Dissertation Abstract - Automated Planning for Planetary Rovers

Juan M. Delfa Victoria
Technische Universität
Darmstadt, Germany
delfa@sim.tu-darmstadt.de

Introduction

Along the brief history of Space Exploration the expertise of Space Agencies has grown, from small rockets, not even able to escape the atmosphere, to missions beyond the solar system. At present, robots are gaining prominence in several scenarios as planetary exploration or ISS exploitation. Satellites are endowed with autonomous systems to improve their performance and safety. On the horizon, we start to catch sight of one of the most challenging mission ever undertaken by the mankind, a manned mission to mars.

Regarding robotic missions, their complexity lay in different factors:

- Spacecraft: Future missions are becoming more and more complex in terms of payload and operations. As an example, Curiosity weights 900 kg, has 17 cameras and around 13 instruments, has nuclear power, the CPU runs at 132 MHz with 4 GB of internal memory while MER weights 185 kg, has 10 cameras and a total of 7 instruments, has solar power, the CPU runs at 20 MHz with 256 MB of internal memory.
- Environment: Regarding rover missions, the inherent uncertainty of an unknown, unstructured and dynamic environment makes planning much harder. It has several consequences: Planning based in a complete understanding of the world is not feasible, initial conditions might be unknown, some of the assumptions considered during planning might be wrong, outcome of the actions is based on estimations rather than exact models and new relevant information for the plan might be discovered only during execution time. As a consequence, re-planning is frequently needed due to flaws in the plan or new opportunistic goals.
- Performance: An improvement in the performance of the spacecrafts becomes mandatory. For example, MER rover is stopped most of the time during traverses while processing pictures, generating the map and calculating the path.
- Duration: Long term missions like MER or Curiosity rovers represent a challenge for the operations team: producing day-to-day command sequences is a very demanding task that requires a huge team of experts. Automated tools could help to ease this problem.
- Communications: The delay of communications in deep space missions makes teleoperation impossible. For example, the round trip of a radio signal from Earth to Mars can take more than forty minutes.
- Low CPU performance: Space-oriented processors have much lower performance than those integrated in conventional computers. As an example, the latest, state-of-the-art space processor runs at around 132 MHz.
- Failure recovery: The possibilities to recover a mission in case of a spacecraft failure are very limited. Therefore, safety and V&V play a major roll in space missions.
- Cost: Increasing the number of missions and their complexity while maintaining the manpower present a number of challenges in terms of operations.

All these facts make robotic missions specially challenging, demanding the introduction of new, more sophisticated technologies. In the past decades Automated Planning & Scheduling (P&S) has become a well studied field. Nevertheless, there is an important gap between academic and real-world systems that needs to be continuously bridged in both directions to make planning theory aware of the complexity of real-world problems and to transfer innovations in theory to applied planners.

In an effort to develop new solutions, ESA supported the development of APSI (Fratini, Pecora, and Cesta 2008), a framework to develop timeline-based automated planning and scheduling applications. In APSI, a planner requires two inputs: **model** and **problem** and produce one output, the **plan**. The model contains a formal description of all the systems which activities must be planned. Each system is modeled as an automaton (named $component$ in APSI) composed of states (component decisions (cd) in APSI) and relations (rlt) between the cd's, which represent a super-set of the classical transitions in automata theory. Each component has related a timeline that represents a more or less flexible sequence of cd's that represent the plan.

A problem is represented as a decision network (dn), that is, a graph that contains a set of initial conditions ic representing facts that the planner does not need to justify, and $goals$, which the planner must justify using the model. Both

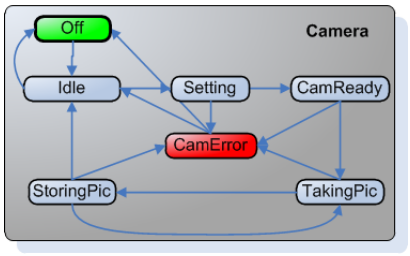

Figure 1: Component Camera automaton

ic's and $goals$ are represented as sets of cd's (the nodes) and/or rlt's (the edges) of the graph.

In case all goals are satisfied, the result is a fully supported dn called plan from which the timelines are extracted.

QuijoteExpress - Novel planner for space missions

The author's thesis "Automated Planning for Planetary Rovers", aims to provide a novel planner able to generate valid plans for temporal problems. Some of the key concepts to be addressed are:

- Plan complexity: Provide novel planning architectures and algorithms to handle NP-complete problems.
- Uncertainty: The planner should be able to generate robust plans for execution under uncertainty. Uncertainty is present in scenarios where the initial conditions are unknown, or where making assumptions is hard due to the environment properties (unknown, unstructured and dynamic).
- Exploit knowledge: Several years of operations resulted in a lot of knowledge of the operators that is written nowhere. The planner must be able to represent and use this knowledge.
- Domain&Problem independent: The solving algorithms and heuristic should retain as much generality as possible in order to be reusable in other scenarios.

The author has implemented a planner called QuijoteExpress that represents a new timeline planning paradigm for real-world scenarios such as the rover-world problem. QuijoteExpress is a heuristically&knowledge-based, domain-independent timeline planner that takes advantages from different planning techniques such as HTN, CSP or Plan Space. It presents the following novelties, further detailed in next subsections:

- HTLN: Hierarchical timeline networks represent an application of HTN for temporal planning.
- Parallelism: Identification of independent parts of the problem that can be solved in parallel.
- Sufficient-plan: It represents a problem that has been partially solved, but still represents a valid output of the planner.
- Heuristically-based: The planner departs from other temporal planners in that it follows a heuristically-based approach, close to traditional Plan Space algorithms.

Hierarchical Timeline Networks

HTLN relies on the idea of merging timeline planning and HTN (Erol, Hendler, and Nau) techniques. With respect to timeline planning, HTLN is based on the formalism of APSI (Fratini, Pecora, and Cesta 2008). Regarding HTN, we have used cyclic hypergraph structures to represent the hierarchical decomposition of goals into sub-goals. Hierarchical Task Networks (HTN) allows the human operator to define a plan in terms of complex goals while the planner is in charge of decomposing them into commands. This technique provides several advantages. First, it represents an improvement on the planner performance, as HTN simplifies the search space. For all the complex goals which decompositions are known, the planner does not need to perform search anymore, as it just need to replace the goal by its decomposition. Second, it also simplifies the modeling which is one of the major problems that engineers need to face to deploy automated tools. Finally, it makes plans easier to understand and validate by humans.

Parallelism

A key concept for the design of the planner is the hypergraph. It allows the representation of complex goals as both, a goal on its own and as a sub-problem. Hypergraphs are also useful to achieve parallel planning. By reasoning over the underlying hypergraph structure of an HTLN problem it is possible to identify independent sub-problems, allowing the use of parallel planning. As explained in (Dechter and Pearl 1987), a graph $G = (V, E)$ has a separation vertex v if there exist vertices a and b, $a \neq v$, $b \neq v$ such that all the paths connecting a and b pass through v (see Figure 2). A graph that has a separation vertex is called separable. Let $V' \subseteq V$, the induced sub-graph $G' = (V', E')$ is called a non-separable component if G' is non-separable and if for every larger V'', $V' \subseteq V'' \subseteq V$, the induced sub-graph $G'' = (V'', E'')$ is separable.

An efficient algorithm to generate valid temporal plans (not considering resource consumption) and computing the minimal network is to first find the non-separable components $C_1..C_m$ and then solve each one of them independently. If all the components are found to be consistent, then the entire network is consistent, and the minimal networks of the individual components coincide with the overall minimal network.

Sufficient plan

Traditional timeline planners search for fully justified plans, which are complete, fully ordered and valid:

- Complete: All variables are grounded. The plan always specifies how to proceed (the timelines have no gaps).
- Fully ordered: All the decisions are sequenced.
- Valid: All the constraints are satisfied.

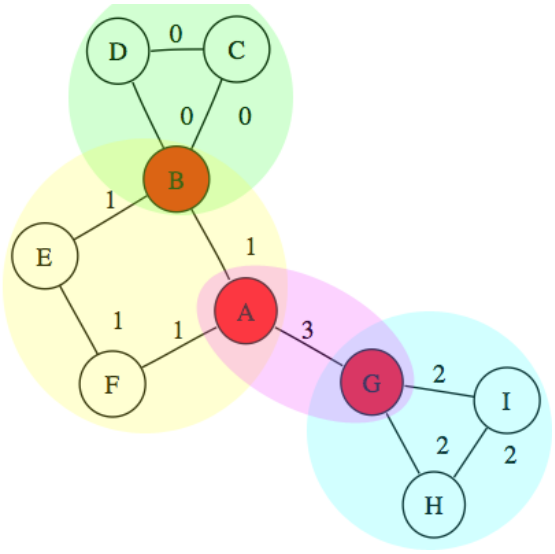

Figure 2: Articulation points in a graph

This definition might represent an unachievable condition for real-world P&S systems. In order to deal with non deterministic, dynamic and partially observable environments, more flexibility is required. Our planning system is based on the concept of Sufficient Plan, defined as follows:

All variables and relations are sufficiently grounded, all fully grounded relations are satisfied, all sub-plans are sufficiently decomposed and all the mandatory goals can be achieved for at least one specific instantiation of the sufficient plan.

The underlying concepts of this definition are:

- Sufficiently grounded: All decisions $d \in D$ and relations $r \in R$ that appear as goals in the problem must specify whether they should be grounded or not at planning time. A cd is grounded when its value and parameters are grounded; a relation is grounded when all its elements are grounded. A partially grounded relation has two important consequences: (1) the relation cannot be satisfied, (2) in case of temporal relations, the resulting dn is partially ordered.
- Sufficiently decomposed: A complex goal should also specify whether it must be or not fully decomposed. It is fully decomposed if all its $sub - cd$'s are fully decomposed and partially decomposed in other case.
- Valid plan: If there is one instantiation of the partial plan where every decision and relation can be grounded, all constraints satisfied, all sub-plans can be fully decomposed and all mandatory goals are achieved.

This represents an extension of the definition provided in (Frank and Jónsson 2003), where a partial plan is fully defined up to a certain point called plan horizon, ignoring activities that fall outside it. In our case, any goal, decision or constraint might be partially defined according to an initial definition. These partially defined elements of the problem should be completed prior to the execution.

With this technique, it is possible to generate plans under situations in which the information about the system or the environment is not complete.

QuijoteExpress

QuijoteExpress extends the AP^2 temporal planner developed in the frame of the ESA Goal Oriented Autonomous Controller Study (GOAC) with the properties presented in the previous section. It is divided in a strategic planner that leads the search and four resolvers used to fix the plan (see Figure 3):

- Unfolder: Add/remove elements (decisions or relations) from the problem in order to make a goal valid.
- Scheduler: Guarantees the consistency of the timelines and resource consumption.
- Timeline completer: Fill in the gaps of the timelines.
- Decomposer: Decomposes a complex goal into sub-goals. The problem can be evolved to different layers, being each layer more detailed than the previous one.

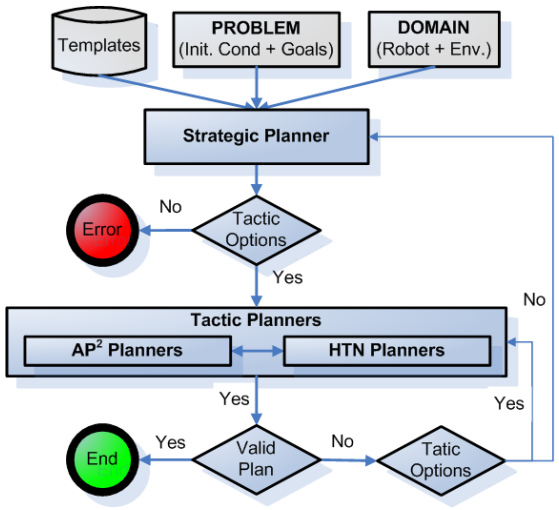

Figure 3: QuijoteExpress structure

QuijoteExpress follows an iterative approach. Given a problem and model as inputs to the planner, it first divides the problem in its independent components and assign each one to a tactical-solver. If all the components are solved, then they are joined and the relations between them repaired in case it is necessary (see Algorithm 1).

For each sub-problem, the tactical-solver first computes the flaws, and then iterates selecting non-deterministically a flaw and a resolver for it. At this point, the planner has constructed an evolution of the problem. In case the problem is not sufficiently decompose, it proceeds to decompose all complex goals in sub-goals, possibly introducing new flaws that will need to be planned (see Algorithm 2).

Figure 4 represents the search tree generated by the planner. Red nodes represent problems with flaws, labeled p^f, blue nodes represent problems not sufficiently decomposed

Algorithm 1: $QuijoteExpress(searchSpace, \delta)$

begin
- **while** $(\neg endCondition)$ **do**
  - $\pi \leftarrow chooseNode(searchSpace)$
  - $\pi[] = calcSeparationVertex(\pi)$
  - $[parallel]$
  - **for** $each \pi[]$ **do**
    - $\pi^1[i] \leftarrow TacticalSolver(\pi[i], \delta)$
  - $\pi^2 \leftarrow joinComponents(\pi^1)$
  - $\pi^{valid} \leftarrow TacticalSolver(\pi^2, \delta)$
  - $calculateScore(\pi^{valid})$
  - **if** $(\pi^{valid}$ *is Solution*$)$ **then**
    - $solutions \leftarrow \pi^{valid}$
  - $\pi^{decomposed} \leftarrow decompose(\pi^{valid})$
  - $searchSpace \leftarrow \pi^{decomposed}$
- **return** $chooseBest(solutions)$

Algorithm 2: $TacticalSolver(\pi, \delta)$

begin
- $flaws \leftarrow OpenGoals(\pi) \cup Threats(\pi)$
- if $flaws = \emptyset$ then return(π)
- select $\phi \in flaws$
- select $r \in Resolvers$
- if $r = \emptyset$ then return($failure$)
- $\pi' \leftarrow r.solve(\pi, \delta, \phi)$
- return $TacticalSolver(\pi', \delta)$

p^{nd}, dark green nodes represent problems sufficiently valid p^{sv}, that is, sufficiently valid plans and light green nodes represent full-valid plans. Each node contains an evolution of the problem and some extra information, including a score assigned by the planner that helps to lead the search or choose between different solution nodes. The edges represent the transition from a less-evolved version to a more-evolved one and contain information about the modifications undertaken. The root represents the initial problem. Each layer is divided in two sub-layers, one that generates fixed problems but not sufficiently decomposed and a second which decomposes the previous one at the cost of introducing new flaws. The planner can stop following different criteria, returning a valid solution if it has found either a p^{sv} or p^v. p^v will be always chosen over p^{sv} regardless of their scores, while the score will be used in case there are several p^v's or in case there are only p^{sv}.

Future work

What to do first, either decomposing a layer or fixing it has not yet been determined. Heuristics to choose the next flaw must be improved and backtracking techniques to the appropriate layer in case no solution is found implemented.

References

Dechter, R., and Pearl, J. 1987. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence* 34:1–38.

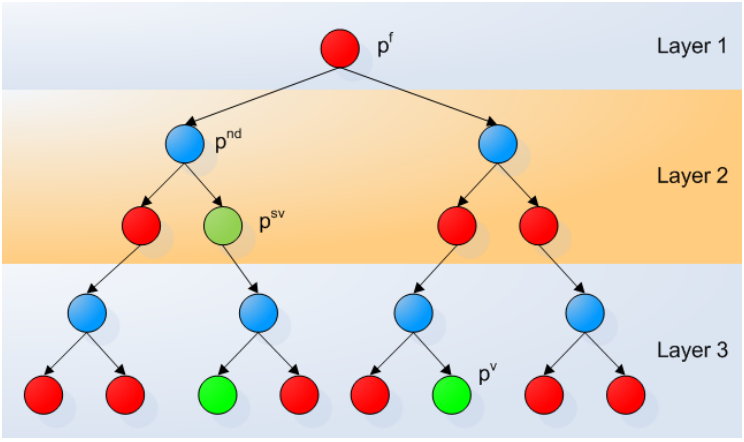

Figure 4: Search Tree

Erol, K.; Hendler, J.; and Nau, D. S. Htn planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94*, 1123–1128. AAAI Press.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Constraints* 8:339–364.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.