinguish themselves by offering innovative proposals with limited traffic loss hours. The Dutch road authority and several provinces of The Netherlands are currently experimenting with this method in the Netherlands.

Acknowledgements

This research is part of the Dynamic Contracting in Infrastructures project and is supported by Next Generation Infrastructures and Almende BV. Matthijs Spaan is funded by the FP7 Marie Curie Actions Individual Fellowship #275217 (FP7-PEOPLE-2010-IEF).

References

d'Aspremont, C., and Gérard-Varet, L. 1979. Incentives and incomplete information. *Journal of Public Economics* 11(1):25–45.

Athey, S., and Segal, I. 2007. An efficient dynamic mechanism. Technical report, UCLA Department of Economics.

Bergemann, D., and Valimaki, J. 2006. Efficient dynamic auctions. *Cowles Foundation Discussion Papers*.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4):819–840.

Boutilier, C. 1996. Planning, learning and coordination in multiagent decision processes. In *Proc. of 6th Conf. on Theoretical Aspects of Rationality and Knowledge*, 195–201.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 73–78.

Cavallo, R.; Parkes, D. C.; and Singh, S. 2006. Optimal coordinated planning amongst self-interested agents with private state. In *Proc. of Conf. on Uncertainty in Artificial Intelligence*, 55–62.

Cavallo, R. 2008. Efficiency and redistribution in dynamic mechanism design. In *Proc. of 9th ACM conference on Electronic commerce*, 220–229. ACM.

Der Spiegel. 2012. A40: Autobahn nach dreimonatiger sperre freigegeben. Online, Sep 30.

Detienne, B.; Dauzère-Pérès, S.; and Yugma, C. 2009. Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling* 1–16.

Grandoni, F.; Krysta, P.; Leonardi, S.; and Ventre, C. 2010. Utilitarian mechanism design for multi-objective optimization. In *Proc. of 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, 573–584. Society for Industrial and Applied Mathematics.

Groves, T. 1973. Incentives in teams. *Econometrica: Journal of the Econometric Society* 617–631.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *Proc. of Conf. on Uncertainty in Artificial Intelligence*, 279–288.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *Int. Conf. on Automated Planning and Scheduling*, 114–121.

Kok, J. R.; Hoen, P.; Bakker, B.; and Vlassis, N. 2005. Utile coordination: Learning interdependencies among cooperative agents. In *Proc. Symp. on Computational Intelligence and Games*, 29–36.

van der Krogt, R. P.; de Weerdt, M.; and Zhang, Y. 2008. Of mechanism design and multiagent planning. In Ghallab, M.; Spyropoulos, C. D.; Fakotakis, N.; and Avouris, N., eds., *European Conf. on Artificial Intelligence*, 423–427.

Melo, F. S., and Veloso, M. 2009. Learning of coordination: Exploiting sparse interactions in multiagent systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 773–780. International Foundation for Autonomous Agents and Multiagent Systems.

Procaccia, D., and Tennenholtz, M. 2009. Approximate mechanism design without money. In *Proc. of ACM Conf. on Electronic Commerce*, 177–186.

Puterman, M. L. 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons, Inc.

Volker, L.; Scharpff, J.; De Weerdt, M.; and Herder, P. 2012. Designing a dynamic network based approach for asset management activities. In *Proc. of 28th Annual Conference of Association of Researchers in Construction Management (ARCOM)*.

Wolpert, D. H.; Tumer, K.; and Frank, J. 1999. Using collective intelligence to route internet traffic. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, 952–958. MIT Press.

Qualitative Planning under Partial Observability in Multi-Agent Domains

Ronen I. Brafman
Ben-Gurion University
Beer-Sheva 84105, Israel
brafman@cs.bgu.ac.il

Guy Shani
Ben-Gurion University
Beer-Sheva 84105, Israel
shanigu@cs.bgu.ac.il

Shlomo Zilberstein
University of Massachusetts
Amherst, MA 01003, USA
shlomo@cs.umass.edu

Abstract

Decentralized POMDPs (Dec-POMDPs) provide a rich, attractive model for planning under uncertainty and partial observability in cooperative multi-agent domains with a growing body of research. In this paper we formulate a qualitative, propositional model for multi-agent planning under uncertainty with partial observability, which we call Qualitative Dec-POMDP (QDec-POMDP). We show that the worst-case complexity of planning in QDec-POMDPs is similar to that of Dec-POMDPs. Still, because the model is more "classical" in nature, it is more compact and easier to specify. Furthermore, it eases the adaptation of methods used in classical and contingent planning to solve problems that challenge current Dec-POMDPs solvers. In particular, in this paper we describe a method based on compilation to classical planning, which handles multi-agent planning problems significantly larger than those handled by current Dec-POMDP algorithms.

Introduction

Many problems of practical importance call for the use of multiple autonomous agents that work together to achieve a common goal. For example, disaster response teams typically consist of multiple agents that have multiple tasks to perform, some of which require or can benefit from the cooperation of multiple agents. In such domains, agents typically have partial information, as they can sense their immediate surroundings only. And because agents are often located in different positions and may even have different sensing abilities, their runtime information states differ. Sometimes, this can be overcome using communication, but communication infrastructure can be damaged, and even if it exists, communication may be costly (in terms of time and resources) and should be reasoned about explicitly.

Decentralized POMDPs (Dec-POMDPs) offer a rich model for capturing such multi-agent planning problem (Bernstein et al. 2002; Seuken and Zilberstein 2008). Dec-POMDPs extend the single agent POMDP model to account for multiple agents with possibly different information states, but the complexity of the Dec-POMDP model has limited its applicability. In this paper we define and study a conceptually simpler model for multi-agent planning that extends the single-agent contingent planning model. We call this new model *Qualitative Dec-POMDP* (QDec-POMDP).

In terms of worst-case complexity, we show that QDec-POMDPs are no easier than Dec-POMDPs. Nevertheless, a multi-agent contingent planning formalism offers two main advantages. First, being geared to propositional (a.k.a. factored) state models, it allows for more convenient model specification, as opposed to flat state models that characterize much of the work on Dec-POMDPs. Second, much like contingent planning, it is more amenable to the use of current classical planning methods, which are quite powerful. Thus, it could allow us to solve much larger problems. Indeed, one of our main contributions is a compilation method from QDec-POMDPs to classical planning, allowing us to tackle domains larger than those that can be solved by current Dec-POMDP algorithms.

Of course, the qualitative contingent planning model is less expressive in that it specifies the possible outcome states without their likelihood. But this is an advantage in cases where it is difficult to specify a richer, quantitative model, or when such models are too complex to solve. Furthermore, a solution to a qualitative model can provide guidance and heuristics for methods that operate on the quantitative model. Alternatively, one could use information from the quantitative model to bias choices made by a qualitative counterpart, e.g., when state sampling techniques are used, thus gradually moving from qualitative to quantitative.

In the next section we introduce the formal QDec-POMDP model. We start with an analysis of the complexity of solving a flat state-space qualitative model. This makes clear the impact of the move from a quantitative Dec-POMDP model, for which complexity results exist in that form. Next, we take a closer look at the issue of belief state representation, which is much more complex than in the single agent case. Here we still consider a flat state space model for semantic clarity. Next, we introduce a factored model, in the spirit of contingent planning models. Focusing on a deterministic variant of this model, we suggest an offline compilation method for its solution, and describe its empirical performance. The earlier discussion of belief states will help us understand an essential simplification made by this model. We end by discussing some of the challenges faced in designing an online algorithm.

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Model Definition

We start with the basic definition of a flat-space QDec-POMDP.

Definition 1. *A qualitative decentralized partially observable Markov decision process (QDec-POMDP) is a tuple* $\mathcal{Q} = \langle I, S, b_0, \{A_i\}, \delta, \{\Omega_i\}, O, G, T\rangle$ *where*

- I *is a finite set of agents indexed* $1, ..., m$.
- S *is a finite set of states.*
- $b_0 \subset S$ *is the set of states initially possible.*
- A_i *is a finite set of actions available to agent* i *and* $\vec{A} = \otimes_{i\in I} A_i$ *is the set of joint actions, where* $\vec{a} = \langle a_1, ..., a_m\rangle$ *denotes a particular joint action.*
- $\delta : S \times \vec{A} \to 2^S$ *is a non-deterministic Markovian transition function.* $\delta(s, \vec{a})$ *denotes the set of possible outcome states after taking joint action* $\vec{a}$ *in state* s.
- Ω_i *is a finite set of observations available to agent* i *and* $\vec{\Omega} = \otimes_{i\in I}\Omega_i$ *is the set of joint observation, where* $\vec{o} = \langle o_1, ..., o_m\rangle$ *denotes a particular joint observation.*
- $\omega : \vec{A} \times S \to 2^{\vec{\Omega}}$ *is a* non-deterministic *observation function.* $\omega(\vec{a}, s)$ *denotes the set of possible joint observations* $\vec{o}$ *given that joint action* $\vec{a}$ *was taken and led to outcome state* s. *Here* $s \in S$, $\vec{a} \in \vec{A}$, $\vec{o} \in \vec{\Omega}$.
- $G \subset S$ *is a set of goal states.*
- T *is a positive integer representing the horizon.*

Our model allows for non-deterministic action effects as well as non-deterministic observations. That is, we allow a set of possible global states to result from a single joint action, and we also allow multiple possible observations per outcome state. Additionally, our model assumes a shared initial belief state, as most Dec-POMDP models. The case where agents have different initial belief states is very important, as it corresponds to the situation in on-line planning, but is also very challenging.

We represent the local plan of each agent using a *policy tree* q, which is a tree with branching factor $|\Omega|$ and depth T. Each node of the tree is labeled with an action and each branch is labeled with an observation. To execute the plan, each agent performs the action at the root of the tree and then uses the subtree labeled with the observation it obtains for future action selection. If q_i is a policy tree for agent i and o_i is a possible observation for agent i, then $q_{i_{o_i}}$ denotes the subtree that corresponds to the branch labeled by o_i.

Let $\vec{q} = \langle q_1, q_2, \cdots, q_m\rangle$ be a vector of policy trees. $\vec{q}$ is also called a *joint policy*. We denote the joint action at the root of $\vec{q}$ by $\vec{a}_{\vec{q}}$, and for an observation vector $\vec{o} = o_1, \ldots, o_m$, we define $\vec{q}_{\vec{o}} = \langle q_{1_{o_1}}, \ldots q_{m_{o_m}}\rangle$.

We later suggest algorithms to compute a joint policy that solves a given QDec-POMDP (guarantee goal reachability).

Complexity of QDec-POMDP

We now analyze the complexity of generating plans in our proposed model, compared with policy generation in the traditional Dec-POMDP model.

We first characterize the set of states reachable via a joint policy $\vec{q}$. Intuitively, if all states reached by time T are goal states, the joint policy is a solution to the QDec-POMDP. To do so, we define the set of pairs of the form (global state, joint policy) that are reachable, denoted by β. The base case, β_0, corresponds to initially possible states and the full depth-T joint policy $\vec{q}$: $\beta_0 = \{(s, \vec{q}) | s \in b_0\}$. We define β_t inductively:

$$\beta_t = \{(s', \vec{q}_{\vec{o}}) \mid (s, \vec{q}) \in \beta_{t-1},\ s' \in \delta(s, \vec{a}_{\vec{q}}),\ \vec{o} \in \omega(\vec{a}, s')\}$$

We can now formally define a solution to a QDec-POMDP using our β notation:

Definition 2. *A given depth-*T *joint policy* $\vec{q}$ *is a solution to a QDec-POMDP* Q *iff* $\forall s : (s, \emptyset) \in \beta_T \ \Rightarrow\ s \in G$.

Note that at time T the remaining policy trees are empty.

Definition 3. *Let QDec-POMDP*$_m$ *denote the problem of finding a joint policy* $\vec{q}$ *that is a solution of a given* m*-agent QDec-POMDP* $\mathcal{Q} = \langle I, S, \{A_i\}, \delta, \{\Omega_i\}, O, G, T\rangle$ *(i.e.,* $|I| = m$*).*

We now analyze the complexity of finding such solutions.

Theorem 1. *For all* $m \geq 2$*, if* $|T| \leq S$*, then QDec-POMDP*$_m \in$ *NEXP.*

Proof. We show that a nondeterministic machine can solve an instance of QDec-POMDP$_m$ using at most exponential time. To start, we guess a joint policy $\vec{q}$. A joint policy includes m policy trees, each of size $O(|\Omega|^T)$. Overall, the size is $O(m|\Omega|^T)$, and because $T < |S|$, the joint policy can be generated in exponential time. Given a joint policy, the update of the belief state β_t can be performed in exponential time: β_t can be larger than β_{t-1} by at most a multiplicative factor of $|S|$, and the update takes polynomial time in the size of β_t. Thus repeating this process T times may require at most exponential time. Finally, all we need is to verify that $\forall s :\ (s, \emptyset) \in \beta_T \ \Rightarrow\ s \in G$. □

Theorem 2. *For all* $m \geq 2$*, QDec-POMDP*$_m$ *is NEXP-Hard.*

Proof. The proof is similar to the one presented by (Bernstein et al. 2002) for Dec-POMDPs. It follows a reduction of the TILING problem (Lewis 1978; Papadimitriou 1994), which is NEXP-complete, to the QDec-POMDP$_2$ problem. We only sketch the argument here.

TILING involves a given board size n (represented in binary), a set of tile types $L = \{tile_0, ..., tile_k\}$, and a set of binary horizontal and vertical compatibility relations $H, V \in L \times L$. A tiling f is *consistent iff* (a) $f(0, 0) = tile_0$, and (b) for all x, y $\langle f(x, y), f(x + 1, y)\rangle \in H$ and $\langle f(x, y), f(x, y + 1)\rangle \in V$. That is, adjacent tiles satisfy the compatibility relations. The decision problem is to determine, given n, L, H, V, whether a consistent tiling exists.

The basic idea is to create a two-agent QDec-POMDP that randomly selects two tiling locations bit by bit, informing one agent of the first location and the other agent of the second location. The agents' local policies are observation-history based, so the agents can base their future actions on the tiling locations given to them. After generating the locations, the agents are simultaneously queried to place tiles at some locations. The QDec-POMDP problem is designed such that the agents reach the goal *iff* their answers to the query are based on some agreed upon solution of the tiling

problem. Here is a brief discussion of the phases of the original proof from (Bernstein et al. 2002) and the relevant changes needed for the QDec-POMDP model.

Select Phase Using nondeterminism, the system generates two random bit positions and values. They are memorized as part of the state and not observed by the agents.

Generate Phase Using nondeterminism, the system generates two tile locations and reveals one to each agent via their observation streams.

Query Phase Each agent is queried for a tile type to place in the location specified to it.

Echo Phase The agents are now required to echo their tile locations. Only one position (not known to the agents) is verified by the system per observation stream. Making an error in the echo phase leads to a dead-end, from which the goal cannot be reached. During the echo phase, the system tracks the adjacency relationship between the tile locations.

Test Phase The system checks whether the tile types provided in the query phase come from a single consistent tiling. If the tile types violate any of the constraints, a dead-end state is reached. Otherwise, the goal is reached.

Similar to the original proof, if there exists a consistent tiling, then there must exist a joint policy for the constructed QDec-POMDP$_2$ that reaches the goal state. Likewise, there is no way to *guarantee* goal reachability without the agents being faithful to a single consistent tiling. □

Note that the QDec-POMDP constructed for the proof is in fact a QDec-MDP (i.e., the observations of the two agents combined provide full information on the state of the system). Therefore, QDec-MDP$_2$ is NEXP-Hard as well.

Corollary 1. *For all $m \geq 2$, both QDec-POMDP$_m$ and QDEC-MDP$_m$ are NEXP-complete.*

It is somewhat surprising that the qualitative model with its different objective (goal reachability versus maximizing expected reward) has the same complexity as the standard Dec-POMDP model. In some sense, this confirms the intuition that the main source of complexity is decentralized operation with partial information, not stochastic actions.

Belief States and Joint Policies

The notion of the agent's belief state plays a central role in algorithms for solving problems with partial observability and in the representation and computation of policies. In this section, we seek to understand belief states in QDec-POMDPs, explain some simplification we make, and use this to provide an alternative representation for a joint policy.

Online Local Belief States

We begin with a definition of a local belief state of an agent in the context of a known joint-policy tree. This definition is useful for reasoning about the information state of an agent online. However, it is not useful for the generation of a joint-policy, as it assumes a fixed policy.

Each agent can maintain a belief state β_i^t at time t that reflects its own experience. The belief state includes all the possible pairs of the form *system state* and *joint policy*. Agent i knows its own policy tree, so all joint policies considered possible in its belief state must agree with its own, actual, policy tree.

The initial belief state of agent i is $\beta_i^0 = \{(s_0, \vec{q}) | s_0 \in b_0\}$ where $\vec{q}$ is the initial vector of policy trees for all the agents. Let a_i^t be the action agent i executes at time t, and o_i^t the observation it obtains. We define β^t inductively as follows:

$$\begin{aligned} \beta_i^t \quad = \quad & \{(s_t, \vec{q}_{\vec{o}}) \mid (s_{t-1}, \vec{q}) \in \beta_i^{t-1}, \\ & s_t \in \delta(s_{t-1}, \vec{a}_{\vec{q}}), \\ & \vec{o} \in \omega(\vec{a}_{\vec{q}}, s_t),\ \vec{o}[i] = o_i^t\} \end{aligned} \tag{1}$$

The only difference between the global update of β^t and the local update is the added condition $\vec{o}[i] = o_i^t$, which means that we only include outcome states s_t that produce the actual observation that agent i obtained. That is, we use the local information of agent i to filter states that are inconsistent with its knowledge.

This belief state update scheme is valid when the joint policy is fixed in advance in the form of policy trees. But if we want to have policies that depend on these local belief states we run into a problem. The actions of the other agents depend on their beliefs that in turn depend on their actions. Without resolving this circularity, it is hard to generate plans conditioned on local beliefs.

Offline, Policy Independent Belief States

Most existing methods for planning under partial observability rely on a "nice-to-manage", policy-independent notion of belief state. These methods include, for example, search in belief state space, the computation of a real-valued function over belief states, as in POMDPs, and the generation of a policy that maps belief states to actions.

In the multi-agent case there is no longer a single belief state, but we can replace that with the notion of a history. A *history* is a sequence of states and actions, of the form $(s_0, a_1, s_1, \ldots, a_n, s_n)$, denoting the initial state, the initial action, the resulting state, etc. If $h = (s_0, a_1, s_1, \ldots, a_n, s_n)$, let $h_s(k) = s_k$ and $h_a(k) = a_k$. Initially, every agent's belief state is $\beta_i^0 = \{(s_0) | s_0 \in b_0\}$. We define $\beta^t_{\beta^{t-1}, a_i, o_i}$, the new belief state of agent i at time t after executing a_i and observing o_i in belief state β^{t-1}, as follows:

$$\begin{aligned} \beta^t_{\beta^{t-1}, a_i, o_i} \quad = \quad & \{(h \circ (\vec{a}_t, s_t)) \mid h \in \beta^{t-1}, \\ & s_t \in \delta(h_s(t-1), \vec{a}_t),\ \vec{a}_t[i] = a_i, \\ & \vec{o} \in \omega(\vec{a}_t, s_t),\ \vec{o}[i] = o_i\} \end{aligned} \tag{2}$$

That is, those histories that extend current histories with a joint action that is consistent with the local action executed by the agent, and with a state which is the result of applying that joint action to the last state of the history, such that this last state and action can induce a joint-observation consistent with an agent's local observation.

In the (qualitative) single agent case, due to the Markovian assumption, one can simply maintain the set of last states of the above histories, rather than the entire history, i.e., maintaining the set of currently possible states. Unfortunately, to the best of our knowledge, such a "truncation" is

not possible in the multi-agent case without sacrificing completeness. The reason is that different histories that led one agent to the same state, lead to different states of information for other agents. Thus, the set of last states in histories considered possible by an agent only approximates its state of knowledge. We will employ this approximation in the planning algorithm introduced later, referring to it as *set-of-possible-states* approximation.

Initially, every agent's belief state is $\beta_i^0 = \{(s_0)|s_0 \in b_0\}$. The estimated set of possible states for agent i at time t given the estimated belief state at time $t-1$, action a_i by the agent, and observation o_i is defined as follows:

$$\begin{aligned}\beta^t_{\beta^{t-1},a_i,o_i} \quad = \quad & \{s_t \ : \ s_{t-1} \in \beta^{t-1}, \\ & s_t \in \delta(s_{t-1}, \vec{a}_t), \vec{a}_t[i] = a_i, \\ & \vec{o} \in \omega(\vec{a}_t, s_t), \ \vec{o}[i] = o_i\}\end{aligned} \tag{3}$$

Global Policy Tree

To describe a joint policy, we used a vector $\vec{q}$ of individual policy trees. An alternative description is a global policy tree, which we denote by q_g. Its definition is identical to that of an individual policy tree, except that nodes are labeled by *joint* actions, and edges are labeled by *joint* observations.

Unfortunately, some general policy trees do not correspond to any joint policy. If two nodes in the global policy tree correspond to branches that would yield the same history for agent i, i.e., agent i cannot distinguish between these branches, the action assigned to i in these nodes must be identical.

Thus, let q_g be a policy tree, and let b_0 be the initial belief state. For every node n, let $\vec{b}(n) = b_1(n), \ldots, b_m(n)$ be the vector of agents' belief states given the history that corresponds to the path to this node. q_g is *executable* if for every agent $i = 1, \ldots, m$ and every two nodes n, n' in g_p, if $b_i(n) = b_i(n')$ then the i^{th} component of the joint actions associated with n and n' must be identical.

Although joint policies are easier to execute – they contain an explicit policy for each agent – global policy trees are a better fit for the compilation approach that we describe below, because they are closer in form to (single-agent) classical plans over joint actions: Instead of generating joint policy trees consisting of m local policies, our translation method will seek a single *executable* global policy tree. To ensure that the global tree is executable, we will enforce the constraint described above while using the set-of-possible-states approximation for agents' state of knowledge. Because this approximation is sound, i.e., two histories that the agent cannot distinguish with will always yield two identical sets of possible states (but not vice versa), we are guaranteed that the global policy tree is indeed executable.

Factored Representation of QDec-POMDP

A factored representation of a planning problem makes it more compact and facilitates development of efficient algorithms that leverage the factored structure. With a few exceptions (Oliehoek et al. 2008; Kumar, Zilberstein, and Toussaint 2011), little work has focused on exploiting such factored models for Dec-POMDPs. Moreover, existing factored Dec-POMDPs use a "flat" state representation *per* agent (one state variable per agent) as opposed to multiple generic state variables that describe compactly the entire state space.

In this section we describe a factored specification of a QDec-POMDP model, motivated by the classical STRIPS and PDDL languages. We propose a PDDL-like representation that is much more compact than the SPUDD-like representation used in some factored Dec-POMDPs. At present, our language does not support non-deterministic observations. Although conceptually simple and easy to define in multi-valued settings, formalizing non-deterministic observations in the boolean STRIPS setting was not straightforward, and is left for future work. In what follows we slightly abuse notation by overloading terms previously defined.

Definition 4. *A factored QDec-POMDP is a tuple* $\langle I, P, \vec{A}, Pre, Eff, Obs, b_0, G\rangle$ *where* I *is a set of agents,* P *is a set of propositions,* $\vec{A}$ *is a vector of individual action sets,* Pre *is the precondition function,* Eff *is the effects function,* b_0 *is the set of initially possible states, and* G *is a set (conjunction) of goal propositions. The state space* S *consists of all truth assignments to* P*, and each state can be viewed as a set of literals.*

The precondition function Pre maps each individual action $a_i \in A_i$ to its set of preconditions, i.e., a set of literals that must hold whenever agent i executes a_i. Preconditions are local, i.e., defined over a_i rather than $\vec{a}$, because each agent must ensure that the relevant preconditions hold prior to executing its part of the joint action. We extend Pre to be defined over joint actions $\{\vec{a} = \langle a_1, .., a_m\rangle : a_i \in A_i\}$ (where $m = |I|$): $Pre(\langle a_1, .., a_m\rangle) = \cup_i Pre(a_i)$.

The effects function Eff maps joint actions into a set of pairs (c, e) of conditional effects, where c is a conjunction of literals and e is a single literal, such that if c holds before the execution of the action e holds after its execution. Thus, effects are a function of the *joint* action rather than of the local actions, as can be expected, due to possible interactions between local actions. For the sake of semantic clarity, we assume that if (c, e) and (c', e') are conditional effects of the same joint action, then c and c' are inconsistent. Here we focus on deterministic effects, but one can model non-deterministic effects simply by allowing for multiple pairs of the form $(c, e), (c, e')$ representing alternative outcomes of the action under the same conditions. The preconditions and effects functions, taken together, define the transition function from one state to another given actions.

For every joint action $\vec{a}$ and agent i, $Obs(\vec{a}, i) = \{p_1, \ldots, p_k\}$, where $p_1, ..., p_k$ are the propositions whose value agent i observes after the joint execution of $\vec{a}$. The observation is private – i.e., each agent may observe different aspects of the world, and we assume that the observed value is correct and corresponds to the post-action value of these variables.

A solution to the factored model is identical to that used for the flat model. We can use joint policy trees or executable global policy trees, as discussed earlier.

Example 1. We now illustrate the factored QDec-POMDP model using a simple box pushing domain (Figure 1). In this example there is a one dimensional grid of size 3, with cells

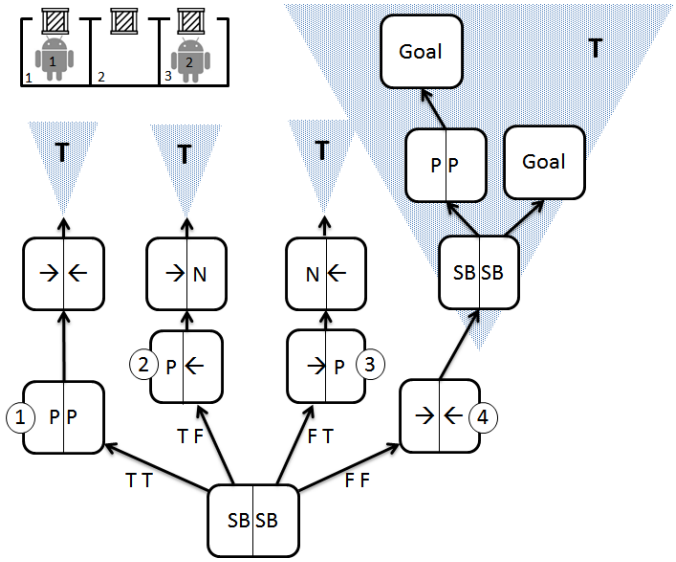

Figure 1: Illustration of Example 1 showing the box pushing domain with 2 agents and a possible joint policy tree with nodes labeled by joint actions. Possible agent actions are sensing a box at the current agent location (denoted SB), moving (denoted by arrows), pushing a box up (denoted P), and no-op (denoted N). On the second level of the tree, nodes marked 1 and 2 must have the same action for agent 1 (push up in this case), because agent 1 cannot distinguish between these two nodes. Likewise for nodes 2 and 4 with respect to agent 2 that cannot distinguish between them.

marked 1-3, and two agents, starting in cells 1 and 3. In each cell there may be a box, which needs to be pushed upwards. The left and right boxes are light, and a single agent may push them alone. The middle box is heavy, and requires that the two agents push it together.

We can hence define $I = \{1, 2\}$ and $P = \{AgentAt_{i,pos}, BoxAt_{j,pos}\}$ where $pos \in \{1, 2, 3\}$ is a possible position in the grid, $i \in \{1, 2\}$ is the agent index, and $j \in \{1, 2, 3\}$ is a box index. In the initial state each box may or may not be in its corresponding cell — $b_0 = AgentAt_{1,1} \wedge AgentAt_{2,3} \wedge (BoxAt_{j,j} \vee \neg BoxAt_{j,j})$ for $j = 1, 2, 3$. There are therefore 8 possible initial states.

The allowed actions for the agents are to move left and right, and to push a give box up. There are no preconditions for moving left and right, i.e. $Pre(Left) = Pre(Right) = \phi$. To push up box j, agent i must be in the same place as the box. That is, $Pre(PushUp_{i,j}) = \{AgentAt_{i,j}, BoxAt_{j,j}\}$. The moving actions transition the agent from one position to the other, and are independent of the effects of other agent actions, e.g., $Right_i = \{(AgentAt_{i,1}, \neg AgentAt_{i,1} \wedge AgentAt_{i,2}), (AgentAt_{i,2}, \neg AgentAt_{i,2} \wedge AgentAt_{i,3})\}$. The only truly joint effect is for the actions that contain a component $PushUp_{i,2}$, where box 2 is the heavy box — $Eff(PushUp_{1,2}, a_2)$ where a_2 is some other action, are identical to the independent effects of action a_2, while $Eff(PushUp_{1,2}, PushUp_{2,2}) = \{(\phi, \neg BoxAt_{2,2})\}$, that is, if and only if the two agents push the heavy box jointly, it (unconditionally) gets moved out of the grid.

We define sensing actions for boxes — $SenseBox_{i,j}$, with precondition $Pre(SenseBox_{i,j}) = AgentAt_{i,j}$, no effects, and $Obs(SenseBox_{i,j}) = BoxAt_{j,j}$. The goal is to move all boxes out of the grid, i.e., $\bigwedge_j \neg BoxAt_{j,j}$.

Compilation-Based Method

We now present a method for solving QDec-POMDP problems using a compilation approach to classical planning. Our approach generates a planning problem whose solution corresponds to an executable global plan tree, branching on agent observations. It relies on the approximate, sound, but incomplete notion of belief state, as discussed earlier.

The compilation method is currently designed for deterministic QDec-POMDPs, i.e., ones where actions have deterministic effects. The method could in principal be expanded to handle non-determinism by embedding the uncertainty of action effects into the uncertainty of the initial belief (Yoon et al. 2008), but this will clearly impact the solution time and size. We hence leave discussion of efficient handling of non-deterministic effects to future research.

A classical planning problem is a tuple $\pi = \langle P, A, s_0, G \rangle$ where P is a set of propositions, A is a set of actions, s_0 is the initial state, and G is a set of goal propositions. We use a translation method inspired by the MPSR translation method (Brafman and Shani 2012) and improves upon it. An important concept in this translation is *distinguishability* between states. We say that we can distinguish at runtime between two states s, s', denoted $\neg s/s'$, if we observed the value of some proposition p which is true in s and false in s'. In our translation we have two types of distinguishability — when a single agent can distinguish between states based on its own observations, denoted $\neg s/s'|i$, and when the combined observations of the agents can distinguish between observations, denoted $\neg s/s'$, as in MPSR.

Given a factored QDec-POMDP problem $\pi = \langle I, P, \vec{A} = \{A_i\}, Pre, Eff, Obs, b_0, G \rangle$ defined as in the previous section we create the following classical planning problem $\pi_c = \langle P_c, A_c, s_{0_c}, G_c \rangle$:

Propositions $P_c = \{p/s : p \in P, s \in S\} \cup \{\neg s/s' : s, s' \in S\} \cup \{\neg s/s'|i : s, s' \in S, i \in I\}$. Propositions of the form p/s capture the value at run time of p when s is the true initial state. Propositions of the form $\neg s'/s|i$ denote that at run-time, if s is the true initial state, then agent i has gathered sufficient data to conclude that s' cannot be the true initial state, i.e., to distinguish between state s and s'. These propositions allow us to define the agent-specific belief state during execution. These will be used later to enforce the constraint on actions at the same level explained in the previous section. Propositions of the form $\neg s'/s$ allow us to distinguish between states that at least one of the agents can distinguish between. These propositions allow us to define the joint belief state during plan construction.

Actions For every joint action $\vec{a}$ and every subset of $S' \subseteq b_0$, A_c contains an action $a_{S'}$. This action denotes the execution of $\vec{a}$ when the set of possible states is S'. $a_{S'}$ has no effect on states outside S'! It is defined as follows: $pre(\vec{a}_{S'}) = \{p/s : s \in S', p \in pre(\vec{a})\} \cup \{\neg s'/s : s' \in$

Table 1: Execution time (seconds) for different box pushing domains, comparing our translation-based QDec-POMDP approach, and two Dec-POMDP solvers, IPG and GMAA-ICE with the Q_{MDP} heuristic. A model is defined by its width (W), length (L), and number of boxes (B). Average depth denotes the average depth of leaves in the policy tree. Expected cost was reported by the GMAA-ICE solver.

Domain W, L, B	$\|S\|$	$\|b_0\|$	QDec-POMDP	Avg depth	IPG IPG	GMAA-ICE	Expected cost
2 , 2 , 2	256	4	12.79	2	450	15.32	2
2 , 3 , 2	1296	4	25.39	2	×	59.67	2
2 , 3 , 3	7776	8	48.42	5	×	732.59	5
3 , 3 , 3	59049	8	66.47	6	×	×	×

$S', s \in b_0 \setminus S'\}$. That is, the preconditions must hold prior to applying the action in all states for which this action applies, *and* the joint knowledge of the agents must be sufficient to distinguish between any state in S' and every state *not* in S'. Thus, the plan can choose action $a_{S'}$ only when the current belief state is S', and all action preconditions are known to hold in S'.

For every $(c, e) \in$ *effects*(a), *effects*$(a_{S'})$ contains the following conditional effects:

- For each $s \in S'$, $(c/s, e/s)$ — the effect applied to every state in S'.
- $\{(p/s \wedge \neg p/s', \neg s/s'|i)\}$ — for every p observable by agent i, and every two states $s, s' \in S'$, if the states disagree on p, then agent i can distinguish between the states following the observation of the value of p.

Initial State $s_{0_c} = \bigwedge_{s \in b_0, s \models l} l/s$ — for every literal we specify its value in all possible initial states.

Goal $G_c = \{\bigwedge_{s \in b_0} G/s\}$ — we require that the goal will be achieved in all states.

In addition we must explicitly enforce the constraints on nodes at the same depth or level, as explained in the previous section. To avoid the dependency on the depth, which is a numeric variable, unsupported by the planners that we use, we enforce the plan construction to proceed in a breadth-first-search (BFS). That is, each level in the tree must be fully constructed before the next level can be started. To achieve that we add for each state s in the initial belief a proposition $LevelDone_s$. For each compiled action $\vec{a}_{S'}$ we add preconditions $\bigwedge_{s \in S'} \neg LevelDone_s$, and unconditional effects $\bigwedge_{s \in S'} LevelDone_s$. Thus, once a state has been handled at the current level of the tree, no action that applies to it can be executed at the current level. To move to the next level, we add an action $ResetLevel$ with preconditions $\bigwedge_{s \in b_0} LevelDone_s$ and unconditional effects $\bigwedge_{s \in b_0} \neg LevelDone_s$. That is, once all states have been taken care for the current level, the $LevelDone$ propositions are reset and the next level begins. Our method adds only $|b_0|$ additional propositions to the translation.

After enforcing a BSF plan construction, we enforce that all agent actions at the current level in different states can be different only if the agent can distinguish between the states. As the ability to distinguish between states is a result of a different observation, this achieves the validity constraint required for global policy trees to become executable, as discussed in the previous section. We add for each agent i and action $a_i \in \{A_i\}$ predicates $constraint_{a_i,s}$, modeling which states are constrained on a_i. For every action $\vec{a}_{S'}$ we add preconditions:

$$\bigwedge_{i \in I, s \notin S'} \neg LevelDone_s \wedge (constraint_{a_i,s} \vee (\wedge_{s' \in S'} \neg s/s'|i))$$

where a_i is the action assigned to agent i in $\vec{a}_{S'}$. That is, for each agent i and state s which is not handled by the action, either s has not yet been handled by any other action, and is hence unconstrained, or there is a constraint of s and it matches a_i, or we can distinguish between s and any other state $s' \in S'$ for which the action does apply. We also add unconditional effects $\bigwedge_{i \in I, s \in S'} constraint_{a_i,s}$, specifying the new constraint induced when selecting the joint action. When a level is done, we remove all constraints in the $ResetLevel$ action, i.e., we add to $ResetLevel$ unconditional effects $\bigwedge_{i \in I, a_i \in A_i, s \in b_0} \neg constraint_{a_i,s}$.

The solution to the classical problem above is a linearization of a joint plan tree (Brafman and Shani 2012).

Example 2. We now describe a portion of the compilation of the box pushing domain described in Example 1. The set of possible initial state can be described as $s_{b_1 b_2 b_3}$ where b_i denotes whether b_i is initially in the grid and must be pushed up. For example, s_{tft} denotes that box 1 and 3 are in the grid, and box 2 is not. The propositions are conditioned on the initial states, and we thus have, e.g., $BoxAt_{j,pos}/s_{ftf}$, and $AgentAt_{i,pos}/s_{ttf}$.

For each subset of states we define one instance of an action. For example, for $S' = \{s_{ttt}, s_{fff}\}$, and action $Left_i$ we will define an action $Left_{i,S'}$ with preconditions $\bigwedge_{s \notin \{s_{ttt}, s_{fff}\}} \neg s_{ttt}/s \wedge \neg s_{fff}/s$. We also need to ensure the BFS expansion, by adding $\neg LevelDone_{s_{ttt}} \wedge \neg LevelDone_{s_{fff}}$. Finally, we ensure that the proper execution tree structure holds by adding $\bigwedge_{s \notin \{s_{ttt}, s_{fff}\}} constraint_{Left_i,s} \vee (\neg s_{ttt}/s'|i \wedge \neg s_{fff}/s'|i)$. The effects of the action are specified only for s_{ttt} and s_{fff}: $(AgentAt_{i,3}/s_{ttt}, \neg AgentAt_{i,3}/s_{ttt} \wedge AgentAt_{i,3}/s_{ttt})$, $(AgentAt_{i,3}/s_{fff}, \neg AgentAt_{i,3}/s_{fff} \wedge AgentAt_{i,3}/s_{fff})$. In addition, we add to the effects $LevelDone_{s_{ttt}} \wedge LevelDone_{s_{fff}}$ so that these states will not be handled again at the current depth. Next, we add the resulting constraint effect $constraint_{Left_i,s_{ttt}} \wedge constraint_{Left_i,s_{fff}}$ ensuring that all states undistinguishable from $\{s_{ttt}, s_{fff}\}$ must also use $Left_i$ at the current tree depth.

Experimental Results

We now provide some proof-of-concept experimental results showing that our algorithm can solve considerable size QDec-POMDP problems. We experiment with a variant of the box pushing problem (Seuken and Zilberstein 2007) where a set of boxes are spread in a grid, and the agents must push each box to a designated location at the edge of the grid (the end of the column it appears in). Each box may be either in a pre-specified location, or at its goal location to begin with, and the agent must be in the same location as the box in order to observe where it is. Agents may move in the 4 primary directions, and can push boxes in these 4 primary directions, if they occupy the same location as the box. Some boxes are heavy and must be pushed by a few agents jointly (in our example, heavy boxes are pushed by 2 agents). Agents can also only observe the location of other agents when they are in the same location. All transitions and observations are deterministic.

We experimented with four box pushing domains. The smallest example that we tried was a 2×2 grid, with 2 boxes and 2 agents and the largest had a 3×3 grid with 3 boxes. Each A_i has 11 possible actions (4 move actions, 4 push actions, observing the other agent, and observing each box), and hence there are 121 joint actions. We ran two Dec-POMDP solvers on this fully deterministic Dec-POMDP problem — the GMAA-ICE algorithm with the Q_{MDP} search heuristic (Oliehoek, Spaan, and Vlassis 2008) using the MADP package[1], and Incremental Policy Generation (IPG) (Amato, Dibangoye, and Zilberstein 2009). The results are presented in Table 1. Our compilation approach solves all the problems using the Fast Downward (FD) classical planner (Helmert 2006), while IPG solves only the smallest instance, and GMAA-ICE solves the smaller instances but not the larger one. Manually observing the trees, we saw that the planner computed the intuitive plan tree.

We acknowledge that this comparison is not entirely fair, because Dec-POMDP solvers try to optimize solution quality, whereas we only seek a satisfying solution. Thus, Dec-POMDP solvers may need to explore many more branches of the search graph, at a much greater computational cost. Furthermore, many Dec-POMDP solvers are naturally anytime, and can possibly produce a good policy even when stopped before termination. It may well be that solvers may reach a satisfying policy, which is the goal in a QDec-POMDP, well before they terminate their execution. That being said, our experiments demonstrate that our approach can provide solutions to decentralized problems and may be competitive with current Dec-POMDP solvers.

Our experiments investigate scaling up in terms of states and the horizon, yet another source of complexity in Dec-POMDP problems is the number of agents. It would be interesting to examine in future work how our approach scales with the number of agents.

An interesting aspect of our approach is the ability to compactly represent large problems. For example, the 3×3 box pushing example that we describe above, required a model size of over 1GB (specifying only non-zero probabilities) in the traditional Cassandra format for Dec-POMDPs, while our factored representation required less than 15KB.

[1] staff.science.uva.nl/~faolieho/madp

Conclusion

We presented a new model for multi-agent planning problems, called QDec-POMDP, which emphasizes valid, rather than optimal solutions, that achieve a given goal, in the spirit of classical and contingent planning. We analyzed the complexity of the new model, concluding that it is as hard as the standard Dec-POMDP model for a given horizon. Then, we presented a factored version of this model, motivated by similar representations used in classical and contingent planning. Our representation is compact and can describe models with tens of thousands of states and about 150 joint actions using file sizes of less than 15KB. We intend to investigate even more compact methods for specifying the effects of joint actions. Next, we described a solution method for deterministic QDec-POMDPs, based on a compilation approach to classical planning. Our method creates a classical planning problem whose solution is a linearized joint plan tree. We demonstrated the advantage of this compilation method over Dec-POMDP solvers using a number of examples. Our approach solves small problems much faster and scales to larger problems compared to existing Dec-POMDP solvers.

In this paper, our focus was on providing an exposition of the model, its properties, and potential. Of course, this is only the first step towards developing more scalable solvers for QDec-POMDP domains. In particular, we know well from contingent planning that it is much harder to scale up offline solution methods. Hence, we intend to explore online planning in QDec-POMDPs. This raises some non-trivial challenges as we will need some mechanism that will allow different agents with different belief states to jointly plan (Wu, Zilberstein, and Chen 2011), unlike the offline case in which a global plan is generated for a group of agents that share an initial belief state. The advantage, however, is that agents can focus on the relevant part of the state space at each planning phase, requiring smaller encodings and smaller plans. In addition, online methods are likely to better deal with non-deterministic effects. A second possible direction for scaling up would allow agents to plan independently, enforcing certain constraints on the joint solution.

Finally, it would be interesting to study variants of the QDec-POMDP model in more detail to identify the sources of its complexity, and, in particular, variants that have lower complexity. For example, we suspect that solving QDec-POMDPs with deterministic transitions might belong to a lower complexity class. Additional insights concerning belief state representation may also help yield more efficient algorithms.

Acknowledgments

Support for this work was provided in part by the National Science Foundation under grants IIS-0915071 and IIS-1116917, the Paul Ivanier Center for Robotics Research and Production Management and the Lynn and William Frankel Center for CS Research.

References

Amato, C.; Dibangoye, J. S.; and Zilberstein, S. 2009. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2–9.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27:819–840.

Brafman, R. I., and Shani, G. 2012. A multi-path compilation approach to contingent planning. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*, 1868–1874.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Kumar, A.; Zilberstein, S.; and Toussaint, M. 2011. Scalable multiagent planning using probabilistic inference. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2140–2146.

Lewis, H. R. 1978. Complexity of solvable cases of the decision problem for the predicate calculus. In *Proceedings of the Nineteenth Annual Symposium on Foundations of Computer Science*, 35–47.

Oliehoek, F. A.; Spaan, M. T. J.; Whiteson, S.; and Vlassis, N. 2008. Exploiting locality of interaction in factored DEC-POMDPs. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 517–524.

Oliehoek, F. A.; Spaan, M. T. J.; and Vlassis, N. A. 2008. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research* 32:289–353.

Papadimitriou, C. H. 1994. *Computational Complexity*. Reading, MA: Addison-Wesley.

Seuken, S., and Zilberstein, S. 2007. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 344–351.

Seuken, S., and Zilberstein, S. 2008. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems* 17(2):190–250.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175(2):487–511.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the Twenty-Third Conference on Artificial Intelligence*, 1010–1016.

Multi-agent Planning based on the Dynamic Selection and Merging of Hierarchical Task Networks

Gonzalo Milla-Millán and **Juan Fdez-Olivares** and **Inmaculada Sánchez-Garzón**
Dpto. de Ciencias de la Computación e I.A., Universidad de Granada, Spain
c/. Daniel Saucedo Aranda, s/n 18071 Granada, Spain
{gmillamillan, faro, isanchez}@decsai.ugr.es

Abstract

In this work we present a Multi-Agent Planning (MAP) approach which uses Hierarchical Task Networks (HTN) planning as the base of a conflict solving process between the local solutions from different local HTN planning problems. Several planning agents encoding the local problems (related to distinct planning domains) cooperate to that end exchanging a selection of the HTNs used to obtain valid local solution plans. A conflict solving agent is in charge of dynamically merging these local HTNs into a new global HTN planning domain, and using it to build and solve a new global HTN planning problem. This global problem contains also exclusive information - not known by any of the local planning agents - about the potential interactions between the local solutions and how to manage them. A preliminary experiment gives results demonstrating the validity of this approach.

1 Motivation

Planning for many real-world problems requires plans to comply with a set of domain-specific operating procedures. That is the case of detailing the steps of a care plan (the strategy to treat a patient) in the medical domain, where a Clinical Guideline (CG) encapsulates the knowledge to assist clinicians about appropriate health care for managing a single disease (Field and Lohr 1993). This knowledge involves operating procedures that can be encoded in a formal CG using a representation based on "*Task-Network Models*" (Peleg et al. 2003), from which care plans must be tailored to specific patients. Implementing these formal CGs requires of a language with the ability to express a significant amount of domain-specific knowledge (including ordering and time constraints, as well as high-level tasks comprising sequential and parallel control flow patterns), for which the classical planning paradigm is inadequate due to its constrained-action representation (Kambhampati et al. 1991; Miksch 1999). However, HTN planning (Erol, Hendler, and Nau 1996) has already been demonstrated to be a suitable approach for it thanks to (1) the possibility of using an HTN planning domain as a formal CG (Gonzalez-Ferrer et al. 2012), and (2) the ability of HTN planning algorithms to tailor this knowledge to the context data (i.e., the patient features and available resources) (Fdez-Olivares et al. 2011).

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

On the other hand, single-disease CGs can not be directly reused for patients suffering from more than one disease, which is known as the *comorbidities* problem (Boyd et al. 2005). That is because each CG encapsulates the knowledge about a single disease and may give recommendations which contradict those in other CGs, so interactions among the local care plans may arise when they must be simultaneously applied to a comorbid patient (a patient suffering from more than one disease). The work in (Tu et al. 2011) presents an ontology with these potential interactions including, for instance, drug interactions (two different drugs which must not be administered to the same patient may be prescribed by different CGs). Thus, the required knowledge for solving the global problem of treating a comorbid patient is distributed in (1) several single-disease CGs (one for each of the diseases that she/he suffers from), and (2) other sources of information with the potential interactions between them (e.g. an external database). Therefore, in order for a global care plan to be valid, it must (1) comply with the local constraints in every involved CG, and (2) avoid the potential interactions between different CG recommendations.

We present in this work a cooperative MAP approach to tackle this issue, which comprises several *planning* agents and a *conflict solving* agent. Each planning agent represents a single-disease specialist and is responsible for a local HTN planning problem encapsulating the knowledge about a single-disease CG (encoded as a formal CG in a local HTN planning domain). This partitioning allows for the distribution of the search for valid local solutions. The planning agents send the HTNs used to obtain their local solutions to the conflict solving agent. Using the local HTNs allows the conflict solving process (which is also based in HTN planning) to comply with all the single-disease constraints. The conflict solving agent has knowledge about the potential interactions between local solutions, and uses this knowledge together with the local HTNs to build a global HTN planning problem, whose solutions are free of interactions and comply with the local constraints. If such a global solution does not exist, the conflict solving agent asks the planning agents for alternative local proposals. The whole process is repeated until a global solution is found or no more local proposals can be given by any of the planning agents.

In the next section, a problem example is introduced to guide the explanation through the rest of the paper.

2 Problem Example

In this section a problem example is introduced to guide the explanation. This example is very simple and not intended to represent a real clinical case. However, it contains key elements involved in the problem of tailoring a care plan to a comorbid patient. Due to its simplicity (most of the plans comprise a single action), it could seem that classical planning could be a valid approach to afford it. However, a real CG comprises much more expert knowledge (including ordering and time constraints, as well as high-level tasks with sequential and parallel control flow patterns) which prevents classical planning to be used due to its constrained-action representation (Kambhampati et al. 1991).

Patient *John* (a 15 years old boy who weighs 35 kilos) has been diagnosed with both diseases X and Y. Two separated CGs exist, which encapsulate the local knowledge about each disease (*guideline-X* and *guideline-Y*, respectively). For the specific case of *John*, *guideline-X* has knowledge about two potential drugs to be administered: $X1$ and $X2$, from which only one of them must be selected (preferably in the given order). Same happens for *guideline-Y*, which has knowledge about three possible drugs for treating disease Y in *John*: $Y1$, $Y2$, and $Y3$, from which only one of them must be selected (preferably also in the given order). Applying the knowledge of each guideline, $X1$ and $Y1$ would be the local treatments for diseases X and Y respectively.

However, $X1$ and $Y1$ are known to interact between them, which means that they must not be administered to the same patient in simultaneous treatments. This is known as a *drug-drug* interaction (Tu et al. 2011). Each separated guideline encapsulates the knowledge about only one of the diseases, so none of them knows anything about this interaction or how to manage it. This knowledge is in an external source out of the guidelines (e.g., an external database).

This example will guide the explanation through the rest of the work. Next section gives the required notions of HTN planning and describes how it can be used to obtain care plans from single-disease CGs. Section 4 follows with a description of the global problem faced in this work in terms of several other local HTN problems. Then our approach to tackle this global problem is presented in section 5. A preliminary experiment is presented together with its results in section 6. Section 7 gathers the related work and section 8 presents our conclusions and plans for future work.

3 HTN Planning for Single-Agent Problem Solving

A single-agent HTN planning problem is composed of a ***domain***, an ***initial state*** and a set of ***goals***[1]. The ***initial state*** is a set of literals that describes the facts that are true at the beginning of the problem. The ***domain*** is designed in terms of a hierarchy of compositional activities which can be primitive or non-primitive. Primitive activities are named **operators** and are similar to those in classical planning, with related *preconditions* and *effects*. Non-primitive activities are symbolic activities named **compound tasks** which need to be decomposed into simpler activities. Decomposition **methods** encode how compound tasks can be decomposed and are defined as m = (t, `Cond`, `Subacts`), being t a compound task; `Cond` a set of applicability conditions; and `Subacts` a partially-ordered set of activities. `Subacts` represents the way that t should be decomposed according to m if `Cond` hold in the current state. Unlike classical planners, ***goals*** in HTN planning are not specified as a well formed formula that must be made true by the planner from the initial state, but in terms of an **initial task network**, which consists of a partially ordered set of ground compound tasks that need to be decomposed applying the decomposition methods until a plan formed of only **actions** - ground operators - is found.

The problem of tailoring care plans to patients from a single-disease CG using HTN planning has already been studied in (Fdez-Olivares et al. 2011; Gonzalez-Ferrer et al. 2012). Figures 1, 2, and 3 illustrate how this can be done for disease X and patient *John* (assuming that he doesn't suffer from any other disease). They represent a single-agent setting where figures 1 and 2 are the initial state and the basic knowledge of the domain respectively. This information is shown in HPDL code (Castillo et al. 2006), which is based on PDDL 2.2 (Edelkamp and Hoffmann 2004) and is able also to manage HTN planning specific features. HPDL has already been shown to be as expressive as standard languages for representing formal CGs (Gonzalez-Ferrer et al. 2012), which makes it an ideal tool for the comorbidities problem which will guide the explanation. Figure 3 uses a graphical notation to depict the hierarchy of activities which formalizes the expert knowledge of the CG. The planning goal for the problem of tailoring a care plan for *John* from *guideline-X* can be represented with the initial task network: `(Treat-X john)`.

4 Global Problem Definition

The global problem must be tackled through several local HTN planning problems for which (1) initial states have some common parts and (2) their local solution plans must be executed in parallel. However, interactions between the local plans may arise due to their parallel execution and the common parts in the initial states. The encapsulated knowledge in any of the local problems does not know anything about these potential interactions. That is the case of the comorbidities example where the local HTN planning problems are those encoding the local management of diseases X and Y respectively; the local solution plans are the separated treatments for each disease; and the common part of the initial states is that relating to patient *John*. Figures 1, 2 and 3 reflect the local problem for disease X, and so do figures 4, 5 and 6 for disease Y. However, the interaction among drugs $X1$ and $Y1$, prevent them to be prescribed together in a global plan for *John*, who suffers from both diseases. None of the local formal guidelines knows anything about this potential interaction. So the global problem consists of obtaining a global plan which solves all the local problems and avoids all the potential interactions among them.

[1] Following (Ghallab, Nau, and Traverso 2004), we use the term "**problem**" to make reference not only to the initial state and the goals, but also to the domain which they relate to.

```
(:objects
    john - patient   X1 X2 - drug)
(:init
  (male john)        (= (age john) 15)
  (drugclass X1 boy) (drugclass X2 boy))
```

Figure 1: Single-agent initial state for managing disease X

```
(:types
    patient drug class - object)
(:constants
    boy man girl woman - class)
(:predicates
  (male ?p - patient)
  (female ?p - patient)
  (drugclass ?d - drug ?c - class))
(:functions
  (age ?p - patient))
(:action administer
 :parameters (?p - patient ?d - drug ?c - class)
 :precondition (drugclass ?d ?c)
 :effect ())
```

Figure 2: Single-agent basic knowledge of the domain for *guideline-X*

5 Using HTNs for cooperative conflict solving

The main reason to use HTN planning as the base for the conflict solving process is that this process must be aware of and comply with the local constraints imposed by the local domains (the single-disease formal CGs), which are already encoded in the local HTNs. The main reason to use a cooperative MAP approach is to distribute the search for valid local solutions between different planning agents, each one representing a single-disease specialist. Thus, the HTN-based conflict solving process gathers the local HTNs used by each specialist to find a valid local solution and merge them with information about the potential interactions. The result of this merging process is a new HTN planning problem (with a new related HTN planning domain) ready to manage conflict-solving information. Using the local HTNs to build the global HTN planning domain guarantees that the global solutions comply with the local constraints.

Furthermore, designing and implementing a new planning domain for the global problem is not possible due to complexity and privacy reasons. For instance, in the comorbidities problem, it would be all but realistic to create a new planning domain for every potential combination of diseases. Real CGs are much more complex than the ones presented in sections 3 and 4, and joining them in a single domain would be a thorough and complex task. Furthermore, the approach here presented is intended for combinations of an arbitrary number of diseases (not only two) which would make that implementation task even harder. Regarding the privacy issues, they refer mainly to the requirement of not disclosing the local operating procedures, which are represented in the HTNs. Though this privacy aspect could not seem so relevant in the comorbidities problem, we expect our approach to be domain independent and think that

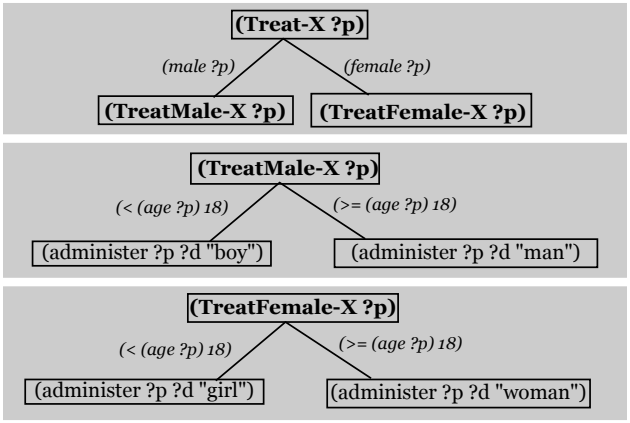

Figure 3: Single-agent HTNs of the domain for *guideline-X*. Activities are surrounded by a rectangle. **Compound tasks** are in bold font and **operators** in regular font. **Decomposition methods** are depicted with a line hanging from their related compound task to their `Subacts` network. The **applicability conditions** of each method are at one side of it.

```
(:objects
    john - patient     Y1 Y2 Y3 - drug)
(:init
  (= (weight john) 35)
  (drugclass Y1 thin)  (drugclass Y2 thin)
  (drugclass Y3 regular))
```

Figure 4: Single-agent initial state for managing disease Y

this feature can be useful in some other domains.

Nevertheless, instead of designing and implementing a new domain, the global problem can be tackled via the local problems plus the information about potential interactions among them. In this section we present our proposal for achieving it, which consists of dynamically creating a new **global** HTN planning problem and using it to solve the potential conflicts among the local solutions via standard HTN planning techniques. This global problem has its own related ***global initial state***, ***global domain*** and ***global goals***, in terms of which we explain how to build it up.

5.1 Global Initial State

The global initial state is built joining the local initial states and adding the information about potential interactions. Figure 7 depicts the global initial state for the example of section 2. It gathers the information from the local initial states (figures 1 and 4) and adds the drug interaction among $X1$ and $Y1$ with the predicate `(interact X1 Y1)`.

At a first glance, it could seem that using this global initial state in the local problems could help somehow in detecting the interactions, but that is not the case. For instance, Y drugs can not be prescribed using the local knowledge of disease X, so interactions will never happen though they are added to the local initial state. Moreover, the privacy issues commented above prevent the inclusion of expert knowledge from *guideline-Y* in *guideline-X* and vice-versa.

```
(:types
    patient drug class - object)
(:constants
    thin regular overweight - class)
(:predicates
  (drugclass ?d - drug ?c - class))
(:functions
  (weight ?p - patient))
(:action administer
 :parameters (?p - patient ?d - drug ?c - class)
 :precondition (drugclass ?d ?c)
 :effect ())
```

Figure 5: Single-agent basic knowledge of the domain for *guideline-Y*

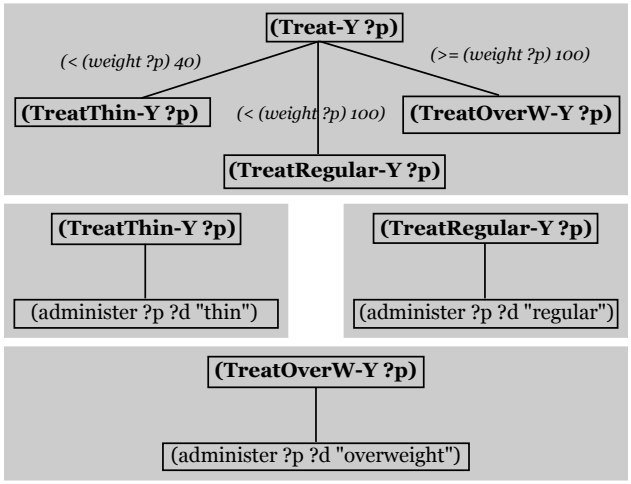

Figure 6: Single-agent HTNs of the domain for *guideline-Y*.

5.2 Global Domain

Broadly speaking, an HTN planning domain can be divided in two main parts: the operators and the HTNs.

The operators of the global domain are created extending the local ones to properly manage information about the interactions. For instance, a possibility to manage drug interactions is shown in figure 8. There, a new operator `cooperative-administer` is created in the global domain, extending the common operator `administer` of the local ones[2]. `cooperative-administer` reflects in the state (through its effects) the drugs prescribed to a patient, and checks in its preconditions that the drug to be prescribed does not interact with any of the drugs already administered to the patient. Looking now at figure 7 it can be seen how the constant representing drug $Y1$ will never be used to instantiate the parameter `?d` of `cooperative-administer` if $X1$ has been already administered to the same patient. The `:derived` predicate reflects the fact that interactions work in both directions (if $X1$ interacts with $Y1$, then $Y1$ interacts with $X1$).

The other main part of an HTN planning domain are the

[2]For ease of explanation, local problems are assumed to have the same basic operators (as for `administer` in figures 2 and 5).

```
(:objects
    john - patient   X1 X2 Y1 Y2 Y3 - drug)
(:init
  (male john)
  (= (age john) 15)  (= (weight john) 35)
  (drugclass X1 boy) (drugclass Y1 thin)
  (drugclass X2 boy) (drugclass Y2 thin)
                     (drugclass Y3 regular)
  (interact X1 Y1))
```

Figure 7: Global initial state

```
(:predicates
  (interact ?d1 ?d2 - drug)
  (administered ?p - patient ?d - drug))
(:action cooperative-administer
 :parameters (?p - patient ?d - drug ?c - class)
 :precondition (and (drugclass ?d ?c)
                    (forall (?d2 - drug)
                      (not (and (administered ?p ?d2))
                                (interaction ?d ?d2))))
 :effect (administered ?p ?d))
(:derived (interaction ?d1 ?d2 - drug)
           (or (interact ?d1 ?d2) (interact ?d2 ?d1)))
```

Figure 8: Operators in the global domain

Hierarchical Task Networks, which encode the hierarchy of activities representing the expert knowledge of the domain. The HTNs of the global domain are built from HTNs subsets of the local domains, being each one of these subsets sufficient to obtain a local solution for its related local problem. Being sufficient means that a local solution can be obtained using only the methods contained in one of these subsets. These local subsets of HTNs are (1) dynamically selected by the planning agents from their local domains, (2) sent by the planning agents to the conflict solving agent, and (3) dynamically merged into the global domain by the conflict solving agent.

The reason to communicate HTNs instead of ground information is twofold: (1) the local HTNs make the conflict solving process aware of the local constraints, and (2) they allow to reduce the communication overload. If a planning agent sends a local (ground) solution and it is not valid for a global one, she must propose another (ground) local solution. However, a set of HTNs may represent several solution plans (e.g. with a different resource assignment), so they can be used together with the global initial state and operators explained above in the global planning problem to reassign conflicting resources.

Dynamic Selection of HTNs In order to compute a local subset of HTNs which is sufficient to obtain a local solution, it can be done bottom-up from the decomposition tree of a local solution plan. The decomposition tree of a plan is formed by the ground decomposition methods applied to the initial task network to obtain the plan. Figure 9 depicts decomposition trees for the local plans `(administer john X1 boy)` (left) and `(administer john Y1 thin)` (right) of our exam-

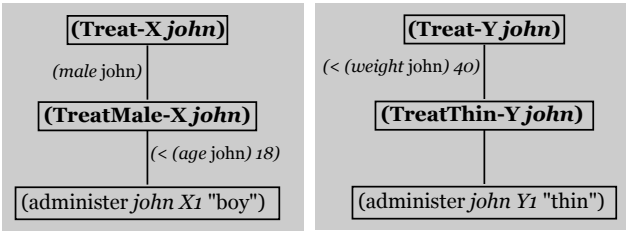

Figure 9: Decomposition trees.

Algorithm 1: Selection of HTNs

Input: HTN planning problem $\mathcal{P} = (s, G, O, M)$, solution plan π for $\mathcal{P}$, decomposition tree $Tree$ of π for $\mathcal{P}$
Output: A subset of HTNs M_s which is sufficient to solve $\mathcal{P}$
Data: Stack of pending (ground) activities $Pending = \emptyset$

1 $M_s = \emptyset$;
2 Add all actions in π to $Pending$;
3 **while** $Pending \neq \emptyset$ **do**
4 $\quad CurrentAct$ = $Pending$.pop();
5 $\quad ParentMethod$ = parent of $CurrentAct$ in $Tree$;
6 $\quad$ Add $ParentMethod$ to M_s;
7 $\quad$ **if** $CurrentAct \neq G$ **then**
8 $\quad\quad ParentAct$ = task of $ParentMethod$;
9 $\quad\quad$ Add $ParentAct$ to $Pending$;
10 return M_s;

ple. Algorithm 1 shows how to compute a sufficient subset of HTNs from a planning problem $\mathcal{P}$, a solution plan π and its related decomposition tree. In the notation $\mathcal{P} = (s, G, O, M)$, s is the initial state, G is the initial task network, O is the set of basic operators, and M is the set of decomposition methods. The HTNs are given by the methods in M. Without loss of generality and for ease of explanation, the initial task network G is assumed to consist of a single ground compound task. Firstly in algorithm 1, all actions in the plan are added to a stack of pending (ground) activities to process (line 2). Then, the parent method of each pending activity is taken from the decomposition tree and added to the subset of HTNs (lines 5 and 6). If the current method does not relate to the goal task, the algorithm looks up in the decomposition tree for the parent task and adds it to the stack of pending activities (lines 7 to 9). The algorithm ends when no more pending activities need to be processed (line 3) and returns the selected subset of HTNs (line 10). In figure 10 the corresponding local subsets of HTNs for `(administer john X1 boy)` (top) and `(administer john Y1 thin)` (bottom) can be seen. Note that the HTNs not used to solve the local problems are thrown away by the algorithm and that each local subset allows for different solution plans (e.g. both $X1$ and $X2$ drugs could be prescribed for patient *John* from the HTNs on top of figure 10).

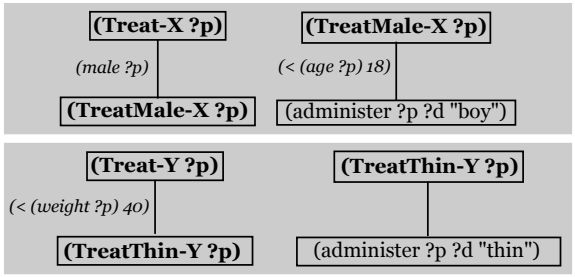

Figure 10: Local subsets of selected HTNs

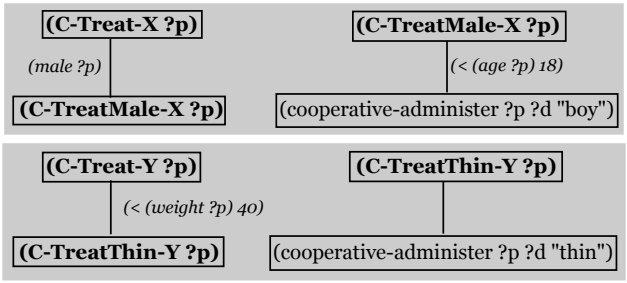

Figure 11: Global HTNs. The prefix "**C-**" stands for "*cooperative*" and has been added to the compound tasks names to differentiate them from the local ones.

Dynamic Merging of HTNs Once that the local subsets of HTNs are built, they must be merged into the global domain. The local HTNs refer to the basic operators of the local problems from which they were computed (through the `Subacts` part of the decomposition methods), but the HTNs of the global problem must refer to the extended global operators which are able to manage the information about interactions between local solutions. An example of this can be seen in figure 11 which depicts the global HTNs that are the objective of this merging step. There it can be seen how the global HTNs are similar to those of the local subsets (as in figure 10), but using the global operators instead of the local ones (e.g. `cooperative-administer` instead of `administer`). Therefore, the information of the local HTNs must be *assimilated* in the global domain. Algorithm 2 shows how this assimilation is done within the whole merging process. The notation for decomposition methods m = (t, `Cond`, `Subacts`) explained in section 3 is used here. For ease of explanation, the HTNs are broken down now in operators and methods (an HTN may refer to an operator in the `Subacts` part of a method and it is needed to know the local operator to find the corresponding global one). This algorithm must be called iteratively for each local subset of HTNs. Initially (before the first local subset of HTNs is assimilated in the global domain), O_G is the set of global operators as explained above, and the set of global methods $M_G = \emptyset$. $ActMap$ and $MethMap$ are data structures which maps activities and methods of the incoming local HTNs to the ones that are created in the global domain. The algorithm can be divided in the following main parts:

1. **Mapping of *local* to *global* operators** (lines 1-3): Correspondence between local (without interaction information) and global (with information about interactions) operators is added to $ActMap$.
2. **Assimilation of global methods and compound tasks** (lines 4-11): Corresponding methods and tasks are created in the global domain for the incoming local ones. The "*assimilation*" steps of lines 7 and 9 do create them, but referencing the global operators instead of the local ones. The `Subacts` part of the methods created in this step is always empty (line 10), because it may refer to an activity that has not been assimilated yet in the global domain. These `Subacts` parts of the methods are copied in the next main step.
3. **Creation of `Subacts` partially ordered sets** (lines 12-17): Once that all the activities have been assimilated in the global domain, the `Subacts` partially ordered sets of each method are properly copied (lines 14-16). Once that all the activities are added to the `Subacts` of a method, order constraints between them are copied (line 17).

Figure 11 depicts the output of this algorithm after being called for the local subsets of HTNs in figure 10 and the global operator in figure 8.

Algorithm 2: Merging local HTNs into the global domain.

Input: HTNs in the global domain: O_G (operators) and M_G (methods), HTNs from a local domain: O_l (operators) and M_l (methods)
Output: HTNs in the global domain: O_G (operators) and M_G (methods)
Data: Maps $ActMap = \emptyset$ and $MethMap = \emptyset$

1 **foreach** $o_l \in O_l$ **do**
2 Get global operator $o_G \in O_G$ related to o_l;
3 $ActMap$.add(o_l, o_G);
4 **foreach** $m_l = (t_l, Cond_l, Sub_l) \in M_l$ **do**
5 $t_G = ActMap$.get(t_l);
6 **if** $t_G = \emptyset$ **then**
7 $t_G =$ Assimilate t_l;
8 $ActMap$.add(t_l, t_G);
9 $Cond_G =$ Assimilate $Cond_l$;
10 Add $m_G = (t_G, Cond_G, \emptyset)$ to M_G;
11 $MethMap$.add(m_l, m_G);
12 **foreach** $m_l = (t_l, Cond_l, Sub_l) \in M_l$ **do**
13 $(t_G, Cond_G, Sub_G) = MethMap$.get($m_l$);
14 **foreach** $u_l \in Sub_l$ **do**
15 $u_G = ActMap$.get(u_l);
16 Add u_G to Sub_G;
17 Copy order constraints from Sub_l to Sub_G;

5.3 Global Goals

One of the key requirements given for the global problem in section 4 is that the local solution plans must be executed in parallel. So the global goal (the initial task network) in the global HTN planning problem is defined as the parallel planning for the local initial task networks. Thus, the global initial task network `[(C-Treat-X john) (C-Treat-Y john)]` (referring the global HTNs in figure 11), where [] denote parallel order, would be the global goal in the example of patient *John*.

The global goals refer to the global HTNs which make use of the global operators. These global operators are ready to detect and manage the interactions thanks to the information included in the global initial state. So the global problem constructed in this way is ready to be used for detecting and managing interactions like the drug interaction between $X1$ and $Y1$. To that end, standard HTN planning techniques can be used to solve the global problem, which automatically will change the resource assignment of $Y1$ to $Y2$ (or $X1$ to $X2$) without need for further local proposals.

5.4 Life-cycle of the MAP Approach

There are several *planning* agents and a *conflict solving* agent in our approach. Each planning agent encodes a local HTN planning problem and is responsible for finding a local solution, computing the related subset of HTNs and sending information about them to the conflict solving agent. The conflict solving agent encodes global knowledge - not known by any of the planning agents - about the potential interactions between local solutions and how to manage them, and uses it together with the received local information from the planning agents in order to build a new global HTN planning problem, whose solutions are free of interactions. In this section a data structure called *TreeStub* is presented as the local information that planning agents send to the conflict solving agent about their local solutions so she can built the global problem. Afterward, the whole life-cycle of the approach is detailed.

TreeStubs The planning agents send information to the conflict solving agent in the form of *TreeStubs*, which are data structures encapsulating all the required local information about a local solution so this information can be used to build a global HTN planning problem as explained above. A *TreeStub* contains a set of HTNs and an initial task network related to a solution of an HTN planning problem. The set of HTNs contains those HTNs used to obtain the solution and can be computed as in algorithm 1. The initial task network is the one from which the solution was deduced. The HTNs are needed by the conflict solving agent to be merged into the global domain. The local initial task network is required to build the global one as in section 5.3.

Life-cycle The life-cycle of our MAP approach consists of the following steps:

1. **Each planning agent selects a subset of HTNs:** A local HTN planning process is performed by each planning agent over its local problem to that end. Each planning agent uses its local solution plan together with its decomposition tree for selecting HTNs as in algorithm 1.
2. **Communication of *TreeStubs* to the conflict solving agent:** Each planning agent builds a *TreeStub* with the HTNs selected in the previous step and her local initial

task network and sends it to the conflict solving agent. In order to preserve privacy, *TreeStubs* can be communicated in an encrypted way. The concept of *obfuscation* from the field of software encryption (Collberg, Thomborson, and Low 1997; Kirk and Jenkins 2004) can be taken and adapted to that end.

3. **The conflict solving agent builds and tries to solve a new HTN global problem:** The conflict solving agent encodes the global initial state and global operators, and uses them together with the information in the incoming *TreeStubs* to build a global HTN planning problem. Algorithm 2 is used to merge the incoming local HTNs into the global domain. The global goal is built as explained in section 5.3. The conflict solving agent tries then to solve the so-built global problem using HTN planning techniques. If a global solution is found, it is guaranteed to (1) comply with the local constraints, thanks to the use of the local HTNs which encode them, and (2) be free of interactions, thanks to the fact that the global operators are ready to detect and manage the interactions included in the global initial state, as explained above.
4. **Alternative proposals:** in case that a global solution is not found, the conflict solving agent asks the planning agents for alternative proposals and the process from step 1 is repeated until a global solution is found or no more local proposals can be given by any of the planning agents. In the worst case, the conflict solving agent would ask to every planning agent for all their possible local proposals and would try every possible combination among them (one proposal by planning agent in each combination). Being the proposals based on the HTNs instead of on the ground plans, the planning agents do not need to communicate all their local plans to the conflict solving agent, because she can use the local HTNs to explore other valid local solutions herself (e.g. using a different resource assignment and/or scheduling).

The key point for obtaining the new proposals referred in step 4 is that the planning agents must not finish their local planning processes when a local solution plan is found. Instead of that, the local planning processes are kept in a *stand-by* mode. When a new proposal is requested to a planning agent (by the conflict solving agent), she triggers a backtracking process from the point where the last solution was found. This backtracking process is controlled by the planning agent to stop when a new local solution plan is found ***and*** *a not already discovered subset of HTNs* has been decomposed to obtain it. Thus, another *TreeStub* with this new subset of HTNs can be sent as an alternative local proposal to the conflict solving agent. For instance, if the interactions `(interact X1 Y2)`, `(interact X2 Y1)` and `(interact X2 Y2)` are added to the global initial state in figure 7 then the local subsets of HTNs depicted in figure 10 can not be used to build a global problem to avoid interactions (no drugs of class *thin* are compatible with any drug of class *boy*). Then, it is the conflict solving agent who must ask for new proposals to the planning agents. The agent in charge of `guideline-X` has not any more valid proposals to make (no more different subsets of HTNs valid for a local solution). However, the agent responsible for `guideline-Y` can find a new set of methods as in figure 12. This new proposal can be used by the conflict solving agent to build a new global HTN problem with the first proposal from `guideline-X` and to give the global solution plan `((administer john X1 boy) (administer john Y3 regular))`, which in fact solves all the local problems (manages every disease) and is free of interactions.

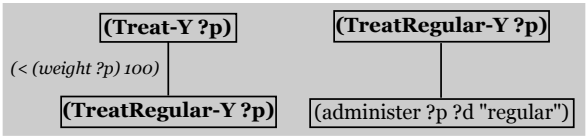

Figure 12: Second proposal of agent in charge of `guideline-Y`

6 Experiments and Results

In order to evaluate the cooperative MAP approach proposed, a preliminary experiment has been carried out for the example of drug-drug interactions in the comorbidities problem presented in section 2. Two different clinical guidelines (*guideline-X* and *guideline-Y*) are represented in separated HTN planning domains with a single but significant difference with figures 1 to 6: instead of directly using the operator `administer`, a more flexible recursive compound task is used to allow for more complex drug dosages as in table 1. This recursive compound task is depicted in figure 13. The global initial state and global set of operators have been manually created as in figures 7 and 8, but reflecting also the drug dosages as in table 1. Two planning agents X and Y have been implemented, each one being responsible for the local management of the disease with the same name. Thus, each planning agent encodes an HTN local problem for one of the diseases. A conflict solving agent has the knowledge of the handcrafted global initial state and operators. The interaction among $X1$ and $Y1$ drugs and how to manage it are not known by any of the planning agents but only by the conflict solving agent. Table 2 shows the local solutions found by agents X (left) and Y (right) respectively. Table 3 shows a global solution given by the conflict solving agent. This global solution was found using the HTNs selected for the local solutions of table 2. As it can be noted, the HTNs subset including recursive compound tasks allow to the conflict solving agent not only to make a different resource assignment, but also to give alternative plans with a different number of actions.

As shown in this experimental evaluation, the cooperative MAP approach presented in this work seems to be a suitable proposal for the automated generation of treatment plans for comorbid patients. This approach avoids the appearance of undesired drug interactions in global care plans. However, the approach is domain independent and can be used there where similar interactions may happen.

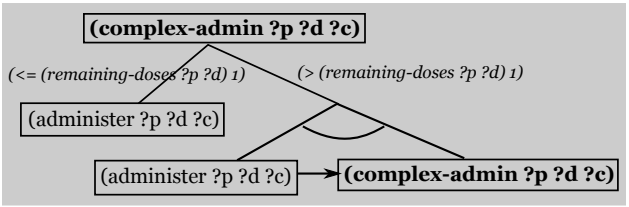

Figure 13: Compound task for a complex drug dosage. The number of doses is decremented each time that an `administer` action is added to the plan.

7 Related Work

Following the taxonomy of MAP approaches in (Durfee 2000), the work here presented can be categorized as *Distributed Planning for Centralized Plans*, same as those of (Kambhampati et al. 1991; Durfee, Kenny, and Kluge 1998; Conry et al. 1991). In (Kambhampati et al. 1991) the global problem of process planning for machined parts is divided into three different stages that must be managed in a sequential way. Those stages can not be distributed and tackled in parallel. In our case the local planning processes are distributed among the planning agents and tackled in parallel by them. (Durfee, Kenny, and Kluge 1998) use procedural knowledge to plan missions for a team of Unmanned Ground Vehicles which must coordinate at execution time to face possible context variances from expected. In our approach, the interactions which need for coordination are known *a priori* and must be managed at planning time to obtain an interaction-free global plan for the long term. In (Conry et al. 1991) the authors present a MAP approach for a multistage negotiation for restoral of transmission paths in a complex communication system. The impact of any local decision is ultimately due to particular resources which cannot be committed to different agents. In our case, the use of one resource by one agent inhibits the use of another (different) resource by other agent.

Different coordination techniques have also been proposed for merging and gathering several individual plans into a single joint plan, as for example the partial global planning framework (Durfee and Lesser 1991; Lesser et al. 2004) which allows agents to communicate their local plans to the rest of agents and merge this information into their own partial global plan to improve it. Their approach is more reactive than ours, which is focused on planning for the long term. Instead of ground information, we show how and why HTNs can be used.

Drug	Drug class	Required doses	Delay b/t doses
X1	boy	2	24h
X2	boy	4	12h
Y1	thin	2	24h
Y2	thin	6	8h
Y3	regular	4	12h

Table 1: Drug dosages.

Drug	Date / Time	Drug	Date / Time
X1	25-Mar / 12:00	Y1	25-Mar / 12:00
X1	26-Mar / 12:00	Y1	26-Mar / 12:00

Table 2: Local treatments for diseases X (left) and Y (right).

Drug	Date / Time
X1	25-Mar / 12:00
X1	26-Mar / 12:00
Y2	25-Mar / 12:00
Y2	25-Mar / 20:00
Y2	26-Mar / 04:00
Y2	26-Mar / 12:00
Y2	26-Mar / 20:00
Y2	27-Mar / 04:00

Table 3: An interaction-free global care plan given by the conflict solver

Regarding the problem of care planning for comorbid patients, it has already been addressed from different single-agent perspectives in works as (Lozano et al. 2010; Abidi 2009; Hing et al. 2010). The strength of agent-based representations has been exploited in projects like GLINDA (Tu et al. 2011) for detecting and repairing interactions and consolidating treatment recommendations. Though there is not much information about this last approach, it seems to not allow for so many tailoring capabilities as HTN planning.

8 Conclusions and Future Work

A MAP approach has been presented which uses HTN planning as the base for a conflict solving process between local planning problems. An experiment has been carried out and gives promising results about the validity of the approach. The approach is domain-independent and can be applied in those application domains where resource interactions between different local plans may arise. The modeling capabilities of temporal HTN planning are expected to serve for solving other types of conflicts such as time constraints between actions in different local solutions (e.g., two laboratory tests recommended by two guidelines may be consolidated on the same visit).

As for future work we plan to automatize those steps which are currently handcrafted, namely the creation of the initial global state and the global operators. On the other hand, we are currently involved in the study and formalization of CGs for real diseases which usually coexist in comorbid patients. These guidelines are much more complex than the ones presented in the example of section 2, and comprise a wider range of interactions (as those in the ontology of (Tu et al. 2011)). Implementing them will allow us to carry out a more thorough and extensive experimentation in order to (1) test the validity of our approach in real scenarios, and (2) make an study of its scalability.

Acknowledgments

This work is supported by the Spanish MICINN project TIN2011-27652-C03-03.

References

Abidi, S. 2009. A conceptual framework for ontology based automating and merging of clinical pathways of comorbidities. *Knowledge Management for Health Care Procedures* 55–66.

Boyd, C. M.; Darer, J.; Boult, C.; Fried, L. P.; Boult, L.; and Wu, A. W. 2005. Clinical practice guidelines and quality of care for older patients with multiple comorbid diseases. *JAMA: the journal of the American Medical Association* 294(6):716–724.

Castillo, L.; Fdez-Olivares, J.; García-Pérez, O.; and Palao, F. 2006. Efficiently handling temporal knowledge in an htn planner. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 63–72.

Collberg, C.; Thomborson, C.; and Low, D. 1997. A taxonomy of obfuscating transformations. Technical report, Department of Computer Science, The University of Auckland, New Zealand.

Conry, S. E.; Kuwabara, K.; Lesser, V. R.; and Meyer, R. A. 1991. Multistage negotiation for distributed constraint satisfaction. *Systems, Man and Cybernetics, IEEE Transactions on* 21(6):1462–1477.

Durfee, E. H., and Lesser, V. 1991. Partial Global Planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks* 21(5):1167–1183.

Durfee, E. H.; Kenny, P. G.; and Kluge, K. C. 1998. Integrated premission planning and execution for unmanned ground vehicles. *Autonomous Robots* 5(1):97–110.

Durfee, E. H. 2000. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press. chapter 3, 121–164.

Edelkamp, S., and Hoffmann, J. 2004. Pddl2. 2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC04), at ICAPS04*.

Erol, K.; Hendler, J.; and Nau, D. S. 1996. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence* 18:69–93.

Fdez-Olivares, J.; Castillo, L.; Cózar, J. A.; and García Pérez, O. 2011. Supporting clinical processes and decisions by hierarchical planning and scheduling. *Computational Intelligence* 27:103–122.

Field, E. M., and Lohr, K. 1993. Guidelines for clinical practice: From development to use. *BMJ* 306:17.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kauffman.

Gonzalez-Ferrer, A.; ten Teije, A.; Fdez-Olivares, J.; and Milian, K. 2012. Automated generation of patient-tailored electronic care pathways by translating computer-interpretable guidelines into hierarchical task networks. Artificial Intelligence In Medicine.

Hing, M. M.; Michalowski, M.; Wilk, S.; Michalowski, W.; and Farion, K. 2010. Identifying inconsistencies in multiple clinical practice guidelines for a patient with co-morbidity. In *2010 IEEE International Conference on Bioinformatics and Biomedicine Workshops (BIBMW)*, 447–452. IEEE.

Kambhampati, S.; Cutkosky, M.; Tenenbaum, M.; and Lee, S. H. 1991. Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 199–205.

Kirk, S. R., and Jenkins, S. 2004. Information theory-based software metrics and obfuscation. *Journal of Systems and Software* 72(2):179–186.

Lesser, V. R.; Decker, K.; Wagner, T.; Carver, N.; Garvey, A.; Horling, B.; Neiman, D. E.; Podorozhny, R. M.; Prasad, M. V. N.; Raja, A.; Vincent, R.; Xuan, P.; and Zhang, X. 2004. Evolution of the gpgp/tæms domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems* 9(1-2):87–143.

Lozano, E.; Marcos, M.; Martnez-Salvador, B.; Alonso, A.; and Alonso, J. 2010. Experiences in the development of electronic care plans for the management of comorbidities. *Knowledge Representation for Health-Care. Data, Processes and Guidelines* 113–123.

Miksch, S. 1999. Plan management in the medical domain. *AI communications* 12(4):209–235.

Peleg, M.; Tu, S.; Bury, J.; Ciccarese, P.; Fox, J.; Greenes, R. A.; Hall, R.; Johnson, P. D.; Jones, N.; Kumar, A.; et al. 2003. Comparing computer-interpretable guideline models: a case-study approach. *Journal of the American Medical Informatics Association* 10(1):52–68.

Tu, S. W.; Nyulus, C.; Martins, S. B.; Jung, H.; Kum, P.; Goldner, J.; Goldstein, M. K.; and Musen, M. A. 2011. GLINDA: GuideLine INteraction detection architecture. http://glinda-project.stanford.edu/.

Deterministic Multiagent Planning Techniques: Experimental Comparison (Short paper)

Karel Durkota[1] and **Antonín Komenda**[2]
karel.durkota@gmail.com, komenda@agents.fel.cvut.cz
[1]Faculty of Electrical Engineering, Czech Technical University in Prague
[2]Dept. of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract

Deterministic domain-independent planning techniques for multiagent systems stem from principles of classical planning. Three most recently studied approaches comprise (i) DisCSP+Planning utilizing Distributed Constraint Satisfaction Problem solving for coordination of the agents and individual planning using local search, (ii) multiagent adaptation of A* with local heuristics and (iii) distribution of the GraphPlan approach based on merging of planning graphs.

In this work, we summarize the principles of these three approaches and describe a novel implementation and optimization of the multiagent GraphPlan approach. We experimentally validate the influence of parametrization of the inner extraction phase of individual plans and compare the best results with the former two multiagent planning techniques.

Introduction

The problem of multiagent planning as defined in (Brafman and Domshlak 2008) is similarly important as classical planning, as it can provide generally usable techniques for intelligent agents, which are required to cooperatively come up with distributed plans. Recently the research community proposed both theoretical treatments and implementations of such distributed multiagent planning (DMAP) techniques.

Similarly to the classical planning, the agents in DMAP cooperatively search for the local sequences of actions, which after execution transform the world from an initial state to a common goal state. The local sequences of actions—the local plans—has to interleave appropriately, as each particular agent cannot possibly solve the problem on its own, but have to base its own actions on the results of actions of the other agents. Furthermore, the agents are motivated to communicate as few information as possible not to put load on the other agents if it is not needed.

Three recently theoretically treated approaches for DMAP are (i) multiagent planning utilizing a solver for Distributed Constrain Satisfaction Problems (DisCSP) for the coordination part and a classical planning for the individual plans denoted as DisCSP+Planning (Brafman and Domshlak 2008), (ii) extension of A* for multiagent systems coined Multiagent Distributed A*, (MA-A*) (Nissim and Brafman 2012a; 2012b) and (iii) Distributed Planning through Graph Merging (DPGM) (Pellier 2010) which uses principally the same factorization scheme for separation of parts of the original planning to more agents as in the previous approaches, defined originally in (Brafman and Domshlak 2008) together with the MA-STRIPS formalization.

First two approaches, namely DisCSP+Planning and MA-A*, were already implemented and experimentally validated. Works describing the implementation and experiments are for DisCSP+Planning (Nissim, Brafman, and Domshlak 2010) and for MA-A* the original papers (Nissim and Brafman 2012a; 2012b). However, according to our knowledge, the Pellier's approach was not implemented and experimentally verified yet. Therefore, our initial focus in context of this work was the implementation of the approach described by Pellier and comparing it with the other two approaches. Since this comparison was not done yet, it was not clear if the GraphPlan (Blum and Furst 1997) approach could be viable in multiagent setting, although the underlying approach in classical planning was outperformed already at (IPC 2004). Especially as in the multiagent setting the communication complexity can be of much more importance than the computational complexity.

Multiagent planning

Planning in a multiagent (MA) systems is by (Brafman and Domshlak 2008) a search for a plan for each agent, assuming that agents have to cooperate in order to reach a global goal. Formally, problem for a set of k agents $AG = \{ag_i\}_{i=1}^{k}$ is given by a quadruple $\Pi = \langle P, \{A_{ag_i}\}_{i=1}^{k}, I, G\rangle$, where P is a finite set of propositions describing facts holding in the world; $I \subseteq P$ is a set of propositions that hold in the initial state; $G \subseteq P$ is a set of propositions that must hold in a goal state; and A_{ag_i} is a set of actions that an agent ag_i can perform. Each action has a standard STRIPS syntax, i.e., $a = \langle pre(a), add(a), del(a)\rangle$, where $pre(a), add(a), del(a) \subseteq P$ and $add(a) \cap del(a) = \emptyset$. An action a can be performed only in a state $s \subseteq P$, which the propositions form $pre(a)$ hold in. Performing an action a will add to the state s propositions from $add(a)$ and remove the propositions from $del(a)$.

DisCSP+Planning-based planner The algorithm from (Nissim, Brafman, and Domshlak 2010) can be

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

described as two interleaving components, a *coordination component* and an *individual planning component*; both of which require the separation of the public and individual actions of each agent. An action a of an agent ag_i is public if there exists an action b of an agent ag_j, $i \neq j$, such that $(\text{pre}(a) \cup \text{add}(a) \cup \text{del}(a)) \cap (\text{pre}(b) \cup \text{add}(b) \cup \text{del}(b)) \neq \emptyset$, otherwise the action is considered individual. This separation defines the multiagent problem factorization.

The *coordination component* deals only with the public actions. It searches for a sequence of the interaction points between the agents' plans. For a given length of the public part of the plan δ, it tries to assign different public actions of each agent in the different time-steps so that the global goal is satisfied. This is the interaction part, so the resulting plan of each agent has to satisfy the requirements put by the rest of the agents and vice versa. These requirements are described in form of *coordination constraints* in the inner DisCSP problem. Solving this problem effectively means solving the multiagent planning problem. If some agent or the team as a whole can not solve the DisCSP, then δ, the length of the coordination public part of the plan is increased by one and the whole process is repeated.

The *individual planning component* forms the other type of constraints for the DisCSP process encoding the requirement of the local parts of the plan. The *individual planning constraints* limits usage of the public actions in the coordination part of the plan such that the gaps between them can be filled by sequences of individual actions of the agents.

Multiagent A* The algorithm proposed in (Nissim and Brafman 2012b) is inspired by the well-known A* algorithm. Similarly to centralized A* the Multiagent Distributed A* (MA-A*) maintains *open lists* for all agents, that keep track of the so far unvisited states, and *closed lists*, that keep track of the already visited states. Each agent also uses a local heuristic to decide which state from the open list it should expand as next. As stated in the paper, each agent can use different heuristics.

In the MA-A*, similarly to the DisCSP+Planning, it is firstly necessary to separate every agents' public and individual actions. The algorithm runs simultaneously for each agent. During the search, the agents send messages to each other to distribute the search at the points, where the other agents can follow. Effectively, it means the messages are sent only for states achieved by public actions. Each such message consists of a state s, its cost value $g_{ag_i}(s)$ and a heuristic estimate $h_{ag_i}(s)$. In decoupled problems, this principle allows distribution of the knowledge about the entire search space among the agents. When an agent receives a message with a state s, it decides either to: visit the state (by adding it to its *open list*), update its knowledge about s, or discard it if it knows better path to the state.

The search terminates if an agent expands a state which is compatible with the goal set G and the resulting plan is ensured to be globally optimal.

Distributed Planning through Graph Merging The algorithm DPGM presented in (Pellier 2010) uses as the main data structure a planning graph together with the distributed versions of algorithms for its building and for extraction of the resulting plan. The distributed extraction consists of a individual CSP and a distributed coordination mechanism. The planner as a whole can be described by following five phases: *global goal decomposition*, *expansion*, *planning graph merging*, *individual plan extraction*, *coordination*. The result is in form of a *coordinated individual solution plan*.

In the first phase, the *global goal decomposition*, each agent creates an individual goal, i.e., a subset of the propositions from the global goal that it can reach. Proposition $p \in G$ is in the individual goal G_{ag_i} of an agent ag_i if exists an action $a \in A_{ag_i}$ such that $p \in add(a)$. If any proposition from the global goal cannot be assigned to any agent, the problem has no solution.

In the next two phases, the *expansion* and the *planning graph merging*, every agent builds an individual planning graph via GraphPlan algorithm (Blum and Furst 1997). Firstly, every agent builds a new layer in their planning graphs. Afterward, relevant actions from the new layer are shared among the agents. The shared actions are included into their respective planning graphs. An action $a_{ag_i} \in A_{ag_i}$ is *relevant* to an agent ag_j in two cases: (i) $pre(a_{ag_i})$ contains a proposition that another action $a_{ag_j} \in A_{ag_j}$ uses in $del(a_{ag_j})$ or $add(a_{ag_j})$, then the action a_{ag_i} *promotes* the action a_{ag_j} or (ii) $del(a_{ag_i})$ contains a proposition that another action $a_{ag_j} \in A_{ag_j}$ uses in $pre(a_{ag_j})$ or $add(a_{ag_j})$, then the action a_{ag_i} *threats* the action a_{ag_j}. The algorithm alternates between the *expansions*, where all the agents build new layers in their planning graphs and *planning graph merging*, where all the agents share their actions until all agents reach their individual goals in their planning graphs or the fixed point is reached.

In the *individual plan extraction* phase, each agent extracts plan(s) from its planning graph. In the centralized GraphPlan algorithm, this is done by compilation of the problem into a CSP and solved by a CSP solver. Each result from the solver is then one resulting plan as Kambhampati showed in (Kambhampati 2000).

In the last two *coordination* phases, before an agent ag_i generates an individual plan, it includes *requirement* and *commitment constraints*—induced by the other agents' plans—into its individual plan extraction CSP problem. The *requirement constraints* are couples (a, l) which describe an action a has to be performed in l-th layer. As $req(\pi_{ag_i})$, we will denote a set of requirement constraints induced by the current partial plan π_{ag_i} of the agent ag_i. A partial plan π_{ag_i} in this phase contains finished parts from agents $1, \ldots, i$ and future actions required by the agent ag_i (and possibly from previous agents) for the following agents $i+1, \ldots, k$. The *commitment constraints* are again couples (a, l) denoting that no action b, which is in *mutex* with action a, can be performed in l-th layer. Let $com(\pi_{ag_i})$ be a set of the commitment constraints induced by the current partial plan π_{ag_i}.

A couple $c = (com(\pi_{ag_i}), req(\pi_{ag_i}))$, besides representing the partial plan π_{ag_i} in form of the action commitments and requirements, describes which agents have already contributed to the plan π_{ag_i} with their individual plans (the performers of the actions in $com(\pi_{ag_i})$) and which agents have not (the performers of the actions in $req(\pi_{ag_i})$ but not in

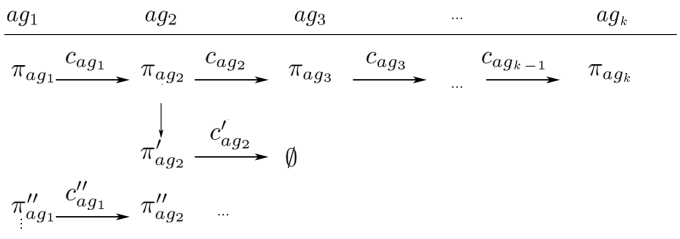

Figure 1: The DPGM plan search process.

$com(\pi_{ag_i})$). Such common partial plan in form of requirement couple c is passed from one agent to another. Each agent extends it with its individual plan and possibly new requirements for the following agents. After c passes all the agents, it contains a global plan consisting of the individual plans of all the agents.

Assume the agents are ordered $ag_1, ag_2, \ldots, ag_k$, as illustrated in Figure 1. The phase starts with an agent ag_1 which generates its first plan π_{ag_1}—lower index indicates the owner of the individual plan. The agent computes the commitment and requirement constraints, denoted as $c_{ag_1} = (com(\pi_{ag_1}), req(\pi_{ag_1}))$ and sends them to the next agent ag_2. ag_2 includes constraints c_{ag_1} into its CSP problem as additional constraints and generates a plan π_{ag_2} which is compatible with the plan π_{ag_1}. Next, ag_2 derives new constraints $c_{ag_2} = (com(\pi_{ag_1}) \cup com(\pi_{ag_2}), (req(\pi_{ag_1}) \cup req(\pi_{ag_2}))$, and passes them to the agent ag_3, and so on. In Figure 1, the search would actually end with generating ag_k's plan π_{ag_k} which illustrates the ideal way of the solution, since no agent had to generate a variance of its plan more than once to fulfill all the requirements from previous agents. If agent ag_3 could not generate plan π_{ag_3}, algorithm would backtrack to agent ag_2, who would generate an alternative plan π'_{ag_2}, as shown in the figure. If all agents ran out of plans, algorithm return to *expansion* and *planning graph merging*, and whole search repeats again.

Implementation

In (Pellier 2010), the author introduces theoretically the algorithm, yet no experiments were carried out to verify its efficiency. Besides the implementation of the algorithm in Java programming language, we included some implementation improvements to speed up the algorithm.

Constraint cache To reduce the search tree, as illustrated in Figure 1, each agent memorizes the constraints it passed to the next agents. If the constraints caused one of the following agents is not able to generate any plan, the requesting agent do not require such constraints for the next agents any more.

For an instance depicted in Figure 1, at point when agent ag_2 generates plan π_{ag_2} together with its constraints c_{ag_2}, he memorizes the constraints c_{ag_2} before passing them to ag_3. Let us assume that the constraints caused ag_3 to be unable to generate any individual plan depicted as $\emptyset$. The agent ag_2 memorizes this information and introduces new constraint into his CSP assignment that will eliminate generating of plans that restricts the agent ag_3 in future as c_{ag_2} did.

For an instance, $c_{ag_2} = (\{\}, \{(a_{ag_3}, 1), (a_{ag_4}, 2)\})$, where $(a_{ag_3}, 1)$ restricts ag_3 to perform action a_{ag_3} in first layer and $(a_{ag_4}, 2)$ restricts ag_4 to perform action a_{ag_4} in second layer. Since ag_3 could not find any plan that satisfies restriction $(a_{ag_3}, 1)$, agent ag_2 should not generate such plans. Thus, ag_2 introduces new constraint into its CSP assignment that will eliminate plans having action a_{ag_3} in first layer. This will cause agent ag_2 to eliminate generating of the plans that agent ag_3 cannot satisfy.

Ordering of agents Ordering of the agents turned out to be crucial for the DPGM algorithm to work efficiently. It is necessary to separate agents that have an individual goal from those that have no goal, but whose cooperation might be required somewhere during the plan. Let AG_g be a set of all agents having an individual goal and let AG_s be a set of all agents without their own goals. As the plan search progresses, the actual agents' ordering dynamically changes. The ordering starts with a random agent ag_i from the set AG_g (the set cannot be empty at this point; if it was, we have no goal, therefore the problem would be unsolvable). Afterward, a plan π_{ag_i} is generated together with constraints c_{ag_i}. Next, agent ag_j is selected by looking which cooperation is required in c_{ag_i}. If there are more such agents, they are prioritized from AG_g over the agents from AG_s. After generating new π_{ag_j} and c_{ag_j}, the process is repeated. If c_{ag_j} has no requirements on the other agents at any point, although AG_g or AG_s are not empty yet, it means that there exists a goal reaching plan without cooperation of the agents remaining in AG_g and AG_s sets. Notice that this may happen even for AG_g, although we said that these are the agents with goals, if a goal proposition can be reached by more than one agent and an agent outside AG_g has reached it.

Removing unnecessary actions Since we used pure PDDL parser to read the domain and the problem, we could end up with useless actions. For example, in the simplest LOGISTICS problem these actions are following: (drive car1 place1 place1) or (fly plane1 airport1 airport1). These actions do not change the state in any way. Preconditions of an action (drive car1 place1 place1) are (at car1 place1), a positive effect is (at car1 place1) and a negative effect is (at car1 place1). A result of this actions is that the car will remain in the same place (so does the action (fly plane1 airport1 airport1) with the plane). These actions can be generalized as $a = \langle pre(a), add(a), del(a) \rangle$, where $add(a) = del(a)$. Moreover, actions (as STRIPS defines them) have a requirement $add(a) \cap del(a) = \emptyset$, but a pure PDDL parser cannot hold this requirement while parsing. Hence, these actions had to be removed by the algorithm.

Experiments

The experiments were carried out on five different domains, where three originated from the International Planning Competition benchmarks adapted for the multiagent planning: ROVERS, LOGISTICS, and SATELLITES. The additional two are: LINEAR LOGISTICS (one package has to be transported step-wise by all agents in a chain) and DECONFLICTION

domain-agents	Minion	Minion srf	Choco	comm.
rover-a2	8.9s	3.8s	11.7s	69kB
rover-a3	6.6s	20.3s	17.4s	234kB
log-a4	0.9s	0.3s	1.2s	34kB
log-a6	0.7s	0.6s	1.3s	136kB
log-lin-a6	0.5s	0.5s	0.3s	167kB
log-lin-a8	0.7s	0.7s	0.5s	417kB
log-lin-a10	0.9s	0.9s	0.7s	849kB
log-lin-a15	1.6s	1.6s	1.8s	2.9MB
deconf-a2	–	1.3s	–	18kB
deconf-a3	0.2s	0.2s	0.1s	13kB
satellite-a6	1.6s	1.5s	4.6s	266kB
satellite-a8	5.0s	4.3s	24.8s	793kB
satellite-a10	14.3s	12.7s	101s	1.8MB

Table 1: Comparison of CSP solvers used in DPGM. The dash – means that the time or memory limit was exceeded.

(robots on a grid are tasked to switch its positions with opposite ones, not colliding with each other). Each domain was tested on several problems with various numbers of agents. All experiments were run on 8-core processor at 3.6GHz with 2.5GB limit on memory and 10 minutes time limit. We used time and communicated bytes as metrics for the comparison of the algorithms.

Comparison of used CSP solvers in DPGM

As DPGM algorithm uses CSP solver for the local plan extraction, we experimented which solver would serve the best. Two CSP solvers were tested: Choco CSP Solver[1] and Minion CSP Solver[2]. Choco solver was tested with its basic setting, while Minion was tested with and without smallest-ratio-first (*srf*) variable order. Table 1 shows the times DPGM took to solve the problems using certain CSP solvers and settings. Although Minion solver showed to be sometimes unstable and did not return any result over the longer period of time, DPGM was faster with it than with the Choco solver. Another Minion's disadvantage is that if a problem is unsolvable, it is inefficiently detected, since it has to go through all the possibilities in the search space. Last column (comm.) in Table 1 shows the communicated bytes among the agents. As the CSP solver is used only for local extraction of a plan, the numbers are the same for all the solvers. In the deconf-a3 problem, even the number of agents is higher than in deconf-a2, the results are better. This is caused by the particular problem instance, where the agents in the a2 case has to pass by each other, and therefore the solution is found not before 4th layer, however in a3 the agents only rotates and therefore the solution is found in 2nd layer.

Comparison of the multiagent planners

In the final experiment, the DPGM algorithm was compared to other two cited algorithms. Table 2 shows the results. As the *srf* setting of Minion showed the best results—especially because of its ability to solve most of the presented problems—we chose it for comparison with the other algorithms.

domain-agents	DPGM	DisCSP+Pl.	MA-A*
rover-a2	3.8s/69kB	1.4s/0.8kB	22.4s/52kB
rover-a3	20.3s/234kB	7.9s/1.7kB	230s/2.5MB
rover-a4	–	62.3s/3.1kB	–
log-a4	0.3s/34kB	0.6/15kB	0.8s/77kB
log-a6	0.6s/136kB	38.5/7.1MB	2.0s/320kB
log-lin-a6	0.5s/167kB	–	1.7s/87kB
log-lin-a8	0.7s/417kB	–	4.7s/254kB
log-lin-a10	0.9s/849kB	–	15.4s/589kB
log-lin-a15	1.6s/2.9MB	–	217s/4.2MB
deconf-a2	1.3s/18kB	N/A	0.9s/15.3kB
deconf-a3	0.2s/13kB	N/A	1.2s/187kB
deconf-a4	–	N/A	3.8s/2.1MB
satellite-a6	1.5s/266kB	4.4/6.5kB	7.4s/270kB
satellite-a8	4.3s/793kB	–	37.5s/964kB
satellite-a10	12.7s/1.8MB	–	189s/2.5MB

Table 2: Results for DPGM, DisCSP+Planning and MA-A* with set-additive heuristic. The dash – means the time or memory limit was exceeded. N/A means the planner did not return a sound plan.

The results show the DPGM to be efficient in decoupled domains which are rather combinatorially easy (LOGISTICS, LINEAR LOGISTICS and SATELLITES). The DisCSP+Planning is efficient in problems which are combinatorially hard from perspective of individual planning (ROVERS), as the internally used planner is highly efficient FastForward. The used implementation of MA-A* with set-additive heuristics was most effective in highly coupled domains (DECONFLICTION).

Final remarks

DPGM showed its strength based on efficient factorization of the problems. However problems as DECONFLICTION, which are coupled and require high combinatorial search, DPGM solves rather inefficiently if at all. An issue that hindered the algorithm was the order how CSP generated the plans. For instance, the first plan that the agent ag_1 generated in the 4th layer of the deconf-a4 problem, consisted of its own actions, leading him to the goal. Additionally, agent generated requirements for the agent ag_2 to prevent future collisions. However, the requirements were unreasonable: instead of requiring one action that would suffice for agent ag_1 to avoid the collision with agent ag_2, he built a whole plan for the agent ag_2. And since ag_1 did not know the ag_2's goal, the plan was usually invalid. We tried to avoid this, by stating to minimize requirements put on other agents in the CSP solver. This approach helped to lower the number of the constraints; however, the solution of such CSP became combinatorially more complex and therefore did not bring much of improvement in the efficiency. Deeper study of these phenomenons remains for future work.

Acknowledgments This work was supported by Czech Science Foundation (GACR) under grant no. 13-22125S.

[1] http://www.emn.fr/z-info/choco-solver/

[2] http://minion.sourceforge.net/

References

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial intelligence* 90(1):281–300.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of ICAPS'08*, 28–35. AAAI.

IPC. 2004. International planning competition (IPC) – Results. http://www.tzi.de/ edelkamp/ipc-4/results.html.

Kambhampati, S. 2000. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in Graphplan. *J. Artif. Intell. Res. (JAIR)* 12:1–34.

Nissim, R., and Brafman, R. 2012a. Multi-agent A* for parallel and distributed systems. In *Proceedings of 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1265–1266.

Nissim, R., and Brafman, R. 2012b. Multi-agent A* for parallel and distributed systems. 43–51.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS'10*, 1323–1330. IFAAMAS.

Pellier, D. 2010. Distributed planning through graph merging. In Filipe, J.; Fred, A. L. N.; and Sharp, B., eds., *Proceedings of ICAART'10*, volume 2, 128–134. IFAAMAS.

A Theory of Intra-Agent Replanning

Kartik Talamadupula[†] and **David E. Smith**[§] and **William Cushing**[†] and **Subbarao Kambhampati**[†]

[†]**Dept. of Computer Science and Eng.**
Arizona State University
Tempe, AZ 85287
{krt,rao} @ asu.edu, william.cushing @ gmail.com

[§]**NASA Ames Research Center**
Moffet Field
CA 94035
david.smith @ nasa.gov

Abstract

When autonomous agents execute in the real world, the world state as well as the objectives may change from the agent's original conception of those things. In such cases, the agent's planning process must modify the existing plan to make it amenable to the new conditions, and to resume execution. The need for inter-agent replanning, in terms of commitments to other agents, is understood in the multi-agent systems community. Such inter-agent replanning also motivates an intra-agent replanning problem for each individual agent. However, the single-agent planning community has mostly limited its view of replanning to reducing the computational effort involved, by minimally perturbing the current plan structure to replan. This is not very appropriate as a general model for intra-agent replanning, which may consist of various techniques that are employed according to the scenario at hand. In this paper, we present a general replanning problem that is built on various types of replanning constraints. We show that these constraints can model different types of replanning, including the similarity-based approaches used in the past and sensitivity to commitments made to other agents. Finally, we show that partial satisfaction planning provides a good substrate for modeling this general replanning problem.

1 Introduction

Many tasks require handling dynamic objectives and environments. Such tasks are characterized by the presence of highly complex, incomplete, and sometimes inaccurate specifications of the world state, the problem objectives and even the model of the domain dynamics. These discrepancies may come up due to factors like plan executives or other agents that are executing their own plans in the world. Due to this divergence, even the most sophisticated planning algorithms will eventually fail unless they offer some kind of support for replanning. These dynamic scenarios are non-trivial to handle even when planning for a single agent, but the introduction of multiple agents – automated or otherwise – introduces further complications. All these agents necessarily operate in the same world, and the decisions made and actions taken by an agent may change that world for all the other agents as well. Moreover, the various agents' published plans may introduce commitments between them, due to shared resources, goals or circumstances. The need for *inter-agent replanning* in terms of these commitments is understood in the multi-agent systems (MAS) community (c.f. Section 2). However, these inter-agent commitments may evolve as the world itself changes, and may in turn affect a single agent's internal planning process.

Given the importance of replanning in dealing with all these issues, one might assume that the single-agent planning community has studied the issues involved in depth. This is particularly important given the difference between agency and execution, and the real-world effectors of those faculties: a single agent need not necessarily limit itself to planning just for itself, but can generate plans that are carried out by multiple executors in the world. Unfortunately, most previous work in the single-agent planning community has looked upon replanning as a *technique* whose goal is to reduce the computational effort required in coming up with a new plan, given changes to the world. The focus in such work is to use the technique of minimally perturbing the current plan structure as a solution to the replanning problem. However, neither reducing replanning computation nor focusing on minimal perturbation are appropriate techniques for intra-agent replanning in the context of multi-agent scenarios.

In this work, we argue for a better, more general, model of the replanning problem. This model considers the central components of a planning problem – the initial state, the set of goals to be achieved, and the plan that does that, along with *constraints* imposed by the execution of that plan in the world – in creating the new replan. These replanning constraints take the form of commitments for an agent, either to an earlier plan and its constituent actions, or to other agents in its world. We will show that this general commitment sensitive planning architecture subsumes past replanning techniques that are only interested in minimal perturbation – the "commitment" in such cases is to the structure of the previously executing plan. We will also show that partial satisfaction planning (PSP) techniques provide a good substrate for this general model of replanning.

In the next section, we discuss some prior and related work from the multi-agent systems (MAS) and single-agent planning communities in order to motivate our work. We then present our formulation of the replanning problem in terms of the problem instance (composed of the initial state and the goals), the plan to solve that particular instance, and the dependencies or constraints that are introduced into the world by that plan, and three models associated with the handling of these *replanning constraints* that are defined in that formulation. Subsequently, we delve deeper into the compo-

sition of those constraints, and discuss the various solution techniques that can be used to satisfy these constraints while synthesizing a new replan. We then describe our experimental setup and present our results.

2 Related Work

Replanning has been an early and integral part of automated planning and problem solving work in AI. The STRIPS robot problem-solving system (Fikes, Hart, and Nilsson 1972), one of the earliest applications of planning and AI, used an execution monitoring system known as PLANEX to recognize plan failures in the world, and replan if direct re-execution was not an option. The replanning mechanism worked by sending the change in state back to the STRIPS system, which returned a sequence of actions that brought the state back to one from which the execution of the original plan could be resumed. This relatively simple procedure encoded the idea that would come to dominate replanning work within the planning community for the next few decades – the notion of *commitment to a plan*. The principle underlying the concept of minimally changing an existing plan is christened plan *stability* by Fox et al. (Fox et al. 2006). In that work, two approaches – replanning from scratch, and repairing the existing plan – and their respective impacts on plan stability are considered. Stability itself is defined as the measure of the difference a process induces between an original plan and a new plan, and is closely related to the idea of *minimal perturbation planning* (Kambhampati 1990) used in past replanning and plan re-use (Nebel and Koehler 1995) work. Fox et al. argue that plan stability as a property is desirable both from the standpoint of measurable quantities like plan generation time and plan quality, as well as intangibles like the cognitive load on human observers of planned activity and the strain on the plan executive.

Other work on replanning has taken a strong stand either for or against the idea of plan repair. Van Der Krogt et al. (Van Der Krogt and De Weerdt 2005) fall firmly into the former category, as they outline a way to extend state-of-the-art planning techniques to accommodate plan repair. For the purposes of this paper, it suffices to note that this work looks at the replanning problem as one of commitment to and maintenance of a broken plan. This work has a strong parallel (and precursor) in planning for autonomous space exploration vehicles, a proven real world application of plan-ning technology. The Casper system (Knight et al. 2001), which was designed to autonomously control a spacecraft and its activities, was designed as a system with a high level of responsiveness, enabled through a technique called *itera-tive repair* – an approach that fixes flaws in an existing plan repeatedly until an acceptable plan is found. At the other end of the spectrum, Fritz et al. (Fritz and McIlraith 2007) deal with changes to the state of the world by replanning from scratch. Their approach provides execution monitor-ing capabilities by formalizing notions of plan validity and optimality using the situation calculus; prior to execution, each step in the (optimal) plan is annotated with conditions that are sufficient for the plan's optimality to hold. When a discrepancy or unexpected change occurs during execution, these conditions are re-evaluated in order to determine the optimality of the executing plan. When one of the conditions is violated, the proposed solution is to come up with a completely new plan that satisfies the optimality (or validity) conditions.

In contrast, the MAS community has looked at replanning issues more in terms of multiple agents and the conflicts that can arise between these agents when they are execut-ing in the same dynamic world. Wagner et al. (Wagner et al. 1999) proposed the twin ideas of *inter-agent* and *intra-agent* conflict resolution. In the former, agents exchange commitments between each other in order to do team work. These commitments in turn may affect an agent's local controller, and the feasibility of the agent's individual plan – this brings up the process of intra-agent conflict resolution. Inter-agent commitments have been variously formalized in dif-ferent work in the MAS community (Komenda et al. 2008; Bartold and Durfee 2003; Wooldridge 2000), but the fo-cus has always been on the interactions between the various agents, and how changes to the world affect the declared commitments. The impact that these changes have *within* an agent's internal planning process has not received significant study. The closest work in the multi-agent planning community to ours is by (Komenda, Nov´ak, and P˘echou˘cek 2012), where the multi-agent plan repair problem is intro-duced and reduced to the multi-agent planning problem; and (Meneguzzi, Telang, and Singh 2013), where a first-order representation and reasoning technique for modeling commitments is introduced.

In this work, we propose to bring these two approaches from two different communities – single-agent planning, and multi-agent systems – together in a unified theory of agent replanning. Our central argument is that it should be the single-agent planning community's brief to heed the changes to the world state and inter-agent commitments, and to generate a new (single-agent) plan that remains consistent with the larger multi-agent commitments in the world. The first step in this endeavor is to re-define the replanning problem such that both single and multi-agent commitments can be represented under a unified framework.

3 The Replanning Problem

We posit that replanning should be viewed not as a technique, but as a *problem* in its own right – one that is distinct from the classical planning problem. Formally, this idea can be stated as follows. Consider a plan Π_P that is synthesized in order to solve the planning problem $P = \langle I, G\rangle$, where I is the initial state and G, the goal description. The world then changes such that we now have to solve the problem $P' = \langle I', G'\rangle$, where I' represents the changed state of the world, and G' a changed set of goals (possibly different from G). We then define the *replanning problem* as one of finding a new plan Π'_P that solves the problem P' subject to a set of constraints ψ^{Π_P}. This model is depicted in Figure 1. The composition of the constraint set ψ^{Π_P}, and the way it is handled, can be described in terms of specific *models* of this newly formulated replanning problem. Here, we present three such models based on the manner in which the set ψ^{Π_P} is populated.

1. M_1 | **Replanning as Restart**: This model treats replanning as 'planning from restart' – i.e., given changes in the world $P = \langle I, G\rangle \rightarrow \ P' = \langle I', G'\rangle$, the old plan Π_P is

completely abandoned in favor of a new plan Π'_P which solves P'. Thus the previous plan induces no constraints that must be respected, meaning that the set ψ^{Π_P} is empty.

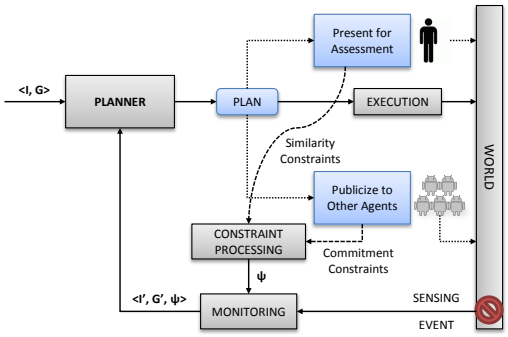

Figure 1: A model of replanning

2. M_2 | **Replanning to Reduce Computation**: When the state of the world forces a change from a plan Π_P to a new one Π'_P, in the extreme case, Π'_P may bear no relation to Π_P. However, it is most desirable that the cost of comparing the differences between the two plans with respect to execution in the world be reduced as far as possible. The problem of minimizing this cost can be re-cast as one of minimizing the differences between the two plans Π'_P and Π_P using *syntactic constraints* on the form of the new plan. These syntactic constraints are added to the set ψ^{Π_P}.
3. M_3 | **Replanning for Multi-agent Scenarios**: In many real world scenarios, there are multiple agents $A_1 \ldots A_n$ that share an environment and hence a world state.[1] The individual plans of these agents, $\Pi_1 \ldots \Pi_n$ respectively, affect the common world state that the agents share and must plan in. This leads to the formation of dependencies, or *commitments*, by other agents on an agent's plan. These commitments can be seen as special types of constraints that are induced by an executing plan, and that must be obeyed when creating a new plan as a result of replanning. The aggregation of these commitments forms the set ψ^{Π_P} for this model.

In the following section, we explore the composition of the constraint set ψ^{Π} (for any given plan Π) in more detail. First, however, we consider a real world application scenario and the application of the three replanning models described above to it, in order to illustrate that these models are broad enough to capture the various kinds of replanning techniques.

3.1 Example: Planetary Rovers

Planning for planetary rovers is a scenario that serves as a great overarching application domain for describing the motivations behind the various models of replanning that we propose in this work. Automating the planning process is central to this application for three reasons: (1) the complex checks and procedures that are part of large-scale or critical applications can often only be fully and correctly satisfied by automation; (2) there are limited communication opportunities between the rover and and the control station; and (3) the distances involved rule out immediate tele-operation, since there is a considerable communication lag between a rover operating on the surface of a distant planet and the control center.

1. M_1: This model is frequently used by planning algorithms that create path and motion plans for the rover's operation. Often, changes to the environment (e.g. the detection of an obstacle such as a rock ahead) will render the currently executing plan useless; in cases where the system needs to react immediately and produce a new plan, creating a completely new plan works better than trying to salvage some version of an existing plan.
2. M_2: In the case of planetary rovers, both computational and cognitive costs are present when it comes to comparing Π and Π'. Changes to an executing plan Π must pass muster with human mission controllers on Earth as well as mechanical and electrical checks on-board the rover itself. It is thus imperative that the replanning model is aware of the twin objectives of minimizing cognitive load on the mission controllers as well as minimizing the computation required on board the rover when vetting a new plan Π' that replaces Π. In this case, the set ψ^{Π_P} will contain constraints that try to minimize the effort needed to reconcile Π' with Π, and the metric used in the reconciliation determines the contents of ψ^{Π_P}. These can be seen as a *syntactic* version of plan stability constraints, as against the *semantic* stability constraints (based on commitments) that we go on to propose.
3. M_3: In a typical scenario, it is also possible that there may be multiple rovers working in the same environment, with knowledge (complete or partial) of the other rovers' plans. This knowledge in turn leads to dependencies which must be preserved when the plans of one (or more) of the rovers change – for example, rover *Spirit* might depend on rover *Opportunity* to transmit (back to base) the results of a scientific experiment that it plans to complete. If *Opportunity* now wishes to modify its current plan Π_O, it must pay heed to the commitment to communicate with *Spirit* – and pass on the data that results from that communication – when devising its new plan Π'_O.

4 Replanning Constraints

As outlined in the previous section, the replanning problem can be decomposed into various *models* that are defined by the constraints that must be respected while transitioning from the old plan Π to the new plan Π'. In this section, we define those constraints, and explore the composition of the set ψ for each of the models defined previously.

4.1 Replanning as Restart

By the definition of this model, the old plan Π_P is completely abandoned in favor of a new one. There are no constraints induced by the previous plan that must be respected, and thus the set ψ^{Π_P} is empty. Instead, what we have is a new problem instance P' whose composition is completely independent of the set ψ^{Π_P}.

[1] Note that this is the case regardless of whether the planner models these agents explicitly or chooses to implicitly model them in the form of a dynamic world.

4.2 Replanning to Reduce Computation

It is often desirable that the replan for the new problem instance P' resemble the previous plan Π_P in order to reduce the computational effort associated with verifying that it still meets the objectives, and to ensure that it can be carried out in the world. We name the effort expended in this endeavor as the *reverification complexity* associated with a pair of plans Π_P and Π'_P, and informally define it as the amount of effort that an agent has to expend on comparing the differences between an old plan Π_P and a new candidate plan Π'_P with respect to execution in the world.

This effort can either be computational, as is the case with automated agents like rovers and robots; or cognitive, when the executor of the plans is a human. Real world examples where reverification complexity is of utmost importance abound, including machine-shop or factory-floor planning; planning for assistive robots and teaming; and planetary rovers (see Section 3.1). Past work on replanning has addressed this problem via the idea of *plan stability* (Fox et al. 2006). The general idea behind this approach is to preserve the stability of the replan Π'_P by minimizing some notion of difference with the original plan Π_P. In the following, we examine two such ways of measuring the difference between pairs of plans, and how these can contribute constraints to the set ψ^{Π_P} that will minimize reverification complexity.

Action Similarity Plans are defined, first and foremost, as sequences of actions that achieve specified objectives. The most obvious way to compute the difference between a given pair of plans then is to compare the actions that make up those plans. (Fox et al. 2006) defines a way of doing this -given an original plan Π and a new plan Π', they define the difference between those plans as the number of actions that appear in Π and not in Π' plus the number of actions that appear in Π' and not in Π. If the plans Π and Π' are seen as sets comprised of actions, then this is essentially the symmetric difference of those sets, and we have the following constraint:[2] $\min\ |\Pi \triangle \Pi'|$.

This method of gauging the similarity between a pair of plans suffers from some obvious pitfalls; a very simple one is that it does not take the ordering of actions in the plans into account at all. Consider the simple plans $\Pi : \langle a_1, a_2 \rangle$ and $\Pi' : \langle a_2, a_1 \rangle$; the difference between these two plans is $\Pi \triangle \Pi' = \emptyset$. However, from a replanning perspective, it seems obvious that these two plans are really quite different, and may lead to different results if the actions are not commutative. In order to account for such cases, we would need to consider the ordering of actions within a plan, and more generally, the causal structure of a plan.

Causal Link Similarity The next step in computing plan similarity is to look not just at the actions that constitute the plans under comparison, but to take the causal structure of those plans into account as well. Work on partial order planning (POP) has embedded a formal notion of causal links quite strongly within the planning literature. Past partial order planning systems (Penberthy and Weld 1992; Joslin and Pollack 1995) have looked at the idea of different serializations of the same partial order plan. Given plans Π and Π', and CL(Π) and CL(Π') the sets of causal links on those plans respectively, a simple constraint to enforce causal similarity would be: $\min\ |\text{CL}(\Pi) \triangle \text{CL}(\Pi')|$. Note that this number may be non-zero even though the two plans are completely similar in terms of action similarity; i.e. $(\Pi \triangle \Pi') = \emptyset$. This analysis need not be restricted to causal links alone, and can be extended to arbitrary ordering constraints of a non-causal nature too, as long as they can be extracted from the plans under consideration.

4.3 Replanning for Multi-agent Scenarios

In a multiperson situation, one man's goals may be another man's constraints. – Herb Simon (Simon 1964)

In an ideal world, a given planning agent would be the sole center of plan synthesis as well as execution, and replanning would be necessitated only by those changes to the world state that the agent cannot foresee. However, in the real world, there exist multiple such agents, each with their own disparate objectives but all bound together by the world that they share. A plan Π_P that is made by a particular agent affects the state of the world and hence the conditions under which the other agents must plan – this is true for every agent. In addition, the publication of a plan Π_P^A by an agent A leads to other agents predicating the success of their own plans on parts of Π_P^A, and complex dependencies are developed as a result. Full multi-agent planning can resolve the issues that arise out of changing plans in such cases, but it is far from a scalable solution for real world domains currently. Instead, this multi-agent space filled with dependencies can be projected down into a single-agent space with the help of *commitments* as defined by (Cushing and Kambhampati 2005). These commitments are related to an agent's current plan Π, and can describe different requirements that come about:

1. when Π changes the world state that other agents have to plan with
2. when the agent decides to execute Π, and other agents predicate their own plans on certain aspects of it
3. due to cost or time based restrictions imposed on the agent
4. due to the agent having paid an up-front setup cost to enable the plan Π

A simple travel example serves to demonstrate these different types of commitments. Consider an agent A_1 who must travel from Phoenix (PHX) to Los Angeles (LAX). A travel plan Π that is made for agent A_1 contains actions that take it from PHX to LAX with a long stopover at Las Vegas (LAS). A_1 is friends with agent A_2, who lives in LAS, and thus publicizes the plan of passing through LAS. A_2 then makes its own plan to meet A_1 – this depends on A_1's presence at the airport in LAS. If there are changes to the world (for e.g., a lower airfare becomes available), there are several commitments that a planner must respect while creating a new plan Π' for A_1. First, there are commitments to other agents – in this case, the meeting with A_2 in LAS. There are also setup and reservation costs associated with the previous plan; for example, A_1 may have paid a non-refundable airfare as part of Π. Finally, there may be a deadline on getting

[2] Given this constraint, the similarity and difference of a pair of plans are inverses, and hence the name 'Action Similarity'.

to LAX, and any new plan has to respect that commitment as well.

At first blush, it seems that the same kinds of constraints that seek to minimize reverification complexity between plans Π and Π' (minimizing action and causal link difference between plans) will also serve to preserve and keep the most commitments in the world. Indeed, in extreme cases, it might even be the case that keeping the structures of Π and Π' as similar as possible helps keep the maximum number of commitments made due to Π. However, this is certainly not the most natural way of keeping commitments. In particular, this method fails when there is any significant deviation in structure from Π to Π'; unfortunately, most unexpected changes in real world scenarios are of a nature that precludes retaining significant portions of the previous plan. For example, in the (continuing) air travel example from above, agent A_1 has a commitment not to the plan Π itself, but rather to the event of meeting A_2. This suggests modeling commitments natively as state conditions (as opposed to casting them as extraneous constraints on plan structure) as goals that must be either achieved or preserved by a plan as a possible replanning constraint. We elaborate on this in Section 5.3.

5 Solution Techniques

So far, we have looked at three different ways in which the replanning problem can be represented, and delineated the differences between these models via the constraints that need to be considered when making new plans in a changed world. We now turn our attention to the planning *techniques* that are (or can be) used to solve these variants.

5.1 T1: Classical Planning

For the *replanning as restart* model, the problem is defined as one of going from a plan Π_P that solves the problem instance $P = \langle I, G \rangle$ to the best new plan Π'_P that is valid for the new problem instance $P' = \langle I', G' \rangle$. I' is the state of the world at which Π_P stops executing to account for the change that triggered replanning; that is, replanning commences from the current state of the world. G' is the same as G unless new goals are explicitly added as part of the changes to the world. The replanning constraint set ψ^{Π_P} is empty, since replanning is being performed from scratch. This new instance is then given to a standard classical planner to solve, and the resulting plan is designated Π'_P.

5.2 T2: Specialized Replanning Techniques

When it comes to *replanning to reduce computation* and associated constraints, techniques that implement solutions that conform to these constraints must necessarily be able to compile them into the planning process in some way. This can be achieved by implementing plan stability metrics – either explicitly by comparing each synthesized plan candidate with the existing plan Π_P, or implicitly by embedding these metrics within the search process. One way of doing the latter is to use a planner such as LPG (Gerevini, Saetti, and Serina 2003), which uses local search methods, and to structure the evaluation function such that more syntactic similarity between two plans – similar actions, for example – is preferred. Such an approach is used by (Srivastava et al. 2007) in the generation of a set of diverse plans where the constituent plans differ from each other by a defined metric; for replanning where search re-use is of importance, the objective can instead be to produce *minimally different* plans within that set. An earlier version of this approach can be seen in the Casper system's iterative repair approach (Knight et al. 2001).

5.3 T3: Partial Satisfaction Planning

We now turn our attention to replanning techniques that can be used when the dependencies or commitments towards other agents due to an agent A's original plan Π (solving the problem instance P) must be maintained. The constraint set $\psi^{\Pi^A_P}$ now contains all those commitments to other agents that were made by the plan Π. We follow Cushing et al. (Cushing and Kambhampati 2005) in modeling commitments as *soft* constraints that an agent is not mandated to necessarily achieve for plan success. More generally, commitments – as reservations, prior dependencies or deadlines – can be modeled as soft trajectory constraints on any new plan Π' that is synthesized. Modeling commitments as soft constraints (instead of hard) is essential because not all commitments are equal. A replan Π' may be valid even if it flouts a given commitment; indeed, it may be the *only* possible replan given the changed state of the world. Soft goals allow for the specification of different priorities for different commitments by allowing for the association of a *reward* for achieving a given goal, and a *penalty* for non-achievement. Both of these values are optional, and a commitment may either be seen as an opportunity (accompanied by a reward) or as a liabiity (when assigned a penalty). The quality of a replan Π' – in terms of the number of commitment constraints that it satisfies – can then be discussed in terms of the *net-benefit*, which is a purely arithmetic value.

An added advantage of modeling commitments as soft goals is that the constraints on plan structure discussed previously in Section 4.2 can be cast as commitments too. These constraints are commitments to the *structure* of the original plan Π, as against commitments to other agents or to other extraneous phenomena like deadlines etc. The advantage in doing this is that new plans and their adherence to commitments can be evaluated solely and completely in terms of the net-benefit of those plans; this makes the enforcement of the replanning constraints during the planning process more amenable to existing planning methods. We thus devise a natural way of combining two distinct quality issues in replanning: (1) how good a replan Π' is for solving the changed problem instance $\langle I', G' \rangle$; and (2) how much Π' respects and balances the given replanning constraints, which may be in service of completely different objectives like reducing the computation involved in verifying a new plan, or commitments to other agents in the world.

To obtain the new problem instance P' from the original problem P, we perform the following transformations: I' is, as before, the state of the world at which execution is stopped because of the changes that triggered replanning. G' consists of all outstanding goals in the set G as well as any other explicit changes to the goal-set; in addition, the constraints from the set $\psi^{\Pi^A_P}$ are added to G' as soft goals, using the compilations described below. The new problem

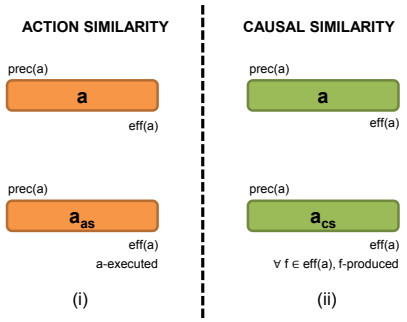

Figure 2: Compiling action and causal similarity to PSP by creating new effects, actions that house those effects, and soft goals on those effects.

instance is then given to a PSP planner to solve for the plan with the best net-benefit, which is then designated Π'^{A}_{P}.

The syntactic plan similarity constraints discussed at length in Section 4.2 can be cast as PSP constraints, in the form of soft goals. In the following, we describe a general compilation of the constraints in $\psi^{\Pi_P{}^A}$ to a partial satisfaction planning problem instance. We follow (van den Briel et al. 2004) in defining a PSP Net Benefit problem as a planning problem $P = (F, O, I, G_s)$ (where F is a finite set of fluents, O is a finite set of operators and $I \subseteq F$ is the initial state as defined earlier in our paper) such that each action $a \in O$ has a "cost" value $C_a \geq 0$ and, for each goal specification $g \in G$ there exists a "utility" value $U_g \geq 0$. Additionally, for every goal $g \in G$, a 'soft' goal g_s with reward r_g and penalty p_g is created; the set of all soft goals thus created is added to a new set G_s.

The intuition behind casting these constraints as goals is that a new plan (replan) must be constrained in some way towards being similar to the earlier plan. However, making these goals *hard* would over-constrain the problem – the change in the world from I to I' may have rendered some of the earlier actions (or causal links) impossible to preserve. Therefore the similarity constraints are instead cast as *soft* goals, with rewards or penalties for preserving or breaking (respectively) the commitment to similarity with the earlier plan. In order to support these goals, new fluents need to be added to the domain description that indicate the execution of an action, or achievement of a fluent respectively. Further, new copies of the existing actions in the domain must be added to house these effects. Making copies of the actions from the previous plan is necessary in order to allow these actions to have different costs from any new actions added to the plan.

Compiling Action Similarity to PSP The first step in the compilation is converting the action similarity constraints in $\psi^{\Pi^A_P}$ to soft goals to be added to G_s. Before this, we examine the structure of the constraint set $\psi^{\Pi^A_P}$; for every ground action $\bar{a}$ (with the names of the objects that parameterize it) in the old plan Π, the corresponding action similarity constraint is $\Psi_{\bar{a}} \in \psi^{\Pi^A_P}$, and that constraint stores the name of the action as well as the objects that parameterize it.

Next, a copy of the set of operators O is created and named O_{as}; similarly, a copy of F is created and named F_{as}. For each (lifted) action $a \in O_{as}$ that has an instance in the original plan Π, a new fluent named "a-executed" (along with all the parameters of a) is added to the fluent set F_{as}. Then, for each action $a \in O_{as}$, a new action a_{as} which is a copy of the action a that additionally also gives the predicate a-executed as an effect, is created. The process of going from the original action a to the new one a_{as} is depicted graphically in Figure 2(i). In the worst case, the number of actions in each O_{as} could be twice the number in O.

Finally, for each constraint $\Psi_{\bar{a}} \in \psi^{\Pi^A_P}$, a new soft goal $g_{\bar{a}}$ is created with corresponding reward and penalty values $r_{g_{\bar{a}}}$ and $p_{g_{\bar{a}}}$ respectively, and the predicate used in $g_{\bar{a}}$ is $\bar{a}$-executed (parameterized with the same objects that $\bar{a}$ contains) from O_{as}. All the $g_{\bar{a}}$ goals thus created are added to G_s. In order to obtain the new compiled replanning instance P' from P, the initial state I is replaced with the state at which execution was terminated, I'; the set of operators O is replaced with O_{as}; and the set of fluents F is replaced with F_{as}. The new instance $P' = (F_{as}, O_{as}, I', G_s)$ is given to a PSP planner to solve.

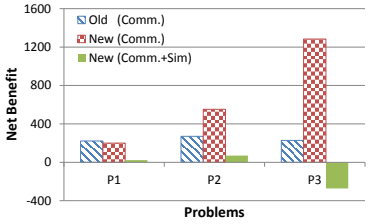

Figure 3: Net-benefit of plans for zenotravel problems, Experiment 1.

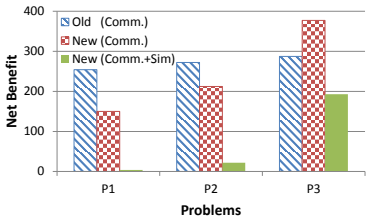

Figure 4: Net-benefit of plans for driverlog problems, Experiment 1.

Compiling Causal Similarity to PSP Causal similarity constraints can be compiled to PSP in a manner that is very similar to the above compilation. The difference that now needs to be considered is that the constraints are no longer on actions, but on the grounded fluents that comprise the causal links between the actions in a plan instead.

The first step is to augment the set of fluents; a copy of F is created and named F_{cs}. For every fluent $f \in F$, a new fluent named "f-produced" is added to F_{cs}, along with all the original parameters of f. A copy of the set of operators O is created and named O_{cs}. Then, for each action in $a \in O_{cs}$, a new action a_{cs} is added; a_{cs} is a copy of action a, with the additional effects that for every fluent f_a that is in the add effects of the original a, a_{cs} contains the effect f_a-produced – this process is shown in Figure 2(ii). Thus in the worst case, the number of effects of every action a_{cs} is twice the number of effects of the original action a, and the size of O_{cs} is twice that of O.

Finally, the causal constraints in $\psi^{\Pi^A_P}$ must be converted to soft goals that can be added to G_s. The constraints $\Psi \in$

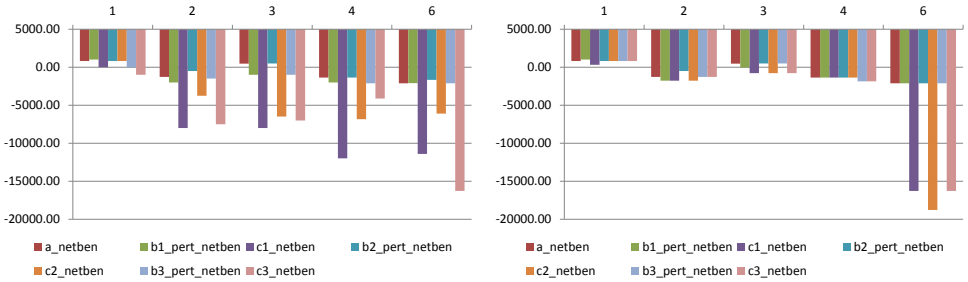

Figure 5: Net-benefit of plans for zenotravel problems, Experiment 2.

$\psi^{\Pi}_{P^A}$ are obtained by simulating the execution of Π from I using the operators in O. Each ground add-effect $\bar{f}_e$ of each ground action $\bar{a_\Pi}$ in Π is added as a new constraint $\Psi_{\bar{f}_e}$. Correspondingly, for each such new constraint added, a new soft goal $g_{\bar{f}_e}$ is created whose fluent corresponds to $\bar{f}_e$, with reward and penalty values $r_{g_{\bar{f}_e}}$ and $p_{g_{\bar{f}_e}}$ respectively.[3] All the goals thus created are added to G_s. The new planning instance to be provided to the PSP planner is thus given as $P' = (F_{cs}, O_{cs}, I', G_s)$, where I' is the state of the fluents when execution was previously suspended.

6 Empirical Study

Generally, work on specialized (single-agent) replanning techniques claims the support of experiments that exhibit either: (1) an advantage over from-scratch replanning in terms of speed or efficiency; or (2) greater plan stability when compared to other techniques. Unfortunately, our improved understanding of replanning as a general problem rather than as any single technique renders such an evaluation unsatisfactory. Since different kinds of replanning problems can be realized from different instantiations of the constraint set ψ, and these constraints can all be weighted as desired, one thing that we *should* evaluate is whether our single general model can model any problem (and technique) in the replanning spectrum. Given access to such a general model, it is also rather straightforward to set up problem instances that favor a specific technique over all other replanning techniques. Such results can be attributed either to the fact that the other techniques ignore the constraints that the first takes into account, or that they use surrogate constraints in order to mimic them. In the following, we describe the setup of three such experiments, and present preliminary results from the first two.

Setting Rewards & Penalties – The compilations outlined in Section 5.3, as well as the results that follow, are sensitive to the actual value of the rewards and the penalties that are assigned to the goals that the planner must achieve. We are currently in the process of conducting a theoretical as well as empirical analysis of the effect of these values on the plans that are produced, as well as the time taken to replan. For the experiments outlined below, we assign a reward of 500 units and a penalty of 1000 units to the regular, state space goals.

[3]Note that in the general case, we would need to consider *consumers* – i.e., actions that consume the causal link – apart from the producers of those links, in order to avoid over-constraining the new problem. However, we assume here that the original plan does not contain any superfluous actions.

Similarity goals are given a reward of 0 units, and a penalty of 1000 units (since they can be seen as commitments to the form of the previous plan).

6.1 Planning System

Since PSP is key to the success of our approach, we used the planner Sapa Replan (Talamadupula et al. 2010), a planner that has been used previously to support applications that require replanning. Sapa Replan additionally handles temporal planning and partial satisfaction. The system contains an execution monitor that oversees the execution of the current plan in the world, which focuses the planner's attention by performing objective (goal) selection, while the planner in turn generates a plan using heuristics that are extracted by supporting some subset of those objectives. Unfortunately, Sapa Replan's support for all of these varied functionalities renders it less scalable to an increase in the number of soft goals that must concurrently be pursued by the planner. This rules out extensive experimentation, as well as the testing of theories such as the one proposed in Experiment 4 (see Section 6.2). We are currently working on using faster planners that can handle larger numbers of soft goals for our experiments.

6.2 Experiments

For our experiments, we compared similarity based replanning against commitment sensitive replanning (Experiment 1) and against replanning from scratch (Experiment 2). In order to set up a compilation from similarity based replanning into PSP, we followed the procedure outlined in Section 5.3 to alter the IPC domains that we considered, and obtain the respective O_{as} operator sets and the P' problems (which we denote P_a for the experiments). We then ran all the prob-lem instances for all our experiments on the same planner.

Experiment 1 – We ran the planner with O_{as} and P_a, and recorded the net-benefit values of the resulting plan π_a. We then created two different versions of this problem, P_b and P_c respectively, to simulate the effect of changes in the world (and hence force "replanning"). The perturbations used to generate P_b from P_a involved deleting facts from the initial state, deleting goals, or both. P_c was a copy of P_a that additionally contained new "similarity" goals on the execution predicates of every action $o \in \pi_a$. Each of these similarity goals carried a penalty with it – one that is levied for every action in π_c that deviates from π_a (and hence lowers π_c's similarity to π_a). We ran the problem on P_b and P_c as well and recorded the net-benefit a nd m akespan values from these runs.

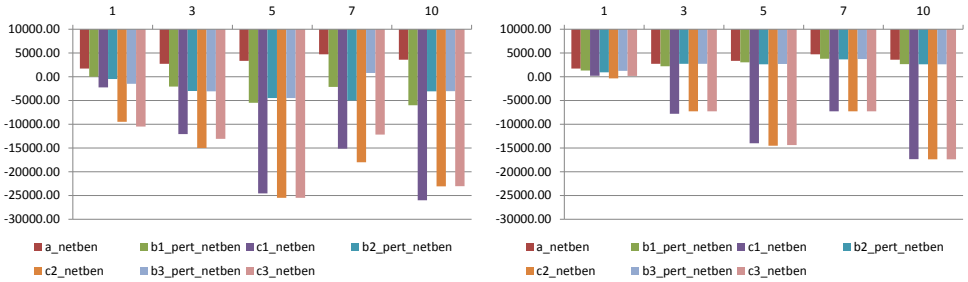

Figure 6: Net-benefit of plans for driverlog problems, Experiment 2.

Experiment 2 – For this experiment, we generated more perturbations to each P_a in a randomized fashion. These perturbations involved deleting facts from the initial state, deleting goals, or both. From each problem instance P_a we produced three perturbed instances $P_{b_1} \ldots P_{b_3}$ by deleting (at random) up to a maximum of 4 facts from the original initial state, and a maximum of 2 goals from the original set of goals to be achieved. Then, once the perturbed instances were generated, we generated three corresponding "similarity" instances $P_{c_1} \ldots P_{c_3}$ by copying $P_{b_1} \ldots P_{b_3}$ respectively, to force the planner to adopt a minimal perturbation approach. This was achieved by adding an additional set of "similarity" soft goals to the instances P_{c_i}; these similarity goals were generated using the same process as for Experiment 1. The addition of a penalty ensured that the planning process tried, as far as possible, to include the actions from the previous plan in the new plan. Due to the way the IPC problem instances are set up, even the smallest random perturbation to the initial state can render the perturbed problem unsolvable. To account for this problem, we created copies of all the P_{b_i} and P_{c_i} instances with only the goals deleted; that is, the initial state was unchanged, and only goal perturbations were introduced.

The resulting net-benefit values are plotted in Figure 5 and Figure 6; the plot on the left denotes those instances where the perturbation was performed on both the initial state as well as the goals, whereas the plot on the right represents those with only goal perturbations. The numbers along the X-axis are the IPC problem number, and the columns (from left to right) denote the instance that produced that value –respectively, P_a, P_{b_1}, P_{c_1}, P_{b_2}, P_{c_2}, P_{b_3} and P_{c_3}. Every pair of columns after the first one (there are three such pairs) signifies a direct comparison between the net-benefit for the perturbed problem, and the perturbed problem with similar-ity constraints added. Since the net-benefit values of many of the resulting plans are negative, the X-axis is shifted fur-ther up along the Y-axis. Due to this, smaller columns in-dicate better plan quality, since those net-benefit values are higher. We preserve the same axes for a given domain, so direct comparisons may be made between the plots on the left and the right.

Experiment 3 – Experiment 2 can also be modified in order to induce a direct comparison between similarity and commitment based replanning. Such an experiment would require a component that extracts commitments from a given plan π in an automated and unbiased manner; these commitments (or some randomized subset therein) would then be added in as goals for the perturbed version of the commitment based problem. The similarity based problem would be perturbed as before, and similarity goals added to it based on the actions in the original plan. Both of these instances can then be run on the same planning system, and the net-benefit values returned may then be compared directly. We have constructed a preliminary version of such a module, and are in the process of collecting results for this experiment.

Experiment 4 – Another experiment that can be performed is to contrast replanning methods that preserve similarity (either action or causal) against methods that instead preserve (state space) commitments. This can be done by fixing the number of commitments C that must be adhered to when transitioning from the original plan Π to a new plan Π'. Suppose that the state space of the original plan Π is given by S, where each element $s \in S$ is a state that results from the execution of actions $a \in \Pi$ starting from the initial state I. When the value of C is zero – that is, no commitments need be preserved when replanning – the net benefit of the new plan Π' will be at its highest value. Then, as the value of C tends toward $|S|$, the net benefit of the plan generated by the commitment preserving approach will steadily change, until at $C = |S|$ it is the same as the previous plan Π (indeed, at this stage, the new plan Π' is the same as Π). We are currently evaluating this experiment across various domains.

7 Conclusion

In this paper, we presented a general model of the single-agent replanning problem, and described three replanning paradigms that are distinguished by the constraints that they are bound to satisfy during the replanning process: replanning as restart, replanning to reduce computation, and replanning for multi-agent scenarios. In particular, we showed how commitment to certain constraints – whether they be from the structure of the previous plan, or understandings with other agents – can influence replanning. We then looked at solution techniques from the single-agent planning community for these three paradigms. Finally, we presented an evaluation of our main claims based on a compilation of the previously defined replanning constraints into a partial satisfaction planning framework.

8 Acknowledgements

We wish to thank anonymous reviewers for helpful comments and suggestions on past versions of this paper. Kambhampati's research is supported in part by the ARO grant W911NF-13-1-0023, the ONR grants N00014-13-1-0176, N00014-09-1-0017 and N00014-07-1-1049, and the NSF grant IIS201330813.

References

[Bartold and Durfee 2003] Bartold, T., and Durfee, E. 2003. Limiting disruption in multiagent replanning. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 49–56. ACM.

[Cushing and Kambhampati 2005] Cushing, W., and Kambhampati, S. 2005. Replanning: A New Perspective. In *Proc. of ICAPS 2005*.

[Fikes, Hart, and Nilsson 1972] Fikes, R.; Hart, P.; and Nilsson, N. 1972. Learning and executing generalized robot plans. *Artificial intelligence* 3:251–288.

[Fox et al. 2006] Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proc. of ICAPS 2006*.

[Fritz and McIlraith 2007] Fritz, C., and McIlraith, S. 2007. Monitoring plan optimality during execution. In *Proc. of ICAPS 2007*, 144–151.

[Gerevini, Saetti, and Serina 2003] Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *J. Artif. Intell. Res. (JAIR)* 20:239–290.

[Joslin and Pollack 1995] Joslin, D., and Pollack, M. E. 1995. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the National Conference on Artificial Intelligence*, 1004–1009.

[Kambhampati 1990] Kambhampati, S. 1990. Mapping and retrieval during plan reuse: a validation structure based approach. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 170–175.

[Knight et al. 2001] Knight, S.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. Casper: Space exploration through continuous planning. *Intelligent Systems, IEEE* 16(5):70–75.

[Komenda et al. 2008] Komenda, A.; Pechoucek, M.; Biba, J.; and Vokrinek, J. 2008. Planning and re-planning in multi-actors scenarios by means of social commitments. In *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, 39–45. IEEE.

[Komenda, Novák, and Pěchouček 2012] Komenda, A.; Novák, P.; and Pěchouček, M. 2012. Decentralized multi-agent plan repair in dynamic environments. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 1239–1240. International Foundation for Autonomous Agents and Multiagent Systems.

[Meneguzzi, Telang, and Singh 2013] Meneguzzi, F.; Telang, P. R.; and Singh, M. P. 2013. A first-order formalization of commitments and goals for planning.

[Nebel and Koehler 1995] Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: a complexity-theoretic perspective. *Artificial Intelligence* 76:427–454.

[Penberthy and Weld 1992] Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, 103–114. Citeseer.

[Simon 1964] Simon, H. 1964. On the concept of organizational goal. *Administrative Science Quarterly* 1–22.

[Srivastava et al. 2007] Srivastava, B.; Nguyen, T.; Gerevini, A.; Kambhampati, S.; Do, M.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *Proc. of IJCAI*, volume 7, 2016–2022.

[Talamadupula et al. 2010] Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)* 1(2):14.

[van den Briel et al. 2004] van den Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the National Conference on Artificial Intelligence*, 562–569.

[Van Der Krogt and De Weerdt 2005] Van Der Krogt, R., and De Weerdt, M. 2005. Plan repair as an extension of planning. In *Proc. of ICAPS 2005*.

[Wagner et al. 1999] Wagner, T.; Shapiro, J.; Xuan, P.; and Lesser, V. 1999. Multi-level conflict in multi-agent systems. In *Proc. of AAAI Workshop on Negotiation in Multi-Agent Systems*.

[Wooldridge 2000] Wooldridge, M. 2000. *Reasoning about rational agents*. MIT press.

Plan Sharing for Multi-Agent Planning

Daniel Borrajo
Departamento de Informática
Universidad Carlos III de Madrid
dborrajo@ia.uc3m.es

Abstract

Multi-agent planning has been shown to be harder than single-agent planning. There have been mainly two alternative approaches: centralized, where a single planner computes a plan for multiple agents; and distributed, where each agent computes a plan, and then plans are merged or coordinated. The centralized approach is not possible in many applications where agents have private goals, actions or states. We describe in this paper an approach to solve deterministic multi-agent planning problems with private information. Our approach first assigns a subset of the public goals to each agent that are added to the private ones. Then, agents iteratively solve problems by receiving plans, goals and states from the previous agents. After generating new plans by reusing previous agents plans, they share the new plans and some obfuscated private information with the following agents. Experiments show that this approach outperforms state-of-the-art techniques in the tested domains.

Introduction

Automated planning deals with the task of finding sequences of actions that achieve a set of goals from an initial state. We are interested on a deterministic multi-agent planning (MAP) setting, where we have a set of agents for which we have to find a solution to a collaborative multi-agent planning problem with private information (de Weerdt & Clement 2009). Generally, there have been two reasons to pose a planning task as a multi-agent planning task: being able to solve problems involving a high number of agents (scaling up); or when privacy matters (agents cannot, or are not willing to, share private information in the form of goals, actions or states). In this paper, we deal with both by decomposing the planning task, and maintaining agents privacy.

One of the most relevant characteristic when analyzing complexity of multi-agent planning tasks is the coupling level among different agents tasks and a key concept is interaction between agents plans. Interaction can be either positive (one agent achieves a (sub)goal that is also achieved by another agent) or negative (an agent deletes at least one (sub)goal achieved by another). The coupling level depends on the amount of interaction. If agents can achieve their goals without creating positive/negative interactions with the rest of agents plans, the tasks are loosely-coupled. As we find more interactions, the tasks become less loosely-coupled and more tightly-coupled.

There have been two main approaches to solve multi-agent planning tasks: centralized or distributed. The centralized approach aims at generating the complete plan for all agents in the same common search episode. But, the complexity grows exponentially with the number of agents in general. Also, the centralized approach cannot be used in some applications. A common case is when agents have relevant private information. This private information can refer to all planning components: goals, actions, types, and/or predicates. For instance, in a transportation logistics application, a company might have divided its operation in several branches. Each branch might receive its list of specific services (goals) to be addressed. The central branch might as well receive some common services to be planned for by any branch. So, there is a mixture of public and private goals.

Distributed planning consists of each agent solving its own planning task. However, given that there are public interacting goals, it might be that one agent, ϕ_i, generates a solution that invalidates another agent solution, ϕ_j, since it did not take into account ϕ_j private and public goals and actions to solve them. Potential solutions are plan merging (Foulser, Li, & Yang 1992) or plan coordination (Cox & Durfee 2005), that can be as hard as centralized planning and are usually only useful for loosely-coupled tasks (Cox & Durfee 2004). Among the recent deterministic distributed planning approaches we find the ones that use iterative plan refinement (Jonsson & Rovatsos 2011), distributed CSP (Brafman & Domshlak 2008; Nissim, Brafman, & Domshlak 2010), A* (Nissim & Brafman 2012), SAT approaches (Dimopoulos, Hashmi, & Moraitis 2012), or partial-order planning (Torreño, Onaindía, & Sapena 2012). A related body of work can be found on non-deterministic approaches using MDP or POMDP (Szer, Charpillet, & Zilberstein 2012).

We describe in this paper MAPR (Multi-Agent Planning by plan Reuse) that occupies a middle ground between distributed and centralized planning. MAPR considers both the agents private and public information. We have been inspired by iterative multi-agent planning techniques as the one pre-

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

sented in (Jonsson & Rovatsos 2011). MAPR first assigns a subset of public goals to each agent, while each agent might have a set of private goals also. Then, MAPR calls the first agent to provide a solution (plan) that takes into account its private and public goals. MAPR iteratively calls each agent with the solutions provided by previous agents. Each agent receives its own goals plus the goals of the previous agents. Thus, each agent solves its own problem, but taking into account the previous agents solutions. Since previous solutions might consider private data, all private information from an agent is obfuscated for the next ones.

We have called it planning by reuse given that each agent reuses information from previous agents. They reuse the goals, and can reuse (or not) the actions in their plans. So, since each agent receives the plan from the previous agent that implicitly considers the solutions to all previous agents, instead of starting the search from scratch, it can also reuse the previous whole plan or only a subset of the actions. We base our approach on recent work on plan by reuse (Fox *et al.* 2006). Experiments show that MAPR outperforms in several orders of magnitude state-of-the-art techniques in the tested domains.

The next section presents the multi-agent planning task we are dealing with. The third section describes MAPR approach. The following section shows the experimental setup and results. The fifth section compares MAPR with state of the art techniques. And the last section draws some conclusions and presents some future work.

Multi-Agent Planning Task

A single-agent STRIPS planning task can formally defined as a tuple $\Pi = \{F, A, I, G\}$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals. Each action $a \in A$ is described by a set of preconditions (pre(a_i)), that represent literals that must be true in a state to execute the action and a set of effects (eff(a_i)), literals that are expected to be added (add effects) or removed (delete effects) from the state after execution of the action. Actions definition might also include a cost $c(a)$ (default is one). In order to compactly represent planning tasks, automated planning uses the standard language PDDL (Planning Domain Description Language). Thus, a planning task Π is automatically generated from the PDDL description of a domain and a problem. The domain contains a definition of a set of generalized actions (defined using variables – parameters, par(a) – whose instantiations with problem objects will lead to actions in A), a set of predicates (whose instantiations will generate facts in F), and a set of types (to characterize the problem objects). A planning problem defines a set of objects (instantiations of types in the domain), an initial state (I), and a set of goals (G). The planning task should generate as output a sequence of actions $\pi = (a_1, \ldots, a_n)$ such that if applied in order would result in a state s, where goals are true, $G \subseteq s$. Plan cost is commonly defined as: $C(\pi) = \sum_{a_i \in \pi} c(a_i)$.

We consider a multi-agent setting, so we have to plan for a set of m agents, $\Phi = \{\phi_1, \ldots, \phi_m\}$. We define the MAP task as a set of planning subtasks, one for each agent, $M = \{\Pi_1, \ldots, \Pi_m\}$. Each planning subtask can be defined as a single-agent planning task, $\Pi_i = \{A_i, F_i, I_i, G_i\}$. All these components have a public part, that can be shared with the rest of agents, and a private part. In order to differentiate those parts, we provide each agent with the list of predicates and types that are private. Thus,

- all actions of each agent will have a private and a public part. An action can be totally public, totally private or a mixture (some literals public and some private)
- propositions in F_i and I_i can be either private, if their corresponding predicate is private, or public otherwise. Those that are private to a specific agent, will be obfuscated when shared among agents
- similarly, goals in G_i can also be either private or public. Again, both will be shared, but private ones will be obfuscated.

As an example, in the Satellite domain of the International Planning Competition (IPC)[1] several satellites must take images from different directions in space. The private types are instrument and mode, since they are only handled by each satellite. And private predicates are: supports, calibration_target, on_board, pointing, power_avail, calibrated and power_on, since they only relate arguments that are private to the satellite; the arguments of those predicates belong to either a private type or the agents type (satellite). Since in MAPR information on other agents is shared, MAPR obfuscates the private information of each agent when sharing it with the rest of agents (details on obfuscation come in the next section).

Multi-Agent Planning in MAPR

The main steps of the MAPR algorithm are to first assign common goals to agents and then iteratively solve each agent problem. Once an agent solves a problem, it obfuscates the private components of the solution and communicates them to the next agent. In turn, the next agent should solve its own problem augmented with the obfuscated private part of the solution of the previous agents and the public part of those solutions. Therefore, MAPR sees multi-agent planning as plan reuse. An important aspect of the algorithm consists on how to assign public goals to agents. As we will explain below, we have used several standard strategies. Figure 1 shows a high-level description of the algorithm. As we will explain later, we use @ to express obfuscated information. It takes as input a multi-agent planning task (domain, problem and agents description as explained in the previous section), a goal assignment strategy, the planner to be used by the first agent, and a second planner (it might be the same one) to be used by the following agents. The reason to use two planners (that could be different) is that the second planner might be a replanning system. Since all inputs and outputs are in PDDL, we can use any state-of-the-art planner. The algorithm is then composed of seven main steps: goal assignment; ordering of agents; first planning episode; obfuscation of the private part of a plan and communicating information to the next agent; merging of a prior agents plan with a planning

[1] http://ipc.icaps-conference.org/

problem; subsequent planning episodes; and termination. As a side comment on the algorithm, in the second and following iterations, when $j = 1$, then $j - 1$ means $j = m$. So, the first agent, instead of generating a new plan using the first planner, it takes as input the obfuscated solution from the last agent on the previous iteration. Next, we will describe in more detail all of these steps except for ordering of agents. In this paper, we left out all analysis of the influence of orderings (of agents, goals, ...). We just chose to use the ones provided in the PDDL files. Next subsections describe in more detail each component.

Function MAPR (M, GA, FP, SP): plan

M: multi-agent planning task
GA: goal assignment strategy
FP: first planner
SP: second planner

Assign subset of public goals to each agent ϕ_i using GA
Order agents
$\pi_1 \leftarrow$**First-Plan**(FP, ϕ_1)
$j \leftarrow 1$
Repeat until **Termination**
  $j \leftarrow j + 1$
  If $j > m$ Then $j \leftarrow 1$
  ϕ_{j-1} **Obfuscates** its private information, $S^{@}_{j-1}$:
    • the plan $\pi^{@}_{j-1}$ and
    • the problem $\Pi^{@}_{j-1} = \{F^{@}_{j-1}, A^{@}_{j-1}, I^{@}_{j-1}, G^{@}_{j-1}\}$
  ϕ_{j-1} **Communicates** $S^{@}_{j-1}$ to agent ϕ_j
  ϕ_j creates a new planning task, ϕ'_j:
    • it **Merges** its assigned problem Π_j and $S^{@}_{j-1}$
  $\pi_j \leftarrow$**Second-plan**(SP, ϕ_j)
If solved, return last plan

Figure 1: High level description of MAPR planning algorithm.

Goal Assignment

Given the total set of public goals G and a set of agents Φ, MAPR first has to assign a subset of goals to each agent to lower the planning complexity of each individual planning episode.[2] For each goal in $g \in G$ and agent in $\phi \in \Phi$, MAPR computes a relaxed plan from the initial state of each agent, I_i, following the well known relaxed plan heuristic of FF (Hoffmann & Nebel 2001).[3] If the relaxed plan heuristic detects a dead-end, then $c(g, \phi) = \infty$. This will define a cost matrix, $c(G, \Phi)$. Next, we have devised four goals assignment schemes.

all-achievable: MAPR assigns each goal g to all agents ϕ_i such that $c(g, \phi_i) < \infty$; that is, if the relaxed plan heuristic estimates g could be reached from the initial state of ϕ_i, g is assigned to ϕ_i.

rest-achievable: MAPR assigns goals iteratively. It first assigns to the first agent ϕ_1 all goals that it can reach (cost less than ∞). Then, it removes those goals from the goals set, and assigns to the second agent all goals that it can reach from the remaining set of goals. It continues until the goals set is empty.

best-cost: MAPR assigns each goal g to the agent that can potentially achieve it with the least cost, $\arg\min_{\phi_i \in \Phi} c(g, \phi_i)$

load-balance: MAPR tries to keep a good work balance among agents. It first computes the average number of goals per agent, $k = \frac{|G|}{m}$. Then, it starts assigning goals to agents as in best cost. When it has assigned k goals to an agent, it stops assigning goals to that agent. The next goals that could be assigned to this agent will be redirected to the second best agent for each goal. At the end, agents will have either all k goals, or $m - 1$ agents will have k goals and one agent will have the remaining goals, $\mid G \mid -k \times (m - 1)$.

In configurations rest-achievable and best-cost, there can be agents for which MAPR does not assign goals.

Planning

Once goals have been assigned to a subset of agents $\Phi' \subseteq \Phi$, planning starts by calling the first agent to solve its planning task. The task will be composed of its private planning task and its assigned public goals. If it does not solve the problem, it just passes the empty plan to the next agent. It could be either because there is no such plan, or because its plan needs some propositions to be achieved by the rest of agents plans. So, it will wait until, eventually, it will be called again to solve again the planning problem, but with some extra information coming from the other agents planning episodes. The following planning episodes can either use a planner or a replanner. In the latter case, apart from the domain and problem definitions, replanners take a previous solution as input (Fox *et al.* 2006).

Obfuscation

If the first agent solves the problem, then it cannot pass the private information directly to the rest of agents. So, it obfuscates (we will use obfuscate indistinctly of encrypt) the private parts and outputs an augmented obfuscated planning problem. There can be potentially many algorithms for encrypting/obfuscating the information. In this paper, we have considered a simple version of this procedure. Depending on the privacy commitment of the planning task, more complex obfuscating algorithms could be used and the difference will be: more time devoted to the obfuscating algorithm (their time complexity is usually much less than the one of planning); and potentially more space of the obfuscated information (any obfuscating algorithm with a space polynomial complexity could be used without affecting the overall multi-agent planning complexity).

In our case, obfuscating is a two steps process. First, a random substitution is generated for the names of all private predicates, actions and objects. For instance, in

[2] There could be in principle many alternatives to implement goal assignment ranging from a coordinator agent who decides without asking the rest of agents, to a model where agents negotiate goals. We opted here for the first approach for implementation simplicity.

[3] The relaxed plan heuristic computes the cost of a plan that could reach the goals from the initial state without considering the deletes of actions.

the Satellite domain, if a plan contains an instantiated action as `(calibrate sat1 inst1 Phen6)`, given that `calibrate` and `inst1` are private, MAPR would generate a random substitution as:

σ ={`(calibrate . g12) (inst1 . g23)`}

The second step in obfuscation consists of applying the substitution to the plan. An augmented obfuscated solution $S_j^@$ consists of the obtained plan and the set of components that are needed by the rest of agents to regenerate that solution. More specifically, if the plan of ϕ_j is $\pi_j = (a_1, \ldots, a_t)$, it communicates $S_j^@ = \{\pi_j^@, A_j^@, I_j^@, G_j^@\}$ to ϕ_{j+1}:[4]

- the set of instantiated actions in the plan, after obfuscating them, $A_j^@$, by obfuscating the actions parameters (par(a_i)), preconditions (pre(a_i)), and effects (eff(a_i)):
  $A_j^@ = \{a_i^@ \mid a_i \in \pi_j, a_i^@ = (\text{par}(a_i) \mid_\sigma, \text{pre}(a_i) \mid_\sigma, \text{eff}(a_i) \mid_\sigma)\}$
  where we use the notation $\alpha \mid_\sigma$ to represent the result of applying substitution σ to formula α.
- the obfuscated plan, $\pi_j^@ = \{a_1^@, \ldots, a_t^@\}$, since we can use planning by reuse in the next iteration instead of planning from scratch.
- all goals (private and public, including goals of previous agents), after obfuscating the private ones,[5]
  $$G_j^@ = \{g^@ \mid g \in G_j, g^@ = g \mid_\sigma\}$$
- initial state, after obfuscating the private information. Since MAPR only needs to pass to ϕ_{j+1} the relevant private part of the state, it only considers the literals that are preconditions of any action in the plan:
  $$I_j^@ = \{f^@ \mid f \in I_j, a_i \in \pi_j, f \in \text{pre}(a_i), f^@ = f \mid_\sigma\}$$

Communication

Each agent communicates $S_j^@$ to the next agent. We assume there is no noise in the communication. The size of the messages depends on: the plan size (number of actions in the generated plans π_i); the number of goals, where $\mid G \mid \approx \mid \pi_i \mid$; and the size of the initial state. Thus, the size of messages is linear with respect to the plan size and initial state size.

Merging

Each agent ϕ_{j+1} receives $S_j^@$ and builds a new planning problem by adding (appending) the instantiated actions to its actions set, the goals to its own goals, the private previous initial state to its own initial state and all new fluents to its own fluents set. So:

$\Pi_{j+1} = \{F'_{j+1}, A_{j+1} \cup A_j^@, G_{j+1} \cup G_j^@, I_{j+1} \cup I_j^@\}$

where $L(A_j^@)$ are all the literals in preconditions and effects of actions in $A_j^@$ and $F'_{j+1} = F_{j+1} \cup G_j^@ \cup I_j^@ \cup L(A_j^@)$.

Termination

Given that each planning task incorporates all previous goals, including the private ones of the previous agents, as soon as the last agent finds a plan achieving all goals, the whole planning process finishes. If the last agent does not find a plan and there is still time, we perform another iteration over all agents again, but with the accumulation of goals. Starting in the second iteration, as soon as an agent finds a solution, then the whole planning task finishes, since it incorporates all goals from all agents. The planning process will terminate with failure only if the time or memory bounds are reached or there is no solution.

Properties

As we mentioned before, our framework is suboptimal planning, so our approach is not optimal. Also, as in the case of MAP-POP, our approach is incomplete (for a different reason). In our case, take for instance the Logistics domain, where each agent has to partially contribute to the solution. For instance, in the standard *agentification* of the domain, trucks and airplanes are agents. If a package has to be delivered to a different city from a post-office of another city, we first need a truck to move it from the source post-office to the airport of that city and then use an airplane to move it to the other city. Thus, the goal of having the package on another city cannot be achieved only by any of these two agents in isolation. So, they return no solution when executed. We need to study in more depth the type of domains for which MAPR is incomplete. Intuitively, MAPR will not work in any domain where achieving one goal needs achieving a set of subgoals (including it) and there is no single agent that is able to achieve all subgoals in isolation.

Finally, MAPR is sound if the first and second planners are. Intuitively, given that all goals (public and private) are propagated, if the last agent solves the problem (in the first round) or any agent solves the problem (in the next rounds), the plan must be applicable from the initial state of the propagated initial state and its application must result in a state that achieves all goals.

Centralized vs. distributed planning

MAPR uses a decomposition scheme for multi-agent planning. Thus, as explained in the introduction, it takes into account two aspects of multi-agent planning: privacy and problem decomposition. An alternative way of using the ideas presented in this paper would be to let each agent initially obfuscate all its private information, send it to a centralized planning agent, and let that agent perform a centralized planning step with all the obfuscated information.[6] This approach would certainly take into account the privacy issue, but would not benefit from the problem decomposition directly (one could always include a problem decomposition approach to the centralized planning task). In the experiments, we include comparison with a simpler centralized approach that takes the original domain and problem, without the initial obfuscation of information by agents. This can serve as a rough comparison between the two approaches (MAPR and an *obfuscating* centralized approach) and also to analyze whether problem decomposition helps on this kind of problems. We leave for future work the exact comparison with such an approach,

[4] Again, all ordering decisions are taken arbitrarily.

[5] Substitution only affects the private goals.

[6] This has been noted by previous reviewers.

including the initial obfuscating step, but results should be similar to what we show.

Experiments and Results

In this section, we describe the experiments we have performed to compare our work with similar work. From an implementation point of view, our agents have been coded as function calls, so there is no overhead due to communication delays. Since the information that is being exchanged among agents is linear with respect to the size of plans and initial states, we do not expect a big overhead in planning time when transmitting it through a different communication channel. We have used the following experimental setup for comparison:

Comparing approaches. We compare against MAP-POP (Torreño, Onaindía, & Sapena 2012), since it represents the current state-of-the-art on suboptimal multi-agent planning. PLANNINGFIRST (Nissim, Brafman, & Domshlak 2010) could only solve the first two problems in the Rovers domain and the first problem in the Satellite domain. Other approaches, as the one presented in (Jonsson & Rovatsos 2011), deal with optimal planning. MAP-POP generates partially ordered plans with the goal of minimizing makespan, while MAPR generates totally ordered plans, with the goal of minimizing plan quality. Thus, we only compare with respect to planning time, since it is difficult to compare makespan to plan cost. We have also included the results of running a centralized approach (LAMA-FIRST). Given that in the IPC there is no multi-agent track, problems do not have private data, so there is no way to benefit from MAPR in those domains compared against a centralized approach. We show the results with LAMA-FIRST as a reference point that will not be possible to use in case of private data, and a close approximation to an obfuscating centralized approach, as mentioned before.

Domains. We have used the domains presented in previous works (Torreño, Onaindía, & Sapena 2012): Rovers and Satellite (we used the definitions of problems that they built from the problems of the IPC). The Satellite domain is loosely-coupled; satellites can operate without interaction. The Rovers domain is more tightly-coupled given that a rover might take a sample that is then unavailable to other rovers. In (Torreño, Onaindía, & Sapena 2012), the authors also report results in the Logistics domain. As explained in the previous section, our agents are not able yet to provide partial solutions to problems (incomplete plans) in this kind of domain.

Finally, we have also defined a new domain, that we call Port, to show how our approach works in a domain where agents have private information (specially goals). It is inspired by a real world port where there is an area in the dock with towers of containers with height of at most k containers (we used $k = 3$ in the experiments) placed in a grid of $n \times m$ (so, in comparison with the Blocksworld domain, there are fixed positions in the table, and a max height of towers). The port has a set of hoists, one for each ship that is waiting to be loaded with containers. Initially, all containers are in the dock, and the goals are to load all of them in the ships; the goals establish to which ship each container should go and on top of which other container. In this domain, we have defined hoists to be the agents, and all the goals to be private. Thus, each hoist has a set of containers that it has to load in its corresponding ship forming towers also specified in the goals. Given that all goals are private, there are no public goals, and all goal assignment strategies obviously perform the same assignment: a null assignment of public goals to each hoist. Also, this domain is more tightly-coupled than the other two, given that there is high interaction among private goals of agents. In order for a hoist to load a container into a ship, it might have to move around the containers in the dock (the container might be the bottom container in a tower) and those containers are the ones that have to be loaded in other ships. So, there are both positive and negative interactions.

Goal assignment. We have used the four defined methods: all-achievable (AA), rest-achievable (RA), load-balance (LB) and best-cost (BC).

Planners. We have used FastDownward code for generating the plan for the first agent (Helmert 2006; Richter & Westphal 2010). More specifically, we have used only one run of lazy greedy best first search with actions costs, and FF and LM-cut heuristics with preferred operators. We call it LAMA-FIRST. We have used LAMA-FIRST and LPG-ADAPT (Fox *et al.* 2006) for the successive planning episodes. While LPG-ADAPT is a replanning technique, LAMA-FIRST is not. We still call planning by reuse the configuration that uses LAMA-FIRST, because it can reuse the actions in the previous plans, given that they are added to the domain model and are needed to achieve private goals of other agents. LPG-ADAPT uses stochastic local search. Currently, there is a discussion on the planning community on how many times should stochastic planners be executed for each problem. We followed current practice (as in the IPC), making only one execution per problem.

Time bound. We have used 600 seconds, given that we wanted to compare with the results in (Torreño, Onaindía, & Sapena 2012) and most problems were either solved in 600 seconds or not solved.

In Table 1 we show the results of time to compute a solution in Rovers. Rows represent IPC problems and number of agents (rovers), #A. The next two columns are the planning time of MAP-POP (MAP) as reported in (Torreño, Onaindía, & Sapena 2012) and LAMA-FIRST (LF). The following four columns are the planning time of the use of LAMA-FIRST as the replanning system using the four different goal assignment strategies. The final four columns represent the planning times using LPG-ADAPT as the replanning system. The empty cells are unsolved problems by the corresponding technique. We used a 2.6GHz Intel Core i7 with 16Gb of RAM (though we run the experiments with a maximum of 4Gb and we did not run out of memory on any problem) running MacOS X.

MAPR solves all IPC instances with almost all configurations in up to three orders of magnitude less time. MAP-POP solves 14 problems. The combination of a powerful planner, LAMA, with the assignment of goals and the handling of private data makes MAPR much more effective than sharing decisions on partial order plans (MAP-POP), or solving the distributed CSP task (PLANNINGFIRST). A big advan-

			LF				LPG-ADAPT			
P (#A)	MAP	LF	AA	RA	BC	LB	AA	RA	BC	LB
1 (1)	0.44	0.00	0.17	0.16	0.16	0.16	0.16	0.16	0.16	0.15
2 (1)	0.34	0.00	0.15	0.15	0.15	0.15	0.16	0.15	0.15	0.16
3 (2)	0.8	0.00	0.36	0.36	0.36	0.35	0.37	0.36	0.37	0.37
4 (2)	0.9	0.00	0.36	0.35	0.35	0.35	0.38	0.36	0.39	0.38
5 (2)	2.15	0.00	0.40	0.41	0.38	0.38	0.40	0.39	0.38	0.39
6 (2)	2.17	0.00	0.44	0.43	0.43	0.43	0.43	0.41	0.41	0.42
7 (3)	3.75	0.00	0.69	0.18	0.39	0.62	0.65	0.18	0.40	0.59
8 (4)	60.35	0.01	1.01	0.42	0.64	0.90	0.92	0.40	0.64	0.90
9 (4)	15.77	0.01	1.01	0.69	0.96	0.90	0.94	0.65	0.89	0.89
10 (4)		0.01	1.09	0.46	0.45	0.96	0.96	0.43	0.43	0.90
11 (4)	10.39	0.01	1.08	0.47	0.72	0.97	0.96	0.44	0.67	0.90
12 (4)	3.17	0.00	1.01	0.65	0.92	0.90	0.95	0.66	0.90	0.92
13 (4)		0.02	1.18	0.54	0.55	1.17	1.03	0.47	0.51	0.98
14 (4)	47.10	0.01	1.14	0.52	0.51	0.98	1.17	0.47	0.48	0.93
15 (4)	20.68	0.01	1.24	0.59	0.54	1.14	1.06	0.51	0.48	
16 (4)	195.97	0.02	1.32	0.28	0.56	1.18	1.12	0.27	0.50	0.99
17 (6)		0.03	2.17	1.23	1.64	2.06	1.66	1.03	1.38	1.63
18 (6)		0.04	2.63	1.21	1.51	2.28	1.84	0.93	1.18	1.71
19 (6)		0.13	2.60	1.54	1.52	3.83	1.79	1.12	1.02	1.87
20 (8)		0.14	5.12	1.34	5.14	3.99	2.76	0.86		2.46

Table 1: Planning time (in seconds) in the Rovers domain.

tage over MAP-POP and other approaches is that we require much less communication. We only communicate after each planning episode finishes. We also see that if we employ a replanning system, as LPG-ADAPT, results improve over using LAMA-FIRST. In relation to goal assignment strategies, the rest-achievable alternative is the fastest one.

If we analyze plans cost, the reference system, LAMA-FIRST, generates a total cost of all solutions equal to 704.[7] For the configurations where we run LAMA-FIRST as the replanning system, the total cost was: AA, 732; RA, 803; LB, 763; and BC, 744. So, if we consider plan quality, the best option consists of assigning every goal to all agents that can achieve it, so they have a global view of all goals and are able to obtain reasonably good costs. Given that the problems are small, the time to compute them is only a fraction longer than the one of the other alternatives. As expected also, the second best option is to assign goals with the best-cost alternative. In case we run LPG-ADAPT as the replanning system, we had the total costs as: AA, 883; RA, 865; LB, 695 (*); and BC, 837 (*). Configurations LB and BC are marked given that they solved one less problem, so the total quality cannot be compared. In any case, LPG-ADAPT most often provides worse solutions than LAMA-FIRST, as expected, given that it performs stochastic local search, while LAMA-FIRST performs best-first search.

Table 2 shows the results of planning time for the Satellite domain. The analysis is similar to the one in the Rovers domain. We obtain again an improvement in some cases of one order of magnitude over MAP-POP. In this domain, given that agents are loosely-coupled (there is nothing they delete that the rest of agents need for their plans), by using a distributed approach we can even get slightly better performance than the centralized approach on some configurations. Each single-agent problem only focuses on the agents goals and actions. Thus, the search space is much smaller and the planner does not have to consider many more alternatives as when it solves the whole problem with the centralized configuration.

Table 3 shows the results in the Port domain. We do not compare against MAP-POP because MAP-POP cannot handle private goals. The first column shows the number of agents (hoists), the second one the number of goals (containers). The third column shows the results (time and quality - solution length -) of a centralized approach, by using LAMA-FIRST, and the next columns the results of our multi-agent approach. Since all goal assignment strategies generate the same problems here (all are private goals), the table only shows results for the best-cost strategy (the results in terms of time are very similar among them, and number of nodes and solution quality are the same). As one can see, the multi-agent approach is very competitive in this case against a centralized approach. It solves the same number of problems (though not the same ones), in some cases it generates solutions faster, and many times even with better quality. Again, the centralized approach is only shown as a reference, since it could not be applied in the real world, given that it is a problem with only private goals.

Related Work

MAP has been approached from both the multi-agent and the planning communities (de Weerdt & Clement 2009). Most previous work in the planning community defined agents as

[7] In this domain, all actions costs are one, so cost is equivalent to solution length.

			LF				LPG-ADAPT			
P (#A)	MAP	LF	AA	RA	BC	LB	AA	RA	BC	LB
1 (1)	0.25	0.00	0.14	0.13	0.13	0.13	0.13	0.13	0.13	0.14
2 (1)	1.18	0.00	0.14	0.14	0.14	0.13	0.14	0.14	0.13	0.15
3 (2)	0.71	0.00	0.36	0.35	0.32	0.33	0.39	0.36	0.37	0.37
4 (2)	0.96	0.00	0.38	0.16	0.16	0.35	0.38	0.16	0.15	0.38
5 (3)	1.72	0.02	0.62	0.60	0.58	0.59	0.59	0.58	0.61	0.62
6 (3)	1.62	0.00	0.58	0.37	0.57	0.58	0.61	0.37	0.60	0.61
7 (4)	2.94	0.01	0.88	0.63	0.58	0.84	0.89	0.63	0.61	0.87
8 (4)	4.21	0.02	0.96	0.43	0.64	0.92	0.91	0.41	0.64	0.90
9 (5)	7.51	0.02	1.27	0.72	0.88	1.16	1.25	0.67	0.89	1.14
10 (5)	5.71	0.16	1.34	0.48	0.68	1.21	1.34	0.45	0.65	1.16
11 (5)	5.27	0.03	1.38	1.35	1.23	1.31	1.22	1.16	1.13	1.19
12 (5)	8.25	0.64	1.62	0.62	0.93	1.54	1.33	0.54	0.77	1.29
13 (5)	17.87	7.33	2.30	1.84	1.35	2.32	1.46	1.14	1.05	1.36
14 (6)	11.93	0.34	1.98	1.59	1.51	1.83	1.55	1.26	1.24	1.56
15 (8)	33.47	0.10	2.68	2.65	2.38		2.14	2.09	2.05	2.11
16 (10)		0.10	3.55	1.28	1.52	3.23	2.63	1.02	1.26	2.70
17 (12)		0.11	4.08	2.16	2.11	3.74	3.18	1.54	1.78	3.21
18 (5)	6.57	0.30	1.51	1.20	1.55	1.53	1.23	0.95	1.23	1.24
19 (5)		1.88	2.18	1.29	2.11		1.46	0.86	1.29	1.40
20 (5)		1.62	2.22	0.86	1.28	2.35	1.60	0.71	0.90	1.46

Table 2: Planning time (in seconds) in the Satellite domain.

		LAMA-FIRST	MAPR (LAMA-FIRST)	MAPR (LPG-ADAPT)
#A	#Goals	Time (Quality)	Time (Quality)	Time (Quality)
2	5	0.10 (16)	1.61 (18)	0.76 (**12**)
2	10	1.40 (28)	2.58 (**20**)	0.77 (28)
2	20	296.60 (94)		
2	30	711.78 (152)	**268.52** (**128**)	**150.07** (242)
5	5	0.23 (12)	1.69 (12)	1.08 (12)
5	10	1.15 (20)	6.36 (20)	18.26 (20)
5	20	234.35 (74)	**67.90** (**64**)	**122.36** (**58**)
5	30	33.94 (92)	75.06 (**62**)	
10	10	2.58 (30)	8.37 (**20**)	3.50 (**26**)
10	20	28.55 (48)	53.39 (**40**)	91.26 (48)
10	30	92.41 (76)	162.41 (**66**)	
10	40		**350.39** (**84**)	
15	10	3.89 (24)	12.26 (36)	5.39 (46)
15	20	37.44 (44)	75.95 (**42**)	221.32 (48)
15	30	164.73 (60)	187.24 (60)	734.04 (90)
20	10	5.49 (24)	19.77 (**20**)	16.24 (**22**)
20	20	32.55 (42)	92.00 (42)	52.96 (58)
20	30	202.00 (68)	**200.45** (**60**)	268.76 (82)
20	40	390.15 (80)	491.46 (80)	

Table 3: Planning time (in seconds) and quality (solution length) in the Port domain. He have highlighted the results where MAPR improves over LAMA-FIRST in either time or quality.

resources and used centralized planning to solve multi-agent planning tasks. Another line of work allowed each agent to generate separate plans, and then tried to merge those plans (Foulser, Li, & Yang 1992). Usually, there was no consideration of private goals or actions. Recently, there has been a renewed interest on developing multi-agent planning techniques that explicitly consider the agents private information in the suboptimal (Dimopoulos, Hashmi, & Moraitis 2012; Nissim, Brafman, & Domshlak 2010; Torreño, Onaindía, & Sapena 2012) and optimal settings (Nissim & Brafman 2012; Jonsson & Rovatsos 2011).

In (Jonsson & Rovatsos 2011), its authors present a MAP approach that is based on an iterative refinement process of successive single-agent planning episodes. They start the planning iterations using an arbitrary plan in the joint-actions space. And then they perform successive single-agent cost-optimal planning steps to obtain better plans. We perform a similar iterative single-agent process, but MAPR differs in that its plans are totally-ordered and suboptimal.

In (Dimopoulos, Hashmi, & Moraitis 2012), the authors present μ-SAT, that uses SAT planning for generating individual agents plans and combining the solutions. It focuses on two agents problems, while we are not restricted to the number of agents in the problem. μ-SAT also considers the task of minimizing makespan, while we deal with totally-ordered plans. In their experiments, though, they only report on suboptimal planning. In (Nissim, Brafman, & Domshlak 2010), the authors implement ideas published in previous papers on using distributed CSPs to solve the planning task. MAP-POP (Torreño, Onaindía, & Sapena 2012) is based on agents that share their public information when performing partial-order planning. They also used an iterative refinement process that is able to work on both loosely-coupled and tightly-coupled domains.

A difference with these approaches is that they generate either partial or parallel plans. In the case of MAPR, our plans are totally-ordered (sequential) plans. In order to compare them with respect to the quality of solutions, we would have to transform totally-ordered plans into less constrained plans. Computing the optimal partially-ordered plan from a totally-ordered plan is NP-hard in general (Bäckström 1998). However, there are suboptimal algorithms that take linear time and have been shown adequate for this task (Veloso, Pérez, & Carbonell 1990). The other approaches are able to share incomplete plans, while MAPR in its current version is only able to share complete plans. Thus, in domains where agents cannot achieve goals just by themselves or complete plans from the other agents, they will be able to solve them. Another main difference with many of the previous approaches is that they do not share the private information, nor they can handle private goals.

While we focus on deterministic MAP, there has been plenty of work on solving non-deterministic MAP tasks (Szer, Charpillet, & Zilberstein 2012). The advantage of these works over MAPR is that they can deal with uncertainty. But, usually, they do not scale up as deterministic MAP.

Conclusions

We have presented in this paper MAPR, a multi-agent planning technique for tasks where agents share public and relevant private information (after obfuscating it for privacy). MAPR calls each agent with the plans, goals and state literals from previous agents in order to iteratively compose plans. Since they can use the private information from previous agents, they can re-achieve the private goals of other agents, while reasoning at the same time on how to achieve the current agent goals as well as the other agents public goals.

We have shown results where this approach improves in up to three orders of magnitude the efficiency on planning time over similar MAP approaches. In the experiments, we have also compared four different strategies for assigning public goals to agents. We have seen that their performance is quite similar independently of the replanning technique we used, though rest-achievable obtains better results in terms of planning time and all-achievable in terms of plan quality. We have also used two different replanning alternatives: a deterministic best-first approach (LAMA-FIRST) and a stochastic local search replanning algorithm (LPG-ADAPT). LPG-ADAPT is usually faster, while LAMA-FIRST provides better plan quality.

As current and future work we are setting up subsets of agents that together can achieve a subset of goals. We expect to solve problems in domains where agents have to collaborate in order to solve the complete problem, such as Logistics. Also, we would like to perform an exact comparison with an obfuscating centralized approach.

Acknowledgments

We would like to thank the help from Alejandro Torreño and Raz Nissim. This work has been partially supported by MICINN projects TIN2011-27652-C03-02 and INNPACTO IPT-370000-2010-008.

References

Bäckström, C. 1998. Computational Aspects of Reordering Plans. In *Journal of Artificial Intelligence Research*, volume 9, 99–137. Morgan Kaufmann Pub. Inc.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS'08*.

Cox, J. S., and Durfee, E. H. 2004. Efficient mechanisms for multiagent plan merging. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, 1342–1343. Los Alamitos, CA, USA: IEEE Computer Society.

Cox, J. S., and Durfee, E. H. 2005. An efficient algorithm for multiagent plan coordination. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, 828–835. IEEE Computer Society.

de Weerdt, M., and Clement, B. 2009. Introduction to planning in multiagent systems. *Multiagent and Grid Systems* 5(4):345–355.

Dimopoulos, Y.; Hashmi, M. A.; and Moraitis, P. 2012. μ-satplan: Multi-agent planning as satisfiability. *Knowledge-Based Systems* 29:54–62.

Foulser, D.; Li, M.; and Yang, Q. 1992. Theory and algorithms for plan merging. *Artificial Intelligence* 57(2-3):143–181.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS'06)*, 212–221.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 114–121.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, 1323–1330.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Szer, D.; Charpillet, F.; and Zilberstein, S. 2012. MAA*: A heuristic search algorithm for solving decentralized POMDPs. *CoRR* abs/1207.1359.

Torreño, A.; Onaindía, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'12)*.

Veloso, M. M.; Pérez, M. A.; and Carbonell, J. G. 1990. Nonlinear planning with parallel resource allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 207–212. San Diego, CA: Morgan Kaufmann.

How to Repair Multi-agent Plans: Experimental Approach

Antonín Komenda[1] and **Peter Novák**[2] and **Michal Pěchouček**[1]
{komenda|pechoucek}@agents.fel.cvut.cz, P.Novak@tudelft.nl
[1]Dept. of Computer Science and Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
[2]Dept. of Software and Computer Technology, Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, The Netherlands

Abstract

Deterministic domain-independent multi-agent planning is an approach to coordination of cooperative agents with joint goals. Provided that the agents act in an imperfect environment, such plans can fail. The straightforward approach to recover from such situations is to compute a new plan from scratch, that is to replan. Even though, in a worst case, plan repair or plan re-use does not yield an advantage over replanning from scratch, there is a sound evidence from practical use that approaches trying to repair the failed original plan can outperform replanning in selected problems. One of the possible plan repairing techniques is based on preservation of the older plans.

This work experimentally studies three aspects affecting efficiency of plan repairing approaches based on preservation of fragments of the original plan in a multi-agent setting. We focus both on the computational, as well as the communication efficiency of plan repair in comparison to replanning from scratch. In our study, we report on the influence of the following issues on the efficiency of plan repair: 1) the number of involved agents in the plan repairing process, 2) interdependencies among the repaired actions, and finally 3) particular modes of re-use of the older plans.

Motivation

Consider a team of heterogeneous robots working together so as to execute a mission in an environment. Since the robots feature heterogeneous capabilities, it might well be that none of them is able to complete the mission on its own, however by a careful coordination and teamwork, they should be able to reach the joint objective. The team of physical robots is embodied in a dynamic environment in which various events and plan execution interruptions occur and most importantly, in which actions of the agents can fail. To execute their mission, the agents must be able to cope with such a dynamics on both, the individual, as well as the coordination level. Here we focus on the problem of multi-agent plan repair which tackles such issue.

There are several approaches capable to drive multi-agent team activities in an environment with an *a priori* unknown dynamics. Firstly, there is a body of literature dealing with and extending models of decentralized partially observable Markov decision processes (Dec-POMDPs) (Bernstein et al. 2002). A Dec-POMDP model leads to computation of a *policy* for the agents in the environment ensuring that by following it, the team reaches the joint goal. The model assumes only partial observability of the environment, a feature capable to capture various eventualities which could occur in the environment. These, however, have to be known *a priori*, so that a probabilistic model of action outcomes can be constructed before planning. Secondly, single-agent contingency (Fu et al. 2011) and conformant (Palacios and Geffner 2009) planning techniques facilitate classical-style planning for domains with non-probabilistic uncertainty in either action outcomes or state the system happens to be in. However, again, in order to plan for actions in such domains, the possible contingencies and action models in the environment must be known before the planning phase. The above discussed approaches do not scale well to larger domains, especially when the model of run-time action failures and events which could occur is *a priori* unknown.

Recently, in (Komenda, Novák, and Pěchouček 2012) the authors proposed an approach of *multi-agent (MA) plan repair* (MA-REPAIR), based on multi-agent planning (MA-STRIPS) as introduced in (Brafman and Domshlak 2008). MA-STRIPS is an approach to planning for teamwork and coordination extending the classical STRIPS-based planning techniques. According to the MA-REPAIR approach, the multi-agent team computes a team plan using a fully decentralized MA-STRIPS planning algorithm, and subsequently executes the plan, while at the same time monitoring of possible failures of plan execution. Upon an occurrence of such a failure, the team stops execution and invokes a plan repair algorithm and fixes the failed joint plan in order to reach a joint goal state from the state in which the failure occurred.

It can be argued that plan re-use in a single-agent context does not yield much advantage with respect to the computational complexity in the worst case (Nebel and Koehler 1995), since costly attempts to fix a failed plan sometimes lead to replanning from scratch anyway. In multi-agent and multi-robot settings, such as those involving teams of underwater or aerial robots, where communication is unreliable and costly, however, it is often the communication which is of higher priority than the computational complexity.

In (Komenda, Novák, and Pěchouček 2013), the authors proposed prefix and suffix-based approaches to MA plan re-

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

pair. Their work showed that these repairing approaches save communication in contrast to replanning from scratch in tightly coupled problems with action failures, however a research question which plan repairing techniques are more appropriate for which planning domains and problems remained unanswered. In this work, we generalize the prefix and suffix-based approaches from their work and present a study on how particular multi-agent plan repair techniques and particular parametrizations perform in different planning domains.

Multi-agent planning & repair

The problem of multi-agent plan repair as defined in (Komenda, Novák, and Pěchouček 2013) is a tuple $\Sigma = (\Pi, \mathcal{P}, s_f, k)$, where

1. $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$ is a MA-STRIPS multi-agent planning problem over
   (a) a set of agents $\mathcal{A} = \alpha_1, \ldots \alpha_n$, each characterized by a set of STRIPS actions (over a propositional language $\mathcal{L}$) it can perform in an environment the agents operate in. I.e., $\alpha_i = \left\{\langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle \mid \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \subseteq \mathcal{L}\right\}$; and
   (b) an initial state $s_0 \in 2^{\mathcal{L}}$ the agents start to operate in, together with a specification of a set of final states $S_g \subseteq 2^{\mathcal{L}}$ characterizing their joint objective(s).
2. an original multi-agent plan $\mathcal{P}$ solving the problem Π, execution of which failed; and finally
3. a state $s_f \in 2^{\mathcal{L}}$ which the system happens to be in, unexpectedly after the plan execution failure occurrence of the plan execution step $k \in 1..|\mathcal{P}|$.

A solution to the MA plan repair problem $\Sigma = (\Pi, \mathcal{P}, s_f, k)$ is a multi-agent plan $\mathcal{P}'$ solving a modified planning problem $\Pi' = (\mathcal{L}, \mathcal{A}, s_f, S_g)$. A multi-agent plan is a sequence of n-tuples, *joint actions*, with $n = |\mathcal{A}|$ corresponding to the number of agents in the team $\mathcal{A}$. Thereby a multi-agent plan is a sequence of synchronized actions of the individual agents. In order to constitute a valid multi-agent plan for a planning problem Π, firstly, each agent executing its corresponding sub-sequence of primitive actions must be capable to perform them, that is, $a_i \in \alpha_i \in \mathcal{A}$ for each individual action in every joint action $(a_1, \ldots, a_n)$ constituting the k-th step of the plan. Secondly, synchronized execution of the sequence of joint actions must lead from the initial state to some final state prescribed by the planning problem definition. That means all the inter-dependencies between the actions of the individual agents imposed by their preconditions must be satisfied along the plan execution.

In a case there are several plans repairing the original failed plan $\mathcal{P}$, the idea is to prefer those solutions, which *preserve*, that is re-use, parts of the original plan as much as possible. More formally, given two plans $\mathcal{P}_1, \mathcal{P}_2$ repairing the same multi-agent plan $\mathcal{P}$, we say that $\mathcal{P}_1$ is more preserving than $\mathcal{P}_2$ iff $\mathit{diff}(\mathcal{P}_1, \mathcal{P}) \leq \mathit{diff}(\mathcal{P}_2, \mathcal{P})$, where diff denotes the edit distance, i.e., the Levensthein distance (Levenshtein 1966) between the two plans given as arguments. Edit distance of two strings equals the minimal number of primitive edits. The adaptation to multi-agent plans corresponds to primitive edits being joint action addition, joint action deletion and a primitive action replacement operations. For further details, refer to the original papers (Brafman and Domshlak 2008; Komenda, Novák, and Pěchouček 2013).

Besides considering straightforward replanning from scratch, that is invoking the underlying multi-agent planner at the point of a failure and then executing the computed plan right away, in (Komenda, Novák, and Pěchouček 2012), the authors present two main approaches to solving multi-agent plan repair problems coined *back-on-track* and *lazy* repair respectively. Informally, the back-on-track approach tries to fix the prefix of the failed plan by computing a plan from the state in which the system happens to be right after the detection of a plan execution failure to some state along an ideal failure-free execution of the original plan. The resulting multi-agent plan re-uses some *suffix* of the original plan, if possible, and extends the plan at its beginning. The idea underlying the lazy repair is complementary. Lazy repair takes the remainder of the original plan, re-uses all its actions which still can be executed according to their preconditions regardless of the outcome and completes the plan to some goal state of the planning problem. This way, the resulting plan is composed of re-used *prefix* parts of the original plan with an appended suffix of some new repaired plan.

They also experimentally show that in some domains, these approaches lead to significant savings of communication, as well as computational resources in comparison to replanning from scratch in selected problems. Both algorithms first formulate a modified multi-agent planning problem and rely on the underlying multi-agent planner to compute a plan fragment used for re-composition into a solution plan repairing the original failed one.

A straightforward extension of the approach is to consider re-use of different parts of the original failed plan by combining parts of the prefix vs. suffix appending plan repair. Consider e.g., a repairing strategy according to which the back-on-track algorithm would not return back to an arbitrary state along the original plan's failure-free execution, but rather would consider only returning to the next few states presumed after the point of failure—this is approximately the way GPS navigation devices tend to fix a route after a missed junction point. Clearly, in some domains, this can be a beneficial approach, however not in all. An alternative idea would be to select only a minimal relevant subset of agents which should participate in the plan repair process, thereby constraining the inter-agent communication only to a subset of the agent team involved. In result, further reduction of communication needed for planning can be achieved.

Given the two above intuitions, we formulate first two hypotheses tackled in this paper.

Hypothesis 1 *Repairing algorithms minimizing the number of agents involved in the plan repairing process tend to generate lower computational and communication overheads than other strategies.*

Hypothesis 2 *Repairing algorithms reusing the original plan as a suffix generate lower computational and communication overheads than the repairing algorithms reusing the*

Algorithm 1 MA plan execution process with generalized multi-agent plan repairing algorithm.

Input: A MA planning problem $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_g)$.
Input: Parameters f and g bounding the maximal length for reusing of the original plan as prefix and suffix.

$\mathcal{P} = \mathsf{MA\text{-}Plan}(\Pi)$
if $\mathcal{P} = \emptyset$ **then return** fail
$k = 1$
repeat
  agents perform $\mathcal{P}[k]$
  if failure detected **then**
    retrieve the current state s from the environment
    // begin: plan repairing $\Sigma = (\Pi, \mathcal{P}, s, k)$
    $f^* = f$; $g^* = g$
    repeat
      $\mathcal{P}_{\mathrm{pre}} = \mathsf{ExecReminder}(\mathcal{P}[k..(k + f^*)], s)$
      $\mathcal{P}_{\mathrm{suf}} = \mathcal{P}[(|\mathcal{P}| - g^*)..|\mathcal{P}|]$
      $\mathcal{P}^* = \mathsf{MA\text{-}Plan}((\mathcal{L}, \mathcal{A}, s \oplus \mathcal{P}_{\mathrm{pre}}, S_g \ominus \mathcal{P}_{\mathrm{suf}}))$
      if $\mathcal{P}^* \neq \emptyset$ **then**
        $\mathcal{P} = \mathcal{P}_{\mathrm{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\mathrm{suf}}$
        break
      end if
    until tested all pairs of $f^* \in f..0$ and $g^* \in g..0$
    if $\mathcal{P} = \emptyset$ **then return** fail
    // end
    $k = 1$
  else
    $k = k + 1$
  end if
until $k > |\mathcal{P}|$

original plan as a prefix in domains featuring actions with long-term dependencies.

A long-term dependency can be visualized as a tree of consecutively dependent actions. If an action in the root of such tree has to be repaired, intuitively, it is a better idea to try to fix it as soon as possible, because not doing so can cause a snowball effect of increasing number of failing actions.

Further, we define the used algorithm, formulate last hypothesis and treat the hypotheses with three experiments.

Generalized repair algorithm

As outlined in the previous section, the algorithm used in the following experiments is a generalization of the lazy and back-on-track approaches with a number of additional modifications described in detail in the respective sections. The algorithm with the complete plan execution, monitoring and repair scheme is outlined in Algorithm 1.

The algorithm takes a multi-agent planning problem Π and two integer parameters f and g limiting the number of actions, which upon a detection of a failure can be reused during plan repair process from the currently executed *plan* $\mathcal{P}$ as a prefix or a suffix of the repairing plan. The process starts with a computation of the initial plan, which is subsequently executed. k denotes the counter of the current step in the plan execution. As the plan execution proceeds, in each step, all the agents perform their individual actions prescribed by the k-th joint action in the plan $\mathcal{P}$, denoted $\mathcal{P}[k]$.

In the case a failure is detected by the team, current state after the failure is retrieved and the plan repairing algorithm for the plan repairing problem $\Sigma = (\Pi, \mathcal{P}, s, k)$ is invoked. In each plan repairing attempt a modified multi-agent planning problem is formulated according to the current values of f^* and g^* prescribing the length of the re-used prefix and suffix of the original plan. In the case a repairing plan is found, the repairing process finishes, otherwise another attempt with a different combination of f^* and g^* is carried out. The resulting repairing plan consists of three components: the preserved prefix of the original plan $\mathcal{P}_{\mathrm{pre}}$ reusing $\mathcal{P}[k..(k + f^*)]$, a newly computed infix $\mathcal{P}^*$ and suffix part $\mathcal{P}_{\mathrm{suf}}$ reusing $\mathcal{P}[(|\mathcal{P}| - g^*)..|\mathcal{P}|]$, again preserving a part of the original plan $\mathcal{P}$.

The preserved prefix part of the original plan corresponds to an *executable reminder* fragment of $\mathcal{P}$, $\mathsf{ExecReminder}(\mathcal{P}, s)$, a selection of still applicable joint actions given the failure which occurred. The actions with unmet preconditions are simply omitted. Additionally, the prefix $\mathcal{P}_{\mathrm{pre}}$ is based only on a part of the original plan effectively re-using f^* actions beginning after the k-th action of the original plan $\mathcal{P}$. The suffix part $\mathcal{P}_{\mathrm{suf}}$ is obtained as the last g^* actions of the original plan $\mathcal{P}$.

Finally, the infix part of the plan is computed by invocation of the underlying multi-agent planner algorithm MA-Plan proposed by Nissim, et al. (Nissim, Brafman, and Domshlak 2010). The initial state of the modified planning problem is the state in which a failure-free execution of the repairing prefix $\mathcal{P}_{\mathrm{pre}}$ would result in starting from the state s and it is denoted as $s \oplus \mathcal{P}_{\mathrm{pre}}$. The set of goal states corresponds to a back-propagation of effects of the preserved suffix component $\mathcal{P}_{\mathrm{suf}}$ from the set of original goals S_g. More formally, the back-propagation operator $\ominus : 2^{\mathcal{L}} \times \bigcup_{a \in \alpha \in \mathcal{A}} a \rightarrow 2^{\mathcal{L}}$ for a single action is defined as $s \ominus a = s \cup \mathsf{del}(a) \setminus \mathsf{add}(a)$. It extends to sequences of actions as follows.

Definition 1 (proposition back-propagation) *Let S' be a set of propositions back-propagated from a set of propositions S using a MA plan $\mathcal{P}$ denoted as $\ominus$ operator extension $S' = S_g \ominus \mathcal{P}$ iff $S' = (\cdots((S \ominus \mathcal{P}[m]) \ominus \mathcal{P}[m-1]) \ominus \cdots) \ominus \mathcal{P}[0]$.*

In the case a multi-agent planner finds a plan for the modified planning problem, the repairing plan takes the form $\mathcal{P}_{\mathrm{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\mathrm{suf}}$ and gets executed from that point on. In the case no repairing plan can be found, the algorithm attempts the repair for a different combination of f^* and g^* until either a repairing plan is found, or it turns out that no repair for the failure exists.

The parametrization of the repairing process based on the f and g parameters opens an interesting research question, how do different combinations of the prefix and suffix preservation parameters influence the efficiency of the plan repairing process. Let m be the length of the re-usable part of the original plan $\mathcal{P}$, i.e., $m = |\mathcal{P}| - k$. Obviously, for $f + g < m$ there will be a gap, which has to be filled by a result of the inner planning process, in other words the original plan was *underused*. Reversely, for $f + g > m$ there

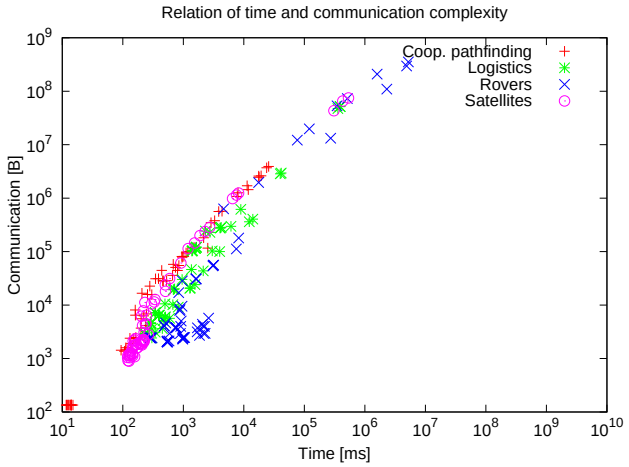

Figure 1: Relation between communicated bytes and computation time for solving the plan repairing problems.

will be an overlap, which has to be reverted, i.e., the original plan was *overused*. Intuitively, these cases are in a sense pathological. In a consequence, we propose the third, final hypothesis of this paper.

Hypothesis 3 *Repairing algorithms overusing or underusing the original plan tend to generate higher computational overheads than other algorithms.*

Experimental setup

The experiments were conducted in a synthetic setting, a simulated world with a group of agents using the plan execution, monitoring and repair loop. The world is fully observable for the agents. All failures of plan execution were generated by the simulator according to a uniform distribution over time and parametrized by a probability p of failure occurrence in each step for each experiment. A failure is generated only if there exists a plan to a goal state, which obviates problems with irreversible actions. The failures are handled by the agents immediately upon detection.

A failure is simulated by not-execution of some of the agent actions from the actual plan step. The individual action is chosen according to an uniform probability distribution over the individual actions within a joint action. As showed in (Komenda, Novák, and Pěchouček 2013), failure models with more radical impacts on the environment (e.g., state perturbations) decrease usability of the plan repairing approaches. Our motivation in this work is to study types of plan repairing, therefore we stick only to action failures.

For the implementation of the experimental setup and the repairing algorithms, we used a centralized world simulator integrating the multi-agent domain-independent planner MA-Plan (Nissim, Brafman, and Domshlak 2010). Each agent runs in its own thread and they deliberate asynchronously. The experiments were executed on 8-core processor at 3.6GHz with Java VM limited to 2.5GB of RAM.

For the experiments, we used four planning domains. Three of them originate in the standard single-agent IPC planning benchmarks. Similarly to the evaluation of the MA-Plan algorithm in (Nissim, Brafman, and Domshlak 2010; Komenda, Novák, and Pěchouček 2013), we chose domains, which are straightforwardly modifiable to a multi-agent setting: LOGISTICS, ROVERS, and SATELLITES. Additionally, we have extended the set of benchmarks by COOPERATIVE PATHFINDING coordination domain on a grid.

The experimental measurements were based on two metrics focusing on the target efficiencies: cumulative time consumed by the particular plan repairing algorithms during a single run of the simulation, i.e., the overall time spent in the algorithm (incl. the underlying planning process) excluding the initial planning phase of the scheme (Algorithm 1). The second metric was communication complexity of the process, that is the volume of communicated information in bytes among the involved agents during the plan repairing processes. Those are mainly the messages generated by the DisCSP solver in the MA-Plan planner and an additional synchronization processes minimizing the number of agents involved in the plan repairing process.

To account for differences in essential computational and communication complexity of the domains, we conducted a relationship experiment between these two measures. Figure 1 depicts the results and demonstrates that there is no essential discrepancy between the computational and communication complexity of the plan repairing solutions. That means, the following results are not biased by problems extremely hard in time and simple in communication and vice versa.

The number of repairing agents

Regardless of the theoretical results presented in (Brafman and Domshlak 2008), showing that the computational complexity of DisCSP-based multi-agent planning is not exponentially dependent on the number of the agents, in practical experiments, we faced a non-negligible dependence of the this number and required communication and computational effort. The first set of experiments analyzes this relation.

Used algorithms To validate Hypothesis 1, we have prepared an extensive set of plan repairing algorithms stemming from the Generalized repair algorithm. They can be divided into three main groups: one without agent count minimization, and two with agent minimization. First of the minimization groups reuse the original plan as a suffix and the other one as a prefix.

The difference among the algorithm instances within one of the groups lies in preference between agent minimization, size of preservation of the original plan and bound on the maximal length of the newly generated repairing plan component $\mathcal{P}^*$. This approach restrain bias prospectively caused by unbalanced influences of the agent minimization on various types of plan repair.

The approach used to minimize the number of involved agents was based on the notion of a set of *supporting agents*. The iterative process from Algorithm 1 was extended with an iteration starting only with a set of agents providing at least one action, which can contribute to the repairing plan by a required proposition(s), i.e., support part of $S_g \ominus \mathcal{P}_{\text{suf}}$.

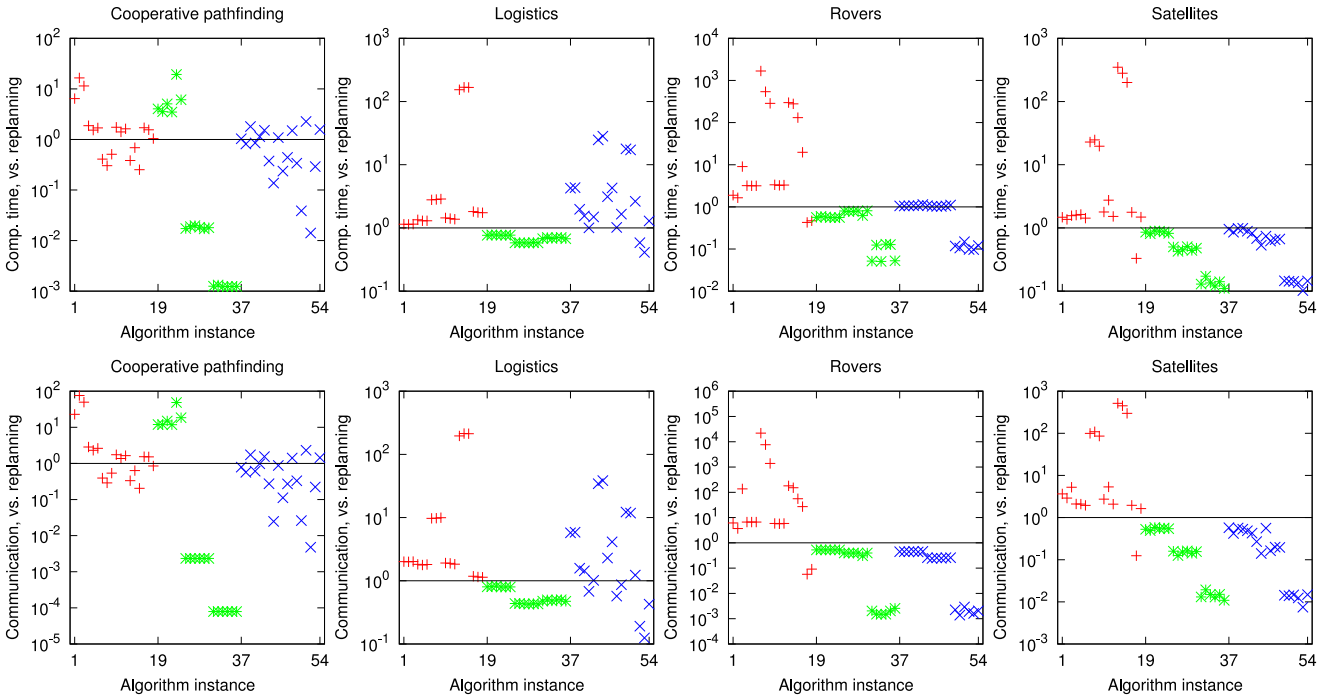

Figure 2: Comparison of various plan repairing algorithms in proportion to replanning (black line at $y = 1$) with failure probability $p = 0.3$. Each point represent a mean of several runs of one of the particular repairing algorithms. The red group contains plan repairing algorithms using only the full set of agents involved in the original planning problem, the green group contains algorithms using various techniques to minimize number of agents involved and preserving suffix of the original plan and the blue group contains algorithms also minimizing number of agents and preserving prefix of the original plan.

If such team of agents is not able to solve the plan repairing problem, the team is extended by additional agents supporting any of the current agents in the team by means of contributing to prepositions in their preconditions. If such additional agent does not exist and the team is still not containing all the agent from $\mathcal{A}$, a random agent is added into the team and the process continues.

Results and Discussion The experiments were conducted in all presented experimental domains and for all combinations of agent counts, i.e., two to four agents giving twelve domain and problem instances. Each of the group contained six variances of the algorithms giving with the problem instances 216 experiments. Each of the experiments was averaged over 5 measurements with different random seeds.

Figure 2 shows results of the first batch of experiments. The first group of repairing algorithms not minimizing number of involved agents (red color) is in most measurements in both computational and communication metrics worse than the baseline replanning algorithm. The suffix preserving algorithms minimizing numbers of agents (green color) is on the other hand nearly in all measurements better in both metrics than the baseline algorithm with an exception in the simplest COOPERATIVE PATHFINDING problems. The group of plan repairing algorithms minimizing the number of involved agents and preserving prefix part of the original plan (blue color) is on tie or better with the replanning in rather loosely coupled domains decreasing the communication and computational overheads with decreasing coupling of the domains. However in tighter coupled domains the algorithms fall behind the replanning baseline. In LOGISTICS domain, only 33% of the algorithms are better by communication overheads and only 18% by means of computational overheads. With increasing coupling the approach lose more. These results *support the first hypothesis*.

Additionally, the results revealed that the prefix-based approaches, as not the best in all agent minimizing approaches, in most of the experiments has one of the best approaches outperforming the best suffix-based approach. In LOGISTICS domain the separation between the best prefix-based and best suffix-based plan repairing algorithm is about a half an order of magnitude in favor of the one prefix preserving approach. On the other hand, in COOPERATIVE PATHFINDING, suffix-approaches gain an order and more.

Repairing of long-term dependencies

The intuition behind the second hypothesis can be rephrased as follows: If an action fails and it has potentially a lot of future dependencies, possibly of other agents or even in the goal, trying to fix it as soon as possible is rather better idea, than ignore it and try to repair it later. The experiments in this section were conducted to validate this concept.

Used algorithms The most straightforward approach here is to compare the two plan repairing algorithms re-using the whole original plan either as a prefix or as a suffix. These

$$
\begin{array}{l}
\mathsf{A}: \\
\mathsf{T}_1: \\
\mathsf{T}_2:
\end{array}
\left(\begin{array}{cccccccccc}
\epsilon & \epsilon & \epsilon & \overline{\mathsf{l}(\mathsf{p},\mathsf{a}_1)} & \mathsf{f}(\mathsf{a}_1,\mathsf{a}_2) & \overline{\mathsf{u}(\mathsf{p},\mathsf{a}_2)} & \epsilon & \epsilon & \epsilon \\
\mathsf{l}(\mathsf{p},\mathsf{d}_1) & \mathsf{m}(\mathsf{d}_1,\mathsf{a}_1) & \overline{\mathsf{u}(\mathsf{p},\mathsf{a}_1)} & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\
\mathsf{m}(\mathsf{d}_2,\mathsf{a}_2) & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \overline{\mathsf{l}(\mathsf{p},\mathsf{a}_2)} & \mathsf{m}(\mathsf{a}_2,\mathsf{d}_2) & \mathsf{u}(\mathsf{p},\mathsf{d}_2) \\
8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 &
\end{array}\right)
$$

Figure 3: A multi-agent plan solving the initial LOGISTICS problem used in the experiments. Empty actions are denoted as ϵ. The overlines mark public actions. The numbers in the last row represent particular counts of steps, i.e., number of actions m, to the end of the plan.

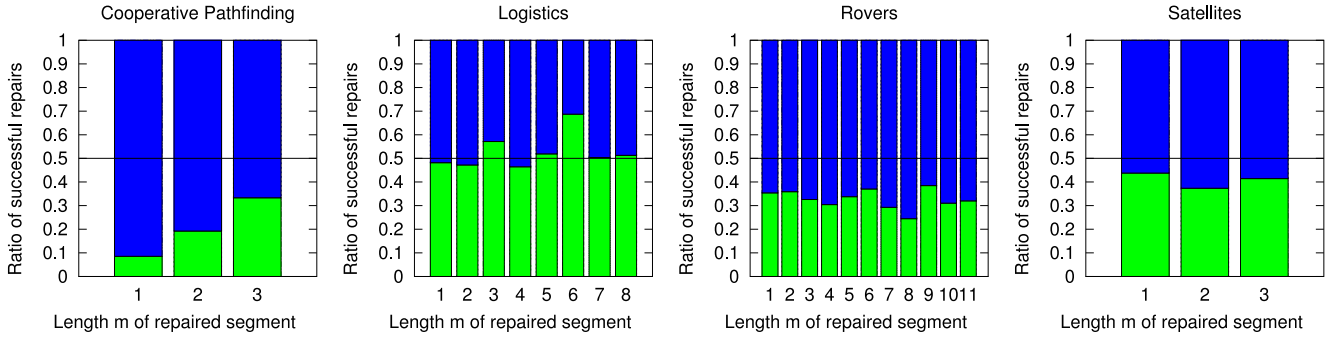

Figure 4: Comparison of success ratio against replanning between prefix-based (blue) and suffix-based (green) plan repairing.

algorithms are again modification of the plan repairing part of Algorithm 1 such that there is no iteration over various f^* and g^*, but only two fixed values. The *pure prefix algorithm* uses fixation $f^* = m, g^* = 0$ and the *pure suffix algorithm* uses only one parameter pair $f^* = 0, g^* = m$.

Furthermore, to be able to demonstrate the behavior and to explain the results, we have to present more details on LOGISTICS. The problem used in the experiments contains three agents controlling two trucks T_1 and T_2 and one airplane A. There are two cities, each with one storage depot (d_1 and d_2) and one airport (a_1 and a_2). The trucks can move $\mathsf{m}(\mathsf{from},\mathsf{to})$ only within their cities, i.e., between one depot and one airport. The airplane can fly $\mathsf{f}(\mathsf{from},\mathsf{to})$ among all airports in the environment, but cannot land at the depots. All vehicles can load $\mathsf{l}(\mathsf{package},\mathsf{location})$ and unload $\mathsf{u}(\mathsf{package},\mathsf{location})$ a package at a location. Initially, there is one package p at one of the depots and the goal is to transport it to the other depot in the other city. The trucks start at the depots and the airplane starts at one of the airports. A typical multi-agent plan solving this particular instance is depicted in the matrix form, see (Komenda, Novák, and Pěchouček 2013) for more detail, in Figure 3.

Results and Discussion To validate Hypothesis 2, we run the pure prefix-based and pure suffix-based repairing algorithms in all the testing domains. We have measured ratio of successful repairs of these two repairing algorithms against replanning by means of computation time. In Figure 4, we summarize the results of these experiments.

In the ROVERS and SATELLITE domains, the plans solving the problem do not contain any significant actions by means of number of future dependencies to the overall count of actions in the plan. In SATELLITES, all actions are private, therefore actions of one agent depend only on other actions of the same agent. Additionally, the individual plans of the agents are relatively short (three to four actions), therefore the private dependencies are never longer than four actions.

Multi-agent plans for the ROVER problems contain several public actions at the end of the plan, representing always only one rover communicating at one time point. Albeit the plans solving the ROVERS problems contain public actions, there are again no long dependencies among the actions. The dependencies in the private part of the plan contain three components, each containing three to four private actions. Consequently, the private dependencies are, similarly to the SATELLITE problems, maximally four actions long. The dependencies among the public actions are even shorter, as there is the same number of public actions as agents, which means maximally three-action public dependencies for three agents. The dependency link between one public action and one dependent private component increases the maximal dependent length to maximally seven actions (four private actions of the component bound to three public actions successively dependent on each other).

In such repairing problem, even if one of the leading actions in a private component fails, lazy approach solves nearly the complete problem only by re-using the original plan. More precisely, it re-uses the original solution for the rest of the private components and all the public actions except one of the failed agent. The results show, the prefix-based repair is always better then the suffix-based and the ratio between these two is stable over different points in the plan. The situation changes in the LOGISTICS domain.

In LOGISTIC with three agents and one package, there is a chain of dependent actions. Particularly, $\mathsf{u}(\mathsf{p},\mathsf{d}_2)$ depends on $\overline{\mathsf{l}(\mathsf{p},\mathsf{a}_2)}$, which depends on $\overline{\mathsf{u}(\mathsf{p},\mathsf{a}_2)}$ and so on to the first action of the plan $\mathsf{l}(\mathsf{p},\mathsf{d}_1)$. The dependency chain has six

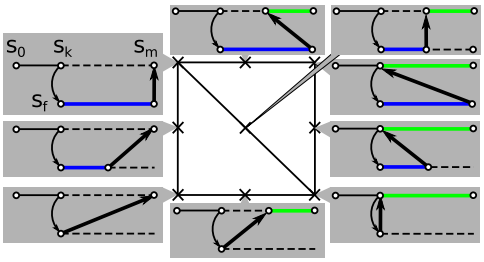

Figure 5: Scheme of a two-dimensional space representing plan repairing algorithms preserving different parts of the original plan and reusing it in different ways. The blue segments represent prefix-based re-usage and the green ones the suffix-based re-usage. The notable states are: initial state s_0, last achieved state s_k induced by the original plan, exceptional state s_f after a failure and the last anticipated state $s_m \in S_G$, provided that the original plan would be executed without a failure.

actions in the experimented plan and occupy the complete length of it. As the results show in Figure 4, there are two distinctive peaks where the suffix repair outperforms the prefix repair, additionally with an increasing trend. The first one is for repairing plans of length $m = 3$ and the other one is for $m = 6$. As we can see in Figure 3, these lengths correspond to the package hand-off points in the plan, more precisely repair of failing unloads $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_1)}$ and $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_2)}$. Ignoring a failure of unloading by a lazy approach causes the package is left in the last vehicle and the rest of the team finishes the executable remainder of the plan, which effectively means the vehicles are moving, but they are not transporting the package. Under such circumstances, the suffix repair only repeats the unload action and successfully continues with the rest of the original plan ending in a goal state.

One can argue that the complement load actions should be repaired more efficiently using this same argumentation as well. This is very true, however this phenomenon is not captured in the results, because of a particular implementation of the MA-Plan planner. The explanation is based on the fact the used planner efficiency is more dependent on small differences in number of involved agents, than the number of planned actions. In the case of $m = 3$, i.e., the $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_2)}$ action, we need 2 agents to do lazy repair, because firstly we reuse executable reminder of the original plan to the last state without the package and than we have to use the planner to generate repairing plan $\mathcal{P}^*$reverting all the moves and planning again to one of the goal states. Such plan has to firstly unload the package from the airplane A and then transport it successfully by the truck T_2 to the goal destination d_2. On the other hand, the back-on-track algorithm only generates a plan repeating the unload action $\overline{\mathsf{u}(\mathsf{p}, \mathsf{a}_2)}$ and afterward continues with the original plan as a suffix. This planning problem involves only one agent, in particular, the airplane A carrying out unload of the package. The same principle can be applied to $m = 6$, but with all three agents for pure lazy repair but only 2 agents for pure back-on-track.

In the last problem of COOPERATIVE PATHFINDING, the length of a sequence of dependent actions correspond to the length of the plan, as all the actions in such plan are public and inter-dependent. Nevertheless, this is quite different "order of dependency", than in SATELLITES for example. In SATTELITES, all the actions are dependent as well, but only within one agent, whereas here, the actions are dependent across the agents. In the experimental results of the COOPERATIVE PATHFINDING a trend arises. In such dense types of inter-dependent problems, the longer are the repaired plans, the more the suffix repairing algorithm gains against the prefix one. Unfortunately with the current available implementation of the multi-agent planner, we were not able to conduct experiments with bigger grid sizes, i.e. longer repaired plans, thus the validation if the trend would continue is left for future work at this point.

The results of these experiments, namely of LOGISTICS and COOPERATIVE PATHFINDING, moderately support the second hypothesis of the paper.

Repair appropriately reusing the original plan

A fundamental principle behind a large group for repairing algorithms, firstly proposed in (Nebel and Koehler 1995), can be described as action ordering preservation or, in other words, re-usage of parts from former plans. This very principle stands also behind the two multi-agent plan repairing algorithms proposed in (Komenda, Novák, and Pěchouček 2013). It is not intuitively clear what is a good strategy for reusing the original plan, moreover related to a particular planning domain. The experiments conducted in these sections provide several insights into this issue.

Used algorithms A battery of plan repairing algorithms was prepared to validate Hypothesis 3. We modify how and how much the algorithms reuse the original plan. Such modifications lead to a two-dimensional discrete space of different plan repairing algorithms, as depicted in Figure 5, representing a structure of the repaired plan.

Each of the nine diagrams in the figure describes a variation on a resulting plan repaired by one particular modification of the algorithm in the context of execution of the original plan. The execution starts with a world in the initial state s_0 and it is anticipated to continue with help of the original plan to the last state s_m, which is one of the goal states, i.e., $s_m \in S_G$. However during execution of an action following a state s_k, execution failed and the state of the world ends up not in the state s_{k+1}, but in a state s_f, out of the anticipated sequence of states and actions. To fulfill the goal, the agents use one of the plan repairing algorithms, which under the condition of perfect execution, would transform the world from s_f to a $s_m \in S_G$.

In Figure 5, there are two dimensions depicted. One of the dimensions represent the number of actions which has to be reused from beginning of the original plan as a prefix corresponding to fixation of the iteration parameter $f^* = f$. The other dimension represents number of actions reused as suffix of the final repairing plan, i.e., fixed iteration parameter $g^* = g$. In the presented scheme, $\mathcal{P}_{\text{pre}}$ from the Algorithm 1 is denoted as a blue line, $\mathcal{P}_{\text{suf}}$ as a green line and $\mathcal{P}^*$as a black thick arrow. Since both the dimensions

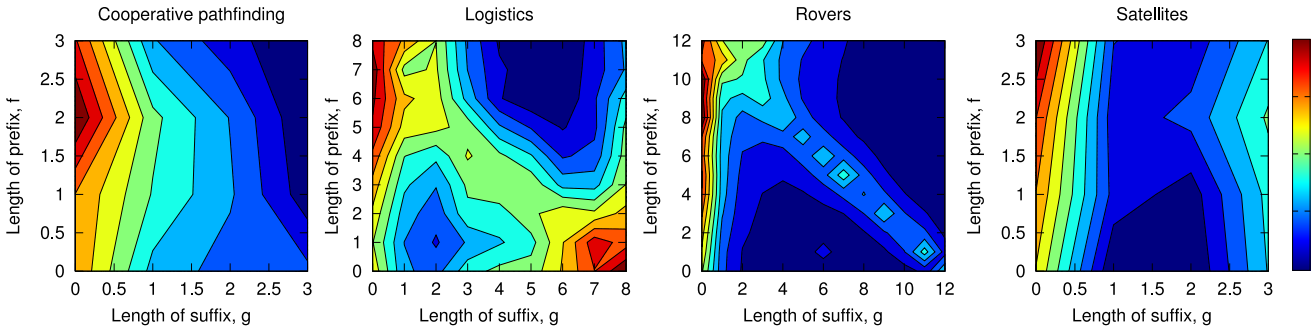

Figure 6: The maps present prefix (f on y-axis) vs. suffix (g on x-axis) preserving repairing algorithms by a success rate against replanning in the repair time for all domains with three agents and $p = 0.3$. Red color represents algorithms more often faster then replanning. The top-left to bottom-right diagonal represent algorithms neither overusing or underusing the original plan.

reuse the same original plan, the space is always a square with a side of the length m.

There are four extremes in the algorithm space. The algorithm at position $(0, 0)$ effectively degenerates from $\mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$ to $\mathcal{P}^*$. Such process correspond to replanning from the scratch. The algorithms at positions $(m, 0)$ and $(0, m)$ represent pure repairs $\mathcal{P}_{\text{pre}} \cdot \mathcal{P}^*$ and $\mathcal{P} \cdot \mathcal{P}_{\text{suf}}$ respectively. The last extreme at (m, m) represent an algorithm, which firstly uses the executable reminder of the original plan, then using a newly generated plan $\mathcal{P}^*$ returns to the anticipated state after execution of the failed action and than reuses the original plan again to get to the goal state, i.e., the algorithm generates a full overlap of the prefix and suffix plans.

Beside the extremes, also the $(0, m), (1, m-1), ..., (m-1, 1)$, $(m, 0)$ diagonal in the space is important from perspective of the ongoing discussion. All the algorithms lying on this diagonal re-use exactly all the actions of the original plan in the original order. Meaning, the original plan is neither overused nor underused. Formally, we define:

Definition 2 *(m-*normal plan repair*) Let $\Sigma = (\Pi, \mathcal{P}, s_{\mathcal{f}}, k)$ be a multi-agent plan repairing problem, then an algorithm* R *is a* m-normal plan repair*, iff* R *solves the problem Σ by a multi-agent plan $\mathcal{P}$ with decomposition $\mathcal{P}_{\text{pre}} \cdot \mathcal{P}^* \cdot \mathcal{P}_{\text{suf}}$ and at the same time $(|\mathcal{P}_{\text{pre}}|, |\mathcal{P}_{\text{suf}}|) \in \{(0, m), (1, m-1), ..., (m-1, 1), (m, 0)\}$.*

Results and Discussion To validate the third and last hypothesis, we used a randomized sampling of the algorithm space and searched for more successful algorithms lying on the m-normal repairing diagonal by the hypothesis. The results are present in Figure 6.

The sampling experimental process measured for each encountered repairing problem the computation time of the replanning algorithm. After this base-line measurement, a tested repairing algorithm was run with a bound on the computation time based on the replanning run-time. If the algorithm performed better, a cell in the result map was incremented by one. In effect, this process rendered the presented normalized results. During the experimental execution and plan repairing, we used different lengths of the original plan, i.e., the repair was done for various m. Therefore, the resulting maps depict a continuous space, as the results with higher and lower m values were merged into the most representative m value corresponding to the initial multi-agent plan generated.

As the maps show, the hypothesis clearly holds for coupled domains with longer plans (LOGISTICS, and ROVERS). In the coupled domain of COOPERATIVE PATHFINDING, the diagonal is also present, but because of shorter repaired plans, it degenerated considerably. In the experiment with SATELLITES, the diagonal is not present.

These results support Hypothesis 3 with an auxiliary observation, that the effect is decreasing as the coupling of the domain decreases.

Final remarks

Based on the experimental results, we can come up with a summary of heuristic approaches in form of simply usable advices decreasing computation and/or communication overheads during repairing of multi-agent plans. These advices can be used for various plan repairing approaches targeting systems with planning agents. The results were verified for plan repairing techniques utilizing preservation of the original plan and using an DisCSP-based multi-agent planner to fill prospective discontinuities in the repairing plan. The advices are:

1. *Prefer smaller numbers of involved agents in the plan repairing process.*
2. *Prefer prefix repairing techniques when repairing failures with long dependencies among different agents.*
3. *Prefer m-normal plan repairing algorithms.*

This work opens several interesting questions left for the future work. Most notably, how would another implementation of the underlying multi-agent planner affect the results and would it be possible to integrate principles from single-agent search effort estimation approaches, e.g., as in (Korf, Reid, and Edelkamp 2001) to provide more precise hints how to repair during the execution and repairing process.

Acknowledgments This work was supported by the U.S. Air Force EOARD grant no. FA8655-12-1-2096.

References

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov Decision Processes. *Math. Oper. Res.* 27(4):819–840.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of ICAPS*, 28–35.

Fu, J.; Ng, V.; Bastani, F. B.; and Yen, I.-L. 2011. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proceedings of IJCAI*, 1949–1954.

Komenda, A.; Novák, P.; and Pěchouček, M. 2012. Decentralized multi-agent plan repair in dynamic environments (Extended Abstract). In *Proceedings of AAMAS*, 1239–1240.

Komenda, A.; Novák, P.; and Pěchouček, M. 2013. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*. DOI: 10.1016/j.jnca.2012.12.011.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artif. Intell.* 129(1-2):199–218.

Levenshtein, V. I. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady* 10:707.

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence* 76(1-2):427–454.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS*, 1323–1330.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Int. Res.* 35(1):623–675.

Fast-Forward Heuristic for Multiagent Planning

Michal Štolba and **Antonín Komenda**
{stolba|komenda}@agents.fel.cvut.cz
Department of Computer Science and Engineering,
Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract

Use of heuristics in search-based domain-independent deterministic multiagent planning is as important as in classical planning. In this work we propose a formal and an algorithmic adaptation of a well-known heuristic Fast-Forward into multiagent planning. Such treatment is important as it solves challenges in decentralization of this and other heuristics based on relaxation of the original planning problem. Such decentralization enables global heuristic estimates to be computed without exposing local information. Additionally, since Fast-Forward heuristic is based on relaxed planning, we propose a multiagent approach for building factored relaxed planning graphs among the agents. We sketch proofs that the results of the distributed version of the algorithm gives the same results as the centralized version. Finally, we experimentally validate different distribution strategies of the heuristic estimate.

Introduction

In recent years the landscape of multiagent planning research has changed by Brafman and Domshlak's formal treatment and promising complexity results of domain-independent deterministic multiagent planning (DMAP) (Brafman and Domshlak 2008) represented as an extension of STRIPS for more agents. An important piece of the puzzle was a decomposition of a planning problem common for all the agents. In principle, the ideas behind relate to the research of planning problem factorization and utilization of such for more efficient solving of classical planning problems. Therefore even for cooperative agents, it is reasonable to hide parts of the information used during planning from other agents as this helps (in loosely coupled problems) the agents to focus only on their parts of the problem.

After this publication, the community started to design and implement first planners using the principles of DMAP described in the Brafman and Domshlak's paper. The first one from Nissim et al. (Nissim, Brafman, and Domshlak 2010) was built on distributed constraint satisfaction problem solver and a forward chaining planner. This approach precisely followed the ideas in (Brafman and Domshlak 2008), however exposed a couple of issues making the approach incomparable in efficiency with current state-of-the-art implementations of classical planners. One of the issues was bad scalability with growing length of the coordination part of the resulting plans. Improvement of scalability was proposed in (Nissim and Brafman 2012) by leaving the DisCSP+Planning approach and moving to a principle which is currently the most successful in classical planning—A* or variations on Best First Search (BFS) with highly informed automatically derived heuristics.

Since the motivation of (Nissim and Brafman 2012) was to propose an optimal planner (MA-A*), the heuristics used were LM-cut (Helmert and Domshlak 2009) with pathmax equation and merge-and-shrink (Helmert, Haslum, and Hoffmann 2007). In the distributed search approach, the heuristics were used only with local information of the respective agent, i.e., with its internal actions, its public actions and projections of other agents' public actions. In discussion of (Nissim and Brafman 2012), the authors state that *"the greatest practical challenge [...] is that of computing a global heuristic by a distributed system"*, which is precisely our focus in this work. According to our knowledge, there is no work proposing efficient planners for DMAP not focused on optimality of the resulting plans. In the field of classical planning, on the other hand, the best performing planners as Fast Downward and LAMA incorporate a fast, but suboptimal search algorithm using non-admissible heuristics.

In this work we propose a formal and algorithmic adaptation of a well-known relaxation heuristic Fast-Forward h^{FF} (Hoffmann and Nebel 2001) into multiagent planning. We argue that such treatment is important as it demonstrates algorithmic challenges in decentralization of computation of h^{FF} and other related heuristics. Additionally, since the h^{FF} heuristic is based on relaxed planning, we propose a multiagent (MA) approach for building factored relaxed planning graphs among the agents. We sketch proofs that the results of the distributed version of the algorithm gives the same results as the centralized version. Finally, we experimentally validate two distribution strategies of the heuristic estimate against the local estimate.

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Multiagent Planning

We consider a number of *cooperative* and *coordinated* agents featuring distinct sets of capabilities (actions), which concurrently plan and execute their local plans in order to achieve a joint goal. The world wherein the agents act is *classical* and the actions are *deterministic*. The following formal preliminaries compactly restate the MA-STRIPS problem (Brafman and Domshlak 2008) required for the following sections.

A MA-STRIPS planning problem is a quadruple $\Pi = \langle \mathcal{L}, \mathcal{A}, s_0, S_g \rangle$, where $\mathcal{L}$ is a set of propositions, $\mathcal{A}$ is a set of *agents* $\alpha_1, \ldots, \alpha_{|\mathcal{A}|}$, s_0 is an initial state and S_g is a set of goal states. A *state* $s \subseteq \mathcal{L}$ is a set of atoms from a finite set of propositions $\mathcal{L} = \{p_1, \ldots, p_m\}$ which holds in s. An *action* an agent can perform is a tuple $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, where a is a unique action label and $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)$ respectively denote the sets of preconditions, add effects and delete effects of a, taken from $\mathcal{L}$. *Act* denotes the set of all actions in the multiagent planning problem Π, i.e., $Act = \bigcup_{\alpha \in \mathcal{A}} \alpha$.

An *agent* $\alpha = \{a_1, \ldots, a_n\}$ is characterized precisely by its capabilities, a finite repertoire of actions $a_i \in Act$ it can preform in the environment. MA-STRIPS problems distinguish between the *public* and *internal* facts and actions. Let $\mathsf{atoms}(a) = \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ and similarly $\mathsf{atoms}(\alpha) = \bigcup_{a \in \alpha} \mathsf{atoms}(a)$. An α-internal and public subset of all facts $\mathcal{L}$ will be denoted as $\mathcal{L}^{\alpha-\text{int}}$ and $\mathcal{L}^{\text{pub}}$ respectively, where $\mathcal{L}^{\alpha-\text{int}} = \mathsf{atoms}(\alpha) \setminus \bigcup_{\beta \in \mathcal{A} \setminus \alpha} \mathsf{atoms}(\beta)$ and $\mathcal{L}^{\text{pub}} = \mathsf{atoms}(\alpha) \setminus \mathcal{L}^{\alpha-\text{int}}$. Facts relevant only for one agent α are denoted as $\mathcal{L}^{\alpha} = \mathcal{L}^{\alpha-\text{int}} \cup \mathcal{L}^{\text{pub}}$ and a *projection* of a state s^{α} to an agent α is a subset of a global state s containing only public facts and α-internal facts, formally $s^{\alpha} = s \cap \mathcal{L}^{\alpha}$. The set of *public actions* of agent α is defined as $\alpha^{\text{pub}} = \{a \mid a \in \alpha, \mathsf{atoms}(a) \cap \mathcal{L}^{\text{pub}} \neq \emptyset\}$ and *internal actions* as $\alpha^{\text{int}} = \alpha \setminus \alpha^{\text{pub}}$. The symbol a^{α} will denote a projection of action $a \in \beta, \beta \neq \alpha$ for agent α, i.e., action stripped of all other agents' propositions, formally $\mathsf{atoms}(a^{\alpha}) = \mathsf{atoms}(a) \cap \mathcal{L}^{\alpha}$.

Note that all actions of an agent α uses only agent's facts, formally $\forall a \in \alpha : \mathsf{atoms}(a) \subseteq \mathcal{L}^{\alpha}$ by definition in (Brafman and Domshlak 2008). The goal set S_G of a multiagent planning problem will be treated as public (Nissim and Brafman 2012), therefore all goal-achieving actions are public. In the following sections, as an algorithm for multiagent planning, we will assume the MA-A* from (Nissim and Brafman 2012), but with a novel distribution of the h^{FF} heuristic.

As a running example, we will use a simple logistics problem (see Figure 1) in a multiagent setting. There are two cities each with two locations A, B and C, D and one package p. A and D represent depots and B,C airports. Three agents represent two cargo trucks t_1,t_2 (moving only within the cities) and one airplane a (moving only between airports B and C). The goal is to transport the package from depot A to the other depot D.

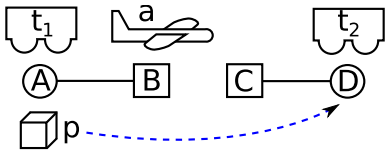

Figure 1: A running example is an instance of LOGISTICS problem with three agents and one package.

Agent Relaxed Planning Graph

Relaxation is a way of simplifying a problem by removing some constraints. In planning, a relaxation is typically obtained by removing delete effect of actions. Solution of such relaxed planning problem is a relaxed plan, which can be used to estimate the cost of a plan in the original problem, e.g., the Fast-Forward heuristic estimation is based on the length of the relaxed plan. A classical technique for finding the relaxed plan is to build a Relaxed Planning Graph (RPG). RPG is a graph representing the reachability of facts and applicability of actions in the relaxed problem.

Building distributed planning graphs (not relaxed) was studied by (Pellier 2010), focusing on distribution of the Graphplan algorithm. Relaxed MA Planning Graphs were recently studied by (Torreño, Onaindia, and Sapena 2012), but in the area of planning with incomplete information and fluent cost estimation.

To obtain a more informed global heuristic estimate in a MA planning problem using the estimation based on a RPG, the RPG has to be decentralized. In this work, we propose a distributed global RPG in form of a set of distinct Agent RPGs. Such Agent RPG (ARPG) contains only facts of its owner agent. The initial state is projection for that agent and since the goals are treated as public, all agents have complete goals in their ARPGs. The usage of actions is straightforward in case of owner agent's internal and public actions which are used equally as in a classical RPG. Additionally, the Agent RPGs are extended by projections of other agents' public actions which were reachable by their particular owners. This extension enables the agents to take other agents' capabilities into account, but only at the time points, where their owners are able to reach them. Similarly to relaxed problems in STRIPS, we define a relaxed multiagent planning problem in MA-STRIPS as a problem stripped of delete effects in all actions of all agents:

Definition 1. An *agent relaxed planning graph (ARPG)* is a directed, labeled and layered graph $\mathcal{R}'^{\alpha} = (P' \cup A', E')$ of one particular agent α for a relaxed multiagent planning task. Let $\Pi = (\mathcal{L}, \mathcal{A}, s_0, S_G)$ be a MA planning task, then a relaxed MA planning task $\Pi' = \langle \mathcal{L}, \mathcal{A}', s_0, S_G \rangle$ contains an altered set of agents $\mathcal{A}'$, s.t., $\forall \alpha \in \mathcal{A}$ and $\alpha = \{a_1, \ldots, a_{|\mathcal{A}|}\}$ there exist a relaxed agent $\alpha' = \{a'_1, \ldots, a'_{|\mathcal{A}|}\}$ and all its actions are relaxed versions of the regular actions $a'_i = \langle \mathsf{pre}(a_i), \mathsf{add}(a_i), \emptyset \rangle$. As in RPG, the nodes of the graph represent proposi-

tions P' and actions A'. The arcs E' represent linkup of propositions and actions.

In the rest of the paper, the discussion will be only about relaxed structures, therefore we will omit the prime signs, which are by convention used to denote relaxed structures.

ARPGs stem from the classical RPGs, therefore an i-th proposition layer and action layer will be denoted as P_i and A_i respectively. The layers alternate, so that $(P_0, A_0, P_1, A_1, \ldots, A_{n-1}, P_n)$ and all layers $P_i \subseteq P$ and all layers $A_i \subseteq A$. The first proposition layer P_0 contains nodes labeled by propositions of the agent's projection of the initial state, formally

$$P_0 = \{p | p \in s_0^\alpha\}.$$

Each action layer contains action nodes for all applicable relaxed actions of the agent α in a state represented by the previous fact layer and external projections of other agents' public actions reachable in the same layer

$$A_i \quad = \quad \{a | a \in \alpha, \mathsf{pre}(a) \subseteq P_i\} \cup \bigcup_{\beta \in \mathcal{A}, \beta \neq \alpha} \{b^\alpha | b \in P_i^\beta\}.$$

In all successive fact layers, the nodes copy the previous fact layer according to the frame axiom and transforms the facts by actions in the previous action layer, since for all relaxed actions $\mathsf{del}(a) = \emptyset$, we can write

$$P_i = P_{i-1} \cup \{p | p \in \mathsf{add}(a), a \in A_{i-1}\}.$$

At least one of the following terminating conditions has to hold for the last fact layer P_n:

- the last fact layer fulfills the goal $S_G \subseteq P_n$,
- or $P_n = P_{n-1}$, meaning there are no additional actions which can extend further fact layers (a fixed-point).

The arcs in ARPG represent applicability and application of actions in relaxed states. We can split the arcs between two fact layers P_i and P_{i+1} into three groups. The first one contains arcs among facts of layer P_i and preconditions of actions in a layer A_i. The second one contains relation between effects of actions and next induced fact layer P_{i+1}. Additionally, there are arcs for all facts from a previous layer effectively representing the frame axioms of the closed world assumption. Formally,

$$\begin{aligned} E_i^{\mathrm{pre}} &= \{(p_i, a_i) | p_i \in \mathsf{pre}(a_i), a_i \in A_i\}, \\ E_i^{\mathrm{add}} &= \{(a_i, p_{i+1}) | a_i \in A_i, p_{i+1} \in \mathsf{add}(a_i)\}, \\ E_i^{\mathrm{frm}} &= \{(p_i, p_{i+1}) | p_i \in P_i, p_{i+1} \in P_{i+1}, p_i = p_{i+1}\} \end{aligned}$$

and $E_i = E_i^{\mathrm{pre}} \cup E_i^{\mathrm{add}} \cup E_i^{\mathrm{frm}}$. Now we will provide an algorithm for distributed building of ARPGs.

Algorithm The algorithm starts with each agent building an ARPG using only its own internal and public actions. An iterative process is then initiated, in which the agents exchange information about their *public* actions and extends their ARPGs with projected public actions of other agents. The algorithm terminates when the goal (or a fixed-point) is globally reached and there are no more messages to process.

Algorithm 1 Distributed build of Agent Relaxed Planning Graphs

Input: An agent's factor of the relaxed MA planning problem $\Pi^\alpha = \langle \mathcal{L}^\alpha, \alpha, s_0^\alpha, S_G \rangle$.
Output: Agent Relaxed Planning Graph $\mathcal{R}$ for α.

1: **init():**
2: $\mathcal{R} \leftarrow P_0 = \{p | p \in s_0^\alpha\}$
3: $\mathcal{R} \leftarrow$build-RPG$(\mathcal{R}, \alpha, S_G)$
4: $\mathcal{S} \leftarrow$map$[\alpha^{\mathrm{pub}}, integer]$
5: ack-count$\leftarrow$idle-count$\leftarrow$0
6: *check*()

7: **check():**
8: **for all** $a \in A_{n-1}$ s.t. a is public **do**
9: **if** $a \notin \mathcal{S}$ **or** $\mathcal{S}[a] >$ earliest layer of appearance of a in $\mathcal{R}$ **then**
10: $\mathcal{S}[a] \leftarrow$earliest appearance of a in $\mathcal{R}$
11: ack-count$\leftarrow$ack-count$+|\mathcal{A}|$
12: $\forall \beta \in \mathcal{A} \setminus \alpha$: *send*(ext-a$[a^\beta, \mathcal{S}[a]]$, to β)
13: **end if**
14: **end for**

15: **receive**(ext-a$[a^\alpha, i \in \mathbb{N}]$, from $\beta \in \mathcal{A} \setminus \alpha$):
16: $\forall \alpha \in \mathcal{A}$: *send*(not-idle, to α)
17: *send*(ack, to β)
18: $\mathcal{R} \leftarrow$extend-RPG$(\mathcal{R}, [a^\alpha, i])$
19: $\mathcal{R} \leftarrow$build-RPG$(\mathcal{R}, \alpha, S_G)$
20: *check*()
21: *goal-reached*()

22: **receive**(ack):
23: ack-count $\leftarrow$ ack-count - 1
24: *goal-reached*()

25: **receive**(idle):
26: idle-count $\leftarrow$ idle-count + 1
27: **if** idle-count $= |\mathcal{A}|$ **then**
28: **return** $\mathcal{R}$
29: **end if**

30: **receive**(not-idle):
31: idle-count $\leftarrow$ idle-count - 1

32: **goal-reached():**
33: **if** $S_G \in \mathcal{R}$ **and** ack-count $= 0$ **and** message queue is empty **then**
34: $\forall \alpha \in \mathcal{A}$: *send*(idle, to α)
35: **end if**

The pseudo-code of the algorithm is given in Algorithm 1 in an event-driven fashion. The events can be caused either by receiving a message or internally by the algorithm itself. We assume that messages sent from one agent arrive in the same order as they were

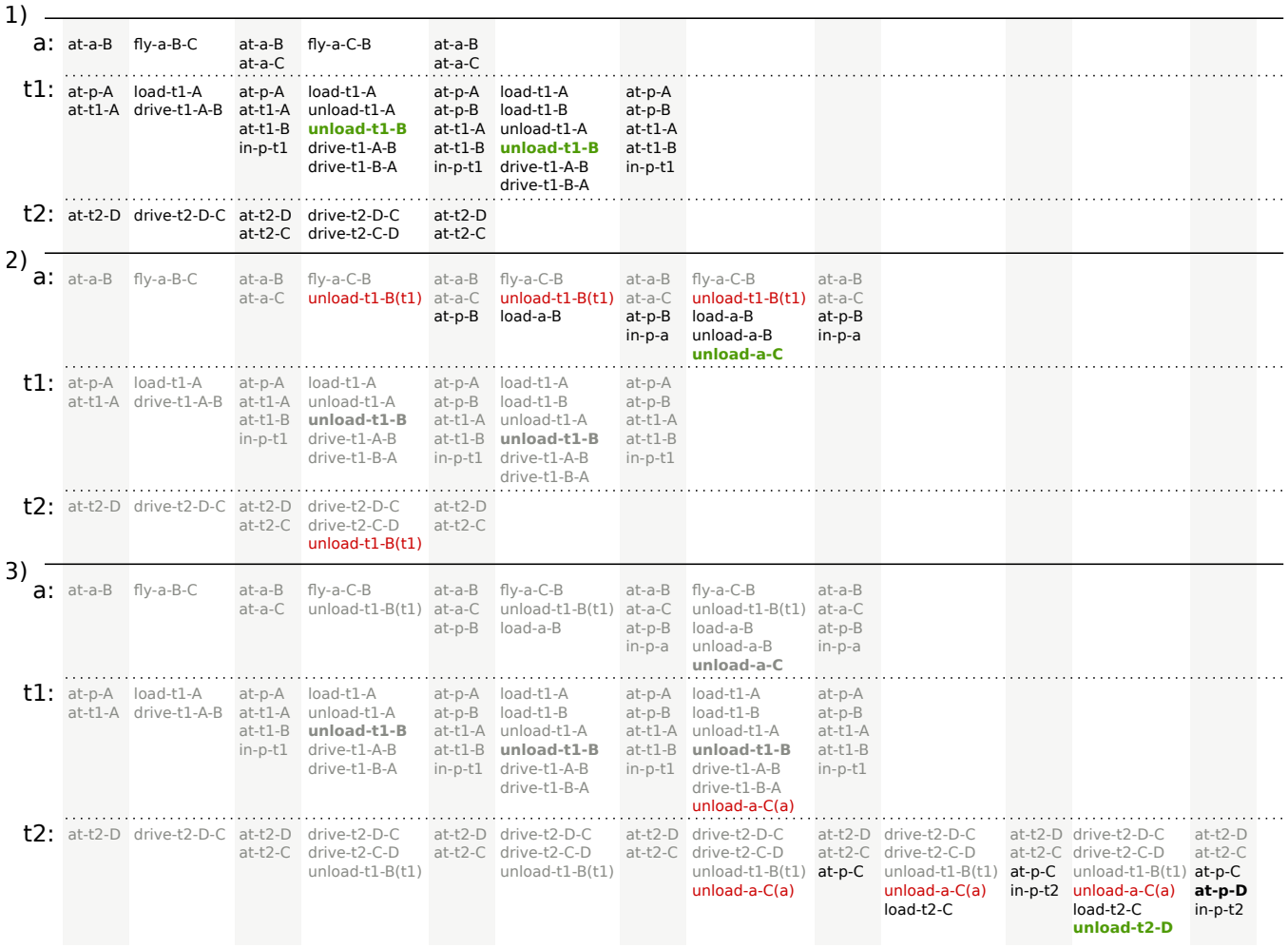

Figure 2: Distributed building of Agent Relaxed Planning Graphs decomposed into iterations.

sent, but we assume no ordering between messages sent from different agents. We will now explain each event handling routine.

In the **init** phase a Relaxed Planning Graph $\mathcal{R}$ is built using only agent's own actions by build-RPG method from the initial state projection s_0^α. A map $\mathcal{S}$ used to store the *earliest layer of appearance*[1] of the agent's public actions is initialized along with other supporting data structures used for synchronized termination of the algorithm. After the initialization phase, reaching of the goal (or a fixed-point) is checked, and if positive, all agents are informed that the agent is idle now. Next, the executed **check** procedure is responsible for checking whether $\mathcal{R}$ contains any public actions. If so, each action is sent to all other agents $\beta \in \mathcal{A} \setminus \alpha$ as a projection a^β with its earliest layer of appearance, unless it was already broadcasted with equal or lower number of layer (this can happen in future **check** calls).

[1]Earliest layer of appearance of an action a in (A)RPG is the first action layer, where a is applicable.

In the next part of the algorithm, there are four message handling **receive** procedures. The first one *ext-a* is executed when a projection of other agent's public action is received. After sending control messages, the action is integrated into $\mathcal{R}$ on the i-th layer by extend-RPG method and the change is propagated by build-RPG, so that all actions newly applicable in the following layers are applied accordingly. Then the built ARPG is checked, whether new public actions (and public actions newly applicable on earlier layers) are reachable and whether the goal or the fixed-point was reached. The last three **receive** procedures maintain the control information needed for distributed termination detection (Mattern 1987). The *acks* counter keeps track of number of sent external actions and postpones termination until all sent actions are processed. If an *idle* message is received, there are no pending *acks* and the number of idle agents is equal to $|\mathcal{A}|$, the algorithm terminates and the resulting ARPG $\mathcal{R}$ is returned. Since *not-idle* and *ack* messages are sent in this particular order (lines 17 and 18) and the messages from one agent

are presumed to keep ordering, the algorithm terminates synchronously when all external actions are processed and no messages are pending.

In Figure 2, the Algorithm 1 is applied on the running example depicted in Figure 1. Although the algorithm is running asynchronously, we can decompose it for clarity into several iterations. In the first iteration, the ARPGs are built using only the actions of the respective agents a, t_1 and t_2 (airplane and two trucks). Notice the bold green action unload-t1-B, which is a public action of the truck t_1, can be applied thanks to the initial position of the package. In the next iteration, projection of the public action is broadcasted and received by other agents. Upon receiving, their ARPGs are updated, which for the airplane means that the ARPG is expanded with further layers. Another public action unload-a-C is applied and therefore broadcasted. In the third iteration, the projection of the airplane's unload action is added to the ARPGs of the trucks. For truck t_1 it has no effect, but it allows truck t_2 to expand the ARPG and reach goal at-p-D. Notice, that when the projected unload-a-C(a) was received by truck t_2, its ARPG was first extended to have enough layers for the action to be added to the correct layer.

Although not shown in Figure 2, the algorithm would continue with one more iteration after broadcasting the public action reached by truck t_2, resulting in all agents having ARPGs with the same number of layers and all having reached the goal. Additionally, the algorithm does not have to terminate when the goal is reached, but can continue until the fixed-point, which can be desirable in some situations and which is also the case when the goal is not reachable.

Proof sketch In this section, we will sketch a proof showing, that the Agent Relaxed Planning Graphs built by Algorithm 1 are *compatible* with a global RPG, meaning that they contain the same actions (with respect to projections) and that the actions are in the same layers. We will use this proven theorem further in a proof of equality of the centralized FF and multiagent FF (MAFF) heuristics.

Firstly, we will formally define the concept of compatibility, then we will show that single iteration of the algorithm does not violate the compatibility, and finally we will show that the algorithm terminates, i.e., the resulting ARPGs are compatible with a centrally built RPG and that no actions are missing or are superfluous.

Let $\Pi = \langle \mathcal{L}, \mathcal{A}, s_0, S_G \rangle$ be a relaxed MA planning task, $\mathcal{R}^\alpha$ be an Agent Relaxed Planning Graph for agent α built from Π using Algorithm 1, having alternating layers $(P_0^\alpha, A_0^\alpha, P_1^\alpha, A_1^\alpha, \ldots, A_{n-1}^\alpha, P_n^\alpha)$ and let $A^\alpha = \bigcup_{i \in \langle 0, n-1 \rangle} A_i^\alpha$ and $A^{\mathcal{A}} = \bigcup_{\alpha \in \mathcal{A}} A^\alpha$. Let $\hat{\Pi} = \langle \mathcal{L}, Act, s_0, S_G \rangle$ be a classical relaxed planning task, $\hat{\mathcal{R}}$ be a classical Relaxed Planning Graph built from $\hat{\Pi}$ having alternating layers $(\hat{P}_0, \hat{A}_0, \hat{P}_1, \hat{A}_1, \ldots, \hat{A}_{n-1}, \hat{P}_n)$ and let $\hat{A} = \bigcup_{i \in \langle 0, n-1 \rangle} \hat{A}_i$. Note that from the monotonicity of (A)RPGs follows $\forall i < j : P_i \subseteq P_j$ and $\forall i < j : A_i \subseteq A_j$.

Definition 2. Let $a \rhd A_i$ denote that an action a is *first applicable* in layer A_i (formally $\mathsf{pre}(a) \subseteq P_i \wedge \mathsf{pre}(a) \nsubseteq P_{i-1}$) regardless of whether the underlying structure is RPG or ARPG.

Definition 3. We define that a set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ is *compatible* with a RPG $\hat{\mathcal{R}}$ iff for each action $a \in A^{\mathcal{A}}$ for which $a \rhd \hat{A}_i$ holds the following:

1) If a is an *internal action* of agent α, then $a \rhd A_i^\alpha$.

2) If a is a *public action* of agent α, then $a \rhd A_i^\alpha$ and $\forall \beta \in \mathcal{A} \setminus \alpha : a^\beta \rhd A_i^\beta$, where a^β is the projection of action a for agent β.

Lemma 4. *A set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ compatible with a RPG $\hat{\mathcal{R}}$ stays compatible with $\hat{\mathcal{R}}$ after application of build-RPG by agent α and successive extend-RPG by all other agents.*

Proof. Let us have a RPG $\hat{\mathcal{R}}$ built from a relaxed planning task $\hat{\Pi}$ and a set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ being built from relaxed MA planning task Π using Algorithm 1. The symbol $A^{\mathcal{A}}$ denotes all actions applied in the algorithm so far and let α^{proj} be the set of projected actions received by agent α so far. Now, agent α applies build-RPG, so that $\mathcal{R}^\alpha$ is updated by $A_i^\alpha = A_i^\alpha \cup \{a | a \in \alpha \cup \alpha^{\text{proj}} : \mathsf{pre}(a) \subseteq P_i^\alpha\}$ (and accordingly P_{i+1}^α), for each layer A_i^α. Let us assume, there exists an extra action $a \in \alpha$ which was newly applied ($a \notin A^{\mathcal{A}}$) and for which $a \rhd \hat{A}_i \wedge a \not\rhd A_i^\alpha$. We can also assume WLOG, that a is first such action (in terms of layer of appearance).

From definition of $a \rhd \hat{A}_i$, where $\mathsf{pre}(a) \subseteq \hat{P}_i \wedge \mathsf{pre}(a) \nsubseteq \hat{P}_{i-1}$ follows that $\mathsf{pre}(a) \subseteq \hat{P}_0 \cup \{p | b \in \hat{A}_{i-1}, p \in \mathsf{add}(b)\}$ and $\mathsf{pre}(a) \nsubseteq \hat{P}_0 \cup \{p | b \in \hat{A}_{i-2}, p \in \mathsf{add}(b)\}$. Because a is first action for which $a \rhd \hat{A}_i \wedge a \not\rhd A_i^\alpha$, for all actions $b \in \alpha$, for which holds $b \rhd \hat{A}_k$ where $k < i$, holds also $b \rhd A_k^\alpha$. Therefore $A_k^\alpha = \hat{A}_k \cap \alpha$ and $P_k^\alpha = \hat{P}_k \cap \mathsf{atoms}(\alpha)$ for all $k < i$ and therefore $\mathsf{pre}(a) \subseteq P_i^\alpha \wedge \mathsf{pre}(a) \nsubseteq P_{i-1}^\alpha$, which means $a \rhd A_i^\alpha$ and that is a contradiction. Now, we can assign $A^{\mathcal{A}} \leftarrow A^{\mathcal{A}} \cup \{a\}$ and repeat the former step.

After broadcasting projections of the newly applied public actions and calling extend-RPG by all other agents, we can show that the second part of Definition 3 also holds. Let a^α be the projection of an action $a \in \beta$ which is broadcasted first. If there exists some i for which $a \rhd \hat{A}_i$ then $\mathsf{pre}(a) \subseteq \hat{P}_i$ and for the projection a^α holds $\mathsf{pre}(a) \subseteq \hat{P}_i \cap \mathcal{L}^{\text{pub}}$. Because for all actions $b \in \hat{A}_{i-1}$ the lemma holds, if b is public, $b^\alpha \rhd A_{i-1}^\alpha$. Because a^α is a projection, $\mathsf{pre}(a^\alpha) \subseteq P_0^\alpha \cup \bigcup_{b^\alpha \in A_{i-1}^\alpha} \mathsf{add}(b)$ and therefore a^α is applicable in layer i (and subsequent layers), which means that the extend-RPG ensures that $a^\alpha \rhd A_i^\alpha$. □

Theorem 5. *When Algorithm 1 terminates, resulting set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ built from Π is compatible with the RPG $\hat{\mathcal{R}}$ built from $\hat{\Pi}$ and there are no additional actions, i.e., $\hat{A} = A^{\mathcal{A}}$. Each public action in $\hat{A}$ has its projected counterparts in $A^{\mathcal{A}}$ and vice versa, i.e., for each public action $a \in \hat{A}$ such that $a \in \alpha$ for some agent α exists projected action a^β for each agent $\beta \in \mathcal{A} \setminus \alpha$ and $a^\beta \in A^{\mathcal{A}}$. There is no projected action $a^\beta \in A^{\mathcal{A}}$ for which there is no original action $a \in \hat{A}$.*

Proof. We will now sketch an induction which shows the compatibility in Theorem 5, based on the Lemma 4. For the initial step of the induction we take all ARPGs containing only the first fact layer P_0^α which is trivially compatible because $A^{\mathcal{A}} = \emptyset$. The induction step is covered by Lemma 4, because each step of the Algorithm 1 can be decomposed as an application of build-RPG and, if there are any applied public actions, broadcasting their projections and application of extend-RPG by all other agents. Even though the algorithm is running asynchronously, Lemma 4 holds because of the monotonicity of (A)RPGs.

Termination of the algorithm follows from the termination of classical RPG, either the algorithm reaches goal or a fixed-point, where no more actions are added. Similarly to classical RPG, building of ARPGs is monotonic, which means that the facts and actions can only be added and because the set of actions is finite, there must be a point where no more actions can be added and the algorithm terminates. The detection of such situation is more complicated in the distributed setting and is described thoroughly in the algorithm section.

The last statement we are about to show is that there are no additional actions, i.e., $\hat{A} = A^{\mathcal{A}}$ (irrespective of the projections of public actions) and that each public action in $\hat{A}$ has its projection in $A^{\mathcal{A}}$ and vice versa. Let us assume, that $\exists a \in \hat{A}$ such that $a \notin A^{\mathcal{A}}$, let us also assume, WLOG, that a is such action appearing in the earliest layer in $\hat{\mathcal{R}}$, say $\hat{A}_i$, and that $a \in \alpha$ for some agent α. We know, that exists minimal $A_{\text{pre}} \subseteq \hat{A}_{i-1}$ such that $\mathsf{pre}(a) \subseteq \{p | b \in A_{\text{pre}}, p \in \mathsf{add}(b)\}$, because $\mathsf{pre}(a) \subseteq \mathcal{L}^{\alpha-\text{int}} \cup \mathcal{L}^{\text{pub}}$, for all actions $b \in A_{pre}$ either $b \in \alpha$ or b is public. Because we assumed, that $A_{\text{pre}} \subseteq \hat{A}$, for each $b \in A_{\text{pre}}$, if $b \in \alpha$ then $b \in A_{i-1}^\alpha$ and if $b \notin \alpha$ then b is public, therefore $b^\alpha \in A_{i-1}^\alpha$. From the said $\mathsf{pre}(a) \subseteq \{p | b \in A_{i-1}^\alpha, p \in \mathsf{add}(b)\}$, which means that a is applicable in A_i^α and therefore a must be applied by the algorithm. If we continue with next such action we end up with $\hat{A} \subseteq A^{\mathcal{A}}$.

Now, we will show that $\hat{A} \supseteq A^{\mathcal{A}}$. Let us assume, that $\exists a \in A^{\mathcal{A}}$ such that $a \notin \hat{A}$ and that a is first such action. Similarly to the previous situation, if a is applicable in some layer A_i^α then there is some set of actions $A_{\text{pre}} \subseteq A_{i-1}^\alpha$, which contains all actions providing preconditions of a. Since all actions $b \in A_{\text{pre}}$ are also in $\hat{A}$, a is applicable in $\hat{A}_i$ and therefore must be applied.

From $\hat{A} \subseteq A^{\mathcal{A}}$ and $\hat{A} \supseteq A^{\mathcal{A}}$ follows that $\hat{A} = A^{\mathcal{A}}$. We have already shown that public actions have their projected counterparts. The only remaining part to show is that there are no projected actions in $A^{\mathcal{A}}$ without their respective original actions in $\hat{A}$. This clearly follows from the algorithm itself, because all projected actions are created only when a public action is added to some A_i^α by agent α and as shown before, such action would also be added to $\hat{A}_i$. □

Multiagent FF Heuristic

With the help of ARPGs, the Fast-Forward heuristic estimate can be straightforwardly adapted to a multiagent setting. We will denoted such heuristic as h^{MAFF}. The multiagent (MA) relaxed plan backing the h^{MAFF} estimate can be in general spread over all ARPGs of the agents in the team as illustrated in Figure 3. The most left achieving actions has to be considered from all agents. In the case of projected public actions, the owner agent has to define part of the the relaxed plan, possibly using his internal actions, to achieve the internal facts of the provided public action. Additionally, the relaxed plan has to share public actions which are required by more agents at the same layers. The private parts of the relaxed plan provided by the other agents can be described by place-holding actions and therefore no private information of the other agents has to be revealed. The final heuristic estimate is the count of actions of the MA relaxed plan.

Definition. Let a MA relaxed plan π be a solution of a MA relaxed problem $\Pi = \langle \mathcal{L}, \mathcal{A}, s, S_G \rangle$, where s is the state, we are estimating the cost for, then $|\pi| = h(s)$ is the *multiagent relaxation heuristic estimate.*

Similarly to the relaxation heuristic estimate h^{FF}, we restrain π for h^{MAFF} according to h^{FF}. A particular π is defined using ARPGs $\mathcal{R}^\alpha = (P \cup A, E)$ of all agents $\alpha \in \mathcal{A}$ built for the state s. From the right (meaning as in Figure 3), the relaxed plan π contains minimal set of actions $A_m^* \subseteq A_m$, achieving the goal facts. The action layer A_m contain actions of all agents in layer m (ignoring projections of actions, since the respective original actions are also included in the same layer). If there is a frame arc $(p_{m-1}, p_m) \in E_m^{\text{frm}}$ of such facts, i.e., $p_m \in S_G$, the fact p_m does not need an explicit achieving action from this particular layer as it will be achieved by an action from an earlier (more left) layer. This principle effectively selects the most-left achievers of a fact as proposed by FF heuristic. The action set A_m^* induces next set of facts across all agents

$$S_m = \{p | p \in \mathsf{pre}(a) : a \in A_m^*\},$$

which has to be achieved by actions from previous action layer A_{m-1} and so on until the action layer A_0 is reached, where all the actions have their preconditions satisfied by the initial state in P_0.

Notice that since this definition works across all ARPGs of all agents, the resulting π may contain actions of different agents.

Figure 3: Multiagent Relaxed Plan

We can compute $h^{\mathrm{MAFF}}(s)$ by first building the set of ARPGs $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ for relaxed MA planning problem $\Pi = \langle \mathcal{L}, \mathcal{A}, s, S_G \rangle$ using Algorithm 1, then simultaneously extracting relaxed plans π_α for each agent using Algorithm 2 and finally summing the lengths of the resulting relaxed plans, excluding projections of other agent's public actions.

Theorem 6. *Let $\pi_\alpha \cap \alpha$ be the computed relaxed plan of agent α restricted only to the agent's actions (excluding all projections of other agent's public actions), then $h^{\mathrm{MAFF}}(s) = \sum_{\alpha \in \mathcal{A}} |\pi_\alpha \cap \alpha| = h^{\mathrm{FF}}(s)$.*

Proof. The fact that $h^{\mathrm{MAFF}}(s) = h^{\mathrm{FF}}(s)$ follows from the previously shown compatibility of the RPG $\hat{\mathcal{R}}$ built for relaxed planning problem $\hat{\Pi}$ and the set of ARPGs built from $R = \{\mathcal{R}^\alpha | \alpha \in \mathcal{A}\}$ for relaxed MA planning problem Π. We first extract relaxed plans $\hat{\pi}$ for $\hat{\Pi}$ and $\{\pi_\alpha | \alpha \in \mathcal{A}\}$ for Π by effectively choosing first achievers of goal facts and of preconditions of previously chosen achievers. It is clear that for some fact p we choose an action a s.t. $p \in \mathsf{add}(a)$ only if $a \rhd A_i$ for some layer A_i and there is no action b s.t. $p \in \mathsf{add}(b)$ and $b \rhd A_j$ for some $j < i$. Because of the compatibility of the RPG and ARPGs, we choose exactly the same actions (and their projections, which are then omitted) for $\hat{\pi}'$ and for $\{\pi'_\alpha | \alpha \in \mathcal{A}'\}$, which means that $|\hat{\pi}'| = \sum_{\alpha \in \mathcal{A}'} |\pi'_\alpha \cap \alpha|$ and therefore $h^{\mathrm{MAFF}}(s) = h^{\mathrm{FF}}(s)$. □

Experiments

The experiments were conducted on an implementation of satisficing version of MA-A* (Nissim and Brafman 2012) with various relaxation heuristic estimates[2]. The algorithm begins with a centralized factorization and a reachability analysis of a centralized planning problem. After the factorization, the agents are started receiving its factors of the problem as an input. The agents run in parallel, each one in its own thread and the messages are delivered by an additional asynchronous messaging thread. Each agent uses an event queue to serialize the computation and reactions to incoming messages. If an agent finds a sound plan (with parts from other agents) it prints it and stops the distributed process.

Since the algorithm is asynchronous, the runs are non-deterministic. Therefore we conducted each experimental run as ten measurements. Each measurement was limited to 8GB of memory for the Java Virtual Machine and to 10 minutes of runtime. Each measurement was run on 8-core processor at 3.6GHz separately. The results from the measurements were averaged.

[2]Technically the implementation is not A* as the heuristics are not admissible, therefore the used algorithm is precisely MA-BestFirstSearch.

Algorithm 2 Distributed extraction of h^{MAFF}

Input: ARPG $\mathcal{R}$ for state s, having layers $(P_0, A_0, P_1, A_1, \ldots, A_{n-1}, P_n)$ and goal $S_G \subseteq \mathcal{L}$.
Output: Relaxed plan R from s to S_G.

```
1:  P ← S_G
2:  R ← ∅
3:  for i = n − 1; i > 0; i ← i − 1 do
4:    P' ← ∅
5:    for p ∈ P do
6:      if p ∉ P_i then
7:        a ← a ∈ A_i, such that p ∈ add(a)
8:        R ← R ∪ {a}
9:        P' ← P' ∪ pre(a), P ← P\{p}
10:       if a is projected action of agent α then
11:         request R^α for s, goal pre(a^orig) from α
12:         R ← R ∪ R^α
13:       end if
14:     end if
15:   end for
16:   P ← P ∪ P'
17: end for
18: return R
```

We used five planning domains, four originating in the single-agent IPC planning benchmarks. Similarly to the evaluation of the algorithms in (Nissim, Brafman, and Domshlak 2010), we chose domains which are straightforwardly modifiable to the multiagent setting: LOGISTICS (similar to the running example, but with more agents), LINEAR LOGISTICS (one package has to be transported stepwise by all agents), ROVERS, and SATELLITES. As in (Komenda, Novák, and Pěchouček 2013), we have extended the set of IPC-based domains by a coordination domain COOPERATIVE PATHFINDING, in which robots on a grid are tasked to switch their positions not colliding with each other. We tested following distribution strategies of FF heuristic estimation:

- h^{FF}_α using only locally built RPGs (including projections of public actions) and local estimation of Fast-Forward heuristic, as proposed in (Nissim and Brafman 2012).
- h^{MAFF} using distributed ARPGs based on Algorithm 1 and distributed extraction of FF heuristic as described in Algorithm 2.
- Lazy h^{MAFF} using only locally built RPGs (including projections of public actions) and distributed extrac-

			h^{FF}_{α}				h^{MAFF}						Lazy h^{MAFF}					
	$\lvert\mathcal{A}\rvert$	l^*	$t[s]$	v	c_{s}	l	$t[s]$	v	c_{s}	c_{r}	c_{h}	l	$t[s]$	v	c_{s}	c_{r}	c_{h}	l
CP	3	6	0.4	99	97	6	6.3	168	166	6.6k	39	6	0.7	117	115	72	40	6
	5	16	88	25k	25k	16	–	–	–	–	–	–	7.6	1.8k	1.8k	120	145	18
	7	24	–	–	–	–	–	–	–	–	–	–	323	52k	52k	168	254	30,9
LOG	4	14	0.6	1.5k	847	14	2.5	505	272	10k	50	14	0.4	365	223	32	45	14
	6	20	6.2	17k	7.4k	20,6	–	–	–	–	–	–	1.3	732	341	52	185	20
LLG	6	18	0.2	134	61	18	4.9	118	54	2.0k	35	18	0.5	130	61	22	21	18
	8	24	0.6	241	113	24	11	216	102	4.6k	64	24	1.2	231	108	30	34	24
	10	30	0.9	381	181	30	27	337	161	8.7k	99	30	2.8	357	170	38	46	30
ROV	2	22	–	–	–	–	88	378	4	25k	4	22	19	482	4	72	4	22
	3	33	–	–	–	–	–	–	–	–	–	–	261	2.0k	14	108	11	33
SAT	4	14	32	32k	369	14	16	941	27	9.8k	22	14,3	0.8	536	29	4	23	14,3
	6	21	–	–	–	–	–	–	–	–	–	–	6.2	1.7k	67	6	54	21,3
	8	–	–	–	–	–	–	–	–	–	–	–	45	4.5k	147	8	104	28,1

Table 1: Experimental results for the heuristics. $|\mathcal{A}|$ is number of agents, l is sequential length of the plan (l^* is optimal), t is duration of the search in seconds, v is a number of visited states, c_{s} is a number of search messages (each of size of a state), c_{r} is a number of messages building ARPGs (each of size of a projected public action) and c_{h} is a number of messages for the heuristic estimate (each of size of a partial relaxed plan). As h^{FF}_{α} do not build distributed ARPGs, c_r and c_h are always zero. The domains are COOP. PATHFINDING (CP), LOGISTICS (LOG), LINEAR LOGISTICS (LLG), ROVERS (ROV) and SATELLITES (SAT). Runs denoted as – did not finish in the limits.

tion of FF heuristic as in Algorithm 2 with additional information on reachability of projected actions.

The implementation we used is a preliminary prototype which is not competitive with the current single-agent state-of-the-art planners, but nevertheless it gives insights in comparison of the heuristics.

The h^{MAFF} is based on the theory presented in the previous section. For each state each of the agents computes the heuristic estimate, i.e., a complete set of ARPGs is built. In order to manage the distribution and asynchronism, the ARPG building algorithm slightly differs from Algorithm 1 so that ARPGs for several different states can be built simultaneously. The heuristic estimate is then extracted using Algorithm 2.

In Lazy h^{MAFF}, the ARPGs are built lazily, i.e., an agent builds an ARPG for its current search state using its actions and projections similarly as in h^{FF}_{α}, then the Relaxed Plan (RP) is extracted and only when some projected action is added to the RP, request is sent to the owner of the original action. The owner then builds an ARPG from the given state to preconditions of the original actions, extracts a RP using the same procedure and sends back the computed RP. The returned RP is then merged with the original one. This effectively forms a distributed recursion algorithm. Such algorithm significantly lowers communication load and enables the agents to search in parallel, especially in loosely coupled problems. In addition to this, reachability analysis using Algorithm 1 can be done before starting the search to improve the estimate of applicability of the projected actions.

The results in Table 1 show, that h^{FF}_{α} is fast and can effectively solve smaller problem instances, but it is not much informed, as illustrated by the number of visited states. This becomes critical in larger problems. On the other hand, h^{MAFF} is better informed, but it has to build all ARPGs for each state which is estimated. This is extremely communication intensive as shown by the number of exchanged ARPG messages (c_{r}). Also the possibilities of parallel computation are reduced by the fact that all agents have to build the ARPGs for each estimated state.

The best performance is given by the Lazy h^{MAFF}. This implementation keeps the heuristic estimate quality of h^{MAFF}, but since it does not build ARPGs during the search, but only local RPGs enriched by the projections of other agents' actions, the RPGs can be built lazily only for those states where any interaction between the agents is needed and only those agents involved build the RPGs. The ARPGs are computed only in an initial reachability analysis, which can be omitted, but which significantly improves the results.

Final Remarks

Our formal treatment and design of algorithms for computing distributed Relaxed Planning Graph for multiagent planning can be seen as a first step towards efficient MA planners based on satisficing state-space search techniques utilizing relaxation heuristics. Furthermore, we have experimentally shown that appropriate implementation of a decentralized estimation of a global relaxation heuristic can radically improve computational and communication efficiency of the planning process as a whole.

Acknowledgments This work was supported by the *U.S. Air Force EOARD* grant no. FA8655-12-1-2096.

References

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of ICAPS'08*, 28–35. AAAI.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of ICAPS'09*. AAAI.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of ICAPS'07*, 176–183. AAAI.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Komenda, A.; Novák, P.; and Pěchouček, M. 2013. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications*. DOI: 10.1016/j.jnca.2012.12.011.

Mattern, F. 1987. Algorithms for distributed termination detection. *Distributed computing* 2(3):161–175.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In van der Hoek, W.; Padgham, L.; Conitzer, V.; and Winikoff, M., eds., *Proceedings of AAMAS'12*, 1265–1266. IFAAMAS.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS'10*, 1323–1330.

Pellier, D. 2010. Distributed planning through graph merging. In Filipe, J.; Fred, A. L. N.; and Sharp, B., eds., *Proceedings of ICAART'10*, volume 2, 128–134. IFAAMAS.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In Raedt, L. D.; Bessière, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *Proceedings of ECAI'12*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 762–767. IOS Press.

FMAP: a Heuristic Approach to Cooperative Multi-Agent Planning

Alejandro Torreño, Eva Onaindia, Óscar Sapena
Universitat Politècnica de València
Camino de Vera s/n
46022 Valencia, SPAIN

Abstract

In this paper we propose FMAP, a novel method for multi-agent planning that relies on an integrated planning-coordination approach. Agents continuously build up refinement plans and coordinate them until they find a joint solution plan. Each agent implements a complete forward-chaining partial-order planning. Multi-agent search is guided by a novel heuristic function based on the concept of Domain Transition Graph and optimized to evaluate plans in a multi-agent context with incomplete information. Experimental results show that this heuristic is competitive when compared to FF's heuristic in single-agent environments, and that FMAP achieves a good performance at solving MAP problems adapted from the IPC testbeds.

Introduction

Multi-agent planning (MAP) extends the typically centralized approach by introducing multiple entities that combine their knowledge and capabilities to solve planning tasks that they are not able to solve by themselves, or at least they can accomplish them better by cooperating with each other (de Weerdt and Clement 2009). In general, solving a cooperative MAP task involves a *planning* procedure by which agents build local plans and a *coordination* activity that combines them into a global solution (Durfee 2001).

The most straightforward approach to MAP divides the planning task into subtasks and allocates each task to an agent. Then, agents compute individual plans and a post-planning coordination process combines these local plans into a global solution (Nissim, Brafman, and Domshlak 2010; Van Der Krogt and De Weerdt 2005; Tonino et al. 2002). This *plan merging* scheme (Ephrati and Rosenschein 1997) allows to fully reuse standard single-agent planning techniques. However, plan merging is unlikely to cope with problems that involve many interactions among agents as merging may introduce exponentially many ordering constraints in problems which require a big coordination effort. Another major research trend in cooperative MAP coordinates plans *during* planning, rather than treating both activities as separate processes. This *integrated planning and coordination* approach provides a unified vision of cooperative MAP and introduces a much more flexible resolution scheme for solving complex problems with a high number of interactions among agents. Different methods follow the integrated approach to MAP. The Partial Global Planning framework (Durfee and Lesser 1991) and its extension, the Generalized Partial Global Planning (Decker and Lesser 1992), allow agents to iteratively communicate their local plans to the rest of agents and then merge this information into their own partial global plan to improve it. The continual planning approach integrates planning and execution and coordinates agents by synchronizing them at execution time (desJardins et al. 1999; Brenner and Nebel 2009). The TFPOP algorithm (Kvarnström 2011) is a centralized planning method that combines temporal planning and forward-chaining partial-order planning for solving MAP problems with few interactions among agents. The best-response planning algorithm (Jonsson and Rovatsos 2011) iteratively improves the quality of the agents' plans through single-agent planning technology.

The work in (Torreño, Onaindia, and Sapena 2012) introduces MAP-POP, an approach to integrated MAP that proves to be more competitive than a plan merging method (Nissim, Brafman, and Domshlak 2010) at solving MAP problems adapted from the International Planning Competition testbeds. MAP-POP combines planning and coordination through a multi-agent refinement planning procedure (Kambhampati 1997). Refinement plans are individually devised through an embedded Partial-Order Planning (POP) system (Younes and Simmons 2003). Planning agents in this approach are partially unaware of the initial state of the world and the information known to the rest of agents, an aspect generally overlooked in most MAP frameworks. While this MAP approach shows promising results, it still presents some limitations regarding performance and lack of theoretical properties.

This paper introduces FMAP, a novel approach to integrated MAP that is inspired on the proposal in (Torreño, Onaindia, and Sapena 2012). Agents in FMAP collaboratively build refinement plans through an embedded customized Forward-Chaining Partial-Order Planning (FPOP) search process (Coles et al. 2010) that ensures completeness and allows for the definition of competitive heuristic functions. The internal FPOP search process is guided by a heuristic function based on the notion of *Domain Transition Graph* (DTG) (Helmert 2006). The features of FMAP (in

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

particular, the partial information managed by the agents) motivated the development of a novel heuristic. In this MAP context, the application of popular heuristic functions such as FF's heuristic (h_{FF}) (Hoffmann and Nebel 2001) would require an important computational effort to evaluate plans; incomplete information obliges agents to interact with each other (for instance, h_{FF} would require the joint construction of a distributed relaxed planning graph (Zhang, Nguyen, and Kowalczyk 2007) instead of a simple one to evaluate a plan). In contrast, DTGs are particularly efficient data structures as they only depend on the planning domain and are built only once throughout the planning process.

The combination of a refinement planning scheme, a FPOP search process and a DTG-based heuristic function gives rise to a complete and reliable planning method that offers a good performance in both single-agent and multi-agent problems. The experimental evaluation shows that FMAP provides a similar performance than the state-based planner JavaFF (Coles et al. 2008) when tested on single-agent problems, and that it is more effective than MAP-POP (Torreño, Onaindia, and Sapena 2012) at solving large MAP problems with many interactions among agents. Moreover, we will show that, even though the DTG-based heuristic has been particularly designed for the evaluation of plans in a MAP context, it returns very similar results when compared to h_{FF} in single-agent problems.

This paper is organized as follows: next section formalizes the notion of MAP task and all the elements involved. Following, we describe the main components of FMAP, including the FPOP search process and the DTG-based heuristic function. Next, we provide the experimental results that evaluate the performance of FMAP when tested with single-agent and multi-agent planning tasks. Finally, we conclude by summarizing our upcoming research lines.

Multi-Agent Planning Task Formalization

Agents in FMAP work under partial observability by adopting the open world assumption, thus assuming that information not represented in an agent's model is unknown to the agent. Although there are some well-known MAP languages in the literature, such as MA-STRIPS (Brafman and Domshlak 2008), we developed our own language to define the particular features of our model, such as the agents' partial information on the world. This language, based on *PDDL3.1* (Kovacs 2011), represents information through *SAS+*-like state variables (Bäckström and Nebel 1995).

The states of the world are modeled through a finite set of *state variables*, $\mathcal{V}$, each of them associated to a finite domain, $\mathcal{D}_v$, of mutually exclusive values. Assigning a variable $v \in \mathcal{V}$ a value of its domain, $d \in \mathcal{D}_v$, gives rise to a *fluent* $\langle v, d\rangle$. A *positive fluent* is a tuple $\langle v, d\rangle$, which indicates that the variable v takes the value d. A *negative fluent* takes the form $\langle v, \neg d\rangle$, indicating that v does not take the value d.

An *action* is a tuple $\alpha = \langle PRE(\alpha), EFF(\alpha)\rangle$, where $PRE(\alpha)$ is a finite set of fluents that represents the preconditions of α, and $EFF(\alpha)$ is a finite set of positive and negative *operations* that model the effects of α. A state S is a set of positive and negative fluents. Executing an action α in a world state S gives rise to a new world state S' as a result of applying $EFF(\alpha)$ over S. Effects of the form $(v = d)$ add a fluent $\langle v, d\rangle$ and a set of fluents ($\langle v, \neg d'\rangle$, $\forall d' \in \mathcal{D}_v \ / \ d' \neq d$) to S'. If ($\langle v, d'\rangle \in S$, $d' \neq d$) or $\langle v, \neg d\rangle \in S$, then $(v = d)$ removes those fluents in state S'. Effects of the form $(v \neq d)$ add a fluent $\langle v, \neg d\rangle$ to S'. If $\langle v, d\rangle \in S$, $(v \neq d)$ also removes $\langle v, d\rangle$ in state S'.

A **MAP task** is defined as a tuple $\mathcal{T}_{MAP} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$, where $\mathcal{AG} = \{1, \ldots, n\}$ is a finite non-empty set of agents; $\mathcal{V} = \{\mathcal{V}^i\}_{i=1}^{n}$, where $\mathcal{V}^i$ is the set of state variables known to an agent i; $\mathcal{A}$ is the set of planning actions of the agents; $\mathcal{I}$ is a set of fluents that defines the initial state of $\mathcal{T}_{MAP}$; and $\mathcal{G}$ is the set of goals of $\mathcal{T}_{MAP}$, i.e., the values of the state variables that agents have to achieve to fulfill $\mathcal{T}_{MAP}$.

As stated above, the complete set of state variables $\mathcal{V}$ may not be known to all agents. An agent i may have a partial observability on the domain of a variable, $\mathcal{D}_v^i \subseteq \mathcal{D}_v$. The observability of a fluent by an agent i is defined by its view on the state variables:

- A fluent $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ is *fully observable* by i if $v \in \mathcal{V}^i$ and $d \in \mathcal{D}_v^i$.
- A fluent $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ is *partially observable* by i if $v \in \mathcal{V}^i$ but $d \notin \mathcal{D}_v^i$. Given a state S, where $\langle v, d\rangle \in S$, i will see instead a fluent $\langle v, \perp\rangle$, where $\perp$ is the undefined value.
- A fluent $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ is *not observable* by i if $v \notin \mathcal{V}^i$.

Each agent $i \in \mathcal{AG}$ has a set of actions $\mathcal{A}^i$ such that $\mathcal{A} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{A}^i$. If $\alpha \in \mathcal{A}^i$ is included in a plan, then agent i is responsible of executing α.

FMAP is a multi-agent refinement planning framework, a general method based on the refinement of the set of all possible plans (Kambhampati 1997). The internal reasoning of each agent is based on a POP search procedure (Barrett and Weld 1994), and hence, agents build *partial plans*.

A **partial plan** is a tuple $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL}\rangle$. $\Delta \subseteq \mathcal{A}$ is the set of actions in Π. $\mathcal{OR}$ is a finite set of ordering constraints ($\prec$) on Δ. $\mathcal{CL}$ is a finite set of causal links of the form $\alpha \overset{\langle v,d\rangle}{\rightarrow} \beta$ or $\alpha \overset{\langle v,\neg d\rangle}{\rightarrow} \beta$, where α and β are actions in Δ. $\alpha \overset{\langle v,d\rangle}{\rightarrow} \beta$ indicates that $(v = d) \in EFF(\alpha)$ supports a precondition $\langle v, d\rangle \in PRE(\beta)$. $\alpha \overset{\langle v,\neg d\rangle}{\rightarrow} \beta$ indicates that $\langle v, \neg d\rangle \in PRE(\beta)$, $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, is supported by an effect $(v \neq d) \in EFF(\alpha)$ or $(v = d') \in EFF(\alpha)$, $d' \neq d$.

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0\rangle$, where $\mathcal{OR}_0$ and $\mathcal{CL}_0$ are empty sets, and Δ_0 contains α_i. Planning agents manage two fictitious actions, α_i and α_f, which do not belong to the action set of any particular agent. α_i represents the initial state of $\mathcal{T}_{MAP}$, i.e., $PRE(\alpha_i) = \emptyset$ and $EFF(\alpha_i) = \mathcal{I}$, while α_f represents the global goals of $\mathcal{T}_{MAP}$, i.e., $PRE(\alpha_f) = \mathcal{G}$, and $EFF(\alpha_f) = \emptyset$. A plan Π for any task $\mathcal{T}_{MAP}$ will always contain α_i.

Planning agents in our model cooperate to solve MAP tasks by progressively refining an initially empty plan Π until a solution is reached. The definition of *refinement plan*

is closely related to the internal forward-chaining partial order planning search performed by the agents. We define a refinement plan as follows:

A **refinement plan** $\Pi_r = \langle \Delta_r, \mathcal{OR}_r, \mathcal{CL}_r \rangle$ over a partial plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$, is a flaw-free partial plan which extends Π, i.e., $\Delta \subset \Delta_r$, $\mathcal{OR} \subset \mathcal{OR}_r$ and $\mathcal{CL} \subset \mathcal{CL}_r$. Π_r introduces a new action $\alpha \in \Delta_r$ over Π, resulting in $\Delta_r = \Delta \cup \alpha$. All the preconditions in $PRE(\alpha)$ are supported by actions in Π; i.e., $\forall p \in PRE(\alpha)$, $\exists\ \beta \xrightarrow{p} \alpha \in \mathcal{CL}_r$, where $\beta \in \Delta$.

Refinement plans in FMAP include actions that can be executed in parallel by different agents. Some MAP models consider that two concurrent actions are mutually consistent if none of them modifies the value of a state variable that the other relies on or affects (Brenner and Nebel 2009). We also consider that the preconditions of two mutually consistent actions have to be consistent (Boutilier and Brafman 2001). Thus, two concurrent actions $\alpha \in \mathcal{A}^i$ and $\beta \in \mathcal{A}^j$ are *mutually consistent* if none of the following holds:

- $\exists (v = d) \in EFF(\alpha)$ and $\exists (\langle v, d' \rangle \in PRE(\beta) \vee \langle v, \neg d \rangle \in PRE(\beta))$, where $v \in \mathcal{V}^i \cap \mathcal{V}^j$, $d \in \mathcal{D}^i_v \cap \mathcal{D}^j_v$, $d' \in \mathcal{D}^j_v$ and $d \neq d'$, or vice versa.
- $\exists (v = d) \in EFF(\alpha)$ and $\exists ((v = d') \in EFF(\beta) \vee (v \neq d) \in EFF(\beta))$, where $v \in \mathcal{V}^i \cap \mathcal{V}^j$, $d \in \mathcal{D}^i_v \cap \mathcal{D}^j_v$, $d' \in \mathcal{D}^j_v$ and $d \neq d'$, or vice versa.
- $\exists \langle v, d \rangle \in PRE(\alpha)$ and $\exists (\langle v, d' \rangle \in PRE(\beta) \vee \langle v, \neg d \rangle \in PRE(\beta))$, where $v \in \mathcal{V}^i \cap \mathcal{V}^j$, $d \in \mathcal{D}^i_v \cap \mathcal{D}^j_v$, $d' \in \mathcal{D}^j_v$ and $d \neq d'$, or vice versa.

Agents address concurrency issues through the resolution of *threats* over the causal links of the plan (Younes and Simmons 2003). Thus, concurrency between any two actions introduced by different agents is guaranteed as refinement plans are always free of flaws.

Finally, a *solution plan* for a FMAP task $\mathcal{T}_{MAP}$ is a refinement plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ that supports the global goals $\mathcal{G}$ of $\mathcal{T}_{MAP}$ by introducing the final action α_f, i.e., $\alpha_f \in \Delta$ and $\forall g \in PRE(\alpha_f)$, $\exists\ \beta \xrightarrow{g} \alpha_f \in \mathcal{CL}$, where $\beta \in \Delta$.

FMAP Refinement Planning Procedure

This section describes the cooperative refinement planning procedure of FMAP, which is a plan-space search where each node of the tree is built through a forward-chaining POP (FPOP). Some existing planning approaches combine the *least commitment strategy* (Weld 1994) of POP with the performance of a forward-chaining search scheme. POPF (Coles et al. 2010) is a single-agent method that departs from a temporal and numeric forward-chaining state-based approach and modifies it to reduce the commitment of temporal order choices. This results in a hybrid proposal that represents a compromise between standard forward-search and POP. TFPOP (Kvarnström 2011) applies centralized forward-chaining POP for multiple agents, keeping a sequential thread of execution for each agent.

Unlike POPF, FMAP works on a plan space, as a classical POP, but each node comprises a forward-built non-linear plan. As opposite to TFPOP, each tree node in FMAP is contributed by several agents, in which parallel branches of the non-linear plan denote parallel threads of execution.

FMAP is a multi-agent search by which agents jointly build and explore a plan-space search tree. Each node in the tree is a refinement plan whose actions have been contributed by different planning agents. Agents independently devise refinement plans over a centralized base plan through an embedded FPOP procedure. The main stages of the FMAP iterative process can be summarized as follows:

- **Base plan selection**: Among all the leaf nodes of the multi-agent plan-space search tree, agents collectively select the most promising refinement plan as the new base plan Π_b.
- **Refinement plan generation**: Each agent individually applies a FPOP search that generates a set of refinement plans over Π_b. A refinement plan introduces a new fully-supported action in Π_b.
- **Refinement plan evaluation**: Agents evaluate the quality of the refinement plans via the DTG-based heuristic function h_{DTG}.
- **Refinement plan communication**: Agents communicate each other the evaluated plans and then they select the following base plan Π_b.

Algorithm 1 summarizes the FMAP procedure. The process starts with an initial information exchange by which agents communicate the fluents defined as *shareable information*. Our MAP language allows us to define, for each agent, the fluents that it can share with the others.

Algorithm 1: FMAP algorithm as applied by an agent i

```
Shareable information exchange
openNodes ← ∅
Π_b ← Π_0
repeat
    RP^i ← FPOP(Π_b)
    ∀ Π_r ∈ RP^i, f(Π_r) = g(Π_r) + h_DTG(Π_r)
    openNodes ← openNodes ∪ RP^i
    ∀j ≠ i, send RP^i to j
    ∀j ≠ i, receive RP^j from j
    openNodes ← openNodes ∪ RP^j
    Π_b ← argmin_{Π_n ∈ openNodes} f(Π_n)
    openNodes ← openNodes \ Π_b
until α_f ∈ Π_b ∨ openNodes = ∅
if α_f ∈ Π_b then
    return Π_b
else
    return No solution
```

After the initial information exchange, agents start the iterative planning procedure. At each iteration, agents select the most promising plan in the $openNodes$ list as the next base plan Π_b (initially, the empty plan Π_0 is selected). Then, each agent i generates a set of refinement plans RP^i over Π_b through its embedded FPOP search engine.

Each refinement plan $\Pi_r \in RP^i$ is evaluated through $f(\Pi_r) = g(\Pi_r) + h_{DTG}(\Pi_r)$, where $g(\Pi_r)$ is the number

of actions of the plan Π_r and $h_{DTG}(\Pi_r)$ applies our DTG-based heuristic function to estimate the cost of reaching a solution plan from Π_r.

Agents communicate each other their refinement plans and store them in the $openNodes$ list. Following, they select the next base plan Π_b as the refinement plan from $openNodes$ that minimizes $f(\Pi_b)$. This iterative process carries on until Π_b becomes a solution plan; i.e., Π_b supports the fictitious final action α_f, or the $openNodes$ list becomes empty, in whose case, agents will have explored the complete search space without finding a solution for the MAP task.

In the next subsections, we analyze the key elements of FMAP, the FPOP search used for the individual generation of the refinement plans and their evaluation through the h_{DTG} heuristic.

Forward-Chaining Partial-Order Planning

In FPOP, a refinement plan, Π_r, generated over a given base plan, Π_b, always introduces a new action α over Π_b. Π_r is a flaw-free partial plan in which the preconditions of the newly introduced action, α, are fully supported. Since refinement plans are built in a forward-chaining fashion and the plan does not contain threats among its actions, it is possible to consistently infer state information from any refinement plan. This allows for the use of powerful state-based heuristic functions, a key advantage over backwards-chaining POP that will be discussed in the following subsection.

Algorithm 2: $FPOP(\Pi_b)$ algorithm for an agent i

```
RP^i ← ∅
if potentiallySupportable(α_f, Π_b) then
    return solutionPlans
for all α ∈ A^i do
    if potentiallySupportable(α, Π_b) then
        Plans ← {Π_b}
        repeat
            Select and extract Π_s ∈ Plans
            Flaws(Π_s) ← unsPre(α, Π_s) ∪ Threats(Π_s)
            if Flaws(Π_s) = ∅ then
                RP^i ← RP^i ∪ Π_s
            Select and extract Φ ∈ Flaws(Π_s)
            Plans ← Plans ∪ solveFlaw(Π_s, Φ)
        until Plans ≠ ∅
return RP^i
```

Algorithm 2 summarizes the FPOP procedure that is invoked by each agent i to generate the refinement plans as explained in Algorithm 1. Firstly, agent i checks whether the fictitious final action α_f can be supported in Π_b. The function $potentiallySupportable(\Pi_b, \alpha_f)$ checks if $\forall(v = d) \in PRE(\alpha_f), \exists\langle v, d\rangle \in EFF(\beta)/\beta \in \Delta(\Pi_b)$. In other words, the agent estimates that α_f is potentially supportable if for every precondition of α_f a matching effect among the actions of Π_b can be found. This means that it is possible to support α_f's preconditions through causal links. The funtion $potentiallySupportable$ is an approximate procedure to determine the possibility of supporting an action in the plan, as it does not actually check the possible conflicts that arise when supporting α_f. Thus, if α_f is potentially supportable, agent i will explore all the possible alternatives to support α_f in a POP fashion, returning a set of $solutionPlans$ that solve the MAP task $\mathcal{T}_{MAP}$.

If α_f is not supportable yet, agent i computes the set of potentially supportable actions out of its set of actions $\mathcal{A}^i$. The *repeat* loop in Algorithm 2 generates, for each supportable action $\alpha \in \mathcal{A}^i$, a set of flaw-free plans stored in RP^i that refine Π_b, each representing a possible way to fully support α.

Agent i manages a list of $Plans$, which initially only contains the base plan Π_b. At each iteration, agent i extracts a plan $\Pi_s \in Plans$ and calculates $Flaws(\Pi_s)$ as the set of the preconditions of α that are not yet supported in Π_s, $unsPre(\alpha, \Pi_s)$, plus the unsolved threats that arise due to the introduction of α in Π_s, $Threats(\Pi_s)$. If $Flaws(\Pi_s) = \emptyset$, Π_s is stored in RP^i as a valid refinement plan over Π_b. Otherwise, the process selects a flaw Φ from Π_s and solves it in a POP fashion, thus generating a set of successors of Π_s, that are stored in the $Plans$ list. Once $Plans$ is empty, the FPOP process concludes and the refinement plans RP^i of each agent i are stored in the FMAP search tree as a result of expanding node Π_b.

The refinement plans obtained with FPOP are then evaluated in order to estimate their quality. For this purpose, we have defined a state-based heuristic function that takes advantage of the features provided by FPOP.

DTG-Based Heuristic Function

One of the major advantages of FMAP over other MAP approaches based on backwards-chaining POP (Torreño, Onaindia, and Sapena 2012) is that it is possible to compute the state resulting from the application of a refinement plan Π_r in the initial state of the MAP task. This allows us to use state-based heuristics while keeping the advantages of backwards-chaining POP. Since FPOP builds up refinement plans through a forward search, and actions in the plan are ensured not to have any conflicts among them, it is possible to *linearize* a partial plan and calculate the state that results from executing it.

Linearizing a refinement plan Π_r involves sequentializing the actions in $\Delta(\Pi_r)$ by establishing a total ordering among them. Consider two actions $\alpha \in \Delta(\Pi_r)$ and $\beta \in \Delta(\Pi_r)$; if $\alpha \prec \beta \in \mathcal{OR}(\Pi_r)$ or $\beta \prec \alpha \in \mathcal{OR}(\Pi_r)$ we keep this ordering constraint in the linearized plan. In case that α and β are concurrent actions, a total ordering is established among them. Since refinement plans do not have conflicts, it is irrelevant how concurrent actions are ordered, as there are not interactions between their preconditions and effects.

From the linearized plan, $linearize(\Pi_r)$, we define the state S_{Π_r} as the finite set of fluents that result from the application of the sequence of actions in $linearize(\Pi_r)$ over $\mathcal{I}$, the initial state of $\mathcal{T}_{MAP}$.

Since FPOP is actually working on a state space defined by the linearization of partial plans, we could make use of heuristics like h_{FF}, the relaxed planning graph

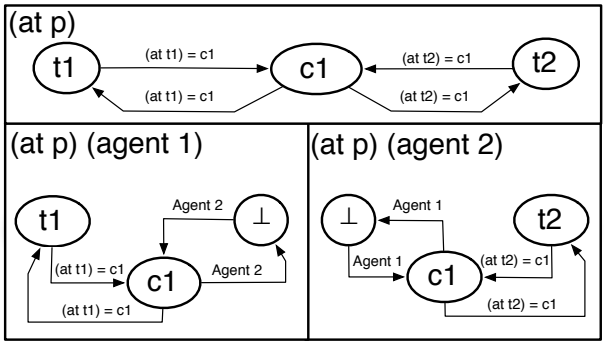

Figure 1: Centralized and distributed DTG example

(RPG) heuristic of FF (Hoffmann and Nebel 2001). However, FMAP is a fully distributed approach to MAP where agents have an incomplete view of the planning task, and this makes the application of h_{FF} not to be adequate to guide our MAP search. The reason is the following; using h_{FF} to estimate the quality of a refinement plan involves agents building a *distributed RPG* (Zhang, Nguyen, and Kowalczyk 2007) as none of the agents has enough knowledge to build a complete RPG by itself. This is a costly process that entails many communications among agents to coordinate which each other and, in addition, this process has to be repeated for the evaluation of each refinement plan. Therefore, the predictable high computational cost of the application of h_{FF} led us to initially discard this choice and opt for designing a DTG-based heuristic (h_{DTG}).

The concept of DTG (Helmert 2004) turns out to be an appropriate data structure to devise a heuristic function for a MAP context. A DTG is a directed graph that represents the ways in which a state variable can change its value. Each transition on a DTG is also labeled with the necessary conditions for the transition to occur; i.e., the common preconditions to all the planning actions that induce the value transition. As opposite to RPGs, DTGs are independent of the particular state of the refinement plan, thus avoiding to recalculate DTGs during the planning process. Therefore, DTGs, as a state-independent data structure, are particularly adequate for MAP.

As for the incomplete information, the unknown value $\perp$ is represented in a DTG as the other values of the corresponding variable. The only difference is that transitions from/to $\perp$ are labeled with the agents that cause that transition. This can be observed in Figure 1, which shows a small DTG example based on the *driverlog* domain. The graph represents the DTG of a state variable $(at\ p)$ that describes the possible locations of a package p (two different trucks $t1$ and $t2$, and one city $c1$). In a centralized problem (upper diagram of Figure 1) all the information is available in the DTG. Consider, though, a multi-agent version of the problem (bottom diagrams of Figure 1) with two different agents 1 and 2 that manage $t1$ and $t2$, respectively. In this situation, agent 1 ignores where package p goes once agent 2 picks it up in $c1$, and vice versa. If agent 1, for example, has to evaluate the cost of achieving the fluent $\langle at\ p, t1\rangle$ and it ignores the location of p, then it will request agent 2 the cost of delivering p in $c1$ and then it will add the cost of moving p from $c1$ to $t1$. Communications among agents are required to evaluate multi-agent plans with unknown information, but DTGs are much more efficient than RPGs as they remain constant along the planning process, so agents can minimize the communication cost by memorizing paths and distances between values.

Algorithm 3: h_{DTG} heuristic calculation for a plan Π_r

$h_{DTG} \leftarrow 0$
$S_{\Pi_r} \leftarrow linearize(\Pi_r)$
$openG \leftarrow PRE(\alpha_f)$
for all $\langle v, d\rangle \in PRE(\alpha_f)$ **do**
  $Values_v \leftarrow \{getValue(v, S_{\Pi_r})\}$
while $openG \neq \emptyset$ **do**
  $\langle v, d_g\rangle \leftarrow \underset{\langle v', d'\rangle \in openG}{\arg\max}\ distMin(v', Values_{v'}, d')$
  $openG \leftarrow openG \setminus \{\langle v, d_g\rangle\}$
  $d_0 \leftarrow \underset{d \in Values_v}{\arg\min}\ |Dijkstra(v, d, d_g)|$
  // $minPath$: value transitions (d_i, d_{i+1})
  $minPath \leftarrow Dijkstra(v, d_0, d_g)$
  for $i \leftarrow 0$ **to** $|minPath| - 1$ **do**
    $\alpha_{min} \leftarrow getMinCostAction(v, d_i, d_{i+1})/$
     $(d_i, d_{i+1}) \in minPath$
    $openG \leftarrow openG \cup \{\langle v, d\rangle \in PRE(\alpha_{min})/$
     $d \notin Values_v\}$
    for all $(v = d) \in EFF(\alpha_{min})$ **do**
      $Values_v \leftarrow Values_v \cup \{d\}$
    $h_{DTG} \leftarrow h_{DTG} + 1$
return h_{DTG}

Algorithm 4: $distMin(v, Values_v, d)$ function

$d_{ini} \leftarrow \underset{d' \in Values_v}{\arg\min}\ |Dijkstra(v, d', d)|$
return $|Dijkstra(v, d_{ini}, d)|$

Algorithm 5: $getMinCostAction(v, d_i, d_j)$ function

$\mathcal{A}_{ij} \leftarrow \alpha \in \mathcal{A}/\langle v, d_i\rangle \in PRE(\alpha) \wedge (v = d_j) \in EFF(\alpha)$
$\alpha_{min} \leftarrow \underset{\alpha \in \mathcal{A}_{ij}}{\arg\min} \sum_{\forall \langle v', d'\rangle \in PRE(\alpha)} distMin(v', Values_{v'}, d')$
return α_{min}

h_{DTG} is a heuristic function that returns, for a given plan Π_r, the number of actions of a relaxed plan between the state S_{Π_r} that results from applying $linearize(\Pi_r)$, as described above, and the set of goals of $\mathcal{T}_{MAP}$, $\mathcal{G}$. h_{DTG} performs a backward search by which actions that support the goals in $\mathcal{G}$ are consecutively introduced in the relaxed plan, until their preconditions are also fully supported. Hence, the underlying principle of h_{DTG} is quite similar to h_{FF}, except for the fact that we use DTGs instead of RPGs to build the relaxed plan.

Algorithm 3 summarizes the h_{DTG} evaluation process for a refinement plan Π_r. First, the process calculates S_{Π_r} by linearizing Π_r. The procedure manages a set of fluents, $openG$, that is initially set to $\mathcal{G}$. During the evaluation process, the preconditions of the actions introduced in the relaxed plan to support the fluents in $openG$ will be also included in $openG$. For each variable v in the MAP task, the process handles a list of values, $Values_v$, which is initialized to the value of v in S_{Π_r}. For each action introduced in

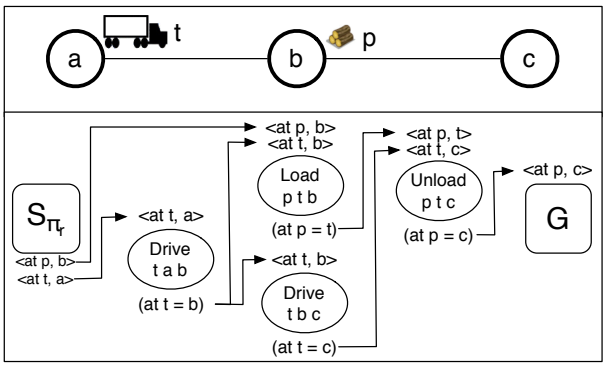

Figure 2: h_{DTG} evaluation process example

the relaxed plan that includes an effect $(v = d')$, $Values_v$ will store d' as a value that can be used to support fluents of $openG$ in further iterations of the process.

An iteration of the h_{DTG} evaluation process, described in Algorithm 3, performs the following stages:

- **Open goal selection**: This stage selects and extracts the fluent $\langle v, d_g \rangle \in openG$ that requires the largest number of value transitions to be supported. Function $distMin(v, Values_v, d)$, shown in Algorithm 4, computes the shortest path length to support a fluent $\langle v, d \rangle$.
- **DTG path computation**: This process calculates the shortest sequence of value transitions in v's DTG from an initial value $d_0 \in Values_v$ to d_g; that is, $minPath = ((d_0, d_1), (d_1, d_2), \ldots, (d_{g-1}, d_g))$. In Algorithm 3, this path is computed through $Dijkstra(v, d_0, d_g)$.
- **Relaxed plan construction**: For each value transition $(d_i, d_{i+1}) \in minPath$, this stage introduces in the relaxed plan the minimum cost action α_{min} that causes that transition; that is, $\langle v, d_i \rangle \in PRE(\alpha_{min})$ and $(v = d_{i+1}) \in EFF(\alpha_{min})$. The cost of an action is computed as the sum of the minimum number of value transitions required to support each of its preconditions, as it can be observed in Algorithm 5. The unsupported preconditions of each α_{min} are stored in $openG$, so they will be supported in forthcoming iterations. For each effect $(v' = d') \in EFF(\alpha_{min})$, the value d' is stored in $Values_{v'}$, so d' can be used in the following iterations to support other fluents in $openG$.

The iterative evaluation procedure carries on until all the open goals have been supported, that is, $openG = \emptyset$, and h_{DTG} returns the number of actions in the relaxed plan.

Figure 2 (upper diagram) shows a small example that illustrates the application of h_{DTG}. This is a transportation example including three different cities a, b and c, a truck t and a package p. The MAP task has two state variables, $(at\ t)$ and $(at\ p)$, that indicate the position of the truck t and the package p, respectively. The goal of the task is to deliver p in city c, that is, $\mathcal{G} = \{\langle at\ p, c \rangle\}$. Let Π_r be a refinement plan such that $S_{\Pi_r} = \{\langle at\ t, a \rangle, \langle at\ p, b \rangle\}$.

The h_{DTG} evaluation procedure of plan Π_r starts by initializing the $Values$ and $openG$ sets as follows: $Values_{(at\ t)} = \{a\}$, $Values_{(at\ p)} = \{b\}$ and $openG = \{\langle at\ p, c \rangle\}$.

The selection stage extracts the only fluent in $openG$ for its resolution, that is, $\langle v, d_g \rangle = \langle at\ p, c \rangle$. The initial value d_0 is the only value in $Values_{(at\ p)}$, $d_0 = b$. The shortest DTG path from b to c is computed as $minPath = ((b, t), (t, c))$, that is, the truck t loads the package p in city b and unloads it in c. Thus, actions $load\ p\ t\ b$ and $unload\ p\ t\ c$ are introduced in the relaxed plan, as they produce the value transitions (b, t) and (t, c), respectively. The preconditions and effects of these actions are used to update the $Values$ and $openG$ sets as follows: $Values_{(at\ p)} = \{b, t, c\}$ and $openG = \{\langle at\ t, b \rangle, \langle at\ t, c \rangle\}$.

The second iteration begins by extracting $\langle v, d_g \rangle = \langle at\ t, c \rangle$ from $openG$, since $Values_{(at\ t)} = \{a\}$ and the distance from a to c is greater than the distance from a to b. Hence, the initial value for $at\ t$ is also $d_0 = a$. The computed path reflects the shortest route that truck t can follow to reach city c from a: $minPath = ((a, b), (b, c))$. The process introduces the actions that produce these value transitions in the relaxed plan: $drive\ t\ a\ b$ and $drive\ t\ b\ c$. The effects of these actions are included in $Values_{(at\ t)}$, resulting in $Values_{(at\ t)} = \{a, b, c\}$. Since $b \in Values_{at\ t}$, $\langle at\ t, b \rangle \in openG$ is also supported, and therefore, $openG = \emptyset$, and $h_{DTG} = 4$. Figure 2 shows the computed relaxed plan.

Completeness and Soundness

The FMAP algorithm builds a multi-agent search tree whose nodes are partial plans. As previously stated, given a base plan, each of the agents generates a set of flaw-free refinement plans that introduce and fully support a new action into the plan. An agent i studies all the possible ways to support each action $\alpha \in \mathcal{A}^i$, giving rise to a set of plans that cover all the possibilities to support the actions in $\mathcal{A}^i$. As all the agents refine base plans concurrently, then all the actions in $\mathcal{A}$ are studied and all the possible refinements over a base plan are generated. Thus, FMAP is a complete MAP method, as it completely explores the search space.

As for soundness, it is important to remark a partial-order plan is sound if it is a flaw-free plan. In our algorithm, we address inconsistencies among partial plans by detecting and solving threats. Thus, FMAP is sound if all the flaws of a refinement plan are correctly detected and solved. Since agents manage an incomplete information model, we should study how visibility over fluents affects the detection of threats.

Let Π be a partial plan and let $\langle v, d' \rangle$ be a fluent in a causal link $cl \in CL(\Pi)$. Let Π_r be a refinement over Π, devised by an agent i, that introduces a new action α_i which is not ordered with respect to cl and has an effect $(v = d'')$. This effect gives rise to a threat over cl as it conflicts with $\langle v, d_1 \rangle$. For Π_r to be sound, agent i should be able to detect and correct such a threat whatever its observability of $\langle v, d' \rangle$ is:

- If i has *full visibility* over $\langle v, d' \rangle$, the threat will be correctly detected.
- If i has *no visibility* over $\langle v, d' \rangle$, then $v \notin \mathcal{V}^i$. In this case, agent i does not have an action α_i with an effect involving variable v, i.e., such a threat can never occur.

- If i has *partial visibility* over $\langle v, d' \rangle$, agent i will see instead a fluent $\langle v, \perp \rangle$. Since $\perp \neq d''$, the threat will be detected and solved.

Therefore, all the threats over partial plans are always detected and resolved, which proves that FMAP is sound.

Experimental Results

In order to assess FMAP performance, we realized both single-agent and multi-agent experiments. Tests were performed with some of the benchmark problems from the International Planning Competitions[1] (IPC). More precisely, we run the STRIPS problem suites for five domains of the 2002 IPC (the *satellite*, *rovers*, *logistics*, *driverlog* and *depots* domains) and two domains of the 2008 IPC (*elevators* and *woodworking*).

The first test compares h_{DTG} and an implementation of h_{FF} in FMAP for single-agent problems (agents are limited to 1). Additionally, we executed the most popular Java-based implementation of the FF planner, JavaFF (Coles et al. 2008). As FMAP is also implemented in Java, we can then achieve a fair comparison between FMAP and the state-based planner JavaFF.

Table 1 summarizes the results of the single-agent planning experimentation[2]. Each experiment is limited to 30 minutes. The table compares the quality of the results obtained with JavaFF, and FMAP with h_{DTG} and h_{FF}. The quality of the results obtained by each planning approach is assessed via four parameters: 1) number of *solved* problems; 2) average number of *actions* of the solution plans; 3) average *makespan*, i.e. duration or number of time steps required to execute a plan; and 4) average execution *time*. Notice that we do not include the makespan values in JavaFF because it returns sequential plans. Actions, makespan and execution times are relative values with respect to the results obtained with FMAP using h_{DTG} (we will simply refer to this approach as h_{DTG} and h_{FF} to the FMAP version with h_{FF}). This way, the notation nx indicates "n times as much as the result obtained with h_{DTG}". For instance, a 2x item in the *Actions* column indicates that, in average, the number of actions of the plans obtained with the approach is twice as much as the number of actions obtained with h_{DTG}. Equivalently for the makespan and computation time. Thereby, a value higher than 1x indicates a better result for h_{DTG} and a value lower than 1x shows a worse result for h_{DTG}.

In general, h_{DTG} and h_{FF} behave similarly in a single-agent context. h_{FF} solves 123 problems out of 160 (77%), while h_{DTG} solves 121 (nearly a 76%). In the *satellite* domain, h_{FF} is slightly faster than h_{DTG}, while h_{DTG} obtains better-quality solutions and solves one more problem. In the *rovers* and *logistics* domains, h_{FF} shows a good performance and improves h_{DTG}'s results by all the parameters we measured, being able to solve the complete *logistics* problem suite.

[1] http://ipc.icaps-conference.org/

[2] All the tests were performed on a single machine with a 2.83 GHz Intel Core 2 Quad CPU and 8 GB RAM.

Results in the *depots* domain favor h_{DTG}, which is six times faster than h_{FF} and obtains better-quality plans in average. However, h_{FF} is able to solve two more *depots* problems than h_{DTG}. The *driverlog* domain presents mixed results; while h_{DTG} improves the average makespan of the solutions, h_{FF} solves one more problem and is slightly faster, providing solutions with fewer actions in average. The *woodworking* domain favors h_{DTG}, which solves six more problems than h_{FF} and is around 19 times faster, whereas the plans of h_{FF} show a slightly lower makespan. Finally, h_{DTG} proves to be a better heuristic for the *elevators* domain. Both heuristics solve the complete problem suite, but h_{DTG} is more than three times faster, and it obtains better solution plans in both number of actions and makespan.

According to these results, we can conclude that both heuristics present a very similar performance in a single-agent context. *Elevators* and *woodworking* domains favor h_{DTG}, while h_{FF} performs better with the *rovers* and *logistics* domains. The rest of domains show mixed results. We can conclude then that, even though h_{DTG} was particularly designed for the evaluation of multi-agent plans, it is very competitive compared to one of the most efficient state-based heuristic functions, h_{FF}.

Table 1 also compares h_{DTG} with JavaFF. As shown in the results, JavaFF obtains much better execution times in four of the domains (around 10 times faster than h_{DTG}). The much more simple calculations of a state-based planner benefit JavaFF's execution times. However, FMAP is still faster in the *driverlog*, *woodworking* and *elevators* domains, which proves the good performance of our approach.

JavaFF is able to solve more problems than h_{DTG} in four of the planning domains as it is generally faster than h_{DTG}. h_{DTG} manages to outperform JavaFF in the *logistics* and *depots* domains. In terms of plan quality, h_{DTG} generally obtains better results that JavaFF, generating shorter plans in average in five of the planning domains. JavaFF only achieves slightly better solutions in the *rovers* and *woodworking* domains.

In conclusion, JavaFF noticeably reduces execution times in some of the domains and solves more problems than h_{DTG}, which is burdened by its complex POP-based search machinery. However, h_{DTG} obtains better-quality solution plans in most cases, which indicates the robustness of our planner.

In multi-agent scenarios, we compared the performance of FMAP with another state-of-the-art MAP planner, MAP-POP (Torreño, Onaindia, and Sapena 2012). As FMAP, MAP-POP is an integrated planning and coordination approach where agents have a partial view of the domain, which makes it appropriate to assess the performance of our MAP method.

We adapted six domains of Table 1 to a multi-agent scenario. In the *satellite*, *rovers* and *elevators* domains we modeled an agent for each `satellite`, `rover` and `elevator`, respectively. In the *driverlog* domain, `drivers` are identified as the agents in each problem. *Logistics* and *Depots* include two different types of

Domain	FMAP					JavaFF		
	h_{DTG}	h_{FF}						
	Solved	Solved	Actions	Makespan	Time	Solved	Actions	Time
Satellite	18 / 20	17 / 20	1,029x	1,172x	0,935x	20 / 20	1,14x	0,402x
Rovers	15 / 20	17 / 20	0,957x	0,915x	0,523x	20 / 20	0,98x	0,039x
Logistics	16 / 20	20 / 20	0,978x	0,964x	0,197x	12 / 20	1,026x	0,107x
Depots	9 / 20	11 / 20	1,076x	1,074x	6,149x	6 / 20	1,16x	0,157x
Driverlog	14 / 20	15 / 20	0,966x	1,015x	0,872x	15 / 20	1,006x	1,26x
Woodworking	19 / 30	13 / 30	1x	0,986x	18,8x	22 / 30	0,85x	1,495x
Elevators	30 / 30	30 / 30	1,007x	1,036x	3,395x	30 / 30	1,028x	4,905x

Table 1: Comparison between FMAP (with DTG and FF heuristics) and JavaFF

Domain	FMAP	MAP-POP			
	Solved	Solved	Actions	MS	Time
Elevators	30 / 30	22 / 30	0,996x	1,364x	8,088x
Rovers	19 / 20	14 / 20	0,957x	0,889x	0,788x
Satellite	18 / 20	18 / 20	0,988x	0,871x	0,156x
Logistics	10 / 20	8 / 20	0,993x	0,95x	1,645x
Driverlog	14 / 20	2 / 20	1,105x	1x	264,187x
Depots	6 / 20	1 / 20	1,2x	1,125x	2,879x

Table 2: Comparison between FMAP and MAP-POP

agents: agents in the *logistics* domain are `airplanes` and `trucks`, while `depots` and `trucks` are modeled as agents in the *depots* domain.

In general, domains *satellite*, *rovers* and *elevators* give rise to simple MAP problems in which there are few interactions among agents. The resolution process for these problems is noticeably simpler. The *driverlog*, *logistics* and *depots* domains lead to much more complex MAP problems with many interactions among agents, which directly affects the number of problems solved by each method.

Table 2 shows the MAP results. The table presents the number of problems *solved* by each method, and MAP-POP's average *time*, number of *actions* and makespan (*MS*), as relative values with respect to the results obtained with FMAP. FMAP solves 97 out of 130 problems, roughly the 75% of the tests, while MAP-POP only solves 65, a poor 50% of the problems in the benchmark. MAP-POP is only able to solve the same number of problems than FMAP in the *satellite* domain. This result is especially significative considering that MAP-POP is not complete, as authors recognize that their method implicitly prunes parts of the search tree (Torreño, Onaindia, and Sapena 2012). Thus, FMAP is able to solve more problems than MAP-POP while keeping completeness.

MAP-POP shows a better performance in the simplest domains, *satellite* and *rovers*. In both cases, MAP-POP solves problems faster and obtains better-quality solutions, an indication that MAP-POP is a better approximation for domains with few interactions among agents. MAP-POP also obtains slightly better plans in the *logistics* domain, but in this case, FMAP is nearly 2 times faster than MAP-POP, and it solves two more problems. FMAP presents better results in the *elevators* domain, solving the complete benchmark and providing solutions with a much lower makespan (whereas the average number of actions is slightly higher), and being 8 times faster than MAP-POP in average.

The major differences between both methods can be found in the two most complex domains, *driverlog* and *depots*. In both cases, FMAP solves far more problems in much less time than MAP-POP, and the average quality of the solution plans also favors FMAP.

In conclusion, FMAP shows to be a more robust method than MAP-POP while keeping soundness and completeness. FMAP is able to solve complex MAP problems which present many interactions among agents, whereas MAP-POP is only competitive with simpler problems, such as the *satellite* ones. In addition, FMAP shows excellent results in both multi-agent and single-agent problems.

Conclusions and future work

FMAP is a general-purpose and flexible MAP model which has been extensively tested on IPC benchmarks, and represents one of the few domain-independent fully-operative MAP approaches nowadays. FMAP contributes to the state of the art in multi-agent planning in several ways. First, it is specifically designed for domains under incomplete information, allowing agents to ignore the variables or specific values of variables of other agents. This allows us to deal with inherently distributed domains (functionally or spatially). Second, FMAP combines the expressiveness and flexibility of a POP with the development of a heuristic for state-based search, exploiting the structure of distributed state-independent DTGs and thus avoiding recalculating distance estimates at each node of the POP tree. This is a relevant issue in MAP approaches as they typically present an overload due to the agents message passing during the solving process. Third, the experimental results show that h_{DTG} is very competitive when compared to h_{FF} and that FMAP outperforms a state-of-the-art MAP method based on backwards POP.

We intend to implement a distributed version of h_{FF}. We would like to test if the power of h_{FF} is somehow outshadowed by the repeated calculation of distributed RPGs. On the other hand, we will work on improving the h_{DTG} heuristic function for that FMAP provides a better performance in the most complex IPC planning domains.

Acknowledgments

This work has been partly supported by the Spanish MICINN under projects Consolider Ingenio 2010 CSD2007-00022 and TIN2011-27652-C03-01.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11(4):625–655.

Barrett, A., and Weld, D. S. 1994. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence* 67(1):71–112.

Boutilier, C., and Brafman, R. 2001. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research* 14(105):136.

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 28–35.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems* 19(3):297–331.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Teaching forward-chaining planning with JavaFF. In *Colloquium on AI Education, 23rd AAAI Conference on Artificial Intelligence*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

de Weerdt, M., and Clement, B. 2009. Introduction to planning in multiagent systems. *Multiagent and Grid Systems* 5(4):345–355.

Decker, K., and Lesser, V. R. 1992. Generalizing the Partial Global Planning algorithm. *International Journal of Cooperative Information Systems (IJCIS)* 2(2):319–346.

desJardins, M.; Durfee, E.; Ortiz, C.; and Wolverton, M. 1999. A survey of research in distributed continual planning. *AI Magazine* 20(4):13–22.

Durfee, E. H., and Lesser, V. 1991. Partial Global Planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks* 21(5):1167–1183.

Durfee, E. H. 2001. Distributed problem solving and planning. In *Multi-agents Systems and Applications: Selected tutorial papers from the 9th ECCAI Advanced Course (ACAI) and AgentLink's Third European Agent Systems Summer School (EASSS)*, volume LNAI 2086, 118–149. Springer-Verlag.

Ephrati, E., and Rosenschein, J. S. 1997. A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence* 20(1-4):13–67.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. *Proceedings of ICAPS* 161–170.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 114–121. AAAI.

Kambhampati, S. 1997. Refinement planning as a unifying framework for plan synthesis. *AI Magazine* 18(2):67–97.

Kovacs, D. L. 2011. Complete BNF description of PDDL3.1. Technical report.

Kvarnström, J. 2011. Planning for loosely coupled agents using partial order forward-chaining. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 138–145. AAAI.

Nissim, R.; Brafman, R.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1323–1330.

Tonino, H.; Bos, A.; de Weerdt, M.; and Witteveen, C. 2002. Plan coordination by revision in collective agent based systems. *Artificial Intelligence* 142(2):121–145.

Torreño, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *20th European Conference on Artificial Intelligence (ECAI 2012)*, volume 242, 762–767. IOS Press.

Van Der Krogt, R., and De Weerdt, M. 2005. Plan repair as an extension of planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 161–170.

Weld, D. 1994. An introduction to least commitment planning. *AI magazine* 15(4):27.

Younes, H., and Simmons, R. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research* 20:405–430.

Zhang, J.; Nguyen, X.; and Kowalczyk, R. 2007. Graph-based multi-agent replanning algorithm. In *Proceedings of the 6th Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Author Index

Keyword Index