ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200362

# Lifted Heuristics for Timeline-Based Planning

**Riccardo De Benedictis** and **Amedeo Cesta**[1]

**Abstract.** This paper discusses the issue of efficient resolution of *timeline-based* planning problems. In particular, taking inspiration from the more classical heuristics for the resolution of STRIPS-like problems, it proposes a new heuristic strategy which, while maintaining the variables lifted, allows more accurate decisions. The concepts presented in this work pave the way for a new type of heuristics which, at present, allow this kind of solvers a significant performance improvement.

## 1 Introduction

Since their early introduction, domain-independent heuristics have immediately proven to be a fundamental ally in solving difficult combinatorial problems such as those related to automated planning. The number of heuristics, introduced in recent years, for the efficient resolution of these problems has grown significantly to the point of constituting a research field (called *heuristic planning*) in its own right. The different approaches that make up a solver's paraphernalia, range from the seminal $h_{add}$ and $h_{max}$ [4] to the more recent developments relying on *delete-relaxation*, like the $h^{FF}$ heuristic [36] and the *causal graph* heuristics [32], on *landmarks*, like in [37, 46], on the *critical path*, like the $h^m$ heuristic [31, 30] or, lastly, on *abstraction*, like in [17] or in [33, 34].

While the above heuristics are significantly heterogeneous among them (although, often, they share some commonalities), they have in common the fact that they have been developed specifically for the resolution of a particular type of problem, characterized by a specific modeling language called PDDL [28], representing a natural evolution of the most long-lived STRIPS [19] formalism. Despite the PDDL, over the years, has been extended through different directions by introducing *durative-actions* and *numeric fluents* [20], *derived predicates* and *timed initial literals* [18], *continuous changes* [21], *state-trajectory constraints* and *preferences* [27] and *object-fluents*[2], the development of heuristics for reasoning with these more expressive formal systems has remained relatively limited to a few cases (e.g., [45, 23]).

Although it significantly departs from the previous ones, the *timeline-based* approach represents a different formalism that, already in its original formulation [44], is able to cover a large part of the above features. Although introduced before the aforementioned formalisms, this specific planning paradigm has always remained a niche within the automated planning community. The fragmentation of the different timeline-based formalisms, indeed, did not allow the emergence of a common language which would have enabled a fair comparison among the different reasoners. Furthermore, analogously to the solvers reasoning upon the previous PDDL extensions, timeline-based planners have to cope with the high expressiveness of the formalisms which, despite making them particularly suited at addressing real-world applications, unavoidably leads to performance issues. The contribution of this paper is, hence, twofold: after providing a new formalization of the timeline-based problem, aiming to embracing the different aspects of the previous formalisms, we propose a new domain-independent heuristic which, inspired by the more classical ones, aims at improving the resolution efficiency.

[1] CNR - Italian National Research Council, ISTC, email: {name.surname}@istc.cnr.it

[2] http://www.plg.inf.uc3m.es/ipc2011-deterministic/attachments/Resources/kovacs-pddl-3.1-2011.pdf

## 2 Timeline-based planning

Timeline-based planning was first introduced in [44, 43] and, since then, many solvers have been proposed like, for example, IxTeT [29], EUROPA [38], ASPEN [10], the TRF [25, 6] on which the APSI framework [26] relies and, more recently, PLATINUm [50]. Some theoretical work on timeline-based planning like [24, 38] was mostly dedicated to identifying connections with classical planning a-la PDDL [20]. The work on IxTeT and TRF has tried to clarify some key underlying principles but mostly succeeded in underscoring the role of time and resource reasoning [7, 40]. The planner CHIMP [49] follows a Meta-CSP approach having meta-Constraints which havely resembles timelines. The Flexible Acting and Planning Environment (FAPE) [16] tightly integrates timelines with acting. The Action Notation Modeling Language (ANML) [47] is an interesting development which combines the HTN decomposition methods with the expressiveness of the timeline representation. Finally, it is worth mentioning that the timeline-based approaches have been often associated to resource managing capabilities. By leveraging on constraint-based approaches, most of the above approaches like IxTeT [41, 40], [8], [48] or [51] integrate planning and scheduling capabilities. Finally, [11] proposes a recent new formalization of timeline-based planning.

In order to better understand what we are talking about when discussing about timeline-based planning, it is important to introduce some basic concepts about *constraint networks*. Some of the timeline-based frameworks like, for example, those described in [48, 24], refer to timeline-based planning in terms of constraint-based planning, further emphasizing the central role that constraints take on within this type of planning. The main ingredients of constraint networks are variables and constraints. Formally,

**Definition 1.** *A* variable *is an object that has a name and is able to take different values.*

A variable (whose name is) $x$ must be given a value from a set that is called the *domain* of $x$ and is denoted by $dom\,(x)$. The domain of a variable $x$ may evolve in time but is always included in a (possibly infinite) set called *initial domain*. Depending on the nature of these domains, variables can be distinguished between *continuous*, having

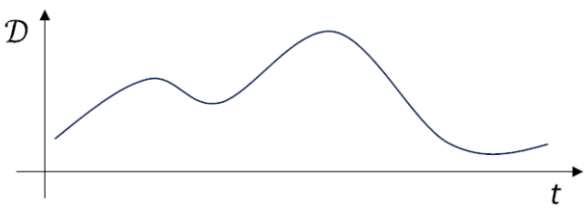
an infinite initial domain usually defined in terms of real intervals, and *discrete*, whose initial domain contains a finite number of values.

**Definition 2.** *A* constraint *is a restriction on combinations of values that can be taken simultaneously by a set of variables.*

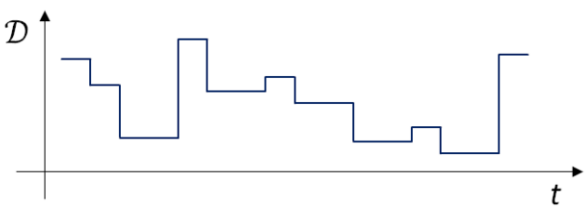
A constraint $c$ is defined over a set of variables which constitute the *scope* of $c$ and are denoted by $scp(c)$. Finally, a structure composed of variables and constraints is called a *constraint network*.

**Definition 3.** *A* constraint network $\mathcal{N}$ *is composed of a finite set of variables, denoted by* $vars(\mathcal{N})$*, and a finite set of constraints, denoted by* $cons(\mathcal{N})$*, such that* $\forall c \in cons(\mathcal{N}), scp(c) \subseteq vars(\mathcal{N})$.

Since constraint networks are fundamentals for timeline-based planning, it is worth introducing some further concepts without going into too much formal details. Specifically, an assignment of values to some or all the variables is called an *evaluation*. Furthermore, an evaluation is said to be *consistent* if it does not violate any constraint. An evaluation is said to be *complete* if it includes all the variables. Finally, given a constraint network, the problem of finding a consistent and complete evaluation is called Constraint Satisfaction Problem (CSP) (refer to [15, 42] for a comprehensive introduction to constraint networks and CSPs).




(a) A continuous timeline.




(b) A step-wise timeline.

**Figure 1**: A continuous and a step-wise timeline.

Constraint networks represent the lowest level elements on which timeline-based planning relies. The main data structure for the timeline-based paradigm is, indeed, the *timeline* which, in generic terms, is a function of time, either discrete or continuous, over a given domain. Formally,

**Definition 4.** *A* timeline $\mathsf{T}$ *is a function*

$$\mathsf{T} : \mathbb{T} \rightarrow \mathcal{D}$$

*where* $\mathbb{T}$ *is the (either discrete or continuous) domain of* time *and* $\mathcal{D}$ *is the (possibly infinite) domain of the timeline.*

It is worth noticing that the previous definition is quite general, not specifying any limitation neither on the time, which can be either discrete or continuous, nor on the domain which can be, in general, of any kind. Specifically, the domain of a timeline can be either symbolic (e.g., "a", "b", "c", etc.) or numeric (e.g., "1", "2", "3", etc.). Additionally, numeric domains can be either integer (e.g., "10", "12", "25", etc.) or real (e.g., "1.23", "2.17", "3.14", etc.). While integer domains can change in time only step-wise, real domains can change both step-wise and continuously. Finally, continuous changes can happen both linearly or non-linearly. Figure 1 (a), for example, represents a continuously updating non-linear timeline over reals. Figure 1 (b), on the contrary, shows a step-wise updating timeline.

Since the definition of timeline is completely general, it is possible to represent, through these, extremely heterogeneous concepts. We need, therefore, a unifying element that allows to represent contents homogeneously, in a way which is agnostic from the nature of the timeline. To this end, we introduce the concept of *token* and establish that values on timelines are a direct consequence of tokens through a timeline extraction procedure (more details soon). Without loss of generality, a token is an "assertion over a temporal interval". Formally,

**Definition 5.** *A* token *is an expression of the form:*

$$n(x_0, \ldots, x_i)_\chi @ [s, e, \tau]$$

*where* $n$ *is a predicate name,* $x_0, \ldots, x_i$ *are the* parameters *of the predicate (i.e., constants, numeric variables or symbolic variables),* $\chi$ *is the* class *of the token (i.e., either a fact or a goal),* $s$ *and* $e$ *are the* temporal parameters *of the token (i.e., constants or variables) belonging to* $\mathbb{T}$ *such that* $s \leq e$ *and* $\tau$ *is the* scope parameter *of the token (i.e., a constant or a symbolic variable) representing the timeline on which the token apply.*

Roughly speaking, the expression on the left of the "@" symbol represents the "assertion" while the expression at its right represents the "interval". In other words, a token $n(x_0, \ldots, x_i)_\chi @ [s, e, \tau]$ asserts that $\forall t$ such that $s \leq t \leq e$, the relation $n(x_0, \ldots, x_i)$ holds at the time $t$ on the timeline $\tau$. Furthermore, given a token $\eta$, we call $pars(\eta)$ its parameters $x_0, \ldots, x_i, s, e, \tau$.

Tokens constitute the main building blocks of timeline-based plans. Regardless of the resolution procedure, indeed, the role of any timeline-based solver consists in introducing new tokens and/or establishing the values of their parameters. A critical aspect to keep in mind, when talking about tokens, is that, in general, their parameters are variables, like those introduced in the Definition 1 and, as such, can be constrained. In other words, in order to reduce the allowed values for the tokens' constituting parameters, and thus decreasing the modeled system's allowed behaviors, it is possible to impose *constraints* among them (and/or among the parameters and other possible variables). Such constraints include temporal constraints, binding constraints between symbolic variables as well as (non)linear constraints among numerical variables (possibly including temporal variables).

The set of tokens and constraints is used to describe the main data structure that is used to represent plans of the timeline-based approach: the *token network*. Formally,

**Definition 6.** *A* token network *is a tuple* $\pi = (\mathcal{T}, \mathcal{N})$*, where:*

- $\mathcal{T} = \{\eta_0, \ldots, \eta_j\}$ *is a set of tokens, such that* $\forall \eta \in \mathcal{T}, pars(\eta) \subseteq vars(\mathcal{N})$.
- $\mathcal{N}$ *is a constraint network.*

Roughly speaking, tokens provide a higher level semantics to (some of) the variables of the constraint network, grouping them into data structures on which the planner can reason.

Finally, as already mentioned, tokens can be partitioned into two classes: *facts* and *goals*. While facts are, by definition, inherently true, goals have to be achieved. Causality, in particular, in the timeline-based approach, is defined by means of a set o *rules* indicating how to achieve goals. Formally,

**Definition 7.** *A* rule *is an expression of the form*

$$n\left(x_0,\ldots,x_k\right) @ \left[s,e,\tau\right] \leftarrow \boldsymbol{r}$$

*where:*

- $n\left(x_0,\ldots,x_k\right) @ \left[s,e,\tau\right]$ *is the* head *of the rule, i.e. an expression in which* $n$ *is a predicate name,* $x_0,\ldots,x_k$ *are the parameters of the head (i.e., numeric variables or symbolic variables),* $s$ *and* $e$ *are the* temporal parameters *of the head (i.e., constants or variables) belonging to* $\mathbb{T}$ *such that* $s \leq e$ *and* $\tau$ *is the* scope parameter *of the head (i.e., a constant or a symbolic variable) representing the timeline on which the rule apply.*
- $\boldsymbol{r}$ *is the* body *of the rule (or the* requirement*), i.e. either a* slave *(or* target*) token, a* constraint *among tokens (possibly including the* $x_0,\ldots,x_k,s,e,\tau$ *variables), a* conjunction *of requirements or a (priced*[3]*)* disjunction *of requirements.*

Specifically, rules define causal relations that must be complied to in order for a given goal to be achieved. Roughly speaking, for each goal having the "form" of the head of a rule, the body of the rule (i.e., further tokens, constraints, conjunctions and disjunctions) must also be present in the token network. An example of rule is given by

$$At\left(?x\right) @ \left[s,e,\tau\right] \leftarrow \left\{ \left\{ \begin{array}{c} \left[e-s \geq 1\right] \wedge \\ \left\{ \begin{array}{c} dt : DriveTo\left(?x\right)_g @ \left[s,e,\tau\right] \wedge \\ \left[\tau == dt.\tau\right] \wedge \left[s == dt.e\right] \wedge \\ \left[?x == dt.?x\right] \end{array} \right\} \vee \\ \left\{ \begin{array}{c} ft : FlyTo\left(?x\right)_g @ \left[s,e,\tau\right] \wedge \\ \left[\tau == ft.\tau\right] \wedge \left[s == ft.e\right] \wedge \\ \left[?x == ft.?x\right] \end{array} \right\} \end{array} \right\} \right\}$$

By means of a combination of slaves, constraints, conjunctions and disjunctions, the above rule states that, in order to be in a given position, our agent must reach it either by driving or by flying.

We have now all the ingredients to define a timeline-based planning problem. In particular, the definition can rely on the above concept of requirement.

**Definition 8.** *A timeline-based* planning problem *is a triple* $\mathcal{P} = (\mathbf{T}, \mathcal{R}, \boldsymbol{r})$*, where:*

- $\mathbf{T}$ *is a set of timelines.*
- $\mathcal{R}$ *is a set of rules.*
- $\boldsymbol{r}$ *is a requirement, i.e. either a (fact or goal) token, a constraint among tokens, a conjunction of requirements or a (priced) disjunction of requirements.*

It is worth highlighting that, conversely to other timeline-based approaches, our formalism makes a clear distinction between tokens and values on timelines. This difference aims at guaranteeing us a further element of generality. The transition from tokens to timelines, however, requires the introduction of a further function which allows to *extract* the timelines from the tokens. Specifically,

**Definition 9.** *An* extraction function $\mathsf{X}_\mathsf{T}$ *is a function for a timeline* $\mathsf{T}$

$$\mathsf{X}_\mathsf{T} : \mathbb{T} \times 2^{\mathcal{T}_\mathsf{T}} \rightarrow \mathcal{D}$$

*where* $\mathbb{T}$ *is the (either discrete or continuous) domain of* time*,* $\mathcal{T}_\mathsf{T}$ *is the set of tokens in the token network, having* $\mathsf{T}$ *in the domain of their* $\tau$ *variable, and* $\mathcal{D}$ *is the domain of the timeline.*

As can be easily seen by comparing Definition 4 with Definition 9, the result of the extraction function is, basically, a timeline. Each type of timeline, indeed, has associated its own timeline extraction procedure which allows to pass from the associated tokens to the resulting timelines. In other words, the timeline extraction procedure assigns to the tokens a higher-level semantic: according to the nature of the timeline, the procedure is able to "recognize the meaning" of the involved tokens. Note that, thanks to the introduction of the above higher-level semantic, not all token configurations lead to consistent timelines. According to the nature of the timeline, indeed, some configurations of tokens might lead to inconsistencies. It is responsibility of the solver to introduce further constraints so as to avoid such inconsistencies. Another way to see a timeline, indeed, is in terms of a *global constraint* (refer, for example, to [15, 42]). In other words, a timeline is a global constraint over the tokens which are applied on it.

Examples of timelines, extracted from tokens, are shown in the Figure 2. Specifically, Figure 2a shows a *state-variable* timeline, a step-wise timeline whose domain depends from the tokens which can be assigned, by means of the $\tau$ variable (omitted, for simplicity), to it. This type of timeline, in particular, introduces an additional constraint that guarantees that different values, on the same timeline, cannot overlap in time. The state-variable of Figure 2a, as an example, has two values that overlap as a consequence of the overlapping of the $At\left(l_1\right)$ and the $GoingTo\left(l_2\right)$ tokens. Such an inconsistency can be solved, for example, by imposing an ordering constraint between the tokens. Another type of timeline, typically used in pure scheduling problems, is the *reusable-resource* (see Figure 2b). This step-wise timeline is characterized by a maximum capacity and by a resource level which changes over time according to how the tokens, representing resource usages, overlap. The resource constraint guarantees that concurrent uses of the resource do not exceed its capacity. Finally, as example of a continuous timeline, the *consumable-resource* timeline (see Figure 2c) is characterized by a maximum capacity and by an initial amount. Similarly to reusable-resources, the resource level changes over time according to how the tokens, representing resource productions and consumptions, overlap, while the resource constraint guarantees that the level never exceeds the resource capacity nor goes below zero.

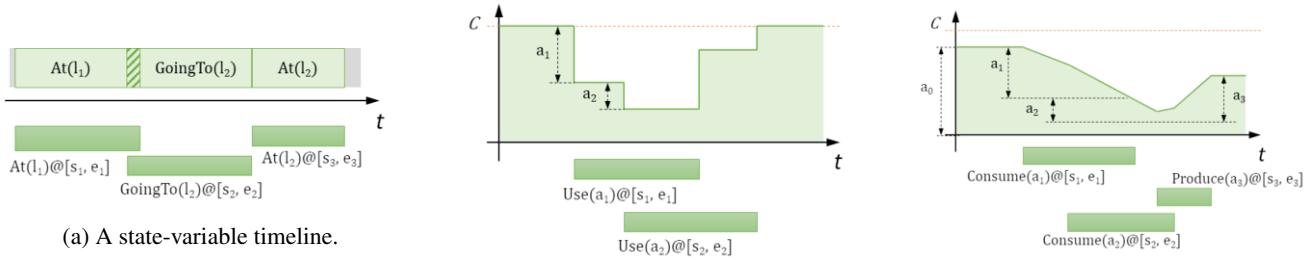
It is worth noticing that, by enabling any implementing solver to reason about timelines agnostically from their specific nature, the above definition allows us to maintain a certain generality. Furthermore, once provided an extraction function and the algorithms for managing the specific global constraint, new types of timelines can be introduced without affecting the solvers' resolution procedures.

The last aspect to consider regards the solution of a timeline-based planning problem. Roughly speaking, a solution is a token network whose all goals have been achieved. Furthermore, at least one consistent and complete evaluation of the underlying constraint network must be available. Notice that, among the constraints of the constraint network, there are also those which are imposed by the timelines. Formally,

**Definition 10.** *A token network* $\pi = (\mathcal{T}, \mathcal{N})$ *is a* solution *for a timeline-based planning problem* $\mathcal{P} = (\mathbf{T}, \mathcal{R}, \boldsymbol{r})$ *if:*

- *there exists a complete and consistent evaluation of the constraint network* $\mathcal{N}$*.*

[3] It is possible, if needed, to associate a cost to the different disjuncts of a disjunction so as to model preferences.




(a) A state-variable timeline.

(b) A reusable-resource timeline.

(c) A consumable-resource timeline.

**Figure 2**: Different timelines extracted by tokens.

– *every goal $g \in \mathcal{T}$ is achieved (i.e., either the goal $g$ is recognized as semantically equivalent of another token or a rule, whose head is compatible with the token $g$, is applied).*

## 3 Reasoning with timelines

Unfortunately, the above definitions do not provide a computable test for building and verifying solutions. This section, therefore, introduces the typical approach for solving timeline-based planning problems. Specifically, common timeline-based solvers strongly rely on partial-order planning [52] for reasoning, while generalizing the concept of *threat* for including any possible inconsistency which might arise as a consequence of the timeline constraints (e.g., different states overlapping on the same state-variable, resources overusages, etc.). Despite this generalization, the search space (and, consequently, the solving algorithm) remains substantially unchanged. In particular, timeline-based solvers rely on the concept of *flaws*, that a token network has, and on the concept of *resolvers*, for solving them. Formally,

**Definition 11.** *A* flaw *in a token network $\pi = (\mathcal{T}, \mathcal{N})$ is either: (i) an open goal (i.e., a goal whose associated rule has not yet been applied or which has not yet been recognized as semantically equivalent to another token), (ii) a threat (i.e., any possible inconsistency arising as a consequence of the timeline constraints) or (iii) a disjunction.*

Intuitively, the main resolution principle consists in refining the token network $\pi$, identifying its flaws and applying resolvers for solving them, while maintaining the constraints $cons(\mathcal{N})$ consistent, until the token network $\pi$ has no more flaws.

**procedure** TP($\pi$)
  $flaws \leftarrow$ OpenGoals ($\pi$) $\cup$ Threats ($\pi$) $\cup$ Disjunctions ($\pi$)
  **if** $flaws = \emptyset$ **then return** $\pi$
  **end if**
  select any flaw $\varphi \in flaws$
  $resolvers \leftarrow$ Resolve ($\varphi, \pi$)
  **if** $resolvers = \emptyset$ **then return** failure
  **end if**
  non-deterministically choose a resolver $\rho \in resolvers$
  $\pi' \leftarrow$ Refine ($\rho, \pi$)
  **return** TP ($\pi'$)
**end procedure**

**Figure 3**: The TP procedure for solving timeline-based planning problems.

Figure 3 specifies a recursive non-deterministic procedure called TP (for Timeline-based Planning) for resolving timeline-based planning problems. Specifically:

– *flaws* denotes the set of all flaws in $\pi$ provided by procedures `OpenGoals`, `Threats` and `Disjunctions`; $\varphi$ is a particular flaw in this set.
– *resolvers* denotes the set of all possible ways to resolve a specific flaw $\varphi$ in a plan $\pi$ and is given by the procedure `Resolve`. The resolver $\rho$ is a particular element of this set.
– $\pi'$ is the new plan obtained by refining $\pi$ according to the resolver $\rho$ as a consequence of the procedure `Refine`.

The TP procedure is called with an initial token network $\pi_0$, characterized by the problem's requirement. Each successful recursion is a refinement of the current plan according to the chosen resolver. In particular, the `Resolve` procedure returns all the resolvers that, in the token network $\pi$, solve the $\varphi$ flaw. These resolvers depend, necessarily, on the type of flaw $\varphi$ and on the current token network $\pi$. In the case of open goals, for example, resolvers represent the application of the corresponding rule or the *unification* with other semantically equivalent tokens (i.e., same predicate name and same, pairwise, parameter values). In the case of excessive concurrent resource usage, resolvers could represent ordering constraints between the tokens. As a consequence, each invocation of the `Refine` procedure might introduce new tokens, new variables and/or new constraints to the token network. Intuitively, refinement operations should be chosen so as to avoid adding to the token network any constraint that is not strictly needed (this is called the *least commitment principle*).

## 4 Toward more effective heuristics

Reasoning within the above formal system is not at all simple[4]. It is worth noting that while the choice of the resolver is a *non-deterministic* step (i.e., it may be required to backtrack on this choice), the selection of a flaw is a *deterministic* step (i.e., there is no reason to backtrack on this choice) as all flaws need to be solved before or later in order to reach a solution plan. Despite the order in which flaws are processed is very important for the efficiency of the procedure, it is unimportant for its soundness and completeness. A deterministic implementation of the TP procedure should rely on algorithms like A* or IDA* so as to avoid that the search may keep exploring deeper and deeper a single path in the search space, adding indefinitely new tokens to the partial plan and never backtracking. As a consequence, choosing the *right* flaw and the *right* resolver becomes a crucial aspect for coping with the computational complexity and hence efficiently generating solutions.

The main difficulty derives from the impossibility of i) having a perfectly defined current state and ii) measuring the distance between

[4] Note that it is possible, in general, to represent through this formalism a self-referential proposition $P$, whose meaning is "$P$ is false", and show quite easily its non-decidability.

this state and a desired state indicated in the formulation of the reasoning problem. For these reasons it becomes particularly inconvenient to use or even adapt, directly, the heuristics developed for classical formalisms. What we propose in this document is, somehow, to separate the temporal aspects from the purely causal ones, which in classical planning are strongly linked to be almost the same thing, and to apply classical heuristics only to the latter. In doing so, the rules of the timeline formalism become the equivalent of the PDDL operators, having the requirements as preconditions and the head of the rule as the only positive effect. Once this paradigm shift has been made, it becomes possible to adapt the heuristics of classical planning. Note that, however, this translation is not trivial: if, on the one hand, there is the simplification of having, for each operator, only a single positive effect (i.e., the solved flaw), on the other hand there is the difficulty of rendering atoms "ground" due to the presence of numerical parameters (representing, for example, the starting and the ending times of the tokens). We are therefore forced to reason about a sort of causal graph having lifted variables.

The overall proposed idea consists in applying, in a coarse way, *all* the possible resolvers for *all* the possible flaws until some termination criteria, i.e., unifications and resolvers which do not add further flaws, is met. Specifically, since flaws and resolvers are causally related (i.e., resolvers might introduce flaws which are solved by other resolvers, etc.) it is possible to build an AND/OR graph for representing such causal relations. By doing so, instead of searching in the space of the token networks, we have a single disjunctive token network containing all the possible plans (or, hopefully, only the "most interesting" ones) that can be found starting from the initial token network $\pi_0$. By exploiting the topology of such a graph it is possible to generate an estimation of "how far" a flaw is from being solved and exploit this estimation for guiding the resolution process. Specifically, taking inspiration from the $h_{add}$ and the $h_{max}$ heuristics introduced in [4], the cost of a resolver, which can be seen as an AND node, can be estimated as the *maximum* (in case of $h_{max}$ heuristic or, in case of the $h_{add}$ heuristic, the *sum*) of the estimated costs of the flaws introduced by the resolver itself plus an intrinsic resolver's cost, while the estimated cost of a flaw, which can be seen as an OR node, can be estimated as the *minimum* of the estimated costs of its possible resolvers. Since all flaws must be solved, the solver chooses, among those that have to be solved, the most expensive flaw (i.e., the one that, most likely, will detect an inconsistency earlier) and will solve it with the least expensive resolver (i.e., the one that, more likely, will lead to a solution).

## 5 The lifted heuristic formulation

Before formally introducing the proposed heuristics, it is worth providing some definitions. Specifically, since the presence of flaws and resolvers, within the current partial solution, is controlled by a set of propositional variables, we refer to flaws by means of $\varphi$ variables (we will use subscripts to describe specific flaws, e.g., $\varphi_0$, $\varphi_1$, etc.) and to resolvers by means of $\rho$ variables (similarly to flaws, we will use subscripts to describe specific resolvers, e.g., $\rho_0$, $\rho_1$, etc.). Specifically, the value of such variables will be used to recognize active flaws that have to be solved (i.e., those flaws whose $\varphi$ variables assume the $true$ value) and applied resolvers (i.e., those resolvers whose $\rho$ variables assume the $true$ value). Additionally, given a flaw $\varphi$, we refer to the set of its possible resolvers by means of $res(\varphi)$ and to the (possibly empty) set of resolvers which are responsible for introducing it by means of $cause(\varphi)$. The latter set is usually constituted by the sole resolver representing the application of the rule which introduced the flaw. Nonetheless, this set can also be empty in case of top-level flaws, in which case the $true$ value is assigned to their controlling $\varphi$ variables or, also, can contain more than one resolver in case the flaw is a consequence of their simultaneous application (e.g., a flaw representing two states overlapping on the same state-variable is activated whenever the rules that introduce the two states are applied simultaneously). Finally, given a resolver $\rho$, we refer to the set of its preconditions (e.g., the set of tokens introduced by the application of a rule) by means of $precs(\rho)$ and to the flaw solved through its application by means of $eff(\rho)$.

The above definitions allow us to formally introduce our heuristics. Specifically, let $G$ be the estimated cost function, the estimated cost of a flaw $\varphi$ and of a resolver $\rho$ are characterized by the following equations:

$$G(\varphi) = min_{\rho \in res(\varphi)} G(\rho)$$
$$G(\rho) = c(\rho) + max_{\varphi \in precs(\rho)} G(\varphi)$$

where $c(\rho)$ is the *intrinsic cost* of the $\rho$ resolver, i.e., a positive number representing the "cost" of disjuncts, in case of priced disjunctions, or the value 1, in other cases.

### 5.1 Enforcing causality constraints

Similar to planning models based on satisfability [39], it is possible to introduce propositional constraints to the $\varphi$ and $\rho$ variables so as to guarantee the causal relations. By doing so, once the graph has been built, it is possible to frame the search space within a given boundary, dropping the computational complexity of the search procedure to a "simpler" NP-hard[5]. Furthermore, the introduction of these variables allows the use of propagation techniques and, in the event of inconsistencies, conflict analysis (and, hence, non-chronological backtracking) techniques, typical of SAT/SMT based solvers. The planning problem is therefore reduced to the assignment of $true$ values to the variables associated to the resolvers while observing the assignment, as a consequence of constraint propagation, of $true$ values to the variables associated to the flaws.

Additionally, we need a gimmick to establish the presence or not of the tokens inside the solution. In this regard, a state variable $\sigma \in \{inactive, active, unified\}$ is associated to each token. Specifically, a partial solution will consist solely of those tokens of the token network which are *active*. Moreover, in case such tokens are goals, the bodies of the associated rules must also be present within the solution. The *unified* tokens do not participate directly in the partial solution, since they are recognized as semantically equivalent to other active tokens, yet, since possibly subject to constraints, they might indirectly influence the "shape" of the solution. Finally, *inactive* tokens do not participate at all in the solution. We refer to tokens, later on, by means of $\sigma$ variables (we will use subscripts to describe specific tokens, e.g., $\sigma_0$, $\sigma_1$, etc.) and to the flaws introduced by tokens by means of the $\varphi(\sigma)$ function.

Specifically, for each flaw $\varphi_i$, we enforce the following causality constraints

$$\varphi_i = \wedge_{\rho_k \in cause(\varphi_i)} \rho_k \quad (1)$$
$$\varphi_i \Rightarrow \vee_{\rho_l \in res(\varphi_i)} \rho_l \quad (2)$$

[5] There is, intuitively, no guarantee that the built graph contains a solution. Similarly to what happens in Graphplan [3], indeed, it might be required the addition of a "layer" to the graph.

guaranteeing that the preconditions of all the applied resolvers are, eventually, satisfied (constraint 1) and that at least one resolver is active whenever the flaw becomes active (constraint 2)[6].

Analogously, for each resolver $\rho_j$, we enforce the causality constraint constraint 3, guaranteeing that the application of a resolver activates (and resolves!) the associated flaw.

$$(\neg\rho_j \vee eff(\rho_j)) \tag{3}$$

The last aspect to consider concerns the update of the $\sigma$ variables as a consequence of the application of a rule application resolver and of a unification resolver. Specifically, each rule application resolver $\rho_a$ binds the $\sigma_a$ variable of the goal token, whose rule has been applied, to assume the $active$ value (formally, $\rho_a == \varphi(\sigma_a) == active$). Finally, for each unification resolver $\rho_u$ representing the unification of a token $\sigma_u$ with a token $\sigma_t$, the constraints $\neg\rho_u \vee \sigma_u == unified$ and $\neg\rho_u \vee \sigma_t == active$ guarantee the update of the $\sigma$ variables while adding $\varphi(\sigma_t)$ to the preconditions of $\rho_u$ guarantees the operation of the heuristic.

## 5.2 An explanatory example

In order to better understand how the heuristic and causality constraints work we introduce, in this section, a very simple example involving an agent moving between different locations either by driving or by flying (which, in turn, requires good weather). Figure 4 shows an example of the graph which is generated for solving the problem of going from $l_0$ (a fact) to $l_1$ (a goal).

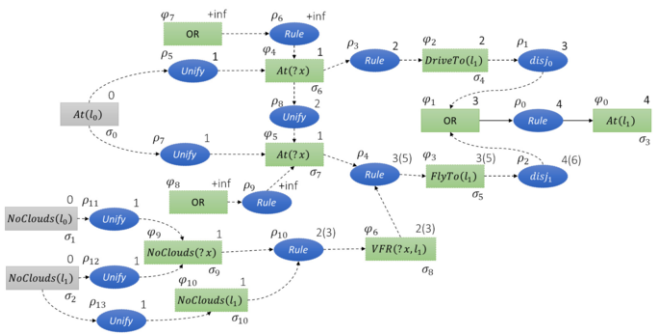



**Figure 4**: An example of causal graph with lifted variables.

Estimated costs for flaws (boxes) and resolvers (ovals) are on their upper right. Notice that, in the example, the flaw $\varphi_0$ can only be solved by resolver $\rho_0$ which is directly applied (solid lines represent what is in the current partial solution). Additionally, since $\varphi_0 = \varphi(\sigma_3)$, the $active$ value is assigned to $\sigma_3$. The first flaw to be solved is, hence, $\varphi_1$, which can be solved either with resolver $\rho_1$, having an estimated cost of 3, or with resolver $\rho_2$ having an estimated cost of 4[7]. Applying, for example, the least expensive resolver $\rho_1$ would lead, as a consequence of constraint propagation, to the activation of the flaw $\varphi_2$ (notice that $precs(\rho_1) = \{\varphi_2\}$ and $cause(\varphi_2) = \{\rho_1\}$) which can be solved with the sole resolver $\rho_3$, which in turn activates the flaw $\varphi_4$ which is solved with resolver $\rho_5$ leading to a solution. Finally, since $\varphi_4 = \varphi(\sigma_6)$, the $unified$ value is assigned to $\sigma_6$.

## 6 Current results

The causal graph, described in the previous section, has been implemented within the ORATIO solver[8]. In order to show the effectiveness of the proposed approach, we tested the solver, enhanced with the above heuristic, on different instances of the GOAC domain. Specifically, the Goal Oriented Autonomous Controller (GOAC) was an ESA initiative aimed at defining a new generation of software autonomous controllers to support increasing levels of autonomy for robotic task achievement. In particular, the domain, initially defined in [26] and more recently cited in [13], aims at controlling a rover to take a set of pictures, store them on board and dump the pictures when a given communication channel was available. The interesting aspect of this domain is that communication can only take place within specific visibility windows that take into account the astronomical motions of the planets/satellites which, in some cases, may stand between the transmitting and receiving stations. The presence of these visibility windows, in particular, requires an explicit modeling of temporal aspects in order to adequately plan the transmission of information and can hence easily be modeled through, and solved by, timeline-based planners. The problem is made more interesting by the presence of constraints which include the available resources (e.g., memory and battery) as well as by having a distance matrix, among the possible locations, which might be not completely connected.

Figure 5 shows the execution times of different timeline-based solvers (i.e., EPSL [9], AP2 [26], J-TRE [14], one of the precursors of ORATIO using a less accurate heuristic [1], and the more recent PLATINUm [50]) as well as a couple of temporal-planning solvers (i.e., OPTIC [2] and COLIN (see [12]), both based on a classic FF-style forward chaining search [35]) in solving different instances of the GOAC problem. In particular, problems are obtained by varying the problem complexity along the number of pictures to be taken and the number of communication windows. Notice that if on the one hand the increasing number of communication windows raises the complexity of the planning problem with a combinatorial effect, on the other hand an higher number of such windows might allow the planner to more easily find room for transmitting. More in general, among all the generated problem instances, the ones with higher number of required pictures and higher number of visibility windows result as the hardest ones. The right mix of causal and temporal aspects makes the GOAC problem particularly complex to the point that some of the planners, beyond a certain number of pictures to collect and data dumps, show serious scalability issues. As shown in the figure, besides being considerably more efficient, compared to other timeline-based planners, ORATIO is also able to solve more complex instances. Compared to the temporal-planning solvers, however, it is clear that, despite significant improvements, there is still a performance gap to fill. Possible explanations of this gap include the maintenance, in the current state of the solvers, of the consistency between the various constraints (which is not required in the forward state space search planners), in addition to the greater effectiveness of the FF heuristics. Another aspect to take into consideration regards the possibility of making the graph more accurate, so as to be able to represent heuristics as $h^2$ [31, 30]. Since it is not possible to recognize the mutual exclusivity between the resolvers directly from

[6] Notice that it is possible to introduce further constraints whenever it is known that the resolvers of a flaw are, among them, mutually exclusive.

[7] In the figure, the estimated costs are represented in the upper right of the flaws/resolvers and are computed through the $h_{max}$ heuristic. Whenever they do not coincide, in parenthesis is also represented the value from the $h_{add}$ heuristic.

[8] https://github.com/pstlab/oRatio

the rules' structure, we have not yet found an effective approach for implementing it.

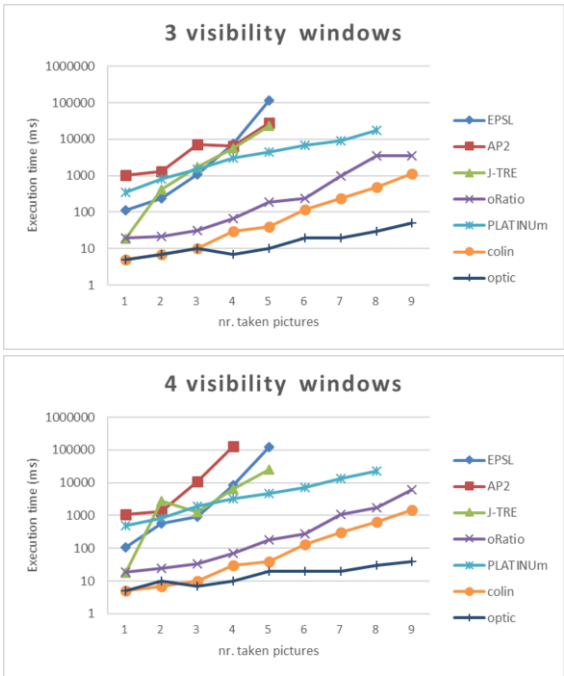



**Figure 5**: Execution times of different solvers to instances, of increasing complexity, of the GOAC problem.

## 6.1 Comments on results

Although, for the moment, there are solvers able to solve the GOAC problem more efficiently than the ORATIO solver, we believe that the current results are nevertheless significant. In the first place, indeed, the heuristic described in this document proposes a complete paradigm shift for timeline-base planners: we pass from heuristics based on the current partial solution (i.e., the current token network $\pi$) to heuristics based on all possible plans of what the planning problem says to us. In so doing, we have the possibility to anticipate the consequences of decisions before they are even taken and this results in more accurate plan synthesis. A second aspect to consider regards the possibility of modeling (and, above all, integrating) different kinds of reasoning which depart from those more closely related to automated planning. By removing the temporal parameters from the tokens, indeed, we obtain a form of reasoning which is similar to constrained logic programming. The proposed heuristics, in particular, remains valid, and paves the way for the efficient integration of different forms of reasoning such as, for example, automated planning and ontological reasoning. To better understand this aspect we can consider as an example the execution of Prolog program, whose efficiency strongly depends on the order in which the goals are defined within the rules. Note that different rules with the same goals in different order are semantically the same. In this case the programmer could be wrong to define the order or, even worse, the order could depend on the value of the parameters unavoidably affecting the performance of the resolution process. Finally, it is worth noting that the generated graph might be used to synthesize explanations (or, also, for proposing alternatives) of the decisions taken by the solver in an Explainable AI fashion [22, 5]. Some of the questions that an "explainable" planner might answer, indeed, are "why did you do action $A$? I would have done action $B$" or "why didn't you do something else?" or, even, "why can't you do that?". In such cases a graphical representation of the graph, appropriately modified (for example, eliminating the inactive part that represents alternative plans), could help the user to understand the planner's decisions, highlighting possible causal relationships between temporally distant decisions and therefore, in large problems, not immediately visible from the resulting timelines. Furthermore, since the graph contains alternative plans, it is possible to exploit it to show consequences of other choices, possibly showing more expensive or even impossible plans, generating, in the latter case, infeasibility explanations thanks to the constraint network's conflict analysis.

## 7 Conclusions

The reasons for introducing a new timeline-based formalism are manifold and range from the possibility to model, through a uniform formalism, continuous changes over time (see, for example, Figure 1a) to make the plans more flexible in the execution phase (relaxing the constraint, present in some formalisms, that forces the timelines to be completely filled over time). Whatever the formalism, reasoning upon these systems remains particularly challenging from a computational point of view. For this reason we have introduced a new heuristic that takes into account, before starting the search, *all* possible resolvers for *all* possible flaws that may emerge from the resolution process, so as to be able to make choices according to a more accurate criterion. Although encouraging, the results show that there is still work to be done. As an example, since it is possible to recognize mutex resolvers by propagating constraints, it is worth to investigate different approaches for representing the $h^2$ heuristic. Analogously, the proper adaptation of landmark-based heuristics, might represent a fruitful path toward the resolution efficiency. We hence believe that, through this document, we can lay the foundations for the definition of a new typology of heuristics for the efficient resolution of timeline-based planning problems.

## ACKNOWLEDGEMENTS

Authors work is partially supported by the INdAM-GNCS project *Metodi formali per tecniche di verifica combinata*, and by SI-ROBOTICS [9]. They are members of the OVERLAY[10] network.

## REFERENCES

[1] Riccardo De Benedictis and Amedeo Cesta, 'Investigating domain independent heuristics in a timeline-based planner', *Intelligenza Artificiale*, **10**(2), 129–145, (2016).
[2] J. Benton, Amanda Coles, and Andrew Coles, 'Temporal Planning with Preferences and Time-Dependent Continuous Costs', in *Twenty-Second International Conference on Automated Planning and Scheduling*, (2012).
[3] Avrim L. Blum and Merrick L. Furst, 'Fast Planning Through Planning Graph Analysis', *Artificial Intelligence*, **90**(1-2), 281–300, (1997).
[4] Blai Bonet and Héctor Geffner, 'Planning as Heuristic Search', *Artificial Intelligence*, **129**(1-2), 5–33, (2001).
[5] Michael Cashmore, Anna Collins, Benjamin Krarup, Senka Krivic, Daniele Magazzeni, and David Smith. Towards Explainable AI Planning as a Service, 2019.
[6] Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, and Angelo Oddi, 'Developing an End-to-End Planning Application from a Timeline Representation Framework', in *IAAI-09. Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference, Pasadena, CA, USA*, (2009).

[9] PON 676–Ricerca e Innovazione 2014-2020–G.A. ARS01_01120
[10] https://overlay.uniud.it

[7] Amedeo Cesta and Angelo Oddi, 'Gaining Efficiency and Flexibility in the Simple Temporal Problem', in *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*, eds., L. Chittaro, S. Goodwin, H. Hamilton, and A. Montanari. IEEE Computer Society Press: Los Alamitos, CA, (1996).

[8] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith, 'A Constraint-Based Method for Project Scheduling with Time Windows', *Journal of Heuristics*, **8**(1), 109–136, (Jan 2002).

[9] Amedeo Cesta, Andrea Orlandini, and Alessandro Umbrico, 'Toward a general purpose software environment for timeline-based planning', in *20th RCRA International Workshop on Experimental Evaluation of Algorithms for solving problems with combinatorial explosion*, (2013).

[10] S. Chien, D. Tran, G. Rabideau, S.R. Schaffer, D. Mandl, and S. Frye, 'Timeline-Based Space Operations Scheduling with External Constraints', in *ICAPS-10. Proc. of the $20^{th}$ Int. Conf. on Automated Planning and Scheduling*, (2010).

[11] Marta Cialdea Mayer, Andrea Orlandini, and Alessandro Umbrico, 'Planning and execution with flexible timelines: a formal account', *Acta Informatica*, **53**(6), 649–680, (Oct 2016).

[12] A. J. Coles, A. I. Coles, M. Fox, and D. Long, 'COLIN: Planning with Continuous Linear Numeric Change', *Journal of Artificial Intelligence Research*, **44**, 1–96, (May 2012).

[13] Amanda Jane Coles, Andrew Ian Coles, Moises Martinez Munoz, Okkes Emre Savas, Juan Manuel Delfa, Tomás de la Rosa, Yolanda E-Martín, and Angel García Olaya, 'Efficiently Reasoning with Interval Constraints in Forward Search Planning', in *Proceedings of the Thirty Third AAAI Conference on Artificial Intelligence*. AAAI Press, (1 2019).

[14] Riccardo De Benedictis and Amedeo Cesta, 'New Reasoning for Timeline based Planning - An Introduction to J-TRE and its Features.', in *ICAART 2012 - 4th International Conference on Agents and Artificial Intelligence*, pp. 144–153. SciTePress, (2012).

[15] Rina Dechter, *Constraint Processing*, Elsevier Morgan Kaufmann, 2003.

[16] Filip Dvorák, Arthur Bit-Monnot, Félix Ingrand, and Malik Ghallab, 'Plan-Space Hierarchical Planning with the Action Notation Modeling Language', in *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Limassol, Cyprus, (November 2014).

[17] Stefan Edelkamp, 'Planning with Pattern Databases', in *Sixth European Conference on Planning*, (2014).

[18] Stefan Edelkamp and Jörg Hoffmann, 'PDDL2.2: The language for the Classical Part of the 4th International Planning Competition', Technical Report 195, Institut für Informatik, (January 2004).

[19] Richard Fikes and Nils J. Nilsson, 'STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving', in *IJCAI*, pp. 608–620, (1971).

[20] Maria Fox and Derek Long, 'PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains', *Journal of Artificial Intelligence Research*, **20**, 61–124, (2003).

[21] Maria Fox and Derek Long, 'Modelling Mixed Discrete-continuous Domains for Planning', *Journal Of Artificial Intelligence Research*, **27**(1), 235–297, (2006).

[22] Maria Fox, Derek Long, and Daniele Magazzeni, 'Explainable Planning', *ArXiv*, (2017).

[23] Santiago Franco, Mauro Vallati, Alan Lindsay, and Thomas Lee McCluskey, 'Improving Planning Performance in PDDL+ Domains via Automated Predicate Reformulation', in *Computational Science – ICCS 2019*, pp. 491–498. Springer International Publishing, (2019).

[24] J. Frank and Ari K Jónsson, 'Constraint-Based Attribute and Interval Planning', *Constraints*, **8**(4), 339–364, (2003).

[25] S. Fratini, F. Pecora, and A. Cesta, 'Unifying Planning and Scheduling as Timelines in a Component-Based Perspective', *Archives of Control Sciences*, **18**(2), 231–271, (2008).

[26] Simone Fratini, Amedeo Cesta, Riccardo De Benedictis, Andrea Orlandini, and Riccardo Rasconi, 'APSI-based Deliberation in Goal Oriented Autonomous Controllers', *ASTRA*, **11**, (2011).

[27] Alfonso E. Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos, 'Deterministic planning in the fifth international planning competition: {PDDL3} and experimental evaluation of the planners', *Artificial Intelligence*, **173**(5-6), 619–668, (2009). Advances in Automated Plan Generation.

[28] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998.

[29] M. Ghallab and H. Laruelle, 'Representation and Control in IxTeT, a Temporal Planner', in *AIPS-94. Proceedings of the 2nd Int. Conf. on AI Planning and Scheduling*, pp. 61–67, (1994).

[30] Patrik Haslum, Blai Bonet, and Héctor Geffner, 'New Admissible Heuristics for Domain-Independent Planning', in *AAAI*, volume 5, pp. 9–13, (2005).

[31] Patrik Haslum and Héctor Geffner, 'Admissible Heuristics for Optimal Planning', in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pp. 140–149. AAAI Press, (2000).

[32] Malte Helmert, 'The Fast Downward Planning System', *Journal of Artificial Intelligence Research*, **26**(1), 191–246, (2006).

[33] Malte Helmert, Patrik Haslum, and Jörg Hoffmann, 'Flexible Abstraction Heuristics for Optimal Sequential Planning', in *ICAPS*, pp. 176–183, (2007).

[34] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim, 'Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces', *Journal of the ACM (JACM)*, **61**(3), 16, (2014).

[35] Jörg Hoffmann, 'FF: The Fast-Forward Planning System', *AI Magazine*, **22**(3), 57–62, (2001).

[36] Jörg Hoffmann and Bernhard Nebel, 'The FF Planning System: Fast Plan Generation Through Heuristic Search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[37] Jörg Hoffmann, Julie Porteous, and Laura Sebastia, 'Ordered Landmarks in Planning', *Journal of Artificial Intelligence Research*, **22**, 215–278, (2004).

[38] A.K. Jonsson, P.H. Morris, N. Muscettola, K. Rajan, and B. Smith, 'Planning in Interplanetary Space: Theory and Practice', in *AIPS-00. Proceedings of the Fifth Int. Conf. on AI Planning and Scheduling*, (2000).

[39] Henry Kautz and Bart Selman, 'Planning as Satisfiability', in *ECAI*, volume 92, pp. 359–363, (1992).

[40] Philippe Laborie, 'Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results', *Artificial Intelligence*, **143**, 151–188, (February 2003).

[41] Philippe Laborie and Malik Ghallab, 'Planning with Sharable Resource Constraints', in *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, pp. 1643–1649. Morgan Kaufmann Publishers Inc., (1995).

[42] Christophe Lecoutre, *Constraint Networks: Techniques and Algorithms*, Wiley-IEEE Press, 2009.

[43] Nicola Muscettola, 'HSTS: Integrating Planning and Scheduling', in *Intelligent Scheduling*, ed., Zweben, M. and Fox, M.S., Morgan Kauffmann, (1994).

[44] Nicola Muscettola, Stephen Smith, Amedeo Cesta, and Daniela D'Aloisi, 'Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture', *IEEE Control Systems*, **12**, (03 1992).

[45] Wiktor Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio, 'Heuristic Planning for PDDL+ Domains', in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pp. 3213–3219. AAAI Press, (2016).

[46] Julie Porteous, Laura Sebastia, and Jörg Hoffmann, 'On the Extraction, Ordering, and Usage of Landmarks in Planning', in *Sixth European Conference on Planning*, (2014).

[47] David E. Smith, Jeremy Frank, and William Cushing, 'The ANML language', in *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, (September 2008).

[48] David E. Smith, Jeremy Frank, and Ari K. Jónsson, 'Bridging the Gap Between Planning and Scheduling', *Knowledge Engineering Review*, (2000).

[49] Sebastian Stock, Masoumeh Mansouri, Federico Pecora, and Joachim Hertzberg, 'Hierarchical hybrid planning in a mobile service robot', in *KI 2015 Proceedings*, pp. 309–315, (2015).

[50] Alessandro Umbrico, Amedeo Cesta, Marta Cialdea Mayer, and Andrea Orlandini, 'Platinum: A new framework for planning and acting', in *AI*IA 2017 Proceedings*, pp. 498–512, (2017).

[51] Gérard Verfaillie, Cédric Pralet, and Michel Lemaître, 'How to model planning and scheduling problems using constraint networks on timelines', *The Knowledge Engineering Review*, **25**(3), 319–336, (2010).

[52] Daniel S. Weld, 'An Introduction to Least Commitment Planning', *AI Magazine*, **15**(4), 27–61, (1994).