Acta Futura 9 (2014) 21-29
DOI: 10.2420/AF09.2014.21

Acta Futura

Planning Mars Rovers with Hierarchical Timeline Networks

Juan M. Delfa Victoria[*1,2,3], Simone Fratini[2], Nicola Policella[2], Oskar von Stryk[1], Yang Gao[3] and Alessandro Donati[2]

[1] *Technische Universität Darmstadt, 64289 Darmstadt, Germany*

[2] *European Space Agency, 64293 Darmstadt, Germany*

[3] *University of Surrey, GU2 7XH Guildford, United Kingdom*

Abstract. Surface operations in distant bodies like Mars present a lot of challenges. Among them, lack of direct communications and a complex environment are the main drivers that push for more autonomy both on-ground and on-board. This paper presents a timeline, hierarchical, heuristically-based and domain-independent planner called QuijoteExpress oriented to solve highly constrained temporal problems.

1 Introduction

Even though automated planning has been used since long in practical problems like traffic control in cities, package transportation or spacecraft operations, many recent scenarios such as Fukushima, the on-going Darpa Robotic Challenge or the Curiosity rover are not using planning techniques beyond path planning.

In Fukushima, robots were mainly tele-operated due to the high complexity of the scenario, but limited communication in some areas heavily shielded, lack of situation awareness or difficult synchronization between the human operators (for some robots an operator guides the robot-base and another the arm) were part of the reasons for the poor performance displayed [12]. In the DRC, automated planning is optional and most teams have focused in control under human supervision, where the operator performs the action in a virtual world and the movements are sent then to the "real" robot, with the exception of walking, for which some teams implemented autonomous systems. Finally, Curiosity plans are generated in a highly manual process involving a huge human team. As in DRC, the most remarkable autonomous system is the Autonav.

In this paper, a new planner called QuijoteExpress is described. The motivation is to provide a planner ready to be used in real-world scenarios. More specifically, we try to fulfil the following goals:

1. Performance: Improve the performance of the planner is key to achieve a good scalability as the complexity of the problem grows and to be able to quickly react by means of re-planning to changing conditions.

2. Handle uncertainty: Due to the lack of information in partially-observable, stochastic and dynamic environments such as Mars or noise coming from faulty sensors, sometimes problems cannot be fully stated. A planner for such domains should be able to generate valid plans in these conditions.

3. User friendly: The communication between the user and the planner is critical in real systems.

[*]Corresponding author. E-mail: delfa@sim.tu-darmstadt.de.

Users should be able to define problems in terms of high level goals that hide the complexity and planners should provide plans easy to understand and verify.

4. Domain-independent: Even though we focus our research on planetary rovers, QuijoteExpress is also suitable for any kind of highly constraint scenario with temporal requirements. The approach is to develop domain-independent search algorithms, encoding the specific knowledge in both Hierarchical Task Network [8] (HTN) *methods* and dedicated heuristics whenever it is required. In this way, knowledge is perfectly isolated and reusability is improved.

QE incorporates two novelties oriented to achieve these goals. First, Hierarchical Timeline Networks (HTLN [6]) are intended to improve the planner performance and the way users interact with the planner (see next section). Second, it can generate partial plans to handle uncertainty.

The rest of the paper is organized as follows. Section 2: short overview of the concepts introduced in HTLN. Section 3: describes the rover scenario used during the tests. Section 4: description of the planner QuijoteExpress and the concept of sufficient plan. Section 5: initial results in comparison to AP^2. Section 6: conclusions and future work.

2 Background

QuijoteExpress is based on APSI*, and extension of APSI [11] that combines temporal and HTN planning in a new planning paradigm called HTLN [7]. The main concepts of HTLN are briefly introduced in the appendix at the end of the paper. Central to HTLN and QuijoteExpress is the way in which complex goals[1] are represented and managed. Most HTN planners like SIPE-2 [14] or O-Plan [13] replace in the problem the complex task by the set of sub-tasks of the method selected. In consequence, some information is lost as the problem does not retain the hierarchical structure of the domain. This information can be in fact useful for a given resolver in charge of fixing flaws of the problem or during backtracking as we will see later in this section.

[1]The conventional name in HTN nomenclature is compound. Along the paper we will use complex instead, as we think it is more representative

QE is based on the construction of hierarchical structures rather than goals replacement, adding new levels of detail as complex tasks are refined into sub-tasks [6]. Given a problem represented as a *decision network* $\mathtt{dn}$ (a hypergraph), it is composed by a set of *decisions* (the nodes), each containing a value and a list of parameters, and *relations* (the hyperedges) among them. The decisions can be complex ($\mathtt{d}^{\mathtt{c}}$) or primitive ($\mathtt{d}^{\mathtt{p}}$). Primitive decisions are always represented in HTLN as component decisions ($\mathtt{cd}$), that is a simple node in the problem graph or in other words constraints on timelines. A $\mathtt{cd}$, that is, a simple node. However, a complex goal $\mathtt{d}^{\mathtt{c}}$ can be represented as a component decision $\mathtt{cd}$ in case it is not decomposed, or as a decision network $\mathtt{dn}$ otherwise. In the last case, $\mathtt{d}^{\mathtt{c}}$ retains the value and parameters of the original (not-decomposed) decision plus the sub-network ($\mathtt{dn}_{\mathtt{dec}}$) which contains a list of sub-tasks and relations among them. A planner can use a decomposed $\mathtt{d}^{\mathtt{c}}$ as an atomic decision ignoring the details, or as a sub-problem. This approach provides several advantages:

- Better constraint propagation: As a $\mathtt{cd}$, single modifications on $\mathtt{d}^{\mathtt{c}}$ affect directly the whole sub-network. In the particular case in which $\mathtt{d}^{\mathtt{c}}$ is *self-contained*, that is, no sub-task $\mathtt{d}^{\mathtt{sub}}$ of $\mathtt{d}^{\mathtt{c}}$ is related to any decision out of $\mathtt{d}^{\mathtt{c}}$, then no constraint propagation will be required in case a relation involving $\mathtt{d}^{\mathtt{c}}$ is modified. For example, changing the position of $\mathtt{d}^{\mathtt{c}}$ will also change the position of all its sub-tasks, as they are ordered relatively to $\mathtt{d}^{\mathtt{c}}$. Besides, it also represent a powerful technique to handle uncertainty. In case it is unknown how to achieve a complex goal $\mathtt{d}^{\mathtt{c}}$ during planning time, $\mathtt{d}^{\mathtt{c}}$ is modelled as a $\mathtt{cd}$. During execution, once the required information is available, a decomposition method is chosen for $\mathtt{d}^{\mathtt{c}}$ and the plan is completed.

- Parallelism: If the problem has a $\mathtt{d}^{\mathtt{c}}$ which is an articulation point of the hypergraph, that is, the hypergraph is divided in two if this node is removed, then $\mathtt{d}^{\mathtt{c}}$ sub-network ($\mathtt{dn}_{\mathtt{dec}}(\mathtt{d}^{\mathtt{c}})$) can be planned in parallel. Moreover, if the result is optimal for each $\mathtt{dn}_{\mathtt{dec}}$, the overall solution will be also optimal [6].

With respect to relations, HTLN supports temporal relations such as $\mathtt{drive}\ \mathtt{before}[\mathtt{l},\mathtt{u}]\ \mathtt{communication}$ [1] and parameter relations such as $\mathtt{pointing}_{\mathtt{camera}} = \mathtt{position}_{\mathtt{target}}$.

 DOI: 10.2420/AF09.2014.21

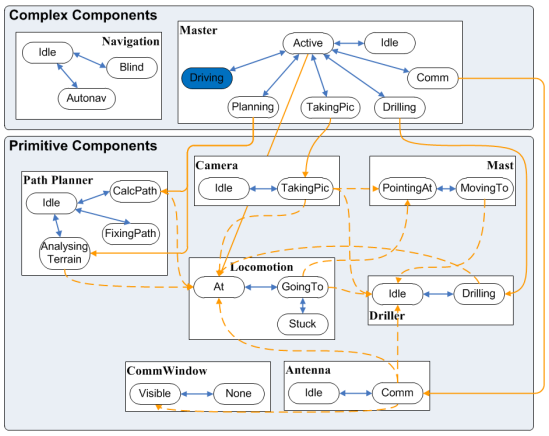

Figure 1: Hierarchical rover model

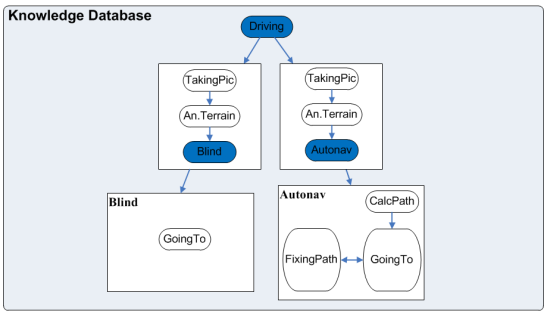

Figure 2: Knowledge database for a Mars Rover

3 Scenario

The rover is a type of robot equipped with a locomotion system 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.

The same problems that make planetary rover missions very challenging form a planning point of view such as uncertain environments, highly constrained plans or no real-time communications represent the main arguments in favour of more autonomy. Given the increasing complexity of future missions, the advantages in terms of science return and costs are undeniable. As an example, MER would have never been so successful without the AUTONAV system [2]. While autonomous navigation is already well understood, other aspects like opportunistic science start to be addressed by new automated systems like AEGIS [9].

Figure 1 presents the model used for our experiments. It emphasizes two properties: a hierarchical structure and complex inter-dependencies between different components of the robot.

The components are divided in two categories. Those primitive are used to model real rover sub-systems such as the antenna or the driller, while the complex are used to model complex behaviours that involve different primitive components. An extensive description of each component and constraint is out of the scope of this paper. However, some components require a brief description.

Within the primitive components, Camera, Mast, Locomotion, Driller and Antenna represent the main rover hardware. Locomotion and Driller are specially relevant, as they impose a lot of constraints to the rest. CommWindow models the communication windows with a satellite, which are imposed as constraints to the planner.

With respect to complex components, Master represents a set of the main activities the rover can accomplish. Even though in this example most of them have a 1:1 mapping with primitive components, Driving and Idle are special. Idle requires all the components to be in the Idle or equivalent initial state while Driving is hierarchically decomposed in either Blind or Autonav as showed in figure 2. Both require as initial step to take a picture of the surroundings and analyse the traversability of the terrain. In case the terrain is easy, the rover blindly goes directly to the target. In case the terrain is complex, a path with intermediate waypoints is calculated and, while driving, it is continuously corrected with information gathered from the sensors. These decomposition recipes are stored as `methods` in the Knowledge Database (`kdb`).

4 QuijoteExpress

QuijoteExpress (QE) is a timeline, hierarchical, heuristically-based and domain-independent planner which extends the AP^2 [4] planner developed in the frame of the ESA Goal Oriented Autonomous Controller Study (GOAC [3]) and is backwards compatible with AP^2, thus allowing QE to run already existing domains. QE is organised in two packages: Planners and Heuristics. The planners are themselves divided in

a strategic and a tactical planner, the last one composed of four *resolvers* ρ, each dedicated to solve a specific type of flaw ϕ. The heuristics are organised in four groups: choose the next SolvingSpaceNode, choose the next flaw in a given SolvingSpaceNode, choose a resolver and choose a decomposition method for a complex goal.

Three resolvers, Unfolder, Scheduler and TimelineCompleter are inherited from AP^2 with some modifications. The unfolder is in charge of adding the supporters of a given goal by applying the domain theory. The scheduler is in charge of adding ordering constraints in the form of temporal constraints while the timeline completer is called to fill up the holes of the timelines, that is, the parts where no decision has been yet assigned.

The rest of this section focuses on the description of the two new planners: the strategic and decomposer.

4.1 QE-Strategic

This planner is in charge of guiding the search. It has been designed as a Producer-Consumer, each running in a separate thread to favour parallel computing thanks to its flaw-oriented approach. The producer is in charge of deciding the next SolvingSpaceNode to be solved, while the consumer analyse the partial solutions provided by the resolvers.

The algorithm 1 shows the cycle of the producer.

```
begin
    while (¬exit_cond) do
        if (pendingJobs(shared_info)) then
            wait_to_finish()
        else
            node ←
            sel_next_solv_node(search_space)
            ρ ← sel_next_resolver(node)
            create_thread(ρ, node, domain)
```

Algorithm 1: producer(search_space, domain, exit_cond)

Given a search-space, the producer chooses the deepest node in the tree as it is expected to be the most evolved and closer to the solution. However, in the future it will be replaced by an A* algorithm with an heuristic based on the number of pending flaws to be resolved. Once the SolvingSpaceNode is selected, a resolver is chosen. Given the fact that each resolver tries to solve all the flaws of its type in the node, the selection follows a cycle depicted in figure 3. First, the Unfolder add new sub-goals to support the goals already existing in the problem and calls the scheduler to order them. Once it can't go any further, the decomposer is called to decompose all complex-goals that must be decomposed (a complex goal might not be required to be decomposed as explained in section 4.2). When the decomposer finishes, the unfolder is called back to add supporters for the new sub-goals. This process iterates until the point where a partial solution is found, in which case the TimelineCompleter is called, or exit_cond becomes true, in which case the algorithm exits. In case the TimelineCompleter adds new sub-goals, then the unfolder is called back and the process is repeated. The exit_cond of the main loops becomes true when there is no pending nodes (partial solutions) in the search space and there is no pending jobs, that means, no thread is still running.

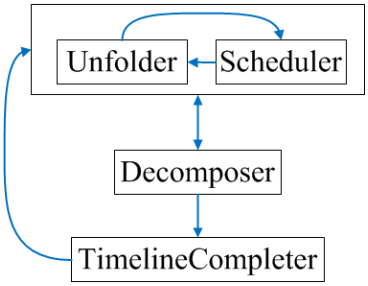

Figure 3: Cycle of calls to the different resolvers

```
begin
    while (¬exit_cond) do
        solution ← wait_next_solution()
        node ← create_new_node(solution)
        add_to_search_space(node)
```

Algorithm 2: consumer(search_space, domain, exit_cond)

The consumer (algorithm 2) waits in a loop until any solver produces a new solution. Equally to the producer, the consumer exits the loop when there is no pending solutions to be managed nor any pending job. Once a job ends and a solver produces a solution, the consumer analyse it. In case it is partial, a new node is created and added to the search space. Otherwise, it is added to the list of solutions.

To guarantee the consistency of the information, both producer and consumer share a thread-safe data struc-

 DOI: 10.2420/AF09.2014.21

ture containing information used by both of them, including the search space, list of pending jobs and solutions.

4.2 QE-Decomposer

QE-Decomposer is in charge of decomposing high level goals into sub-goals.

```
begin
    for (all goals to decompose) do
        goal ← goals
        methods ←
        get_decompositions(goal)
        dn_dec ← select_candidate(methods)
    solution ← create_solution(dn_dec)
    return solution
```

Algorithm 3: decomposer()

The decomposer (algorithm 3) receives as input a dn with at least one complex node to be decomposed d^c_{dec}. Even though the resolver will have to decompose all $d^c_{dec} \in dn$, the order is important as the decomposition of one node might restrict the possibilities of the next ones. Therefore, the next node d^c to be decomposed is heuristically selected based on the number of methods in which it can be decomposed. Then, one of the methods is selected, also heuristically. In this case, domain-dependent heuristics are preferred over general ones because the different methods encode expert knowledge that require domain-dependent information to choose among them. Once the method is selected, the decomposition network dn_{dec} is created and added to the list of decompositions that represent the solution. When the planner finishes, the solution containing all the decompositions is sent back and captured by the consumer which will analyse it.

4.3 Sufficient planning

HTLN allows the planners to create partial solutions in which not every complex node has to be decomposed. This solution is called *Sufficient Plan* [7]. To support this capability, the problem description language used in APSI (PDL) had to be slightly modified. For each d^c of the problem, the user can specify an *exclusion list* that indicates to the planner which decisions, in case they are used as sub-tasks of d^c, do not need to be decomposed. As an example, the first Driving goal g1 shown bellow indicates that it should not be decomposed at all, while g2 express that in case it contains either Autonav or Blind, these sub-goals should not been decomposed.

- g1 < goal, Driving > Master.Drive (?xi, ?yi, ?xf, ?yf, ?travers);
- g2 < goal, Auto, Blind > Master.Drive (?xi, ?yi, ?xf, ?yf, ?travers);

Internally, each decision d of a problem dn_i contains two variables. mustDecomposed is a boolean condition set to false in case the goal was in the list mentioned before or true otherwise. decompositionLevel indicates the present level of decomposition of a complex goal, which is calculated with two different algorithms depending on whether d is a component decision (algorithm 4) or a decision network (algorithm 5).

```
begin
    if (¬d ∈ Goals) then
        decompositionLevel ← TOTALLY
    else if (isPrimitive(d)) then
        decompositionLevel ← TOTALLY
    else if (mustDecomposed) then
        decompositionLevel ←
        NOT_SUFFICIENTLY
    else
        decompositionLevel ←
        SUFFICIENTLY
```

Algorithm 4: recalculateDecomposition(d ∈ CD)

For a d ∈ CD, if the decision is primitive or a fact (is not in the goals list), its decomposition level is TOTALLY. If d ∈ CD must be decomposed, then its level of decomposition is NOT_SUFFICIENTLY because a decomposed decision should be d ∈ DN. In any other case, d is SUFFICIENTLY decomposed.

```
begin
    min_elem ← lowest decomposition level among all
    d ∈ dn_i
    if (mustDecomposed(dn_i)) then
        minim_dn ← NOT_SUFFICIENTLY
    else
        minim_dn ← SUFFICIENTLY
    decompositionLevel(dn_i) ←
    max(min_elem, min_dn)
```

Algorithm 5: recalculateDecomposition(d ∈ DN)

For a d ∈ DN, it is first calculated the minimum decomposition level of all its sub-tasks and the minimum possible value for d, which is NOT_SUFFICIENTLY in

case it must be decomposed and SUFFICIENTLY otherwise. The level of decomposition of d is the maximum of these two values. In case the new value changes respect the previous one, the new value is propagated up to the parents of d, $dn^{sup}(d)$.

5 Initial results

We have implemented two different rover domains in a hierarchical and flattened style in order to compare QE and AP^2. The only modification between the two models has been the translation of the decompositions as synchronizations, which have similar semantics, and no additional constraints have been required. Some properties of the model are shown in table 1.

Property	RD-C	RD-S
Number of timelines	9	7
Total number states	27	19
Number decompositions	4	2
Number synchronizations	14	6

Table 1: Rover domain properties

The first domain, called *RD-C* (Rover Domain Complex) fully represents the model presented in section 3. The second one named *RD-S* (Rover Domain Simple) has removed a level of decompositions, as Driving is directly decomposed in the primitive tasks without the Blind and Autonav intermediate layer. Moreover, the Navigation and Drilling components have been removed.

The tests have been run in an Intel Core i5 M540 at 2.53 GHz computer with 3 GB RAM. The OS is Windows 7 Enterprise 32 bits. To understand the level of impact of the different decomposition methods in the performance an, both QE and AP^2 were configured to choose randomly the decomposition/synchronizations to apply. Notice that for the rover domain, the decomposition of a Driving goal as Autonav impose way more constraints than Blind.

We have created 7 problems named *Problem-1* to *Problem-7* with increasing level of complexity. Each problem *Problem-x* contains a list of *facts* defining the initial state for each component and *x* high-level goals, i.e. goals from the Master component. Being Driving the most complex goal, we have alternated between Blind and Autonav driving goals to cover all the possibilities during testing.

A simplified instantiation of *Problem-4* for the domain *RD-C* is shown bellow.

```
PROBLEM Rover-Problem (DOMAIN Rover_Domain){
 f1 <fact> Locomotion.At(?x=0, ?y=0);
 f2 <fact> Mast.PointingAt(?pan=0, ?tilt=0);
 f3 <fact> Camera.CamIdle();
 f4 <fact> Driller.DrillIdle();
 f5 <fact> Planner.PlannerIdle();
 f6 <fact> Antenna.CommIdle();
 f7 <fact> Navigation.NavIdle();
 f8 <fact> Master.Idle();

 f9 <fact> CommWindow.Visible() AT [0,5];
 f10 <fact> CommWindow.Visible() AT [40,80];

 g1 <goal> Master.Drive(?x=1, ?y=1, ?trav=easy);
 g2 <goal> Master.Pic(?x=1,?y=1,?pan=2,?tilt=2);
 g3 <goal> Master.Drill(?x=1, ?y=1, ?depth=1);
 g4 <goal> Master.Drive(?x=4, ?y=4, ?trav=hard);

 g1 BEFORE [1,+INF] g2;
 g2 BEFORE [1,+INF] g3;
 g3 BEFORE [1,+INF] g4;
```

Most of the parameters are self-explanatory. In order to simulate the inputs from sensors and the output from AnalyseTerrain in charge of deciding which type of navigation the rover should do, we added an extra parameter to driving that specifies by hand the type of terrain. Few modifications were required for the simpler domain *RD-S*. Drilling activities are replaced by TakingPicture and facts *f4* and *f6* were removed from the initial state.

It is important to mention that QE has been used without exploiting its multi-threading capability. The results showed that both planners were using intensively just one of the four cores of the processor, therefore we consider that this capability can boost the planner performance in the future.

For each run, we have registered the execution time to the first solution found and the number of nodes and edges in the solution network. The results comparing QE and AP^2 are shown in Figure 4.

In the first domain, *RD-C*, AP^2 starts to have problems even with very simple problems due to the big branching factor. While AP^2 has to branch for all possible disjunctive synchronizations, QE goes straight to the solution by using the decomposition methods.

In the second scenario, AP^2 managed to solve up to *Problem-5*. The number of edges increments notoriously faster than the number of nodes, crucial to understand the degradation in AP^2 performance as the problem complexity grows. Moreover, QE consistently generate less edges than AP^2. It is also interesting to remark that the peak in time doesn't come with the second Driving activity, which impose itself a number

 DOI: 10.2420/AF09.2014.21

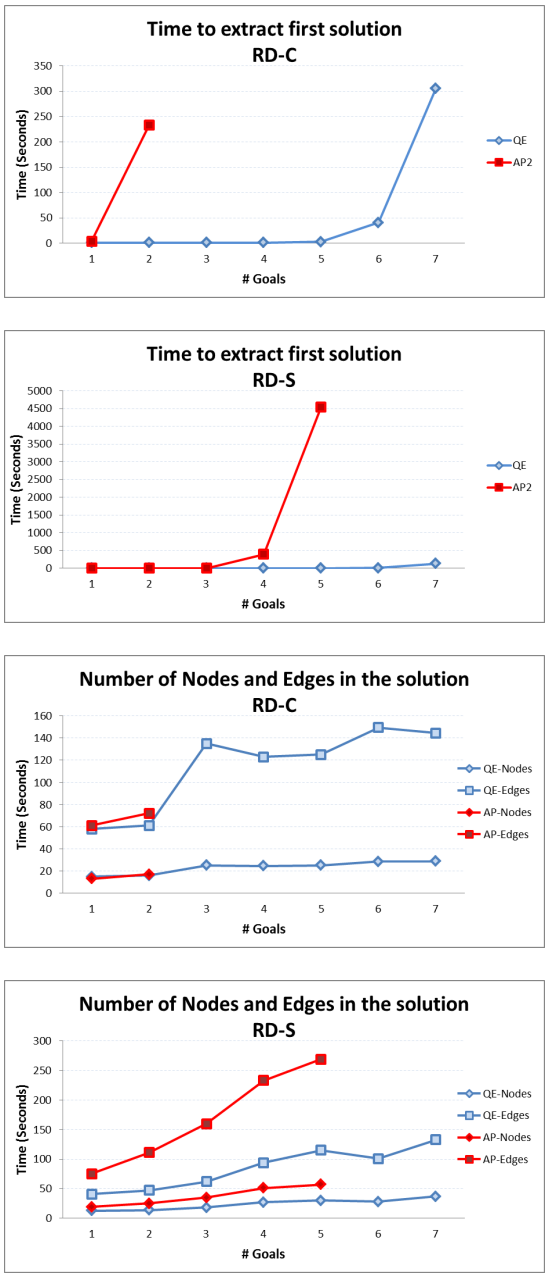

Figure 4: Comparison between QE and AP^2.

of sub-tasks and sub-relations, but with the next activity which is `TakingPicture`. Even though further analysis should be accomplish, our first impression is that after the second `Driving`, the planners (Unfolder and Scheduler) need to do a lot of processing to insert `TakingPicture` and unify some of its parameters. The same effect can be seen in for QE in the first scenario.

Nonetheless, these represent preliminary results that require more experiments to have a better understanding of how both planners explore the search space and generate the solutions.

6 Conclusions and future work

We have presented QuijoteExpress, a solver based in APSI*, an extension of the APSI framework that exploits Hierarchical Timeline Networks for temporal problems. Even though further experiments should be conducted, the results provided from running 6 different problems for 2 different domains in both QE and AP^2 are even better than our original expectations, showing an important improvement on performance. We will conduct more experiments in the future to analyse which are the stronger and weaker points for each planner and evaluate the effect of sufficient planning and parallel planning approaches.

There is a number of lessons learnt extracted from the tests with the rover domain.

- The capability of HTN planners to reduce the branching factor and therefore the size of the search space is crucial to understand the great benefit in terms of performance.

- HTN represents an overhead during the modelling process. The design and validation/verification of the four models implemented (hierarchical and non-hierarchical) was tedious and error prone. This fact is particularly relevant in HTN, where the model can present hidden constraints/dependencies between elements in different levels of abstractions. In consequence, we consider that it will be crucial in the future to develop new tools able to assist during the construction of complex models.

- Debugging and understanding the output of the planner is almost impossible for a non-expert. HTN plays an important role in this two aspects. During the the runs with the hierarchical model, we could identify a fault condition just observing the high-level goals and their relations. It would have taken much more time in case we would have needed to go through all the primi-

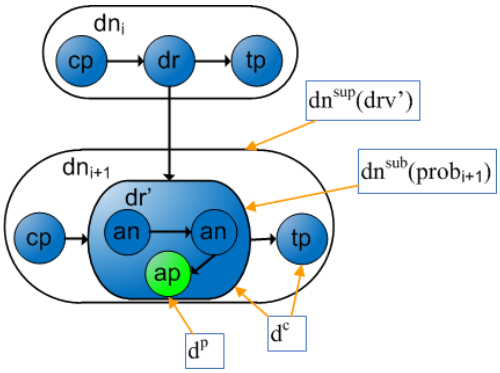

Figure 5: Hierarchical dn structure

tive components in a flat domain. In the same way, it is easier for the user to understand the plan by examining only the timelines related with complex components.

- Modern domain description languages are not expressive enough. In our case, even with the help of a time-oriented language, we were highly constrained. Extending expressiveness or moving towards the use of standard programming languages, as it happens with SMASH for ROS, will be required to construct realistic models.

- It is important to provide tools that allow users to represent knowledge. Domain-dependent knowledge must be contained in dedicated structures to maintain the system reusable, in our case in the methods and heuristics. Regarding method selection, we believe that general heuristics such as method timespan, number of complex tasks, etc. are not appropriate because the selection use to depend on the same specific knowledge the method represents. For example, in the rover domain, the selection between Blind and Autonav should be done by an expert-system according to the information gathered by the rover sensors.

In the future, our intention is to further improve the planner performance in two ways: implementing search heuristics and enable parallel planning. Creating an agile, anytime planner is critical to use it in scenarios that demand continuous re-planning such as Mars Rovers, telescopes or rescue robots. Comparing QE with other planners using similar techniques such as ASPEN [5] or Europa [10] would be important to understand where we are. So far we have just run simulations with virtual models, but it is our intention to start soon doing tests with real robots which should be of great relevance to understand how to evolve the system.

Acknowledgments

This research has been co-funded by the Networking/Partnering Initiative (NPI) between ESA-ESOC, TU Darmstadt and University of Surrey. 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”.

APPENDIX A – Nomenclature

Acronym	Description
C_i	Component i
cd	Component Decision
rlt	Relation
dn	Decision Network
$w = (N, E)$	Hypergraph used to represent a dn. N is the list of nodes and E the list of hyperedges
d^p	Primitive task
d^c	Complex task
D^p	Set of all primitive tasks
D^c	Set of all complex tasks
D^{all}	Set of all tasks
m	Method, implemented in HTLN as a dn
$dn_{dec}(d)$	Method selected to decompose a decision $d \in D^c$
$dn^{sup}(d)$	Parent decision of d. It will be either a dn or $\emptyset$ in case d is the problem network
$dn^{sub}(d)$	Child decision of d. It will be either $dn^{sub}(d) = \emptyset$ if $d \in D^p$ or $dn^{sub}(d) = dn$
ic	Initial condition
ρ	Procedure intended to fix a flaw ϕ in a given problem dn_i
d^+, d^-	List of decisions returned by the resolver ρ to be added/retracted to the problem dn_i
rlt^+, rlt^-	List of relations returned by the resolver ρ to be added/retracted to the problem dn_i
$\rho(from, \sigma)$	Decomposition resolver that decomposes the decision $from \in dn_i$ in the sub-network to using the substitution σ

APPENDIX B – Hierarchical dn structure

Figure 5 illustrates the concepts presented before. dn's are represented as ellipses, cd's as circles and rlt's as segments. The problem contains three compound goals

 DOI: 10.2420/AF09.2014.21

(cp, dr, tp) which stand for *calculate path*, *drive* and *take picture* respectively, ordered between them by *before* temporal constraints (represented as arrows). Drive is the only d^c already decomposed. Therefore, it is represented not as a cd but as a dn (dr') containing three sub-tasks, where an stands for *autonav* and ap *approach*. While ap is primitive, an is compound and will require to be further decomposed adding another nested dn into drive. $dn^{sup}(d)$ represents the dn one layer up of d and $dn^{sub}(d)$ one layer bellow d. Being dn_i the problem network, $dn^{sup}(dn_i) = \emptyset$, that is, a problem has no layer above and $dn^{sub}(d) = \emptyset$ if d is in the last layer.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, Nov. 1983.

[2] J. J. Biesiadecki and M. W. Maimone. The mars exploration rover surface mobility flight software driving ambition. In *IEEE Aerospace Conference (IAC)*, number March, 2006.

[3] A. Ceballos, S. Bensalem, A. Cesta, L. S Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. V. Winnendael. A goal-oriented autonomous controller for space exploration. *ASTRA*, 2011.

[4] A. Cesta, S. Fratini, A. Orlandini, and R. Rasconi. Continuous Planning and Execution with Timelines. In *International Symposium on Artificial Intelligence (i-SAIRAS)*, number 1, 2012.

[5] S. Chien, G. R. Rabideau, R. L. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. A. Estlin, B. Smith, F. W. Fisher, A. C. Barrett, G. Stebbins, and D. Tran. ASPEN - Automated Planning and Scheduling for Space Mission Operations. In *International Conference on Space Operations (SpaceOps)*, pages 1–10, 2000.

[6] J. M. Delfa Victoria, N. Policella, G. Yang, and O. V. Stryk. Design Concepts for a new Temporal Planning Paradigm. In *ICAPS Planning & Scheduling for Timelines (PSTL), 2012*, 2012.

[7] J. M. Delfa Victoria, N. Policella, G. Yang, and O. V. Stryk. QuijoteExpress - A Novel APSI Planning System For Future Space Robotic Missions. In *ASTRA*, 2013.

[8] K. Erol, J. Hendler, and D. S. Nau. Semantics for Hierarchical Task-Network Planning. Technical report, 1994.

[9] T. A. Estlin, B. J. Bornstein, D. Gaines, R. C. Anderson, D. R. Thompson, M. Burl, R. Castano, and M. Judd. AEGIS Automated Science Targeting for the MER Opportunity Rover. *International Symposium on Artificial Intelligence Robotics and Automation in Space (i-SAIRAS)*, pages 1–25, 2010.

[10] J. D. Frank and A. K. Jonsson. Constraint-based Attribute and Interval Planning. *Journal of Constraints Special Issue on Constraints and Planning*, 8(4):339–364, 2003.

[11] S. Fratini and A. Cesta. The APSI Framework: A Platform for Timeline Synthesis. *Proceedings of the 1st Workshops on Planning and Scheduling with Timelines PSTL-12*, 2012.

[12] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma. Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots. *J. Field Robot.*, 30(1):44–63, Jan. 2013.

[13] A. Tate, B. Drabble, and R. Kirby. O-Plan2: an open architecture for command, planning and control. In *Intelligent Scheduling*, volume 1, pages 213–239. 1994.

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