Integrating Resource Management and Timeline-Based Planning

Alessandro Umbrico,[1] Amedeo Cesta,[1] Marta Cialdea Mayer,[2] Andrea Orlandini[1]
[1] Istituto di Scienze e Tecnologie della Cognizione, Consiglio Nazionale delle Ricerche, Italy
Email: name.surname@istc.cnr.it
[2] Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, Italy
Email: cialdea@ing.uniroma3.it

Abstract

This paper describes how explicit resource reasoning is added within an existing temporal planning framework that uses timelines as plan representation. The work is grounded on a recent formalization of timeline-based planning that here is extended to model and reason over resources. The formal account is then fully implemented as an extension of the PLATINUM planner and tested to demonstrate its new capabilities.

Introduction

Planning and Scheduling (P&S) address the same problem focusing on two different aspects. *Planning* concerns the synthesis of actions an agent must perform to achieve some objectives (*which* actions to perform). *Scheduling* concerns the sequencing of a set of known actions in order to satisfy some temporal and/or resource requirement constraints (*when* to perform actions). Very often in the literature the two processes are considered as separate and distinct phases of a solving process. Some planning frameworks (Ghallab and Laruelle 1994; Barreiro et al. 2012; Fratini et al. 2011) have proposed their integration in a unified solving process. Under such a perspective, they are part of the same solving mechanism and therefore an agent can interleave P&S decisions during plan generation. Somehow for quite some time such a capability has been pursued as key for adapting planners behavior to the features of different real-world scenarios and generate more effective solutions. Despite several interesting results a clean uniform management of P&S is still to be achieved, especially with respect to the management of reservoir resources. This work extends some recent formal results on timeline-based planning (Cialdea Mayer, Orlandini, and Umbrico 2016; Umbrico, Orlandini, and Cialdea Mayer 2015) and pursues the integrated management of discrete and reservoir resources in the public domain temporal planner called PLATINUM (Umbrico et al. 2017). This work provides the following contributions: (a) a formal definition of resource management problem consistent with (Cialdea Mayer, Orlandini, and Umbrico 2016) and (b) the implemented counterpart of the resource management formalization as an extension of PLATINUM. An experimental analysis is reported and discussed in order to show how PLATINUM works with resources. A final section completes the paper by identifying key directions for future improvements.

The Theoretical Standpoint

This section extends the formal framework presented in (Cialdea Mayer, Orlandini, and Umbrico 2016) to include the definition of resources and related constraints. Such an extension amalgamates concepts consistent with previous works like (Drabble and Tate 1994; Cesta and Stella 1997; Cesta, Oddi, and Smith 1998; Laborie 2003). The considered class of resources is quite general, i.e., it includes the *reservoir resources* (see also (Lehrer 1993)). A reservoir resource is a multi-capacity resource that can be consumed and/or produced by activities. A reservoir has a capacity ranging in a given interval and may have an initial level. The typical example of a reservoir is a fuel tank. Special kinds of reservoir resources are discrete ones. A discrete resource is used over some time interval: a certain quantity of resource is consumed at the start time of an activity and the same quantity is released at its end time. From the theoretical standpoint, the treatment of the most general class of resources therefore includes discrete ones, although in practice the treatment of the latter ones can be easier. We first report the basic notions regarding flexible timelines and plans (Cialdea Mayer, Orlandini, and Umbrico 2016).

A timeline-based planning domain contains the characterization of a set of *state variables*, representing the components of a system. A *state variable* x is characterized by the set of values it may assume, denoted by $\mathrm{values}(x)$, possible upper and lower bounds on the duration of each value, and rules governing the correct sequencing of such values. A *timeline* for a state variable is made up of a finite sequence of valued intervals, called *tokens*, each of which represents a time slot in which the variable assumes a given value. In general, timelines may be *flexible*, i.e., the start and end times of each of its tokens are not necessarily fixed time points, but may range in given intervals. For the sake of generality, temporal instants and durations are taken from an infinite set of non negative numbers $\mathbb{T}$, including 0. The notation $\mathbb{T}^\infty$ will be used to denote $\mathbb{T} \cup \{\infty\}$, where $t < \infty$ for every $t \in \mathbb{T}$. Tokens in a timeline for the state variable x are denoted by expressions of the form x^i, where the superscript indicates the position of the token in the timeline. Each token x^i is characterized by a value $v_i \in \mathrm{values}(x)$, that will be denoted by $\mathrm{val}(x^i)$, an end time interval $[e_i, e'_i]$ referred to as $\mathrm{end\_time}(x^i)$, and a duration interval $[d_i, d'_i]$ (as usual, the notation $[x, y]$ denotes the closed interval $\{t \mid x \leq t \leq y\}$). The start time interval $\mathrm{start\_time}(x^i)$ of the token x^i is $[0, 0]$ if x^i is the first token of the timeline (i.e. $i = 1$), otherwise, if $i > 1$, $\mathrm{start\_time}(x^i) = \mathrm{end\_time}(x^{i-1})$. So, a token has the form $x^i = (v_i, [e_i, e'_i], [d_i, d'_i])$ and a timeline is a finite sequence of tokens $x^1, \dots, x^k$. The metasymbol FTL (FTL_x) will henceforth be used to denote a timeline (for the state variable x), and **FTL** to denote a set of timelines. A *scheduled timeline* is a particular case where each token has a singleton $[t, t]$ as its end time, i.e., the end times are all fixed. A *schedule* of a timeline FTL_x is essentially obtained from FTL_x by narrowing down token end times to singletons (time points) in such a way that the duration requirements are fulfilled. In general, STL (or STL_x) and **STL** will be used as meta-variables for scheduled timelines and sets of scheduled timelines, respectively. In a given timeline-based domain, the behavior of state variables may be restricted by requiring that time intervals with given state variable values satisfy some temporal constraints. Such constraints are stated as a set of *synchronization rules* which relate tokens on possibly different timelines through temporal relations between intervals or between an interval and a time point. Such relations refer to token start or end points, that will henceforth be called *events*.

Definition 1 (Events and temporal relations)**.** *Let* **FTL** *be a set of timelines and* $\mathrm{tokens}(\mathbf{FTL})$ *the set of the tokens in* **FTL**. *The set* $\Upsilon(\mathbf{FTL})$ *of the* events *in* **FTL** *is the set containing all the expressions of the form* $\mathrm{start\_time}(x^i)$ *and* $\mathrm{end\_time}(x^i)$ *for* $x^i \in \mathrm{tokens}(\mathbf{FTL})$.

A temporal relation *has one of the following forms:*

$$p \leq_{[lb,ub]} p' \qquad p \leq_{[lb,ub]} t \qquad t \leq_{[lb,ub]} p$$

where $p, p' \in \Upsilon(\mathbf{FTL})$, $t, lb \in \mathbb{T}$ *and* $ub \in \mathbb{T}^\infty$.

Intuitively, $p \leq_{[lb,ub]} p'$ states that the token start/end point denoted by p occurs from lb to ub time units before that denoted by p'; $p \leq_{[lb,ub]} t$ states that the token start/end point denoted by p occurs from lb to ub time units before the time point t and the third relation that it occurs from lb to ub time units after t. Temporal relations are used to state the synchronization rules of the planning domain. Here, it is sufficient to say that such rules allow to state requirements of the following form: for every token x_0^i where the state variable x_0 assumes the value v_0, there exist tokens $x_1^{i_1}, \dots, x_n^{i_n}$ where the state variables $x_1, \dots, x_n$ hold some given specified values, and all these tokens are related one to another by some given temporal relations. Unconditioned synchronization rules are also allowed, and are useful for stating both domain invariants and planning goals.

When considering planning problems with resource constraints, beyond the set of state variables and the set of synchronization rules that must be satisfied, the specification of a timeline-based planning domain must contain a characterization of these resources. It consists of a set R of resource names, and each $r \in \mathsf{R}$ is associated to its capacity, i.e., an interval of values the resource may assume. Usually, assuming that resource values are in $\mathbb{R}$, such an interval is given by specifying the minimal and maximal value the resource may assume. State variables and resources will be referred to as the *system components*. The domain specification contains also information on what affects each resource value. We assume here that the value of a resource may change only at token start and end times, i.e., that it cannot increase or decrease *during* the development of an activity which affects it. In other terms, the change of resource availability at the start or end time of a token is considered to be instantaneous: continuous changes are not handled. In the domain specification the influence of activities on resources is described by saying which values affect a given resource (either consuming or producing it) at their start or end times, i.e., at the start or end time of tokens with those values. The expression used to state these facts has one of the following forms: $\mathrm{start\_time}(v)\ \mathrm{affects}_d\ r$, stating that, when a token with value v starts the resource r increases the value d (if $d < 0$, the resource is reduced); or $\mathrm{end\_time}(v)\ \mathrm{affects}_d\ r$, which means that the same happens at the token end time.

Definition 2 (Resources)**.** *A set of* resources Res *is modeled by a pair* $(\mathsf{R}, \mathrm{range})$, *where* R *is a set of identifiers called resources, and* $\mathrm{range} : \mathsf{R} \to (\mathbb{R} \cup \{-\infty\}) \times (\mathbb{R} \cup \{\infty\})$ *is a function mapping each resource* $r \in \mathsf{R}$ *to an interval* $[r_{min}, r_{max}]$, *with* $r_{min} < r_{max}$. *A* resource event *is an expression of either the form* $\mathrm{start\_time}(v)\ \mathrm{affects}_d\ r$ *or* $\mathrm{end\_time}(v)\ \mathrm{affects}_d\ r$, *where* $v \in \mathrm{values}(x)$ *for some state variable* x, r *is a resource name and* $d \in \mathbb{R}$, $d \neq 0$. *A* resource specification *for a set of state variables* SV *is a triple* $(\mathsf{R}, \mathrm{range}, \mathsf{ResE})$, *where* $(\mathsf{R}, \mathrm{range})$ *is a set of resources and* ResE *a set of resource events involving resources in* R *and values of state variables in* SV.

The intended meaning of resource ranges of the forms $[-\infty, r_{max}]$ or $[r_{min}, \infty]$ is that there is no bound to the resource consumption or production, respectively. When considering a set ResE of resource events, it is assumed that for any resource r and value v, there exists at most a value d such that $\mathrm{start\_time}(v)\ \mathrm{affects}_d\ r \in \mathsf{ResE}$, and at most a value d' such that $\mathrm{end\_time}(v)\ \mathrm{affects}_{d'}\ r \in \mathsf{ResE}$. The definition of a planning domain given in (Cialdea Mayer, Orlandini, and Umbrico 2016) is then extended by adding a resource specification $(\mathsf{R}, \mathrm{range}, \mathsf{ResE})$ to its components SV (a set of state variables) and $\mathcal{S}$ (a set of synchronization rules). A planning problem, beyond the components already introduced in (Cialdea Mayer, Orlandini, and Umbrico 2016), also specifies a *resource initializer*, i.e., a function establishing the initial amount of each resource. The initial value of each resource may in fact vary in different problems for the same planning domain.

Definition 3 (Resource initializer)**.** *A* resource initializer $\mathrm{init} : \mathsf{R} \to \mathbb{R}$ *is a function assigning an allowed initial value to each resource: for all* $r \in \mathsf{R}$, *with* $\mathrm{range}(r) = [r_{min}, r_{max}]$, $r_{min} \leq \mathrm{init}(r) \leq r_{max}$.

The next step is to characterize those sets of *scheduled* timelines that are valid with respect to a given planning domain. They must obviously satisfy all the synchronization rules of the domain (see (Cialdea Mayer, Orlandini, and Umbrico 2016) for the formal definition), but also the resource specification: whenever a token start or end time modifies some resource, the value of the latter must neither drop below its minimum, nor raise above its maximum. To this aim, the behavior of each resource in parallel to the temporal evolution of a given set **STL** of scheduled timelines has to be defined and it is called the resource *profile*.

Let $(\mathsf{R}, \mathrm{range}, \mathsf{ResE})$ be a resource specification for a given set of state variables SV. When a set of timelines for SV is considered, the meaning of resource events can be concretized and projected to the tokens in the timelines. If x^i is a token in a given set of timelines, the following abbreviations will be used in the sequel: $\mathrm{start\_time}(x^i)\, \mathrm{affects}_d\, r$ means that ResE contains $\mathrm{start\_time}(\mathrm{val}(x^i))\, \mathrm{affects}_d\, r$, and $\mathrm{end\_time}(x^i)\, \mathrm{affects}_d\, r$ stands for $\mathrm{end\_time}(\mathrm{val}(x^i))\, \mathrm{affects}_d\, r \in \mathsf{ResE}$. When considering a given resource specification $(\mathsf{R}, \mathrm{range}, \mathsf{ResE})$ and a set **FTL** of timelines, it is sometimes necessary to single out the token start and end times which affect a given resource. For each resource $r \in \mathsf{R}$ and $p \in \Upsilon(\mathbf{FTL})$, the *variation* of r at p, $\delta_r(p)$, is defined as follows:

$$\delta_r(p) = \begin{cases} d & \text{if } p\, \mathrm{affects}_d\, r \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, $\delta_r(p)$ is the (positive or negative) quantity that is added to r in the token start/end time denoted by p. The set $\Upsilon_r(\mathbf{FTL}) \subseteq \Upsilon(\mathbf{FTL})$ is the set of the events in **FTL** affecting r: $\Upsilon_r(\mathbf{FTL}) = \{p \in \Upsilon(\mathbf{FTL}) \mid \delta_r(p) \neq 0\}$.

The function δ_r and $\Upsilon_r(\mathbf{FTL})$ are independent from the value of the token start/end time, but depend only on the token names, their values and the resource events. Therefore, they are defined for both scheduled and non-scheduled timelines. On the contrary, resource profiles can only be defined for scheduled timelines. Assuming that each resource has a given initial value (specified by a resource initializer), the profile of r for a set **STL** of scheduled timelines is a function mapping each token start/end point p to the value of r in the corresponding time point in **STL** and is computed as the sum of the initial resource value plus the variations of r at all token start and end times preceding p. As each $p \in \Upsilon(\mathbf{STL})$ has a specific fixed value in **STL**, $\Upsilon(\mathbf{STL})$ is totally ordered according to such values: if $p, p' \in \Upsilon(\mathbf{STL})$, the relation $p \leq_{\mathbf{STL}} p'$ holds when the value of p in **STL** is less than or equal to the value of p' in **STL**.

Definition 4 (Resource profile)**.** *Let* **STL** *a set of scheduled timelines for the state variables in* SV *and* $(\mathsf{R}, \mathrm{range}, \mathsf{ResE})$ *a resource specification for* SV. *The* profile *of the resource* $r \in \mathsf{R}$ *in* **STL** *with initial value* d_0 *is the function* $\pi_{r,\mathbf{STL}} : \mathbb{R} \times \Upsilon(\mathbf{STL}) \rightarrow \mathbb{R}$ *such that:*

$$\pi_{r,\mathbf{STL}}(d_0, p) = d_0 + \sum_{p_i \leq_{\mathbf{STL}} p} \delta_r(p_i)$$

The consistency of a set of scheduled timelines with a set of resources takes into account the corresponding profiles.

Definition 5 (Resource consistency for scheduled timelines)**.** *Let* $\mathsf{ResSpec} = (\mathsf{R}, \mathrm{range}, \mathsf{ResE})$ *be a resource specification and* init *a resource initializer. A set* **STL** *of scheduled timelines is consistent with* $(\mathsf{ResSpec}, \mathrm{init})$ *if for every* $r \in \mathsf{R}$ *with* $\mathrm{range}(r) = [r_{min}, r_{max}]$ *and for every* $p \in \Upsilon_r(\mathbf{STL})$*:* $r_{min} \leq \pi_{r,\mathbf{STL}}(\mathrm{init}(r), p) \leq r_{max}$.

Finally, it can be defined when a set **STL** of scheduled timelines is valid with respect to a planning domain $\mathcal{D}$ and a resource initializer init: beyond satisfying the synchronization rules of the domain, **STL** must be consistent with the resource specification of $\mathcal{D}$. Considering flexible plans, the main component of a flexible plan is a set **FTL** of timelines, representing different sets $\mathbf{STL}_i$ of scheduled ones. It may be the case that not every $\mathbf{STL}_i$ satisfies the synchronization rules of the domain or is consistent with the resource constraints. The fundamental leading guide in (Cialdea Mayer, Orlandini, and Umbrico 2016) is the aim of defining plans so that they encapsulate all the information needed for execution. Consequently, a plan has to be equipped with additional information about the temporal relations in order to be coherent with the domain specification and guarantee that every set of scheduled timelines represented by a given flexible plan Π is valid.

A *flexible plan* Π is a pair $(\mathbf{FTL}, \mathcal{R})$, where **FTL** is a set of timelines and $\mathcal{R}$ is a set of temporal relations, involving token names in some timelines in **FTL**. An *instance* of the flexible plan $\Pi = (\mathbf{FTL}, \mathcal{R})$, is any schedule of **FTL** that satisfies every relation in $\mathcal{R}$. In order for a flexible plan $\Pi = (\mathbf{FTL}, \mathcal{R})$ to satisfy a synchronization rule it must be the case that $\mathcal{R}$ contains temporal relations guaranteeing what the rule requires. For the formal definitions the reader is again referred to (Cialdea Mayer, Orlandini, and Umbrico 2016), where it is also proved that whenever a flexible plan satisfies (in this sense) all the syncronization rules of a domain, then also any of its instances does. Here, the aim is to obtain a similar goal concerning resources: how can a flexible plan be characterized in such a way that all its instances are bound to be consistent with a given set of resources? Analogously to the satisfaction of synchronizations, the only way how a plan can restrict the set of its instances is by means of the set $\mathcal{R}$ of the temporal relations. In the case of resources, such set must *imply* that there are no *peaks* in resource usage, i.e., that none of its instances **STL** may have a resource profile $\pi_{r,\mathbf{STL}}$ where the value of r falls outside the allowed interval $[r_{min}, r_{max}]$ in some points. In order to check whether this condition holds for a given plan Π, something similar to a *flexible resource profile* can be computed for each resource r. In fact, at any time point, the value of r can not be computed precisely, but it can be established that it will fall in a given interval.

In other terms, though an exact resource profile does not exists for r in a flexible plan Π, what can be computed are an *optimistic* resource profile and a *pessimistic* one: the first one gives, for each event, the maximal value the resource can have, and the second one the minimum. The events of a scheduled timeline are totally ordered according to their values, but in a flexible plan they are only partially ordered. The partial order on $\Upsilon(\mathbf{FTL})$ of a plan $\Pi = (\mathbf{FTL}, \mathcal{R})$ can be deduced from both **FTL** and the temporal relations in $\mathcal{R}$, i.e., from the order among tokens in the same timeline, from the start and end times of tokens in different timelines, when they convey enough information, and from the temporal constraints in $\mathcal{R}$. Due to space restrictions, we only mention here some sample cases: start and end times of tokens in the same timeline are obviously totally ordered. Moreover, for instance, if $p = [t_i, t'_i]$, $p' = [t_j, t'_j]$ and $t'_i \bowtie t_j$ for $\bowtie \in \{<, \leq\}$, then $p \bowtie p'$. Other relations are derivable from $\mathcal{R}$. For example, if $p \leq_{[lb,ub]} p' \in \mathcal{R}$ with $lb > 0$, then $p < p'$. The notation $\Pi \vdash p \bowtie p'$, for $\bowtie \in \{\leq, <, =\}$, is used to assert that $p \bowtie p'$ in the partial order induced by Π.

Given an event p of the plan $\Pi = (\mathbf{FTL}, \mathcal{R})$, the partial order induced by Π allows for partioning $\Upsilon(\mathbf{FTL})$ into three sets: the set of the events p' that are bound to be scheduled before or simultaneously with p, those which must be scheduled after p, and the others, whose order with respect to p is not determined. The first and last ones are those that must be taken into account when setting bounds to the value a resource r may have at p. Moreover, only the events in $\Upsilon_r(\mathbf{FTL})$ are relevant.

Definition 6 (Before and Unknown events). *If* $(\mathsf{R}, \text{range}, \mathsf{ResE})$ *is a resource specification,* $\Pi = (\mathbf{FTL}, \mathcal{R})$ *a flexible plan and* $p \in \Upsilon$*, then* $B_r(p) = \{p' \in \Upsilon_r(\mathbf{FTL}) \mid \Pi \vdash p' \leq p\}, U_r(p) = \{p' \in \Upsilon_r(\mathbf{FTL}) \mid \Pi \not\vdash p' \leq p \textit{ and } \Pi \not\vdash p < p'\}$.

Let now **STL** be an instance of the plan Π and $p \in \Upsilon(\mathbf{STL}) = \Upsilon(\mathbf{FTL})$. Obviously, $p' \leq_{\mathbf{STL}} p$ for every $p' \in B_r(p)$. Among the elements of $U_r(p)$, some will be scheduled before or simultaneously with p in **STL** and some after it. Let $b_r(p) \subseteq U_r(p)$ be the set of the first ones:

$$b_r(p) = \{p' \in U_r(p) \mid p' \leq_{\mathbf{STL}} p\}$$

In other words, the set $\{p' \mid p' \leq_{\mathbf{STL}} p\}$ can be partitioned into two sets: $B_r(p) = \{p' \in \Upsilon_r(\mathbf{FTL}) \mid \Pi \vdash p' \leq p\}$ and $b_r(p) = \{p' \in \Upsilon_r(\mathbf{FTL}) \mid p' \leq_{\mathbf{STL}} p \text{ and } \Pi \not\vdash p' \leq p\}$ (the events which are scheduled before or simultaneously with p in **STL** but could also be scheduled after p, according to the plan Π). Therefore the profile of the resource r in **STL**, with initializer init, has, for each event p, the value determined as follows:

$$\begin{aligned}\pi_{r,\mathbf{STL}}(\text{init}(r), p) &= \text{init}(r) + \textstyle\sum_{p_i \leq_{\mathbf{STL}} p} \delta_r(p_i) \\ &= \text{init}(r) + \textstyle\sum_{p_i \in B_r(p)} \delta_r(p_i) \\ &\quad + \textstyle\sum_{p_i \in b_r(p)} \delta_r(p_i)\end{aligned}$$

Pessimistic and optimistic estimates of $\pi_{r,\mathbf{STL}}(\text{init}(r), p)$ can be given by minimizing or maximazing, respectively, the unknown value of the second sum. In order to do this, the set $U_r(p)$ is partitioned into the set of events consuming the resource r and those which produce it:

Definition 7 (Producers and Consumers). *Let* $\mathsf{ResSpec}$ *be a resource specification and* Π *a flexible plan. Then:* $C_r(p) = \{p' \in U_r(p) \mid \delta_r(p') < 0\}, P_r(p) = \{p' \in U_r(p) \mid \delta_r(p') > 0\}$.

In simpler words, both $C_r(p)$ and $P_r(p)$ contain events whose order w.r.t. p is not determined by the partial order induced by the plan, and $C_r(p)$ contains those which affect r negatively (the consumers) while $P_r(p)$ contains the resource producers. The optimistic estimate consists in considering the consumption of resources as late as possible and the productions as early as possible. The pessimistic estimate can be obtained by reasoning the opposite way.

Definition 8 (Optimistic and pessimistic profiles). *Let* $(\mathsf{R}, \text{range}, \mathsf{ResE})$ *be a resource specification,* $\Pi = (\mathbf{FTL}, \mathcal{R})$ *a flexible plan,* init *a resource initializer and* $r \in \mathsf{R}$*. The optimistic and pessimistic profiles of the resource* r *are the functions* ORP_r *and* PRP_r*, respectively, mapping* $\Upsilon(\mathbf{FTL})$ *to* $\mathbb{R}$*, defined as follows:*

$$\begin{aligned}ORP_r(p) = \quad & \text{init}(r) + \textstyle\sum_{p' \in B_r(p)} \delta_r(p') \\ & + \textstyle\sum_{p' \in P_r(p)} \delta_r(p') \\ PRP_r(p) = \quad & \text{init}(r) + \textstyle\sum_{p' \in B_r(p)} \delta_r(p') \\ & + \textstyle\sum_{p' \in C_r(p)} \delta_r(p')\end{aligned}$$

The optimistic resource profile assumes that every producer of the resource r that may be scheduled before or simultaneusly with the given event p is actually scheduled that way, while the resource consumers are postponed. In the pessimistic one, the role of producers and consumers is inverted. The conformance of a plan to a set of resources and its validity w.r.t. a planning domain can finally be defined.

Definition 9 (Plan validity). *A flexible plan* $\Pi = (\mathbf{FTL}, \mathcal{R})$ *satisfies the resource specification* $(\mathsf{R}, \text{range}, \mathsf{ResE})$ *with resource initializer* init *if for any resource* $r \in \mathsf{R}$ *with* $\text{range}(r) = [r_{min}, r_{max}]$*, and any* $p \in \Upsilon_r(\mathbf{FTL})$*, it holds that* $PRP_r(p) \geq r_{min}$*, and* $ORP_r(p) \leq r_{max}$.

A flexible plan $\Pi = (\mathbf{FTL}, \mathcal{R})$ *is* valid *with respect to a planning domain* $\mathcal{D} = (SV, \mathcal{S}, \mathsf{Res}, \mathsf{ResE})$ *and the resource initializer* init *iff the following conditions hold:* **FTL** *is a set of timelines for the state variables* SV*;* Π *satisfies all the synchronization rules in* $\mathcal{S}$*;* Π *satisfies the resource specification of* $\mathcal{D}$ *with resource initializer* init.

It must now be proved that every instance of a valid plan Π is also valid w.r.t. a given domain and resource initializer. As far as synchronization rules are concerned, Theorem 1 in (Cialdea Mayer, Orlandini, and Umbrico 2016) does the job. The next results shows that plan validity propagates to its instances also for what concerns resources. This means that any execution of a flexible plan is bound to satisfy all the domain constraints. Proofs are omitted because of space restrictions.

Theorem 1. *If the flexible plan* Π *satisfies the resource specification* $\mathsf{ResSpec}$ *with initializer* init*, then any instance of* Π *is consistent with* $(\mathsf{ResSpec}, \text{init})$.

The following definition relates flexible plans and planning problems.

Definition 10 (Solution plans). *Let* $\mathcal{P} = (\mathcal{D}, \mathcal{G}, H, \text{init})$ *be a planning problem and* $\Pi = (\mathbf{FTL}, \mathcal{R})$ *a flexible plan.* Π *is a* flexible solution plan *for* $\mathcal{P}$ *if the following conditions hold:* Π *is valid with respect to* $\mathcal{D}$ *with* init*;* Π *satisfies the synchronization rule* $S_{\mathcal{G}}$ *representing the goal* $\mathcal{G}$*; the end time of the last token of every timeline in* **FTL** *is* $[H, H]$ *(i.e., the system is planned up to the problem horizon).*

Now the question is: how can plans satisfying a given resource specification be computed? The conditions in Def. 9 are not helpful in this respect. The following result gives a different characterization of a plan Π that does not satisfy a given resource specification. It identifies which *flaws* have to be solved to refine the plan and obtain a valid one, whenever it is possible to do so, i.e. when there exists at least an instance of Π that is consistent with the resource specification.

Theorem 2. *Let* $\mathsf{ResSpec} = (\mathsf{R}, \mathrm{range}, \mathsf{ResE})$ *be a resource specification,* init *a resource initializer and* $\Pi = (\mathbf{FTL}, \mathcal{R})$ *a flexible plan such that, for some* $r \in \mathsf{R}$ *and* $p \in \Upsilon_r(\mathbf{FTL})$*, either* $PRP(p) < r_{min}$ *or* $ORP(p) > r_{max}$*. If* **STL** *is an instance of* Π *and* **STL** *is consistent with* $(\mathsf{ResSpec}, \mathrm{init})$*, then:*

1. *if* $PRP_r(p) < r_{min}$ *then one of the following two conditions holds: (a) there exists* $p' \in C_r(p)$ *such that* **STL** *is an instance of* $\Pi' = (\mathbf{FTL}, \mathcal{R} \cup \{p' > p\})$*; (b) there exists* $p' \in P_r(p)$ *such that* **STL** *is an instance of* $\Pi' = (\mathbf{FTL}, \mathcal{R} \cup \{p' \leq p\})$*.*
2. *if* $ORP_r(p) > r_{max}$ *then one of the following two conditions holds: (a) there exists* $p' \in P_r(p)$ *such that* **STL** *is an instance of* $\Pi' = (\mathbf{FTL}, \mathcal{R} \cup \{p' > p\})$*; (b) there exists* $p' \in C_r(p)$ *such that* **STL** *is an instance of* $\Pi' = (\mathbf{FTL}, \mathcal{R} \cup \{p' \leq p\})$*.*

Theorem 2 justifies a non-deterministic algorithm that, applied to a partial plan Π, iteratively chooses a resource r and an event such that one of the two cases 1 or 2 of the theorem holds; then either a consumer $p' \in C_r(p)$ or a producer $p' \in P_r(p)$ is chosen (whose existence is guaranteedby the theorem whenever the plan can be refined up to a valid one), and the corresponding temporal relation is added to the plan. The loop continues until either both 1 and 2 are false, hence Π satisfies the resource specification, or the algorithm stops with a failure (when either the set of relations of the plan is contradictory, or for some r and p either $\mathrm{init}(r) + \sum_{p' \in B_r(p)} \delta_r(p') < r_{min}$ or $\mathrm{init}(r) + \sum_{p' \in B_r(p)} \delta_r(p') > r_{max}$). Termination is guaranteed by the fact that each iteration reduces the set $U_r(p)$ for some resource r and event p.

Resources in PLATINUM

PLATINUM[1] is a general-purpose planning and execution framework capable of dealing with *temporal uncertainty* (Umbrico et al. 2017) that complies with the formalization in (Cialdea Mayer, Orlandini, and Umbrico 2016). The framework is able to deal with *uncontrollable dynamics* at both planning and execution time. The PLATINUM solving process pursues a *plan refinement approach* which consists in iteratively refining a partial plan by reasoning in terms of *flaws* that must be solved. Flaw selection is supported by dedicated heuristics that guide the planning procedure. A PLATINUM-based planner relies on a set of data structures and algorithms called respectively *components* and *resolvers*. Components model the types of features that may compose a planning domain. They specify the set of states and constraints that characterize the temporal behaviors of a particular type of domain feature. Resolvers are dedicated algorithms that encapsulate the logic for building valid temporal behaviors of a particular component. The reader may refer to (Umbrico et al. 2017) for a more detailed description of the framework and the solving approach. However, it is important to point out that resolvers are not responsible for making decisions during the search process. They are responsible for detecting flaws on a component and computing all possible solutions of such flaws in order to guarantee completeness of the search. Each solution of a flaw represents a branch in the search and it is up to the planner deciding which flaw to solve and which solution to apply for search expansion (i.e., plan refinement).

The types of flaws a PLATINUM-based planner is capable to deal with depend on the set of components and resolvers available in the framework. PLATINUM provides *state variables components* and the related resolvers that allow a planner to build valid *timelines* according to the semantics proposed in (Cialdea Mayer, Orlandini, and Umbrico 2016). Thus, PLATINUM has been extended by adding new components and new resolvers in order to properly deal with *discrete* and *reservoir* resources.

Resources as new Components

Discrete and reservoir resources have been studied in scheduling literature, e.g., (Bartush, Mohring, and Radermacher 1988; Cesta, Oddi, and Smith 2002; Lombardi and Milano 2012). Such works rely on a a-priori known static set of activities composing a plan and usually P&S are integrated as two distinct and loosely coupled phases of the solving process. Such distinct separation between P&S leads to rigid and not efficient solving processes especially when reservoir resources are taken into account. In fact, reservoir resources may affect the set of decisions that compose a plan and not only the set of constraints. Therefore, monolithic P&S phases could generate a number of invalid intermediate results and increase the need for backtracking. For example, the planning phase could generate an intermediate solution which is not valid with respect to reservoir resources. Then, the following scheduling phase could add some production activities to fix the plan. Such activities would be integrated "too late" in the intermediate (rigid) solution which could not be flexible enough to accomodate such changes. Thus, the planning phase could be forced to retract part of the found solution. The aim here is to realize a tighter integration between P&S in order to achieve a faster computation and also a better control of the search process. Inspiration comes from previous works like (Laborie and Ghallab 1995; Cesta and Stella 1997; Laborie 2003) but the whole concept is integrated in PLATINUM.

PLATINUM has been extended with new components and resolvers in order to model discrete and reservoir resources and encapsulate the logic for detecting flaws and computing possible solutions. In this way, discrete and reservoir resource management is integrated in the partial plan refinement procedure at "flaw level" and, thus, the solving process is capable of dynamically interleaving P&S decisions. The current set of components and resolvers follow the formal characterization in the first part of this paper. As a consequence of Theorem 1, the generated flexible plans are *robust* at execution time. In particular, every instance **STL** of a plan is valid with respect to the resource capacity constraints. In the case of discrete resources, the developed resolver follows the iterative simplification procedure elicited by Theorem 2. The approach takes inspiration from the formalization of RCPSP/max proposed in (Bartush, Mohring,

[1]https://github.com/pstlab/PLATINUm

and Radermacher 1988), and used in (Laborie and Ghallab 1995) plus the flattening techniques of (Cesta, Oddi, and Smith 2002) and derivatives. Furthermore the pursued robustness is in line with the approach of (Policella et al. 2004). A peak is a set of resource expressions called *Critical Set* (CS) which violates the capacity constraints of a resource. A peak is solved by identifying and solving *Minimal Critical Sets* (MCSs). An MCS is a subset of the resource expressions composing a CS such that the set of expressions obtained by removing any expression from the MCS does not violate the capacity constraints of the resource. In the case of reservoir resources the situation becomes more complex and the cited techniques cannot be applied directly. A peak can be solved by taking into account scheduling decisions like discrete resources but also planning decisions that introduce new tokens into the timelines (i.e., modifying the partial plan itself). Then, a flaw-based solving approach and the resulting tight integration between P&S decisions plays a key role to address such complex problems in an flexible and effective way.

Discrete Resource Management

A discrete resource r is a particular type of resource that can be consumed and produced by the same state variable value. A state variable value v decreases the capacity of r of a quantity d as soon as it starts. The same value v increases the capacity of r of the same quantity d as soon as it ends. Each state variable value v affecting a discrete resource r entails two resource expressions. Consumers are represented by resource expressions of the form $\mathrm{start\_time}(v)\ \mathsf{affects}_d\ r$, where $d < 0$. Producers are represented by resource expressions of the form $\mathrm{end\_time}(v)\ \mathsf{affects}_d\ r$, where $d > 0$. Alg. 1 describes the procedure of a resolver for a discrete resource $r \in \mathsf{R}$ which takes as input the set of events affecting r, $\Upsilon_r(\mathbf{FTL})$, and returns a set of flaws Φ_r with feasible solutions. Given a set of events $\Upsilon_r(\mathbf{FTL})$ affecting a resource r, a resolver computes the pessimistic and optimistic profiles, in order to find peaks (i.e., flaws with respect to the refinement procedure) and compute the possible solutions. The symmetry between consumers and producers on discrete resources allows us to take into account only the pessimistic profile. Plans cannot violate the resource maximum capacity constraint because values affecting the resource cannot produce a quantity of resource higher than the quantity consumed. Consequently, it is possible to ignore the case $ORP_r(p) > r_{max}$ with respect to Definition 9 to detect flaws.

For each event p affecting a resource r (row 3), the resolver computes the pessimistic profile and adds the related critical set to the set of flaws (rows 3-9), when the peak condition is satisfied (row 4). The events that compose CS (row 5) are computed by taking into account $C_r(p)$ (see Definition 7) which represents the set of consumptions p' whose order is not determined with respect to p. Then, the resolver must compute a set of feasible solutions for each critical set detected. A peak on a discrete resource can be solved by posting precedence constraints between some of the tokens composing the critical set (rows 7-9). There is a number of precedence constraints that must be posted to solve a peak

Algorithm 1 Discrete resource management

```
1: function DetectAndSolveFlaws($r \in \mathsf{R}, \Upsilon_r(\mathbf{FTL})$)
2:     $\Phi_r \leftarrow \emptyset$                                    ▷ critical sets causing peaks
3:     for $p \in \Upsilon_r(\mathbf{FTL})$ do                          ▷ analyze events affecting $r$
4:         if $PRP(p, \Upsilon_r(\mathbf{FTL})) < r_{min}$ then         ▷ check pessimistic profile
5:             $\Phi_r \leftarrow \Phi_r \cup criticalSet(PRP(p, \Upsilon_r(\mathbf{FTL})))$
6:     for $CS \in \Phi_r$ do
7:         for $MCS \in sample(CS)$ do                                  ▷ analyze minimal critical sets
8:             for $p' \in MCS$ do                                      ▷ compute MCS solutions
9:                 $CS \leftarrow exprScheduling(p', MCS \setminus \{p'\})$
10:        $CS \leftarrow sort(CS, preservedSpaceHeuristics)$
11:    return $\Phi_r$
```

and different possible combinations of such constraints. The resolver identifies such constraints by leveraging the concept of MCS in order to decompose a CS into a "simple critical set" which can be solved by posting a single precedence constraint. Thus, each critical set CS is sampled in a number of MCSs (row 7) each of which is analyzed in order to identify a feasible precedence constraints that solve the considered MCS and therefore simplify the "original" CS (rows 8-9). For each event $p' \in MCS$ the resolver computes a set of precedence constraints that schedule p' with respect to the other events of the MCS, $MCS \setminus \{p'\}$ (row 9). The algorithm takes into account all the possible solutions of all the possible MCS that can be extracted from a CS. The *preserved space heuristic* defined in (Laborie 2003) is used in order to evaluate CSs according to the (average) reduction of the search space induced by the associated solutions (row 10). Such information can be exploited by the general solving procedure to evaluate flaws and select which one to solve for plan refinement.

Reservoir Resource Management

Differently from discrete resources, a reservoir resource r can be simultaneously consumed or produced by different state variable values. A state variable value v decreases the capacity of r of a quantity d as soon as it starts. A state variable value v' increases the capacity of r of a quantity d' as soon as it ends, where $v \neq v'$. Consumers are represented by resource expressions of the form $\mathrm{start\_time}(v)\ \mathsf{affects}_d\ r$, where $d < 0$. Producers are represented by resource expressions of the form $\mathrm{end\_time}(v')\ \mathsf{affects}_{d'}\ r$, where $d' > 0$. Alg. 2 describes the procedure of a resolver for a reservoir resource $r \in \mathsf{R}$ which takes as input the set of events affecting r, $\Upsilon_r(\mathbf{FTL})$, and returns a set of flaws Φ_r with feasible solutions. In this case, producers and consumers are not symmetric. Different state variable values can consume or produce different quantities of a resource. Thus, differently from Algorithm 1, it is necessary to consider both pessimistic and optimistic profiles to compute and solve peaks.

For each event p affecting r, the resolver computes the pessimistic ($PRP(p, \Upsilon_r(\mathbf{FTL}))$) and optimistic profiles ($ORP(p, \Upsilon_r(\mathbf{FTL}))$) and adds the related critical sets to the set of flaws (rows 3-7) by evaluating peak conditions (row 4 for pessimistic profiles and row 6 for optimistic ones). Given a set of peaks (i.e., the flaws Φ_r), a resolver computes solutions by leveraging the concept of MCS. Each CS

Algorithm 2 Reservoir resources management

```
1:  function DetectAndSolveFlaws($r \in \mathsf{R}, \Upsilon(\mathbf{FTL})$)
2:      $\Phi_r \leftarrow \emptyset$                                        ▷ critical sets causing peaks
3:      for $p \in \Upsilon_r(\mathbf{FTL})$ do                              ▷ analyze events affecting r
4:          if $PRP(p, \Upsilon_r(\mathbf{FTL})) < r_{min}$ then             ▷ check pessimistic profile
5:              $\Phi_r \leftarrow \Phi_r \cup criticalSet(PRP(p, \Upsilon_r(\mathbf{FTL}))$
6:          if $ORP(p, \Upsilon_r(\mathbf{FTL})) > r_{max}$ then             ▷ check optimistic profile
7:              $\Phi_r \leftarrow \Phi_r \cup criticalSet(ORP(p, \Upsilon_r(\mathbf{FTL})))$
8:      for $CS \in \Phi_r$ do
9:          for $MCS \in sample(CS)$ do                                      ▷ analyze minimal critical sets
10:             for $p' \in MCS$ do                                          ▷ solve MCS through scheduling
11:                 $CS \leftarrow exprScheduling(p', MCS \setminus \{p'\})$
12:             for $p' \in MCS$ do                                          ▷ solve MCS through planning
13:                 $CS \leftarrow exprPlanning(p', MCS \setminus \{p'\})$
14:             for $p' \in MCS$ do                                          ▷ solve MCS through prod/cons update
15:                 $CS \leftarrow exprUpdate(p', \Upsilon_r(\mathbf{FTL}))$
16:     return $\Phi_r$
```

is sampled in order to extract a list of MCS whose solutions are computed according to three different "scenarios" (row 9) that represent three different types of solution that can solve an MCS (rows 10-15). Specifically, each MCS can be solved by posting precedence constraints (rows 10-1), by adding producers or consumers (rows 12-13) or by updating the amount of resource consumed/produced by the events affecting the resource (rows 14-15). In the first case, for each $p' \in MCS$ the resolver computes a set of precedence constraints that schedule p' with respect to $MCS \setminus \{p'\}$ (row 1). If $\delta_r(p') < 0$ (consumer) precedence constraints take into account the events $p'' \in MCS \setminus \{p'\}$ that also belong to $P_r(p')$. Otherwise, if $\delta_r(p') > 0$ (producer) precedence constraints take into account the events $p'' \in MCS \setminus \{p'\}$ that also belong to $C_r(p')$. In the second case, for each $p' \in MCS$ the resolver adds new expressions and therefore new events p^* to $\Upsilon_r(\mathbf{FTL})$ that produce, if $\delta_r(p') < 0$ or consume, if $\delta_r(p') > 0$, the amount of resource needed to execute p' (row 13). Finally, in the third case (row 15), for each $p' \in MCS$ the resolver posts consumption or production constraints that update the resource profile by increasing or decreasing the amount of resource available in order to "execute" p'. In this case, an MCS is solved by neither scheduling nor adding resource events. Specifically, the resolver takes into account the events that are scheduled before p', $B_r(p')$. If $\delta_r(p') < 0$ (consumer) the resolver increases the quantity of resource produced by events $p'' \in B_r(p')$. If $\delta_r(p') > 0$ (producer) the resolver decreases the quantity of resource consumed by events $p'' \in B_r(p')$.

Experiments and Assessment

An experimental assessment of the extended PLATINUM framework was performed in order to demonstrate the feasibility of the envisaged P&S approach. Experiments were performed on a revised version of the satellite planning problem described in (Cialdea Mayer, Orlandini, and Umbrico 2016). The problem consists of a satellite which is orbiting around a target planet and a ground station on Earth for communication. The satellite can perform two types of operations (i.e., planning goals): (a) *science operations* to acquire data about known targets, (b) *communication operations* to send data to the ground station. Science operations are performed by pointing the satellite towards the target planet. Communication operations are performed by pointing the satellite towards the Earth when the ground station is visible/available. Some operations of the satellite are not controllable. Communication operations are modelled as *partially controllable values* because the actual duration of a data transfer may be affected by external factors. A discrete resource models the bandwidth of the communication channel, and a reservoir resource models the battery level of the satellite. Each communication requires a certain bandwidth amount and simultaneous communications cannot exceed the maximum bandwidth. The battery limits the number of science and communication operations the satellite may perform over time. Both science and communication operations consume a certain amount of battery when executed. When the battery level goes below a minimum level, a recharging operation is needed. Battery recharging is performed through solar panels and the planning model also considers Sun's visibility windows. Such visibility windows constrain the number and the temporal occurrence of recharging operations. Then, four planning domains were defined considering: no resource discrete resource only; reservoir resources only; both discrete and reservoir resources. On the above domains, a number of problem instances were defined by varying the number of: science operations the satellite must perform; windows available for communication; windows available for battery recharging. The number of science operations ranges from 1 to 5, each of which implies a distinct communication operation. The number of communication and recharging windows ranges from 1 to 3. Thus, a total number of 45 problems were defined combining such parameters. Experiments were designed with the aim of "stressing" the capability of the framework to integrate P&S choices in increasingly complex scenarios. The use of different windows for communication and recharging activities highly increase the number of combinations between P&S choices.

Two different PLATINUM configurations were run: **dfs**, applying a depth-first-search strategy and looking for a valid plan without considering any particular metric; **greedy**, applying a greedy search strategy which tries to minimize the plan's *makespan*. Both configurations use the hierarchy-based heuristics defined in (Umbrico, Orlandini, and Cialdea Mayer 2015) to support flaw selection. Each run was repeated 3 times considering average solving times, with a total number of 1080 performed experiments. The experiments ran on a MacBook Pro endowed with Intel Core i5 2.4GHz and 8GB RAM with a timeout of 300 seconds.

Fig. 1 and 2 show the results of **dfs** and **greedy** configurations. Fig. 1 aggregates data concerning the communication and recharging windows to show the performance of the planners with respect to the number of goals considered. Regardless of the number of available windows, the time needed to solve a problem increases with the number of science and communication operations the satellite must perform. The **dfs** configuration was always able to find a solution while the **greedy** configuration was several times

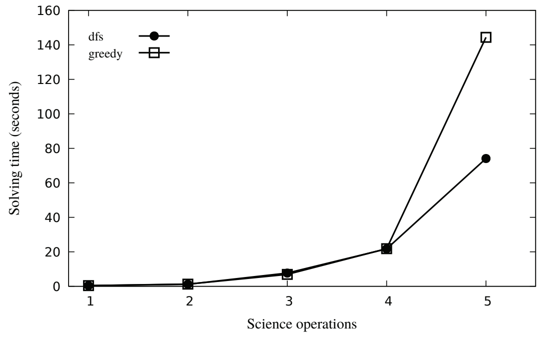

Figure 1: Average solving time of the planner configurations

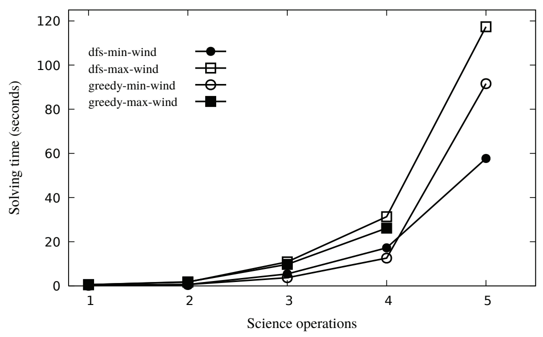

Figure 2: Average solving time of the planner configurations with respect to the min and max number of windows

(20) in *timeout*. The **dfs** outperforms **greedy** because **dfs** returns the first valid solution found while **greedy** explores a higher number of partial plans to find the plan with minimum makespan. This is especially true when several communication and recharging windows are available leading to a less efficient solving process but capable of generating “more efficient” plans. Indeed, the average *makespan* of the solutions generated by **greedy** (264) is better than the average *makespan* of the solutions generated by **dfs** (275).

Fig. 2 shows a more detailed view of the results: the average results obtained on the problems with the minimum and maximum number of available communication and recharging windows. The *dfs-min-wind* and *dfs-max-wind* show the average solving time of **dfs** on the problems with one communication and one recharging windows and the problems with three communication and three recharging windows respectively. The same holds for the *greedy-min-wind* and *greedy-max-wind* with respect to **greedy**. The diagram shows that a higher number of available windows increases the problem complexity. A higher number of communication and recharging windows affects the branching factor by increasing the possible P&S choices. The **greedy** configuration is not able to generate a plan for the problem with five goals, one communication window and one recharging window (*timeout*) but, it is able to generate a plan for the problem with five goals, three communication windows and three recharging windows. A motivation for such result is that six windows allow the planner to better characterize that

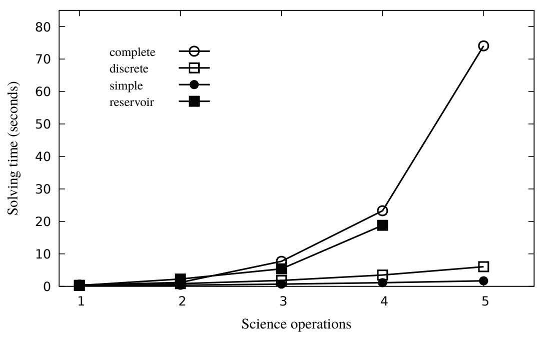

Figure 3: Aggregate solving time for different domain versions.

partial plans with respect to the makespan and therefore better guide the search towards a solution. The case with two windows instead leads the planner to generate many *equivalent* partial-plans (with respect to makespan) and it is not able to efficiently move towards a solution.

Finally, Fig. 3 shows the aggregated results obtained by running **dfs** on different satellite domain versions . The introduction of reservoir resources significantly increases the complexity of planning problems. Indeed, the problems defined on the *complete* and the *reservoir* versions of the domain are the most difficult to solve because of a larger branching factor entailed by the multiple alternatives to manage reservoir resources (see Alg. 2). It is worth noticiting that this complexity is softened for the complete domain version since the presence of discrete resources reduce the general branching factor. This is why no solution has been found (timeout) in the case of 5 goals and reservoir domain.

These experiments show that the proposed approach effectively solves complex problems that require a tighter P&S integration. The different results obtained with **dfs** and **greedy** suggest that the hierarchical solving approach represents a good starting point. However, it is necessary to further develop heuristics as well as search strategies in order to generate plans with specific objective functions (e.g., low *makespan*) in a more efficient way.

Conclusions and Future Works

This paper presents an extension of a recent characterization of the timeline-based planning approach introducing discrete and reservoir resources. The paper also presents an extension of PLATINUM, an existing timeline-based planner, introducing the capability of managing different types of resources and integrating them into a general plan refinement procedure. A set of experiments have shown the capabilities of the extended framework of integrating P&S during plan synthesis in an effective way. The experiments also show the need of investigating the definition of more informed heuristics as well as search strategies that better control the search to improve efficiency. A comparison with EUROPA (Barreiro et al. 2012) and other state-of-the-art hybrid systems like, e.g., CHIMP (Stock et al. 2015) or FAPE (Bit-Monnot 2016) represents a possible future work.

References

Barreiro, J.; Boyce, M.; Do, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; and Smith, D. 2012. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *ICKEPS 2012: the 4th Int. Competition on Knowledge Engineering for Planning and Scheduling*.

Bartush, M.; Mohring, R.; and Radermacher, F. 1988. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research* 16:201–240.

Bit-Monnot, A. 2016. *Temporal and Hierarchical Models for Planning and Acting in Robotics*. Ph.D. Dissertation, Doctorat de l'Université Federale Toulouse Midi-Pyrenees.

Cesta, A., and Stella, C. 1997. A Time and Resource Problem for Planning Architectures. In Steel, S., and Alami, R., eds., *ECP-97. Recent Advances in AI Planning, 4th European Conference on Planning, Toulouse, France, September 24-26, 1997, Proceedings*, volume 1348 of *Lecture Notes in Computer Science*, 117–129. Springer.

Cesta, A.; Oddi, A.; and Smith, S. 1998. Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *AIPS-98*, 214–223.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristics* 8(1):109–136.

Cialdea Mayer, M.; Orlandini, A.; and Umbrico, A. 2016. Planning and execution with flexible timelines: a formal account. *Acta Informatica* 53(6-8):649–680.

Drabble, B., and Tate, A. 1994. The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner. In *AIPS-94. Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, University of Chicago, Chicago, Illinois, USA, June 13-15, 1994*, 243–248.

Fratini, S.; Cesta, A.; De Benedictis, R.; Orlandini, A.; and Rasconi, R. 2011. APSI-based deliberation in Goal Oriented Autonomous Controllers. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *2nd Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS)*, 61–67.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, 2 Volumes*, 1643–1651.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artif. Intell.* 143(2):151–188.

Lehrer, N. 1993. KRSL Reference Manual 2.0.2 – ARPA/Rome Laboratory Planning and Scheduling Initiative. Technical report, ISX Corporation.

Lombardi, M., and Milano, M. 2012. A min-flow algorithm for minimal critical set detection in resource constrained project scheduling. *Artificial Intelligence* 182(Supplement C):58 – 67.

Policella, N.; Smith, S.; Cesta, A.; and Oddi, A. 2004. Generating Robust Schedules through Temporal Flexibility. In *ICAPS-04*, 209–218.

Stock, S.; Mansouri, M.; Pecora, F.; and Hertzberg, J. 2015. Online task merging with a hierarchical hybrid task planner for mobile service robots. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 6459–6464.

Umbrico, A.; Cesta, A.; Cialdea Mayer, M.; and Orlandini, A. 2017. PLATINUm: A New Framework for Planning and Acting. In Esposito, F.; Basili, R.; Ferilli, S.; and Lisi, F. A., eds., *AI*IA 2017 Advances in Artificial Intelligence*, 498–512. Cham: Springer International Publishing.

Umbrico, A.; Orlandini, A.; and Cialdea Mayer, M. 2015. Enriching a temporal planner with resources and a hierarchy-based heuristic. In *AI*IA 2015, Advances in Artificial Intelligence*. Springer International Publishing. 410–423.