Design Concepts for a new Temporal Planning Paradigm

J. M. Delfa Victoria*
University of Surrey, UK

Nicola Policella
ESA-ESOC, Germany

Yang Gao
University of Surrey, UK

Oskar von Stryk
Technische Universität Darmstadt, Germany

Abstract

Throughout the history of space exploration, the complexity of missions has dramatically increased, from Sputnik in 1957 to MSL, a Mars rover mission launched in November 2011 with advanced autonomous capabilities. As a result, the mission plan that governs a spacecraft has also grown in complexity, pushing to the limit the capability of human operators to understand and manage it.

However, the effective representation of large plans with multiple goals and constraints still represents a problem. In this paper, a novel approach to address this problem is presented. We propose a new planning paradigm named HTLN, intended to provide a compact and understandable representation of complex plans and goals based on Timeline planning and Hierarchical Temporal Networks. We also present the design of a planner based on HTLN, which enables new planning approaches that can improve the performance of present real-world domains.

1 Introduction

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 this paper we consider the planetary rover as an example of a real-world problem dealing with critical operations, uncertainty and complex systems and goals that can be easily generalised to many other real scenarios on Space and Earth like rescue robots or autonomous vehicles.

The scope of this work is to define a new timeline planning paradigm for real-world scenarios such as the rover-world problem, aimed to manage temporal problems with uncertainty. The objective is to create a Planning & Scheduling System (PSS) able to generate robust plans for execution, or more specifically, responsive to the uncertainty and dynamics of the environment. A second objective is to produce more understandable plans for human experts.

Regarding its design, the PSS must retain sufficient generality in order to be used to design a knowledge-based, domain-independent timeline planner which can take advantages from different planning techniques such as HTN, CSP or MC.

After studying the problem, we have identified a number of key planning technologies from both the academic and applied worlds that represent the ingredients of a new planning paradigm called Hierarchical Timeline Networks (HTLN). It is based on hierarchical hypergraphs to represent the structure of problems, where complex goals in the upper layers of the hierarchy are decomposed in more specific subgoals, grouped together in sub-hypergraphs in lower level layers.

The paper is organised as follows: first, the planetary rover problem is described, then the proposed approach is discussed and the mathematical background of HTLN is introduced. Next, the design of a planner based on HTLN paradigm is presented. The paper concludes with some final remarks and a discussion of future work.

2 Planning for Autonomous Rover Missions

The planetary rover is a type of robot equipped with a locomotion system, typically wheeled, to move across hazardous terrain. Its hardware is divided between *payload* and *platform*. The former includes all the instrumentation dedicated to perform science, while the remaining sub-systems in support of these activities are considered the platform. They can serve different purposes both on Earth and space, such as rescue missions, surveillance or planetary exploration.

We use as a reference a planetary rover scenario with a single robot that must perform a traverse through uneven, unknown terrain towards a target, which can be a rock or geographical feature. The rover then performs a number of scientific activities such as taking pictures or studying the chemical composition of samples extracted with a driller.

This scenario shares with other problems in the robotic domain a number of specific characteristics, listed in Table 1, making it very hard from a planning point of view.

In this context, additional effort is required in the management of the plan complexity for two reasons. First, as the complexity of plans increases, the capability of humans to understand and manage them decreases. However, human operators need to understand the outcome of the planners and the reasons leading to generate it. This problem can be addressed by making the planner goal-based

Property	Description	Field
Uncertainty - Dynamic environment	The environment can spontaneously change its state due to external events	Both
Uncertainty - Partial observability	Some aspects of the state of the world are unknown. It has three consequences: Planning based in a complete understanding of the world is not feasible, some of the assumptions considered during planning might be wrong and new relevant information for the plan might be discovered only during execution time	Both
Uncertainty - Non determinism	Not possible to estimate with precision the outcome of the robot actions	Both
Hw/Sw/Problem complexity	The complexity of space missions has increased exponentially (Dvorak 2009; Bajracharya, Maimone, and Helmick 2008; Aghevli et al. 2006; McCurdy 2009)	Both
Highly constrained	Robotic operations required highly constrained models to avoid malfunctions	Both
Restricted communications	Some scenarios do not allow continuous or real-time communications with the robot. For example, the round trip of a radio signal to Mars takes around forty minutes	Both
Low CPU performance	Space-oriented processors have much lower performance than those integrated in conventional computers (Berger 2009)	Space
Safety	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	Space

Table 1: Properties of autonomous robots exploration on space and Earth

(Dvorak, Amador, and Starbird 2008; Dvorak et al. 2007; Morris et al. 2006), where goals are organised in a hierarchical structure that contains different levels of abstraction (Ghallab, Nau, and Traverso 2004). Navigating deeper in the structure allows the operator to learn how complex goals decompose in sub-tasks. Second, algorithms might be also benefited, as it is easier to isolate and fix parts of the plan that fail as a group. Different techniques such as intelligent backtracking and fast-forwarding can be used in this context.

A higher level of autonomy is crucial to increase the science return and decrease mission costs at the same time (Muscettola et al. 1998; Chien et al. 2006; Davies et al. 2006). However, autonomy comes at a price: the higher the level of autonomy desired, the higher the complexity of the system and level of detail in the knowledge to be added. The problem is that during the design of a planning system, part of the knowledge of the human experts is not captured. For this reason the system should be able to collaborate with experts in a mixed-initiative strategy (Bresina et al. 2003; 2005), where humans can manually add new elements to the problem and force the planner to satisfy them.

Finally, it must be possible to express the temporal evolution of events and actions. Temporal planners combined with CSP techniques have demonstrated to be very effective for problems involving dynamic environments and have become the main technology in the space domain (Frank and Jónsson 2003; Fratini, Pecora, and Cesta 2008; Ghallab and Laruelle 1994). The plans generated must be flexible to increase the robustness for execution under uncertainty. As a consequence, the PSS should be able to generate partially defined plans to be further completed once the information required is available, possibly at execution time, based on the least commitment principle.

3 Related work in AI Planning

Even though the techniques mentioned before are suitable to address some of the problems presented in Table 1, none of them completely satisfies all the characteristics and requirements described above. In the specific case of space robotics, the high cost of a mission plus the fear of losing the spacecraft due to software malfunctions have so far prevented a deeper integration of these technologies with the exception of two notorious attempts: Deep Space 1 (Muscettola et al. 1998; Jonsson et al. 2000) and Earth Observation 1 (Chien et al. 2004; 2005).

Significant work has been conducted in the field of CSP planning, where different versions of arc and path-consistency algorithms have been used in several planners (Mackworth 1977; Bessière 1994; Mohr and Henderson 1986; Singh 1995). Even though stating the planetary rover as a pure CSP problem is possible, this is not straightforward and can result in a complex representation. For this reason, alternative techniques should be taken under consideration to represent time, uncertainty or goal decomposition.

With respect to hierarchical task networks, HTN planners have been successfully applied in real problems (Wilkins and desJardins 2000) such as SIPE-2 (Wilkins et al. 1995) or O-Plan (Tate, Drabble, and Kirby 1994) but they show some limitations in dealing with uncertainty as well as temporal domains and do not allow interaction with human experts. Even though SIPE-2 can define vagueness with respect to interval relations in terms of minimum and maximum duration, our objective is to have a more powerful mechanism to represent partial plans.

Regarding timeline planners, there are several examples that have been used in the space sector, such as Aspen (Chien et al. 1997; Fukunaga et al. 1997), Europa (Frank and Jónsson 2003) or IxTet (Ghallab and Laruelle 1994; Laborie and Ghallab 1995), but they are not oriented to uncertainty or do not provide hierarchical task representation.

For planning under uncertainty two techniques have been widely used: Markov Decision Processes (MDP) (Cassandra, Kaelbling, and Littman 1994; Boutilier, Dean, and Hanks 1999) and Model Checking (MC) (Clarke, Long, and McMillan 1989; Bertoli et al. 2001). The rover problem presents uncertainty in all its possible dimensions (Table 1) making it a crucial aspect of the PSS design. By assigning costs and rewards, it is possible to represent desires about goals, while non determinism is expressed by means of probabilities assigned to the different choices available to achieve a goal. However, in MDP the policies (pairs states-actions) are defined beforehand, which do not represent a good approximation for real scenarios where the number of states might be infinite. It seems better to provide a model of the system and let the planner calculate how to achieve a specific state, taking into account the model in a similar way to MC, where policies only contain the states involved in transitions to achieve the goals.

4 Theoretical background

HTLN relies on the idea of merging timeline planning and HTN 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 (Figure 2).

4.1 Timeline planning principles

In APSI, a PSS 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.

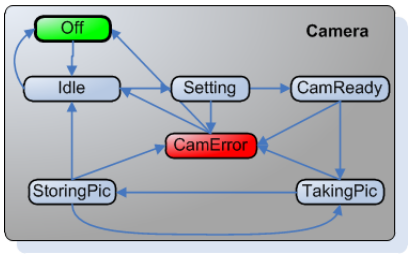


Figure 1: Component Camera automaton

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 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.

4.2 Hypergraph theory

Hypergraphs have been widely studied and applied in different areas such as the definition of complex data structures or optimization (Rugg 1983; Berge 1990; Gallo et al. 1993). They combine graph and set theories (Rugg 1983); it is a generalization of a graph where a hyperedge can connect any number of nodes and a set whose elements are the nodes it connects. It is formally defined as $H = (N, E)$, where N is the set of nodes/hypernodes and E is the set of hyperedges. The model of hypergraph used here diverges from the traditional as it contains in addition a second way for grouping nodes (besides the hyperedge) called hypernode (Damaschke 2009). While hyperedges are used to specify relations (such temporal or parameter relations) between nodes, a hypernode is used to represent the group of nodes in which a complex goal is decomposed. Therefore, a hypernode can be managed by the planner as a single node (seen as a black box) or as a sub-problem.

HTLN structure is organised horizontally with nested hypergraphs and vertically by mean of directed hierarchical hypergraphs.

The former are used to represent the decomposition of a complex element (problem or goal) in sub-elements. A super-hypergraph can contain several sub-hypergraphs and so forth until all the elements of each sub-hypergraph are simple. The idea behind this structure is to help the planner to manage all types of element (complex or simple) in a common way. However, this structure is not sufficient to represent the evolution of a problem. During the planning process, each complex goal is replaced by one of its possible decompositions (sub-goals). It is important to keep track of this process to backtrack in case the planner finds no solution. To represent this relation, a tree data structure seems to be more appropriate than nesting hypergraphs. Nevertheless, a tree can be generalised as a directed hypergraph removing the constraints by which nodes must have at most one parent and no cycles allowed. The resulting structure is a directed hierarchical set of hypergraphs. Each hypergraph has a set of parents (except the root) and children (except the leaves), where a parent represents a less evolved version of the child.

Figure 2 represents the hierarchical hypergraph structure. Ellipses represent problems and sub-problems while circles represent goals. Arrows represent the decomposition of goals into sub-goals while lines represent binary relations (temporal and parameter). For the sake of simplicity, there is no relation involving more than two nodes. The nomenclature used (represented as labels in the figure) is the following:

- p_i and p_{i+1}: Problem at level i and its decomposition
- cd^s: Simple node (definition in Section 5)
- cd^c: Complex node (definition in Section 5)
- $d^{\uparrow}$ and $d^{\downarrow}$: Node to be decomposed (parent) and its decomposition (child). Formally: $d^{\uparrow}(to) = d^{\downarrow} \wedge d^{\downarrow}(from) = d^{\uparrow}(to)$
- dn^{sup} and dn^{sub}: $super - dn$ and one of its $sub - dn$'s
- r_{dec}: Decomposition relation, where the source is a cd^c and the target is a $sub - dn$. Even though a cd^c can be decomposed in different $sub - dn$'s, just one is used in the problem, being the rest inactive $d^{\downarrow}$
- r_{temp}, r_{param}: Temporal and parameter relations respectively

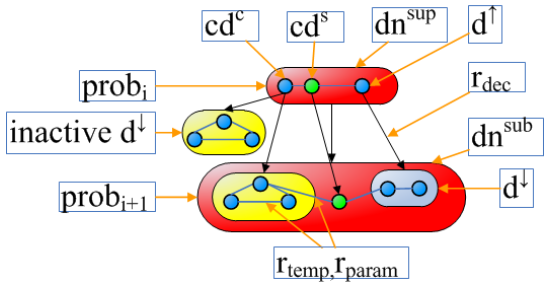


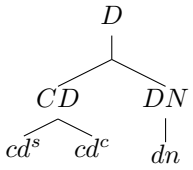
Figure 2: HTLN structure

Following section will present the novelties of HTLN and some formal definitions.

5 Hierarchical Timeline Networks (HTLN)

HTLN extends the APSI formalism in the following areas:

- Uses a HTN approach, allowing the definition of complex goals and its decomposition in sub-plans. HTLN defines three types of decisions which organisation is displayed in the tree bellow:
  - cd^s: Simple component decision that can be directly "executed". It is represented as a node of the graph that cannot be further decomposed
  - cd^c: Complex component decision that must be decomposed in order to be "executable". It is represented as a node of the graph, that is, a node with a set of decomposition relations
  - dn: A decision network represents a special type of decision represented as a hypernode
- Common representation for all relation types, extended as n-ary relations between sets of cd's
- HTLN allows partially defined, partially ordered plans in order to handle uncertainty (see Section 6)



5.1 Decision Network (dn)

A dn is represented as a hypergraph which nodes are a set of cd's and edges are the set of rlt's. In order to allow the definition of hierarchical structures, it can be decomposed in two different ways (see Figure 2):

- Vertical (level of abstraction): A parent dn ($dn^{\uparrow}$) represents a less evolved version of the child ($dn^{\downarrow}$), that is, $dn^{\downarrow}$ contains at least the decomposition of one $dn^{\uparrow}$ complex goal. Each dn can have several parents, as two $dn^{\downarrow}$ can be unified in one single dn, and several children, because a dn might be decomposed in different ways. There is one single dn at the highest level in the hierarchical structure, called $problem_0$, which represents the root of the structure, and several dn's at the lowest level, called $problem_i$ (where i is the level in the hierarchy) which represents the leaves of the structure. In case the planner finds a solution, it will be in any of the leaves of the structure
- Horizontal (nesting): A $super-dn$ (dn^{sup}) can be decomposed in an unlimited number of $sub-dn$'s (dn^{sub}), nested in different levels. At the same time, each dn can belong to several dn^{sup}, as different dn^{sup} can share common goals. There is one single dn at the highest nesting level, called $problem_i$, where i represents the hierarchical level and several dn's at the lowest level, called $sub-problem_{ij}$, where j represents the nesting depth. A dn represents itself a complex goal that can be managed as any other goal by the planner

  It is formally defined as follows:

$$dn = \Big\langle id, \{values\}, \{interval\}, \{nodes\}, \{edges\}, \{props\} \Big\rangle \quad (1)$$

  where:

- $\{values\}$: Values associated to the dn inherited from its parent $d^{\uparrow}$, in case dn represents the decomposition of a cd^c with value
- $\{interval\}$: Indicates the range of minimum and maximum time that the execution of this dn should take. It is either defined as a time horizon for a dn that represents a problem, inherited from its parent (as it happens for $\{values\}$ or derived from its $elements$ in case it represent a partial problem
- $\{nodes\}$: Set of cd's and dn^{sub}'s involved in the dn
- $\{edges\}$: Set of rlt's in which either the dn or any of its elements are involved
- $\{props\}$: A dn inherits the properties of cd's. The value of each property is assigned or retrieved with a specific function, formally defined as follows: $f_{property}(dn)$

A dn is partially ordered in case any of its cd's are not ordered by means of temporal relations with a predecessor and successor. The predecessor of the first element and the successor of the last element is their dn, related by means of $Starts$ and $Ends$ temporal relations respectively (see Algorithm 4). During the specification of a problem or during planning time, a dn can be added or deleted by the planner or human user.

5.2 Component Decision (cd)

A cd on a $component$ defines that a state of the component automaton holds for a given interval of time and can be used to express goals and constraints. It is formally defined as follows:

$$cd = \Big\langle id, type, \{values\}, \{rlts\}, \{props\} \Big\rangle \quad (2)$$

where:

- id: Identifies uniquely the cd
- $type$: Type of cd as defined in Section 5
- $\{values\}$: Each state has associated a value defined over a qualitative or quantitative domain. Notice that the same cd representation is used to represent actions of a component or consumption/production of certain resource
- $\{rlts\}$: Set of relations that affect this cd
- $\{props\}$: A cd can have specific properties such as relevance, uncertainty, parameters, etc. which values are assigned or retrieved via specific functions, formally: $f_{property}(cd)$

Like in dn's, a cd can be added or deleted by the planner or human user.

***dn*'s as *cd*'s**

A decision network $dn_1^{\downarrow}$ that represents a decomposition of a goal $d_1^{\uparrow} \in CD^c$ inherits the values, relations and properties of $d_1^{\uparrow}$.

As a consequence, a planner can apply its normal operators over a set of cd's viewed as a special node called hypernode represented by the dn. This approximation makes the

Type	Constraint	Directed	Arity
Parameter	Yes	No	n-ary
Temporal	Yes	No	n-ary
Decomposition	No	Yes	n-ary

Table 2: Relations in HTLN

PSS more powerful, as it can perform planning over groups of cd's, and simpler, as no special code is required to handle dn's.

5.3 Relation (rlt)

A relation is used to describe a common property between some cd's. It is represented as a hyperedge that joins together the cd's involved in the relation. A $constraint$ is a special type of relation that limits the possible states in which a $component$ can be and the duration of this status. All rlt's in HTLN are n-ary in order to construct more understandable problems and plans for humans (Little and Ghafoor 1990; 1993).

A relation is formally defined as follows:

$$rlt = \left\langle id, type, \{elements\}, \{props\} \right\rangle \quad (3)$$

where:

- id: Identifies uniquely the rlt
- $type$: The relations supported by HTLN are showed in Table 2
- $\{elements\}$: A relation contains a number of elements $e_i \mid 0 < i \leq n$. A n-ary relation is applied iteratively to each consecutive pair of elements in the following way: $e_i\ rlt\ e_{i+1}, \forall i \in [0, n)$. An element is formally described as follows: $e_i = \left\langle cd, \{values\}, \{constraints\} \right\rangle$
  - cd: Component decision affected by the relation. It can be a cd^c, cd^s or dn
  - $\{values\}$: Values used to delimit the relation. Depending on the type of relation, these values will be used for a different purpose. As an example, in the case of temporal relations, the intervals of the relation are defined as a set of pairs of values
  - $\{constraints\}$: Each relation might have appended some constraints
- $\{props\}$: A rlt, as a cd, can have specific properties, formally: $f_{property}(rlt)$

Each of the relations are introduced in detail below.

Parameter (f_{param}). A parameter relation is used to constraint the value between the parameters of different cd's of the dn. It is represented as a n-ary non-directional constraint. Formally

$$rlt = \left\langle id, f_{param}, \{elements\}, \{props\} \right\rangle \quad (4)$$

where:

- f_{param}: There are two type of relations: $f_{param} \in (=, \neq)$
- $\{elements\}$: List of parameters involved in the constraint

Temporal (f_{temp}). In HTLN, temporal constraints represent a superset covering other types of constraints like transitions in conventional automata theory or synchronizations between components (Muscettola 1994). Temporal relations can be applied to any kind of $node$, that is, to dn's, cd^c's and cd^s's.

A temporal constraint is represented as a n-ary, non-directional constraint. Formally:

$$rlt = \left\langle id, f_{temp}, \{elements\}, \{props\} \right\rangle \quad (5)$$

where:

- f_{temp}: HTLN uses the set of Allen temporal relations: $Equals$, $Before\,[l, u]$, $Meets$, $Starts\,[l, u]$, $Ends\,[l, u]$, $Overlaps\,[l, u]$ and $Contains\,[l_1, u_1], [l_2, u_2]$, being $[l, u]$ a time interval, where l is the lower bound and u the upper bound. In the special case of $Contains$, the first interval defines the time relation between the starting points and the second the relation between the ending points
- $elements$: List of elements involved in the constraint

Decomposition (f_{dec}). It represents a relation that is applied not only to decisions $d \in D$, but also to other relations in order to generate a more evolved version of the problem. It is represented as a $1 : n$ directional relation from a decision $n_1 \in D$ to a set of decisions $\{n_2\}$. The type of n_2 depends on the type of n_1 as follows:

- $n_1 \in CD^s \Rightarrow n_2 \in CD^s$: All simple elements are just copied in the evolved version of the problem
- $n_1 \in CD^c \cup DN \Rightarrow n_2 \in DN$: A complex element, either a cd^c or dn will be decomposed as a dn

The decomposition relation is formally represented as follows:

$$rlt = \left\langle id, f_{prop}, \{\{from\}, \{to\}\}, \{props\} \right\rangle \quad (6)$$

The decomposition function relies in some supporting functions that are presented bellow:

- $n_1(from)$: Returns the list of parents of n_1
- $n_1(to)$: Returns the list of children of n_1
- $n_1(active)$: Indicates whether the decision n_1 is active or not
- $f_{sup}(n_1)$: Returns the list of $super - dn$ of n_1
- $f_{sub}(n_1)$: Returns the list of $sub - dn$ of n_1
- $x(nodes)$: List of nodes of x, where x might be a relation or decision network
- $d(edges)$: List of relations of a decision $d \in D$
- $d(params)$: List of parameters of the decision d
- f_{dec}: Decomposition function (which follows the rules defined before)
- $f_{temp}^{Starts}(n_1, n_2)$: Defines a $Start\_Start$ temporal relation between two decisions
- $f_{temp}^{Ends}(n_1, n_2)$: Defines an End_End temporal relation between two decisions

- $f_{search}(domain, n_1)$: Return the decomposition dn's for n_1
- $x \stackrel{+}{=} y$: Addition of the element y to the list $\{x\}$
- $x = y$: Assigns to x the value of y

In addition to the elements presented in Section 4.2, two more are used:

- $domain$: Model of the system containing a description of all components, its simple decisions $cd^s \in CD^s$, complex decisions $cd^c \in CD^c$, decompositions $dn \in DN$ and relations $rlt \in R$
- $\chi(rlt)$: Log that stores all the decompositions performed during planning in support of backtracking in case the planner cannot find a solution

The following algorithms are in charge of decomposing a dn into a more evolved version. This decomposition represents an extra move of the planner in order to transform a problem into a plan.

Algorithm 1: $f_{decNode}(d^{\uparrow}, domain)$

begin
- $d^{\downarrow}(active) = true$
- $f_{createDec}(d^{\uparrow}, d^{\downarrow})$
- $dn^{\uparrow} = f_{sup}(d^{\uparrow})$
- **if** $dn^{\uparrow} \neq$ *null* **then**
  - $dn^{\downarrow} = dn^{\uparrow}(to)$
  - $dn^{\downarrow}(nodes) \stackrel{+}{=} d^{\downarrow}$
- **else**
  - $dn^{\downarrow} = d^{\downarrow}$
- **if** $d^{\uparrow} \in DN$ **then**
  - $\forall d \in d^{\uparrow}(nodes), d \in D \Rightarrow f_{decNode}(d, domain)$
  - $\forall f_r \in d^{\uparrow}(edges), f_r \in \{R_{param}, R_{temp}\} \Rightarrow f_{decRel}(f_r)$

Decompose a decision $d^{\uparrow} \in D$ in $d^{\downarrow} \in D$. The algorithm is illustrated in Figure 3.

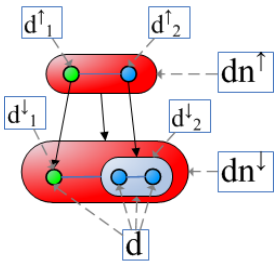


Figure 3: Decomposition of a dn

The algorithm receives as inputs the node to be decomposed ($d^{\uparrow}$) and the decomposition node ($d^{\downarrow}$), which will be selected in different ways depending on the type of $d^{\uparrow}$ (see 5.3:

- $d^{\uparrow} \in CD^s$ (simple decision): $d^{\downarrow}$ contains a copy of $d^{\uparrow}$
- $d^{\uparrow} \in CD^c$ (complex decision): A list of dn candidates to decompose $d^{\uparrow}$ is retrieved from $domain$. One dn is heuristically selected from this list
- $d^{\uparrow} \in DN$ (decision network): A new dn is created to store the decomposition elements of $d^{\uparrow}$

The decision $d^{\downarrow}$ selected for the decomposition is marked as active to distinguish it from the rest of options in which $d^{\uparrow}$ could be decomposed. Then a decomposition relation between $d^{\uparrow}$ and $d^{\downarrow}$ is defined. Following, the $super-dn$'s of $d^{\uparrow}$ and $d^{\downarrow}$, named $dn^{\uparrow}$ and $dn^{\downarrow}$ respectively, are identified. Notice that $d^{\uparrow}$ represents the problem p_i in case it has no $super-dn$. In this case, $d^{\downarrow}$ has been previously initialized to a new dn which represents the evolution of the problem called p_{i+1}. In case $dn^{\uparrow} \neq$ null, then $d^{\downarrow}$ is added to $dn^{\downarrow}$, that means, the decomposition node of $d^{\uparrow}$ is added to a problem that represents an evolution of the problem in which $d^{\uparrow}$ is defined. In case $d^{\uparrow}$ represents a dn, we further decompose each of its cd's and rlt's.

Algorithm 2: $f_{decRel}(f_r^{\uparrow})$

begin
- new $f_r^{\downarrow} \in R$
- $\forall d^{\uparrow} \in f_r^{\uparrow}(nodes)$
- **begin**
  - $d^{\downarrow} = d^{\uparrow}(to)$
  - $f_r^{\downarrow}(nodes) \stackrel{+}{=} d^{\downarrow}$
- $dn^{\downarrow}(edges) \stackrel{+}{=} f_r^{\downarrow}$

Decompose a relation $f_r^{\uparrow} \in R$. The algorithm is illustrated in Figure 4.

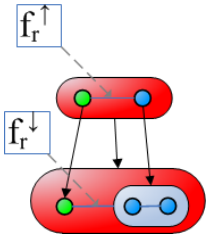


Figure 4: Decomposition of a relation

First, a new relation $f_r^{\downarrow}$ is created of the same type as $f_r^{\uparrow}$. For each node $d^{\uparrow}$ involved in $f_r^{\uparrow}$ we search its decomposition node $d^{\downarrow}$ in $dn^{\downarrow}$ and assign it to $f_r^{\downarrow}$. Once $f_r^{\downarrow}$ contains all the decomposition nodes of $f_r^{\uparrow}(nodes)$, the relation $f_r^{\downarrow}$ is added to the list of relations of $dn^{\downarrow}$.

Regarding parameter relations, the decomposition of $f_r^{\uparrow}$ in $f_r^{\downarrow}$ is very simple as $d_2^{\downarrow}$ will inherit the parameters of $d_2^{\uparrow}$. With respect to temporal relations (see Algorithm 4), the decomposition of $f_r^{\uparrow}$ in $f_r^{\downarrow}$ is equally simple, as $d_2^{\downarrow}$ will inherit the temporal relations (including the temporal duration) of $d_2^{\uparrow}$.

Propagate a relation $f_r \in \{R_{param}, R_{temp}\}$. During planning, two more algorithms will be required to propagate parameter and temporal relations between a dn and its cd's. In order to study how they propagate consider, without loss of generality, a relation like $f_r^{\uparrow}$ in Figure 4 involving two elements, one of which is a cd^c. The generalisation of this example to n-ary relations involving different cd^s's and cd^c's is straightforward.

Algorithm 3: $f_{propParamRel}(dn^{sup}, d^{new})$

begin
  $\forall par_i \in d^{new}(params) : \exists par_j \in dn^{sup} \wedge par_i = par_j$
  begin
    new $f_{param}^{equals} \in R_{param}$
    $f_{param}^{equals}(from) = par_i$
    $f_{param}^{equals}(to) = par_j$
    $dn^{\downarrow}(edges) \stackrel{+}{=} f_{param}^{equals}$

The propagation of parameter relations is presented in Algorithm 3. In case there is a parameter par_i in the new cd equal to a parameter par_j of the dn (for example, both refer to the speed of the rover), an $equal$ parameter relation is created between them and added to the set of relations of dn^{sup}.

Regarding temporal relations, in order to maintain the consistency between $d_2^{\downarrow}$ and its cd's, the temporal relation must be propagated any time a new element is added to $d_2^{\downarrow}$. Algorithm 4 takes care of it.

Algorithm 4: $f_{propTempRel}(dn^{sup}, d^{new})$

begin
  if $dn^{sup}(order)$ *change_to* $\neg fullyOrdered$ **then**
    $dn^{sup}(edges) \stackrel{-}{=}$
    $\{f_{temp}^{Starts}(dn^{sup}, n_{first}), f_{temp}^{Ends}(dn^{sup}, n_{last})\}$
    $dummy_1, dummy_2 =$ new $cd \in CD^s$
    $dn^{sup}(nodes) \stackrel{+}{=} \{dummy_1, dummy_2\}$
    $f_{temp}^{Starts}(dn^{sup}, dummy_1)$
    $f_{temp}^{Ends}(dn^{sup}, dummy_2)$
    $dn^{sup}(edges) \stackrel{+}{=}$
    $\{f_{temp}^{Starts}(dn^{sup}, dummy_1), f_{temp}^{Ends}(dn^{sup}, dummy_2)\}$
  else if $dn^{sup}(order)$ *change_to* $fullyOrdered$ **then**
    $dn^{sup}(edges) \stackrel{-}{=}$
    $\{f_{temp}^{Starts}(dn^{sup}, dummy_1), f_{temp}^{Ends}(dn^{sup}, dummy_2)\}$
    $dn^{sup}(nodes) \stackrel{-}{=} \{dummy_1, dummy_2\}$
    $f_{temp}^{Starts}(dn^{sup}, n_{first})$
    $f_{temp}^{Ends}(dn^{sup}, n_{last})$
    $dn^{sup}(edges) \stackrel{+}{=}$
    $\{f_{temp}^{Starts}(dn^{sup}, n_{first}), f_{temp}^{Ends}(dn^{sup}, n_{last})\}$

In case a new element is added to dn^{sup}, the status of dn^{sup} is checked. If it has changed to $\neg fullyOrdered$, the temporal relations between the first/last cd and dn^{sup} are removed and two dummy cd^s's are created to enforce the temporal relations between dn^{sup} and its cd's:

- $f_{temp}^{Starts}(dn^{sup}, dummy_1)$: Indicates that dn^{sup} will start at the same time as the first of its cd's
- $f_{temp}^{Ends}(dn^{sup}, dummy_2)$: Indicates that dn^{sup} will end at the same time as the last of its cd's

If the status of dn^{sup} has changed to $fullyOrdered$, the $dummy$ cd's are replaced by the first and last cd of dn^{sup} and the new relations $f_{temp}^{Starts}(dn^{sup}, n_{first}), f_{temp}^{Ends}(dn^{sup}, n_{last})$ are added to the list of edges of dn^{sup}.

6 Design of a planner based on HTLN

In this section we summarize the guidelines we identified in order to exploit the HTLN paradigm. Traditional timeline planners search for complete valid plans, which can be defined as follows: *All variables are grounded, the constraints are satisfied, all intervals have been sequenced and the plan has been fully specified to the end of time.*

This definition might represent an unachievable condition for real-world P&S systems. Our PSS 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 cd^c should also specify whether it must be or not fully decomposed. A cd^c 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, giving the responsibility to the executive to fill in the gaps prior to the execution.

During the plan expansion, the planner has to add (and eventually delete) cd's in order to justify the already existing goals in the network. As dn's are managed in HTLN like cd's, the planner can reason over groups of goals. In fact, the planner can add $sub - dn$'s stored in the KB using the same methods applied to cd's.

By reasoning over the underlying graph of an HTLN problem, it is possible to identify separate 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. A graph that has a separation vertex is called separable. Let $V' \subseteq V$, the induced subgraph $G' = (V', E')$ is called a non-separable component if G' is non-separable and if for every larger V'', $V' \subseteq V'' \subseteq V$, the induced subgraph $G'' = (V'', E'')$ is separable. An efficient algorithm for determining consistency 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 components are consistent, then the entire network is consistent, and the minimal networks of the individual components coincide with the overall minimal network. Taking advantage of HTLN structure, we can use this technique to isolate independent sub-problems dn^{sub} and perform parallel planning, where each planner takes care of a different independent sub-problem.

6.1 Planner design

In classical planning, two types of operators are used to build a solution from a partial plan: expansion (horizontally) of the partial plan and fixing flaws. However, in HTLN a plan is a hierarchical structure which divides the problem in different levels of abstraction. In order to generate a valid plan, the planner must generate a valid dn at each level of abstraction. Therefore, a planner for HTLN should have a third type of procedure, the decomposition, which expands the plan vertically.

At the same time, the planner requires the following properties for the cd's: $f_{isSGround}(cd)$, $f_{isSDec}(cd)$, which specify whether the cd is sufficiently grounded or decomposed, respectively, and $f_{active}(cd)$, that specifies which cd is active between a set of exclusive cd's. With respect to rlt's, $f_{isSatisfied}(rlt)$ indicates whether the relation is satisfied or not.

Given a problem and model as inputs to the planner (presented in Algorithm 5), it first divides the problem in independent sub-problems by means of the separation vertices. Each sub-problem is then evolved in parallel and added to the plan, which is refined while taking into consideration all the partial modifications.

The strategy to evolve a problem consists of two steps. First, the flaws (threads or open-goals) at one level of the problem are computed and fixed (if possible). Once the layer is valid, the problem is evolved to the next level decomposing a complex goal in subgoals. This task is undertaken by the $vDecomposition$ method shown in Algorithm 6, which decomposes cd's and rlt's using the algorithms shown in Section 5.3.

7 Conclusions and Future Work

Real-world problems, and particularly space robotics, manage complex and critical systems that present important divergences with respect to theoretical problems. In order to introduce automated P&S systems in this area, the experts need to understand and trust the outcomes of the planners.

Algorithm 5: HTLNPlanner(problem, domain, time)

```
begin
    while (time) do
        subdns = calcSeparationVertex(problem)
        [parallel]
        plan ± evolveP(subdns_i, domain, time)
        refinePlan(plan)
        if (plan.score > best.score) then
            best ← plan
    if (¬best) then
        return fail
    return solution
```

Algorithm 6: evolveP(problem, domain, time)

```
begin
    while (time) do
        flaws = computeFlaws(problem, domain)
        fixFlaws(problem, domain, flaws, time)
        if (flaws ≠ ∅) then
            return fail
        f_decNode(problem, domain)
        if (¬problem.isSDec ∨ ¬problem.isSGround)
        then
            return fail
        return problem
    return fail
```

At the same time, we have to increase the flexibility of planners and plans in order to increase its robustness during plan execution in uncertain environments with noise information. Finally, it is becoming crucial to provide instruments to represent and store the knowledge that will lead the planner through NP-complete search-spaces using incomplete and not sound algorithms. We consider that the way problems and plans are defined and represented constitutes a major concern that might help to address these issues. This is the reason that motivated the conception of HTLN as a new paradigm that defines complex goals as the building blocks of problems and provide advanced features to define and connect them. In the future, we are planning to extend the HTLN paradigm in order to include fuzzy temporal intervals and relations as a complementary tool to represent uncertainty in execution. A planner which exploits the HTLN paradigm and follows the guidelines presented in Section 6 is also under development.

Acknowledgements. This research has been co-funded by the Networking/Partnering Initiative (NPI) between ESA-ESOC and TU Darmstadt and by the Spanish Trainee program, funded by the Ministry of Spain. It also receives support from the German Research Foundation (DFG) within the Research Training Group 1362 "Cooperative, adaptive and responsive monitoring in mixed mode environments".

References

Aghevli, A.; Bachmann, A.; Bresina, J.; Greene, K.; Kanefsky, B.; Kurien, J.; McCurdy, M.; Morris, P.; Pyrzak, G.; Ratterman, C.; Vera, A.; and Wragg, S. 2006. Planning applications for three mars missions with ensemble. In *Proceedings of International Workshop on Planning and Scheduling for Space (IWPSS)*.

Bajracharya, M.; Maimone, M.; and Helmick, D. 2008. Autonomy for mars rovers: Past, present, and future. *Computer* 41(12):44 –50.

Berge, C. 1990. Optimisation and hypergraph theory. *European Journal of Operational Research* 46(3):297–303.

Berger, R. 2009. Development of electronics for use in outer space. Presentation.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artif. Intell.* 65:179–190.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH* 11:1–94.

Bresina, J. L.; Jonsson, A. K.; Morris, P. H.; and Rajan, K. 2003. Mixed-initiative activity planning for mars rovers. *Challenges* 2–3.

Bresina, J. L.; Jonsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Mixed-initiative planning in mapgen: Capabilities and shortcomings. In *Proceedings of the Int. Conf. on Planning and Scheduling (ICAPS) Workshop on Mixed-Initiative Planning and Scheduling*, 8.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, AAAI'94, 1023–1028. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F. W.; Barrett, T.; Stebbins, G.; and Tran, D. 1997. Aspen - automated planning and scheduling for space mission operations. *Jet Propulsion* 1–10.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Lee, R.; Mandl, D.; Frye, S.; Trout, B.; Hengemihle, J.; D'Agostino, J.; Shulman, S.; Ungar, S.; Brakke, T.; Boyer, D.; Van Gaasbeck, J.; Greeley, R.; Doggett, T.; Baker, V.; Dohm, J.; and Ip, F. 2004. The eo-1 autonomous science agent. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, 420–427. Washington, DC, USA: IEEE Computer Society.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castaño, R.; Davies, A.; Mandl, D.; Frye, S.; Trout, B.; D'Agostino, J.; Shulman, S.; Boyer, D.; Hayden, S.; Sweet, A.; and Christa, S. 2005. Lessons learned from autonomous sciencecraft experiment. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, 11–18. New York, NY, USA: ACM.

Chien, S.; Doyle, R.; Davies, A.; Jonsson, A.; and Lorenz, R. 2006. The future of ai in space. *Intelligent Systems, IEEE* 21(4):64 –69.

Clarke, E.; Long, D.; and McMillan, K. 1989. Compositional model checking. In *Logic in Computer Science, 1989. LICS '89, Proceedings., Fourth Annual Symposium on*, 353 –362.

Damaschke, P. 2009. Multiple hypernode hitting sets and smallest two-cores with targets. *J. Comb. Optim.* 18(3):294–306.

Davies, A. G.; Chien, S.; Doggett, T.; Ip, F.; and Catano, R. 2006. Improving mission survivability and science return with onboard autonomy. In *Proceedings of the Intl Planetary Probe Workshop 4, Jet Propulsion Laboratory,*.

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

Dvorak, D. L.; Amador, A. V.; and Starbird, T. W. 2008. Comparison of goal-based operations and command sequencing.

Dvorak, D. L.; Ingham, M. D.; Morris, J. R.; and Gersh, J. 2007. Goal-based operations: An overview. In *AIAA Infotech, Rohnert Park, CA*, 16.

Dvorak, D. L. 2009. Nasa study on flight software complexity. Technical report, NASA.

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.

Fukunaga, A. S.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Aspen: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space*.

Gallo, G.; Longo, G.; Pallottino, S.; and Nguyen, S. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics* 42(2-3):177 – 201.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 61–67.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Jonsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in Interplanetary Space: Theory and Practice. In *Artificial Intelligence Planning Systems*, 177–186.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of the 14th interna-*

tional joint conference on Artificial intelligence - Volume 2, 1643–1649. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Little, T. D. C., and Ghafoor, A. 1990. Synchronization and storage models for multimedia objects. *IEEE J. Sel. Areas Commun.* 8(3):413–427. NewsletterInfo: 38.

Little, T., and Ghafoor, A. 1993. Interval-based conceptual models for time-dependent multimedia data. *Knowledge and Data Engineering, IEEE Transactions on* 5(4):551 –563.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99 – 118.

McCurdy, M. 2009. Planning tools for mars surface operations: Human-computer interaction lessons learned. In *Aerospace conference, 2009 IEEE*, 1 –12.

Mohr, R., and Henderson, T. C. 1986. Arc and path consistency revisited. *Artif. Intell.* 28(2):225–233.

Morris, J. R.; Ingham, M. D.; Mishkin, A. H.; Rasmussen, R. D.; and Starbird, T. W. 2006. Application of state analysis and goal-based operations to a mer mission scenario. In *Proceedings of SpaceOps Conference, Rome, Italy*, 12.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: to boldly go where no ai system has gone before. *Intelligence* 103(1-2):5–47.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Rugg, R. D. 1983. Building a hypergraph-based data structure. *AUTOCARTO 6* 2:211220.

Singh, M. 1995. Path consistency revisited. In *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, TAI '95, 318–. Washington, DC, USA: IEEE Computer Society.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-plan2: an open architecture for command, planning and control. *Intelligent Scheduling* 1:213–239.

Wilkins, D. E., and desJardins, M. 2000. A call for knowledge-based planning. *AI MAGAZINE* 22:99–115.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental Theoretical Artificial Intelligence* 7(1):121–152.