IPSS: A Hybrid Approach to Planning and Scheduling Integration

María Dolores Rodriguez-Moreno, Angelo Oddi, *Member*, *IEEE*, Daniel Borrajo, and Amedeo Cesta

Abstract—Recently, the areas of *planning* and *scheduling* in Artificial Intelligence (AI) have witnessed a big push toward their integration in order to solve complex problems. These problems require both reasoning on which actions are to be performed as well as their precedence constraints (planning) and the reasoning with respect to temporal constraints (e.g., duration, precedence, and deadline); those actions should satisfy the resources they use (scheduling). This paper describes IPSS (Integrated Planning and Scheduling System), a domain independent solver that integrates an AI planner that synthesizes courses of actions with constraint-based techniques that reason based upon time and resources. IPSS is able to manage not only simple precedence constraints, but also more complex temporal requirements (as the Allen primitives) and multicapacity resource usage/consumption. The solver is evaluated against a set of problems characterized by the use of multiple agents (or multiple resources) that have to perform tasks with some temporal restrictions in the order of the tasks or some constraints in the availability of the resources. Experiments show how the integrated reasoning approach improves plan parallelism and gains better makespans than some state-of-the-art planners where multiple agents are represented as additional *fluents* in the problem operators. It also shows that IPSS is suitable for solving real domains (i.e., workflow problems) because it is able to impose temporal windows on the goals or set a maximum makespan, features that most of the planners do not yet incorporate.

Index Terms—Planning, scheduling, temporal reasoning, constraint satisfaction problem.

1 Introduction

Planning and scheduling have always been two very related but separate areas. Since some problems allow a strict separation between planning and scheduling, a possible approach to solving the complete planning problem (a valid plan with temporal and resource information) consists of assigning time and resources (using a scheduler) to the selected and ordered activities (generated by a planner). However, in other cases close to real applications, resource assignments affect the selection and ordering of activities and the problem is hard to solve by a simple sequencing of planning and scheduling. For this motivation, the AI community is showing an increasing interest in integrating features from both fields in order to define new algorithms which are able to find better quality solutions or to solve planning problems, with time and resource constraints, which cannot be easily solved by state-of-the-art planners.

Generally, in AI the term *planning* is used to describe the construction of a sequence of formally described world-states. A planning *domain* consists of a set of *operators* or action types. In AI planning systems, world states are often described in terms of predicates (*fluents*). The best-known formalism for planning is STRIPS [16], now represented in the standard language PDDL3 [19], in which operator effects are defined by an *add list* of fluents which become true when the operator is executed and a *delete list* of fluents which become false when the operator is executed. Problems that planners have when dealing with time and resource constraints relate to the fact that, although the time representation (e.g., durative actions) has been one of the main objectives in the language for the International Planning Competitions,[1] resource constraints are not modeled and handled differently from the rest of planning knowledge. In addition, solving time-resource dependent tasks implies the use of some measure of optimality, such as makespan, and there are few planners that can handle optimality criteria (though more and more planners are now incorporating some way of handling them, such as [14], [20], [21], [23], [36]).

Scheduling is an optimization task where limited resources are allocated over time among a partially ordered set of activities such that an objective function is optimized. From this perspective, the integration of Planning and scheduling systems provides an alternative way to deal with planning problems with time and resource constraints. In fact, the problem can be addressed in two ways:

- representing resource constraints within a STRIPS-like formalism, adding *fluents* to the domain description and modifying operators accordingly, and
- synthesizing an *extended* algorithm, which incorporates time and resource reasoning capabilities.

This work describes a novel framework called IPSS (Integrated Planning and Scheduling System) that follows the latter approach to solving planning problems with *time* and *multicapacity* resource constraints. In particular, our approach integrates a classical planner with a constraint-based module that models temporal and resource features

- M.D. Rodriquez Moreno is with the Departmento de Automática, Universidad de Alcala, 28871 Alcalá de Henares, Madrid, Spain. E mail: mdolores@aut.uah.es.
- A. Oddi and A. Cesta are with ISTC CNR, Italian National Research Council, Via S. Martino della Battaglia 44, I 00185 Rome, Italy. E mail: {angelo.oddi, amedeo.cesta}@istc.cnr.it.
- D. Borrajo is with the Departamento de Informática, Universidad Carlos III de Madrid, 28911 Leganés, Madrid, Spain. E mail: dborrajo@ia.uc3m.es

Manuscript received 19 Sept. 2005; revised 15 May 2006; accepted 20 July 2006; published online 18 Oct. 2006.
For information on obtaining reprints of this article, please send e mail to: tkde@computer.org, and reference IEEECS Log Number TKDE 0390 0905.

1. http://ipc.icaps conference.org/.

using state-of-the-art CSP (Constraint Satisfaction Problem solving) techniques. In the subdivision of work with this hybrid schema, the planner drives the search that decides which actions are in the plan. It receives continuous feedback from the CSP part that represents the temporal features, like duration of activities and the deadline for a solution, in a temporal constraint network. The CSP also handles resources usage like *agents performing actions* that are represented and reasoned upon as binary resources in a CSP representation (although multicapacity resources can also be handled). The two reasoning modules create a continuous loop that shows not only good modeling features, but also an interesting problem solving performance, as shown in the experimental part of this paper. In addition to integrated planning and scheduling, the proposed representation and the associated solver are also able to cope with different quality criteria, control knowledge to reduce the search, and learning tools.

The paper is organized as follows: Section 2 reviews the main approaches that integrate planning and scheduling. Section 3 presents the IPSS architecture and its components. Then, Section 4 presents an experimental setting and discusses the results. Finally, conclusions and future work are outlined.

2 The Integration of Planning and Scheduling

In some real-world domains, there is a class of planning problems that contains both the representation of activities (operators) as well as time constraints (as durations or time windows) and resources (as agents or fuel). In general, we can solve this class of problems with a *planning algorithm* or with the integration of planning and scheduling algorithms. In the current literature, the latter approach has evolved along two lines: *loosely coupled* and *strongly coupled* algorithms. In this section, we describe these approaches in some detail.

2.1 Loosely Coupled Algorithms

Within this approach the two subproblems of planning and scheduling are solved one after the other. The planner generates a sequence of atemporal operations that describes the precedence relations among activities. All the time-related and resource information is removed from the domain, letting the scheduler handle them. The input plan can be either a Partial Order plan or a Total Order plan. Total Order plans are sequences of operator instantiations (activities from the point of view of scheduling). This means a tight set of precedence constraints for the scheduler. However, not every precedence ordering between plan steps is necessary for maintaining its consistency, given that sometimes two activities can be executed in parallel. So, generally, these type of plans (and the planners that generate them [33]) are not the best candidates for a weak integration. This drawback can be solved using planning algorithms that either generate Partial Order plans directly [12], [33] or convert Total Order plans into Partial Order plans, by using a deordering algorithm, as described in [3], [38]. This last solution has a disadvantage: Finding an optimal deordering, an equivalent plan with the minimum number of ordering constaints as well as a plan with the minimal possible length, is an NP-hard problem. It is still possible to remove ordering constraints in polynomial time until the plan remains feasible and convert it into a *minimal deordered plan*, which is not optimal, but might represent a good compromise when the selection of the ordering constraints is performed under tailored heuristic criteria (in Section 3.2.2, we present an example of this criteria). In other recent integrations such as RealPlan [36], the resource allocation issue is solved separately from the planning process and the quality of the plan is measured by plan length.

Despite the simplicity of this integration schema, its main drawback is the lack of *interaction* between the two solving phases; when there is no solution to the final scheduling problem, a new input plan has to be generated from scratch and no information is provided from the "scheduling side" to the planning search in order to drive it toward a promising solution or to remove infeasible search paths with regard to the time and/or resource constraints.

2.2 Strongly Coupled Algorithms

In this approach, planning and scheduling problems are reduced to a uniform representation without the decomposition over two sequential subproblems as described above. The advantage of this approach is based on the idea that the reduction of scheduling and planning to a unique formalism facilitates the development of a set of solving algorithms which can take full advantage of the sharing of the planning and scheduling knowledge. The common formalism for representing planning and scheduling problems is the so-called Constraint Satisfaction Problem [26], which provides a general device for representing complex states, causal justifications, resource, and time constraints. Within the current literature and among the systems which can be classified in this category, we can mention HSTS [27] and IXTET [21] among others.

HSTS [27] is an integrated planning and scheduling system that breaks down a problem domain into a set of *state variables*. A state variable represents a *component* of the working domain which can assume a set of values over time, each one in a given time interval. The integrated planning and scheduling problem consists of finding a set of temporal evolutions—that is, sequences of interval values assumed over time by the state variables (components)—which assumes a set of specific values over a set of temporal intervals; these values are the *goals*.

IXTET [21] is a least commitment planner where the time logic relies on a restricted interval algebra represented by Time Points (TPS) using a Simple Temporal Problem (STP). Given a problem and a set of tasks (operators), the system generates a solution by successive refinements of the initial problem. A partial plan is composed of a set of temporal propositions (events, asserts, or resource usage), a time map (for the management of the temporal constraints post in each instance), and a table of variables (for the management of constraint posts on plan variables).

Resource conflicts are detected and solved through Minimal Critical Sets analysis [24] of a specific representation of the temporal constraints. A resource analysis module is integrated into the planning system to select a certain number of smallest Minimal Critical Sets and compute their minimal resolvers, whereas the other planning analysis modules focus on threats and pending subgoals.

2.3 What's in between?

In the current planning literature there are also other systems which adopt a schema for integrating planning and scheduling which lies in the *middle* of both classes of systems introduced in the previous two sections. We mainly identify two trends for planners in this area:

- GraphPlan-based [4] planners can easily extend the graph to represent temporal distances (durations) by means of time-graphs. Time is specified on each edge of the time-graph through a pair of numbers indicating the upper and lower bounds of the elapsed times between the TPS represented by the vertices connected by the edge. Among planners in this group, we can mention TPSYS [18] or SAPA [14].
- Planners not based on GraphPlan need to integrate methods to represent time to handle temporal events. It is a common practice to use techniques from the CS area, particularly, Temporal Constraint Satisfaction. In this group, we can mention CRIKEY [22], VHPOP, or the system that will be explained in detail in the next section, IPSS [31], [34].

  CRIKEY [22] is a temporal planning architecture which aims at recognizing scheduling aspects in a planning problem instance and dynamically separates these aspects from the strictly logical constraints. The system is based on FF [23] and a Simple Temporal Network (STN) solver and can deal with PDDL2.2 complexities.

  VHPOP [39] is a Partial Order planner loosely based on UCPOP [29]. It is fully compatible with level 1 of PDDL+. There is also support for planning with durative actions (level 3 of PDDL+) and this is accomplished by adding an STN to the regular plan representation of a Partial Order planner. The STN records temporal constraints between actions and supersedes the simple ordering constraints usually recorded by partial order planners. The use of STNs permits actions with duration interval constraints.

As we will show in the next sections, our working framework is more flexible than the ones described here. The combination of a planning algorithm and a Constraint Satisfaction Problem (CSP) is a powerful one: We can easily add search control knowledge and new inference (propagation) rules to the CSP representation in order to prune infeasible choices during the search. Our algorithm is more valuable from an applicative point of view. From the results, it is shown that classical planning algorithms are not well-suited for reasoning about resource and temporal constraints.

3 IPSS: A HYBRID APPROACH

The goal of our approach was to build a flexible, integrated system capable of planning and scheduling reasoning. The system is composed of two problem solvers where each one maintains its own knowledge representation, adequate for the type of reasoning that was required. So, the planner reasons about goals and operators that achieve those goals and establishes causal orderings among them. The scheduler is responsible for reasoning about the time and resource constraints of the problem. Hence, the essence of our problem solving approach is a two-level architecture integrating a constraint-based problem and a planning component which incrementally build a solution by interlacing planning and scheduling steps.

From a planning point of view, the search process can consider a more abstract vision of the planning problem (goals, operators, and establishing causal orderings among them) because time and resource reasoning tasks are delegated to the scheduler. Hence, the planning search process is relieved from establishing preconditions which represent time and resource constraints, which provides benefits in terms of the planning search complexity. The constraint-based component reasons about fundamental properties of time and resource constraints, such as satisfaction and entailment (or propagation). It contains and represents the time and the resource constraints accumulated at some computation step and supports various queries and operations over these constraints. Finally, this separation has some obvious software engineering benefits. It is possible to modify these components independently, e.g., one can easily change the search strategies of each component once the interfaces have been properly defined.

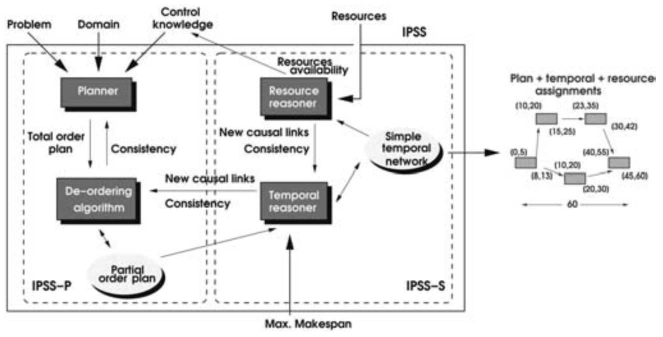


Fig. 1. IPSS architecture.

3.1 IPSS Software Architecture

Fig. 1 shows a high-level view of the architecture. The left part, IPSS P, corresponds to the planning component. It is comprised of the current implementation of two subsystems: a Total Order planner and a deordering algorithm. The right part, IPSS S, corresponds to the scheduling component and incorporates two reasoners: A temporal reasoner based on a Simple Temporal Network (STN) for the time related reasoning and a resource reasoner.

The inputs for the new architecture are the same ones as for the planner (domain theory, problem description, and control knowledge) plus two new ones referring to new capabilities: maximum makespan and a list of resources to be used.

There are two ways in which the problem solvers interact:

- Every time the planner applies an instantiated operator, adding it to the current incomplete plan, the scheduler is called on to identify whether any temporal or resource constraint is violated. If so, the planner backtracks, removing the operator from the plan. If no constraint is violated, the planning search continues.
- At any moment, the planner can determine what the status is of resource consumption or temporal constraints in order to make decisions. Thus, more informative decisions can be made.

In order to implement a first version of this generic architecture, we have used QPRODIGY [6], an extension of PRODIGY [37] to handle quality metrics, for the planner component, and a Constraint-Based Scheduler [10], [35] for the scheduling component. In the next sections, we describe in more detail each subcomponent and then we explain how they are integrated and what knowledge they exchange.

3.2 The Plan Reasoner

The plan reasoner, IPSS P (see Fig. 1), is composed of the QPRODIGY planner and the deordering algorithm. Both will be explained next.

3.2.1 The Planning Algorithm

IPSS P is based on a descendant of PRODIGY, QPRODIGY [6]. More specifically, QPRODIGY redefines PRODIGY4.0 (nonlinear version of the PRODIGY planner). Both PRODIGY and QPRODIGY explore the same space, using the same decision points on that search tree. PRODIGY searches a state-space bidirectionally, from an initial situation S toward the goals G (when applying the operators) and vice versa (when performing the subgoaling) [37]. It uses an augmented STRIPS representation that incorporates most PDDL2.2 features, plus some others (such as control knowledge language or functions for constraining operators variables values).

The difference between PRODIGY and QPRODIGY is that the latter uses a declarative language for expressing knowledge about different quality metrics in each domain operator. So, the user might define a list of cost metrics and compute the value for each metric in each operator. There are no predefined metrics, given that users can define their own. We believe this is conceptually cleaner than defining them in the effects list as is currently done in PDDL. Also, the user is not constrained by the type of functions to be used to compute such cost metrics, given that QPRODIGY can call any arbitrary function. Once it has cost metrics in the operators, QPRODIGY can also take as input a cost metric and a cost bound for that metric (optional). Then, it performs a branch-and-bound search on top of PRODIGY so that only solutions with less cost than the cost bound are searched for. If no cost bound is given, it will search for a valid solution. Also, since PRODIGY can search for more than one solution; once it finds the first one, its cost will be the new cost bound for the next solutions.

QPRODIGY, like PRODIGY, follows a four step decision cycle, as shown in Fig. 2. First, the system must decide whether to apply the applicable planning operators to the current situation S (forward mode) or to work on a goal in G (backward mode). In forward mode, the first currently applicable operator is applied if the actual cost of the already applied operators does not surpass the cost bound of the current cost metric (in PRODIGY, the operator is applied without having in mind the cost).[2] In backward mode, an unachieved goal $g \in G$ must be selected. Then, an operator O able to fulfill g must be chosen. And, finally, its unbound variables must be bound. Each one of these four decision points is backtracking points. So, the user can define control rules (*if-then* rules) to help the planner to take the right alternative at those points and avoid backtracking. Control rules can select, reject, or prefer alternatives at decision points. Select and reject rules are used to prune parts of the search space, while prefer rules determine the order of exploring the remaining parts. First, PRODIGY applies all select rules whose if-part matches the current situation, and creates a set of candidate branches of the search space. Next, it applies all reject rules that match the current situation and prunes the rejected candidates. Finally, it applies prefer control rules to determine the order of exploring the remaining candidate branches.

```
Procedure QPRODIGY ($S, G, M, B, C, P$)

$S, G, M, B, C, P$ are the initial state, goals, cost
metric, cost bound, control knowledge and plan.
$g$ is a goal
$O$ is an operator name
$O_b$ is an instantiated operator
$b$ is a binding

If there is no more time Then Return $P$
Else If $G \subseteq S$
    Then If Multiple-solutions=True
        Then $B \leftarrow \sum_{O_j \in P}$ cost($O_j, M$)
            Backtrack (continue search
                with new cost bound $B$)
        Else Return $P$
    Else (*) Choose forward or backward
            mode according to $C$:
        If in forward mode
        Then Select $O_b$ using $C$
            ($O_b$ preconditions are true in $S$)
            If the cost($O_b, M$) +
                $\sum_{O_j \in P}$ cost($O_j, M$)< $B$
            Then $S$ =apply($O_b, S$)
                recompute $G$
                $P = P + O_b$
            Else Backtrack
        Else(*) Select a goal $g \in (G - S)$
                using $C$
            (*) Select an operator $O$ that can
                satisfy $g$ using $C$
            (*) Select a binding $b$ for
                grounding $O$ using $C$
            Recompute $G$
QPRODIGY($S, G, M, B, C, P$)
```

Fig. 2. QPRODIGY four steps decision cycle. Steps marked with $(*)$ represent nondeterministic choices; that is, the planner can backtrack to the corresponding decision point.

The reasons to choose QPRODIGY are manyfold. Among them, we can highlight the possibility of defining and handling different quality metrics [6], reasoning about multiple criteria [1], flexibility to easily define new behaviors, capability to represent and reason about general numeric variables on fluents, definition of functional constraints on variable values in preconditions of operators, explicit definition of control knowledge as well as its automatic acquisition through different machine learning techniques [37], or explicit rationale of each problem solving episode through the search tree.

3.2.2 Deordering Total-Order Plans

QPRODIGY incrementally generates totally ordered plans composed of the operators that have been applied so far in each search tree node. These plans are usually over constrained with respect to precedence constraints. So, from a scheduling point of view, in order to efficiently reason about time and resources it is better if unnecessary precedence constraints are removed from these partially built total-order plans. Therefore, a deordering algorithm is applied that generates as output an equivalent partial order (PO) plan, so plans can represent parallel execution of operators. This is referred to as *deordering* of the solution.

2. In the latest versions of PRODIGY (FLECS), this is also a decision point that determines which ground operator will be applied. We do not consider it in this paper, though.

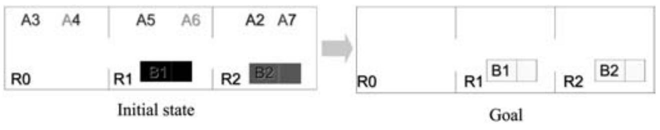


Fig. 3. A problem in the ROBOCARE domain where several agents must make two beds.

This module follows a similar approach to the one presented in [38]. It computes causal links among the operators in the totally ordered plan and identifies threats that might appear among them [30].

- *Causal links generation*. Given a new applied instantiated operator O_b and a current plan, for each precondition of O_b, it finds an instantiated operator O'_b already in the plan that satisfies the preconditions of O_b. Then, it creates a causal link between O'_b, the *producer*, and O_b, the *consumer*. To compute the links, our algorithm starts searching from the first operator applied in the plan (origin). This *heuristic* selects producers as close to the origin as possible and tries to minimize the makespan. We have called it the *Minimal Deordered Link* heuristic.
- *Threat analysis*. If the operator O_b deletes any condition previously established in any link, then the deordering algorithm decides which threat solving strategy to apply (promotion or demotion). If any of the other instantiated operators in the plan deletes any condition established in the new links for the operator, a new causal link is found for the corresponding precondition of the operator, starting from the next applied operator that satisfies the precondition. This is repeated until there are no more threats. In the worst case, the link would be established with the last applied operator.

The complexity of the deordering algorithm is $O(n^2)$, n being the number of applied operators.

Let us clarify how this algorithm works with the problem of Fig. 3. It belongs to the ROBOCARE domain [11]. It is a multiagent (robots) domain for generating user services for human assistance in a closed environment such as a healthcare institution or a domestic environment. Basically, the domain consists of robots that have to perform tasks such as clear beds, serve meals, make beds, or move people from one room to another. But, there are some restrictions to consider. For example, one agent cannot perform two operations at the same time or, in order to make the bed, it must be clean.

The problem of Fig. 3 has two goals: make beds B1 and B2, using any of the six available agents. B2 is already cleared, but B1 is not. One of the possible solutions given by QPRODIGY is shown in Fig. 4. We have added, on the left part, an operator identifier and, on the right part, the duration for each operator. The duration values are internally computed by QPRODIGY using the cost functions definitions provided in the operators in the domain description.

```
3: (make-bed agent7 bed2 room2) --> Dur = 4
4: (clear-bed agent6 bed1 room1) --> Dur = 5
5: (make-bed agent6 bed1 room1) --> Dur = 3
```

Fig. 4. QPRODIGY solution to the problem of Fig. 3.

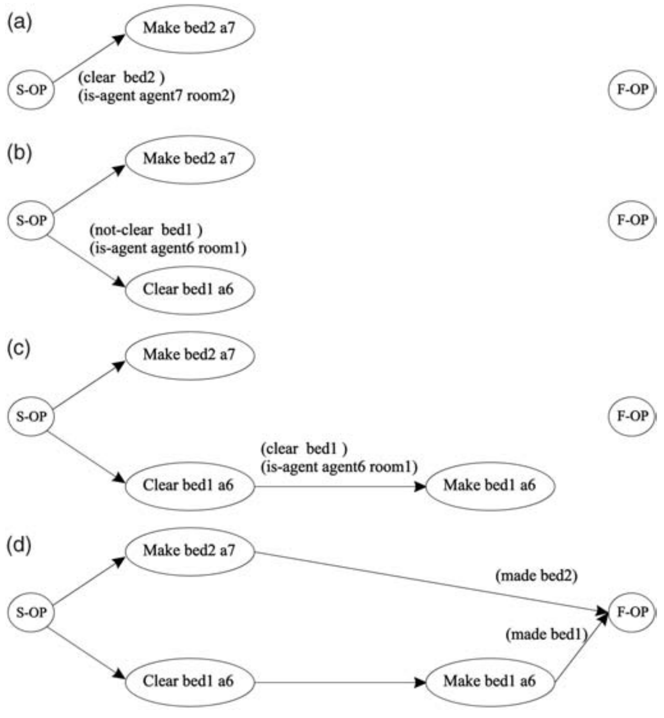


Fig. 5. The deordered solution of Fig. 4.

Fig. 5 shows the deorderings after each operator in the solution is applied during the search. The operators whose identifiers are $S\ OP$ and $F\ OP$ correspond to the initial and goal states, respectively, and are numbered as operator 1 and operator 2. As is shown in the solution to the problem (see Fig. 4), the first applied operator has the identifier number 3, the second operator 4, and so on.

After the planner applies operator 3 or

$$(\texttt{make bed agent7 bed2 room2}),$$

it calls the deordering algorithm that generates a causal link that satisfies the preconditions of this operator and it starts from the $S\ OP$ operator. Therefore, a link is established between $S\ OP$ and 3. Again, no threat is detected.

When applying the second operator,

$$(\texttt{clear bed agent6 bed1 room1})$$

or operator 4, the deordering algorithm generates a link between the origin and the new operator ($S\ OP \rightarrow 4$). Since operator 3 does not delete any condition established by operator 4, there is no need to calculate the next operator that supports the preconditions of operator 4.

After applying the last operator

$$(\texttt{make bed agent6 bed1 room1})$$

or operator 5, the deordering algorithm starts looking from operator $S\ OP$, but it does not support its preconditions. So, it looks for operators 3 (it does not support them either) and 4. Finally, it returns the link ($4 \rightarrow 5$), which also does not generate any threat. As a last step, links to the goal operator (or operator $F\ OP$) are added from operators that achieve the goals ($3 \rightarrow F\ OP$ and $5 \rightarrow F\ OP$).

Because in the previous example there are no threats, let us consider that we have just one robot to perform two activities: make two beds that are clean and placed in two different rooms (see Fig. 6). The solution given by

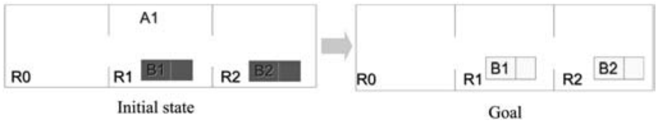


Fig. 6. A threat example in the ROBOCARE domain.

QPRODIGY is shown in Fig. 7. Because there is only one resource, the deordering algorithm will give a TO plan as a solution to the deordering process, as Fig. 8 shows.

As mentioned before, the operators whose identifiers are $S\ OP$ and $F\ OP$ correspond to the initial and goal states, respectively. After the planner applies

$$(\texttt{make bed agent1 bed1 room1})$$

or operator 3, it calls the deordering algorithm that generates a causal link that satisfies the preconditions of operator 3, $(\texttt{is agent agent1 room1})$ and $(\texttt{clear bed1})$. Therefore, a link is established $(S\ OP \rightarrow 3)$ and any threat is detected in this first call. When applying the second operator, $(\texttt{goto room agent1 room1 room2})$, the deordering algorithm generates a link between the origin and the new operator $(S\ OP \rightarrow 4)$ because the preconditions have been met $(\texttt{is agent agent1 room1})$. But, this link threatens the previous one because there is a delete effect $(\texttt{not}\ (\texttt{is agent agent1 room1}))$ in operator 4 that deletes the precondition established in the link $(S\ OP \rightarrow 3)$. The algorithm detects this situation and establishes, instead, the link ($3 \rightarrow 4$) (that is, promotion). The next step is to calculate the links of operator 5. In this case, just operator 4 establishes the two preconditions of operator 4 $(\texttt{is agent agent1 room2})$ and $(\texttt{clear bed2})$, so the algorithm adds a ($4 \rightarrow 5$) link for each condition. As a last step, links to the goal operator (or operator number $F\ OP$) are added from operators that achieve the goals ($3 \rightarrow F\ OP$ and $5 \rightarrow F\ OP$).

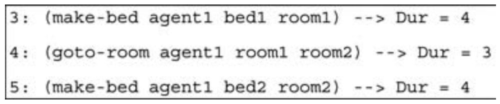
3.3 The Scheduler Reasoner

The scheduling problem is represented as a Constraint Satisfaction Problem (CSP) and the scheduler reasoner, IPSS S, breaks it in two subproblems: a basic GROUND CSP to reason about temporal constraints and a META CSP to reason on resource constraints (see the right part of Fig. 1).

3.3.1 The Temporal Reasoner

This component represents the set of temporal constraints as a Simple Temporal Problem [13]. In particular, significant events such as start/end time of operators are represented as temporal variables tp_i called time points. Each temporal constraint has the form $a \leq tp_i - tp_j \leq b$, where tp_i and tp_j are time points and a and b are constants. Then, if tp_s and tp_e are the start and end time of an operator, Op, its duration constraints can be represented as $d \leq tp_e - tp_s \leq d$, where d is its duration.

Currently, the IPSS architecture supports a set of incremental functionalities for adding and removing both time points and constraints of the form $a \leq tp_i - tp_j \leq b$ in a Simple Temporal Network (STN). A further and fundamental function is the verification of the consistency of the set of temporal constraints represented in an STN. These incremental functions are used inside the algorithm of Fig. 17, where, as part of the backtracking process, a partial solution can increase and reduce its size many times before a solution is found. In particular, the sequence of Steps 1.2 and 1.3 can be seen as the generation of new time points and constraints and Step 1.4 is the consistency checking. The retraction of the last elements is done in case of failure by the backtracking step. The time complexity for adding (including the consistency checking) or removing a constraint is $O(ne)$, where n equals the current number of time points and e is the number of constraints. In fact, we use a Bellman-Ford based implementation for this kind of algorithm in line with [8].



```
3: (make-bed agent1 bed1 room1) --> Dur = 4
4: (goto-room agent1 room1 room2) --> Dur = 3
5: (make-bed agent1 bed2 room2) --> Dur = 4
```

Fig. 7. QPRODIGY solution to the problem of Fig. 6.

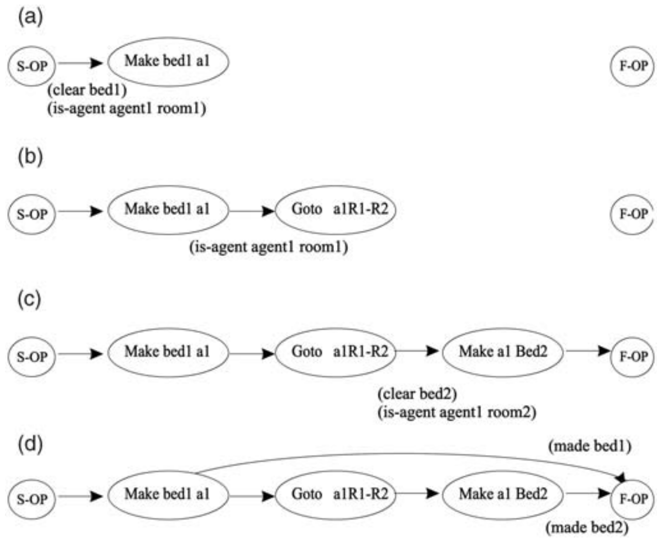


Fig. 8. The deordered solution of Fig. 7.

3.3.2 The Resource Reasoner

The previous algorithm only checks temporal inconsistencies, but it does not check whether plans are using the same resource for performing two actions at the same time. As introduced above, there have mainly been two ways of handling resources in planning and scheduling:

- *Implicit definition of resources*. They are defined as another type of domain (in the best of cases since, in domains such as the blocksworld, the arm is not even defined) and reasoning about them during planning. Examples are trucks and airplanes, rovers, satellites, etc. This implies that the planner has to reason about them and special care has to be taken when defining the domain theory to not allow the same resource to perform two actions at the same time. While planners that generate TO plans can handle this (no parallel actions), planners that generate Partial Order plans do have problems if actions requiring the same resource are applied at the same time.
- *Explicit definition of resources*. This has been the major approach for schedulers and this is the approach we follow in this paper. Actions do not have to specify that an agent cannot do two things at the same time. We just specify that an agent has a limited capacity.

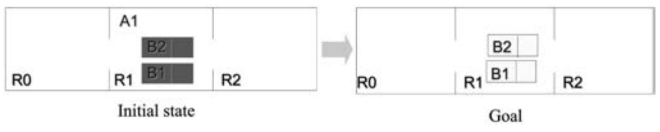


Fig. 9. A threat example in the ROBOCARE domain.

In the META CSP, the resource conflicts are reviewed and addressed. In particular, conflicts are called *peaks* [9], [10]. A peak is any subset of overlapping activities requiring the same resource and such that the sum of the resource requirement is over the resource's capacity. In general, a peak is eliminated by *leveling* it, that is, by posting one or more precedence constraints between pairs of activities contributing to the peak.

The choice of which pair of activities to order is made after the so-called Minimal Critical Set analysis. A Minimal Critical Set is a resource conflict where each proper subset is not a resource conflict. Hence, the philosophy under the META CSP is based on isolating Minimal Critical Sets, so a single precedence relation between any pair of activities in the Minimal Critical Set eliminates the resource conflict. The META CSP is defined by taking the current set of Minimal Critical Sets as decision variables and the set of precedence constraints that can be feasibly posted between some pair of activities in a given Minimal Critical Set as the domain of the corresponding variable. If the META CSP has no assigned variables, there are no remaining resource conflicts and a solution has been found. But, if the set of possible orderings of a conflict is empty, then there is no feasible solution.

To limit the number of Minimal Critical Sets, special attention has to be paid to the the META CSP due to exponential time to complete the computation of the whole set of Minimal Critical Sets. To address this problem, two main ideas have been integrated [9], [10]:

1. After propagation in the GROUND CSP, the earliest start times of all temporal variables are extracted; this is used as a basis for computing the Minimal Critical Set (i.e., the META CSP).
2. A polynomial Minimal Critical Sets sampling method is introduced to extract a subset of Minimal Critical Sets and overcomes the exponential worst case of complete enumeration.

Having generated a Minimal Critical Sets choice set, the next step is to identify one particular Minimal Critical Set for resolution. If at least one solvable Minimal Critical Set remains, the selected Minimal Critical Set is removed by selecting and posting an additional precedence constraint between two of the competing activities in the Minimal Critical Set. In particular, to select and solve a conflict, the heuristic followed is to choose the most constrained variable and set the value (a precedence constraint) that is least constraining. This is measured by computing the temporal flexibility, that is, the number of temporal positions that the time variables in a solution may be assigned with respect to each other. The less flexibility a Minimal Critical Set has, the more critical it is to solve it first. Only those Minimal Critical Sets closest to an unsolvable state are chosen.

Here is an example that more clearly shows the resources conflict resolution. The problem proposed is shown in Fig. 9. It has two goals: (`made bed1`) and (`made bed2`) using the only agent placed in the same room where the two beds are. To simplify the problem, `bed1` and `bed2` are cleared.

```
3: (make-bed agent1 bed1 room1) --> Dur = 4
4: (make-bed agent1 bed2 room1) --> Dur = 4
```

Fig. 10. A QPRODIGY solution to the problem of Fig. 9.

One of the two possible solutions given by QPRODIGY is shown in Fig. 10. In the domain definition, the $\mathtt{make-bed}$ operator does not require the agent to be free in order to make a bed; thus, for any POP algorithm, it could potentially perform both actions at the same time (our deordering algorithm just computes the links: $(S\ OP \longrightarrow 3)$, $(S\ OP \longrightarrow 4)$, $(3 \longrightarrow F\ OP)$, and $(4 \longrightarrow F\ OP)$). So, the solution (make two beds at the same time by the same agent) will not be valid if we do not have in mind the resource conflicts.

The IPSS algorithm checks resource conflicts between pairs of activities and imposes precedence relations between them. Therefore, it is called when more than one operator is on the list of operators that consume the same resource. Considering *agent1* as a resource, we have just one list of operators. The two operators in the solution are on the same list, so there is a conflict. Because both operators can be used, the algorithm chooses the precedence ordering given by the planner, that is, it establishes the $(3 \longrightarrow 4)$ link.

3.4 Properties of the IPSS Algorithm

In this section, we analyze the soundness, completeness, and admissibility of the IPSS algorithm.

3.4.1 Soundness of the IPSS Algorithm

The head plan construction in QPRODIGY always returns a valid plan defined as a sequence of operators that achieves the goals from the initial state. When we use a quality metric, there is nothing that prevents it from removing soundness given that it only reasons about the cost of solutions and not about goal-operators relationships, so QPRODIGY is also sound.

In the case of IPSS, it also constructs a head-plan as its precedent and it applies a deordering to the solution that is always a valid one because all the causal links are proven to be safe. Each time a new set of links is computed, the algorithm checks whether any of the previous links are threatened. If this occurs, we apply any of the methods used in POP algorithms to solve threats. Also, the TN provides a correct time and resource assignment to the TPs that compose it. Thus, IPSS is sound.

3.4.2 Completeness of the IPSS Algorithm

PRODIGY is based on bidirectional planning, that is, a combination of goal-directed backward chaining with simulation of plan execution. Experiments have shown that it is an efficient technique, but it is not complete [17]. In this paper, its authors describe how to make it complete.

In IPSS, we are going to analyze each of the layers that we have added to the search in PRODIGY in order to evaluate their degree of completeness.

- The *Meta*-CSP layer. It calculates the precedence constraints between activities that consume the same resource. The algorithm has in mind all the activities involved in the consumption of the same resource, but, when there is more than one possible order it chooses the link following a logical order, that is, if links $3 \longrightarrow 4$ and $4 \longrightarrow 3$ are possible, it returns the

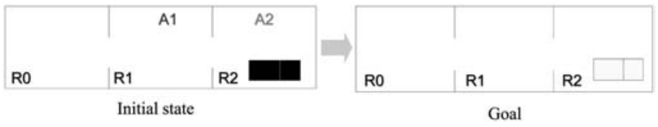


Fig. 11. Another example in the ROBOCARE domain.

3 $\longrightarrow$ 4 link because it always chooses the chronological order imposed by PRODIGY in the operators and does not backtrack over that decision. So, the *Meta*-CSP layer is not complete.

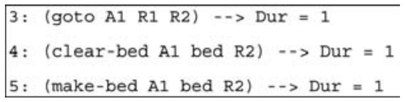
- The *Ground*-CSP layer. It consists of a propagation algorithm and selection of variables and values. In [7], it is shown that the propagation algorithm is correct and complete because a TN is arc-consistent if and only if it is also temporally consistent. This only happens when an assignment of values to the temporal variables exists. Due to the fact that variables and values assignments are embedded within a backtracking framework, the CS algorithm is basically complete.
- The *Deorder-layer*. As described in Section 3.2.2, the deordering algorithm starts computing the link that determines that the preconditions of the last operator added to the head plan are supported by the effects of an operator in the list of applied operators. The links search follows the order of the operators in the head plan, that is, it starts from the origin and continues until the last operator is applied. But, the use of the *Minimal Link Deordered* heuristic can sometimes lead to incompleteness (because we do not backtrack over its decision), so the deordering algorithm is incomplete. If we use the temporal information that the TN returns when we have imposed a temporal horizon, we can see that any deordering different from the one calculated (let us call it $\mathcal{LR}$) will be inconsistent if $\mathcal{LR}$ is. The inverse does not necessarily hold. If the *Ground*-CSP returns an inconsistency for $\mathcal{LR}$, we can compute another link farther from the one calculated, but the TN will again return an inconsistency. In all the experiments that are presented in the next section, we have followed this heuristic and they show that, when QPRODIGY finds a solution, so does IPSS with the heuristic.

3.4.3 Optimality, Quality, and Efficiency of the IPSS Algorithm

The QPRODIGY algorithm is not optimal given that it is not complete. So, IPSS cannot be optimal. IPSS, as QPRODIGY, can improve the quality of the first solution found, given a time bound, when we set the *multiple solution* mode. In that case, the makespan and the cost of the first solution are used as a bound to the next one and so on. With respect to efficiency, the three layers added to the original algorithm usually increase the time to find a solution, given that the time needed for computing deorderings and time/resource inconsistencies is $O(n^2)$.

3.5 Some Benefits of the Approach

Up to now, the only interaction between IPSS P and IPSS S has been by providing new causal links and inconsistency checks from the scheduling component to the planner component. Given the flexibility of the approach, we have devised two initial ways of making use of this integration.



```
3: (goto A1 R1 R2) --> Dur = 1
4: (clear-bed A1 bed R2) --> Dur = 1
5: (make-bed A1 bed R2) --> Dur = 1
```

Fig. 12. One posible QPRODIGY solution to the example of Fig. 11.

The first one uses the temporal network information and the second one the resource reasoning.

3.5.1 Using the Temporal Reasoner

From what has been described up to now, apart from imposing some temporal constraints on the operators, the scheduler does not provide anything new for the planner. Given that, for each causal link, it always adds the most unconstrained temporal range, from a temporal point of view, it does not imply any constraint on the computed causal links. If we provide a maximum makespan (H), then it would be $[0, H]$. In this case, the temporal and resource reasoners can change the behavior of the planner, delivering inconsistent solutions. Let us try to solve a different problem of the same ROBOCARE domain. This is shown in Fig. 11 and we will impose a makespan of two time units.

The first stage starts when the planner explores the search space building a node tree with each node belonging to one of the decision points depicted in Fig. 2. The only goal to work on is (`made bed`) and the only operator that achieves that goal is `make bed`. Let us suppose that the planner chooses `A1` as a binding for this operator. The QPRODIGY solution for this option is shown in Fig. 12.

Because the initial conditions are not yet satisfied, (`not (clear bed)`) and (`in_room A1 R1`), the algorithm chooses another operator that achieves the first goal condition, (`clear bed`). In order to clear the bed, the agent and the bed must be in the same room. The algorithm decides to move `A1` to room R2. At this point, the operator (`goto`) can be applied as the initial conditions are satisfied. The deordering algorithm is executed and the two time points corresponding to the operator are added to the STN: the start *(s)* and end time *(e)* of the operator. At step a in Fig. 13, the *Ground*- and *Meta*-CSP return consistency. The next applied operator is (`clear bed`); again, it is added to the STN with success (step b). But, when the next applicable operator is added, (`make bed`), the GROUND CSP layer returns inconsistency as it goes beyond the imposed makespan (step c). This forces the planner to backtrack and discard nodes 3, 4, and 5. As a next step, the planner binds (`make bed`) with `A2`. The QPRODIGY solution for this option is shown in Fig. 14.

The initial conditions are not yet satisfied (`not (clear bed)`), so the planner chooses the operator (`clear bed`) by `A2`. At this point, the operator can be applied, the deordering algorithm is called again and the two time points corresponding to this operator are added to the STN. The two CSP subsystems return consistency as step a in Fig. 15 shows. Then, operator (`make bed`) is also applied, deordered, and applied to the STN (step b) which achieves the goal by the imposed deadline.

The question is where do we get this upper limit on the makespan from? We have devised two simple potential ways of providing it. First, we can impose a maximum makespan H to each problem in a domain dependent way. As an example, in the ROBOCARE domain, it can be computed as:

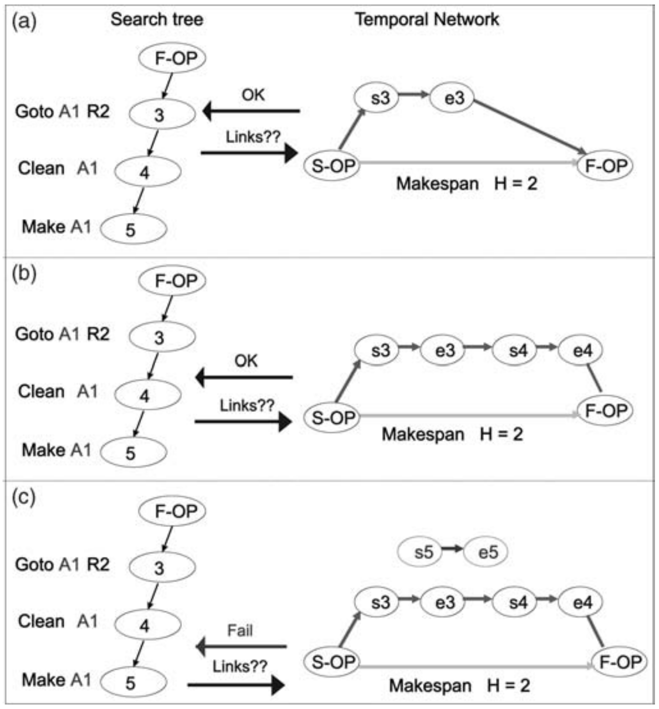


Fig. 13. First step in the solution.

$$H = N_{op} \cdot \left\lceil \frac{N_G}{N_A} \right\rceil, \qquad (1)$$

where, in (1), N_{op} is the number of operators, N_G is the number of goals, and N_A is the number of agents. Second, the planner we are using, QPRODIGY, can compute more than one solution, performing a branch-and-bound search. Therefore, the first solution provides a second limit on the makespan given by the formula and then the makespan can be improved.

3.5.2 Using the Resource Reasoner

The second idea that uses this type of integration consists of providing resource availability from the resource reasoner to the planner. One of our goals is to minimize the makespan at the same time that we compute a valid plan. If we use the feedback from the META CSP, we can aim at efficiently using all the available resources. As a first approach to using such feedback, we have implemented a resource (load) balancing heuristic in the form of *domain dependent* control rules to choose resource bindings that minimize the makespan and at the same time avoid resource conflicts.

A control rule is a production (if-then rule) that tells the system which choices should be made (or avoided) depending on the current state. There are three types of rules: selection, preference, or rejection. One can use them to choose an operator, an instantiation of an operator (binding), a goal, or deciding whether to apply an operator, or continue subgoaling. First, it applies all select rules whose if-part matches the current situation and creates a set of candidate branches of the search space that must be considered. A branch becomes a candidate if at least one select rule points to this branch. If there are no select rules applicable in the current decision point, then, by default, all branches are considered candidates. Next, it applies all reject rules that match the current situation and prunes every candidate pointed to by at least one reject rule. After using select and reject rules to prune branches of the search space, it applies prefer control rules to determine the order of exploring the remaining candidate branches.

```
6: (clear-bed A2 bed R2) --> Dur = 1
7: (make-bed A2 bed R2) --> Dur = 1
```

Fig. 14. The other QPRODIGY solution to the example of Fig. 11.

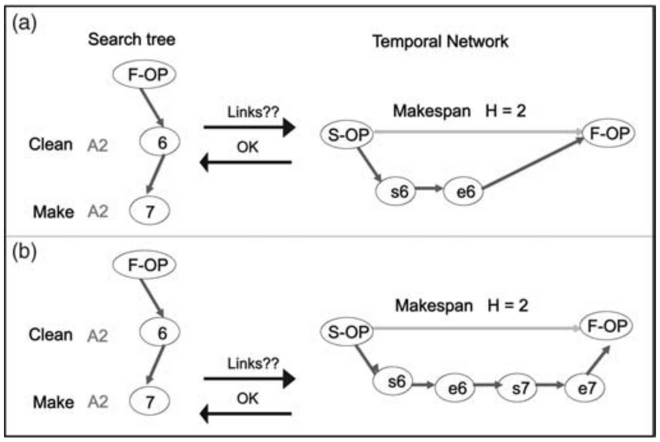


Fig. 15. Second step in the solution.

In IPSS, the control rules consult the META CSP module through a condition on their left-hand side to know which resource is used less at that time point.

Fig. 16 shows an example of a control rule for the `make bed` operator in the ROBOCARE domain. The rule says that IF we have just inserted an operator `make bed` ($<$ `a0` $>$ $<$ `b0` $>$ $<$ `r0` $>$) to achieve an unsatisfied goal literal (`unmade` $<$ `bb` $>$), the `resource less used p` metapredicate implements the consult to the META CSP and the return value is set in the $<$ `aa` $>$ variable THEN it should instantiate the variable $<$ `a0` $>$ with the value $<$ `aa` $>$ given by the META CSP, and the value of the $<$ `b0` $>$ variable with the $<$ `bb` $>$ value.

As the results on Section 4 show, these control rules can greatly improve the makespan.

3.6 The IPSS Algorithm

In the last sections, we have described how each of the different subsystems that compose IPSS works. In this section, we will describe the IPSS algorithm. Fig. 17 shows a description of the complete algorithm. The difference with respect to the QPRODIGY algorithm (see Fig. 2) is highlighted in italics. As explained in previous sections, if the planner decides to apply an action instead of immediately applying it, it first calls the deordering, GROUND CSP and META CSP algorithms. If the new Partial Order plan resulting from the deordering is time and resource consistent, then IPSS applies the operator. If not, it fails, forcing backtracking.

```
(control-rule Bindings_Make-Bed
  (if (and (current-goal (unmade <bb>))
           (current-operator Make-Bed)
           (resource-less-used-p <aa>)))
  (then select bindings ((<a0> . <aa>) (<b0> . <bb>))))
```

Fig. 16. A ROBOCARE control rule to bind resources.

```
Procedure IPSS $(S, G, M, B, C, P)$
---------------------------------------------
$S, G, M, B, C, P$ are the initial state, goals, cost
metric, cost bound, control knowledge and plan
$g$ is a goal
$O$ is an operator name
$O_b$ is an instantiated operator
$b$ is a binding
---------------------------------------------
If there is no more time Then Return $P$
Else If $G \subseteq S$
    Then If Multiple-solutions=True
        Then $B \leftarrow \sum_{O_j \in P}$ cost$(O_j, M)$
            Backtrack (continue search
                with new cost bound $B$)
        Else Return $P$
    Else (*) Choose forward or backward
            mode according to $C$
        If in forward mode
        Then Select $O_b$ using $C$
            ($O_b$ preconditions are true in $S$)
            If the cost$(O_b, M)$ +
                $\sum_{O_j \in P}$ cost$(O_j, M) < B$
            Then update $P$ applying $O_b$,
                deorder $P$ and
                update GROUND-CSP
                with new precedences
                If $P$ is time consistent
                Then compute new
                    precedences
                    using META-CSP
                    If $P$ is resource
                      consistent
                    Then $S$ =apply$(O_b, S)$
                        recompute $G$
                        $P = P + O_b$
                    Else fail
                Else fail
            Else Backtrack
        Else(*) Select a goal $g \in (G - S)$
                using $C$
            (*) Select an operator $O$ that can
                satisfy $g$ using $C$
            (*) Select a binding $b$ for
                grounding $O$ using $C$
            Recompute $G$
IPSS$(S, G, M, B, C, P)$
```

Fig. 17. IPSS decision cycle.

In this algorithm, the temporal-resource inconsistencies are discovered as soon as they occur and this is due to the integration between the planner and the scheduler which interchange information at every decision point. Then, as we show in the experimental section, the possibility of finding better quality plans increases thanks to the capability of the CSP system to perform different kinds of solution analyzes that can be used to guide the planning search.

We observe that, although the QPRODIGY planner can handle time and some type of resources (consumable, producible and resources with single capacity), with the IPSS algorithm we can:

- Reason about activities in parallel. This is an important feature when trying to minimize the makespan.
- Perform different levels of reasoning, that is, we can decide when to reason about resources: during the planning or scheduling processes or both.
- Handle multicapacity renewable resources. Due to the modularity of the approach, we can use any algorithm in the META CSP without affecting the other subsystems or replace the IPSS P part of IPSS (see Fig. 1) by any other planning algorithm.

4 Experiments

In this section, we describe the IPSS versions, the benchmark we have used to test, and the obtained results by comparing them with other planners.

4.1 Domains and IPSS Configurations

As described in previous sections, we have used the ROBOCARE domain [11] for the experiments. The ROBO CARE domain is one of the domains that considers the object *robot* as a resource of binary capacity and treats it separately from causal reasoning. The modeling of the ROBOCARE domain does not consider in the operators if the agents are busy doing something else or not. So, the solution given by PO planners will not be valid. They can generate solutions in which the same agent performs different actions in parallel, which is impossible, but preconditions of operators do not represent this fact. Planners that generate sequences of instantiated operations (TO) solve this problem by not allowing the parallel execution of actions.

The second domain, TI considered is the example of installing a new telephone line as described in [32]. Here, the different activities have to be performed by different workers who should move to the zone where the spares or lines are, in order to accomplish the work. In this case, workers are also considered as a resource of binary capacity.

In order to compare our approach against Partial Order planners, we have also coded these domains forcing each action to require that the agent/worker not be busy performing some other action. After the execution of the action, the agent (robot/worker) is freed by a dummy action with duration zero (the *free-agent* action). In the initial conditions, we have added the predicate *(not-busy <robot>)* for each <robot> in the problem (and also for each worker in the second domain). We have called these domains the ROBOBUSY and TI OCCU domains.

We have used three different configurations of our system: IPSS that uses the integrated planner and scheduler, IPSS-R that incorporates a load balancing heuristic (Section 3.5.2) and IPSS-Q/IPSS-R-Q that searches for more than one solution using branch and bound.

4.2 Results for the ROBOCARE Domain

From all the planners of the Third International Planning Competition[3] that we can compare to, we have chosen METRIC FF [23] for its very good performance. With respect to the type of domains of the competition, IPSS can solve Strips, Time, and Complex problem types. The planners that can solve these types of problems are: LPG [20], MIPS [15], SHOP2 [28], TALplanner [25], TLplan [2], and TP4 [5]. We have discarded TP4 because it is an optimal planner and ours is not, so the comparison would not be fair. We also discarded TALplanner and TLplan because they use

3. http://ipc.icaps conference.org/.

TABLE 1
Number of Problems Solved by Each Planner versus the Number of Agents (Timeout 120 Seconds)

System	A1	A2	A3	A4	A40	A50	Solved	%
LPG	33	33	33	33	33	33	462	100%
FF	33	33	33	33	33	33	462	100%
IPSS-R	27	33	33	33	33	33	456	99%
IPSS	27	30	32	33	33	33	452	97%
MIPS	22	24	27	26	0	0	253	55 %

TABLE 2
Number of Problems Solved by Each Planner versus the Number of Goals (Timeout 120 Seconds)

System	G1	G2	G3	G30	G40	G50	Solved	%
LPG	42	42	42	42	42	42	462	100%
FF	42	42	42	42	42	42	462	100%
IPSS-R	42	42	42	42	39	39	456	99%
IPSS	42	42	42	42	39	35	452	97%
MIPS	30	30	30	13	1	0	253	55 %

domain-specific search control information to control the search and SHOP2 because it needs a handcrafted definition of the domain. But, any of the other mentioned planners could have been used. We have decided to use LPG, METRIC FF, and MIPS.

We randomly generated 11 sets of problems, with three problems in each set, increasing the number of goals (1, 2, 3, 4, 5, 10, 15, 20, 30, 40, and 50). Then, for each set, we increased the number of agents (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, and 50). The total time given to solve each problem (time bound) was 120 seconds.

Given that LPG bases its search on a nondeterministic local search, we ran each problem five times and considered the best solution, the median solution, and the worst solution. This is represented in the tables by LPG Min, LPG Med (it will be considered the baseline to compare the results) and LPG Max, respectively. Tables 1 and 2 show the number of solved problems by each planner considering the number of robots: A1 stands for problems with one agent, A2 with two agents, and so on; and goals: G1 stands for problems with one goal, G2 with two goals, and so on. As can be seen, FF and LPG are fast and efficient planners that solve all problems. IPSS R and IPSS solve almost all the problems and the worst performance is by the MIPS planner.

The IPSS configurations have bad performance when the number of goals is high and when one agent is used (39 problems are solved versus 42 in IPSS R and 35 versus 42 in IPSS that also represents bad performance with two agents). Since none of the added algorithms can provide knowledge that improves the Total Order behavior, they effectively work as the QPRODIGY planner.

Here, we have used the makespan as the quality measure to compare the generated plans. We have only considered the makespan of problems solved by all configurations except for MIPS (because it solved half of the problems). Fig. 18 shows the makespan for all problems averaged over each number of goals. The best results are obtained by IPSS R. It has an improvement of 47 percent over FF and of 17 percent over LPG-med.

Fig. 19 shows the number of operators used by each planner. FF obtains the plans with a smaller number of operators and the LPG family with more operators. This is one example of a domain in which obtaining shorter plans does not imply obtaining better quality. So, we are trading solution length for quality of solutions in terms of makespan.

Fig. 20 shows the time in seconds needed by the systems to solve the problems. In this case, the IPSS configurations have worse performance because IPSS is based on QPRODIGY, which is not a heuristic planner (it does not have domain independent heuristics as FF or LPG have). Also, it is coded in CommonLisp. Thus, in this case, we are trading time to solve for quality of solutions.

But, it obtains better results in the makespan thanks to not considering the predicates *(busy ...) and (not (busy ...))* in the planning process. In fact, due to the use of the *busy* predicates, the augmented number of preconditions needed by the operators tend to create PO solution graphs with a higher number of links among the operators, with a subsequent possibility of obtaining the longest paths in the graph or, equivalently, greater makespan for the output solutions. Therefore, it shows that the type of integration we pursue in this paper provides good results, even when not using the best options for its search strategies.

However, given that both LPG and IPSS can improve quality over time and to be fair in the comparison with respect to time needed to obtain a good solution, we run a

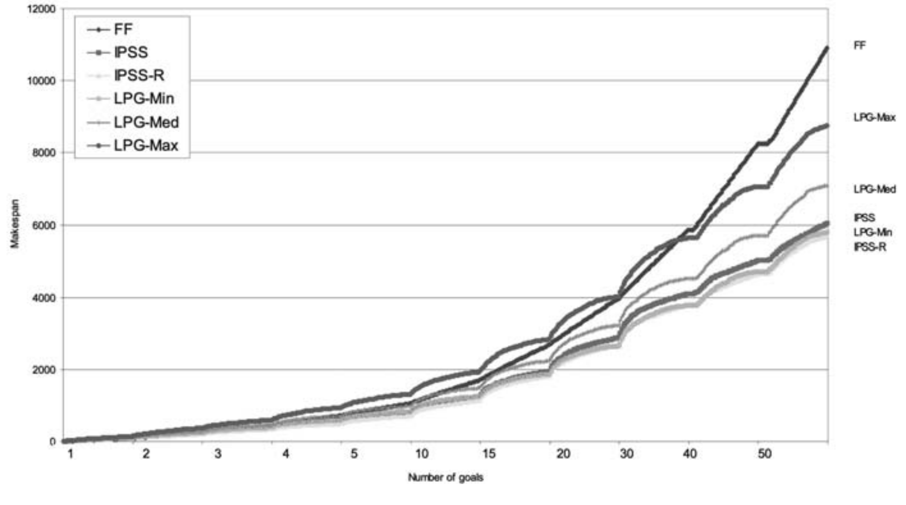


Fig. 18. Makespan for all systems in the robocare domain.

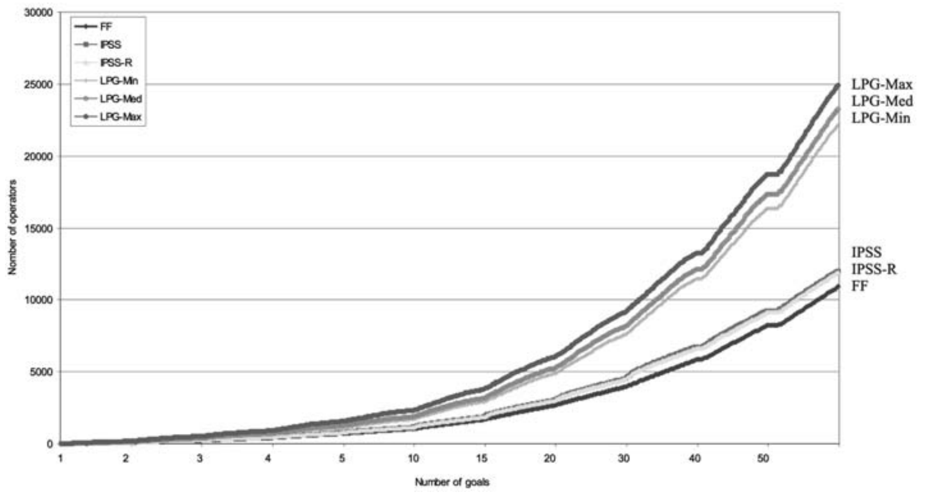


Fig. 19. Number of operators used by each system in the robocare domain.

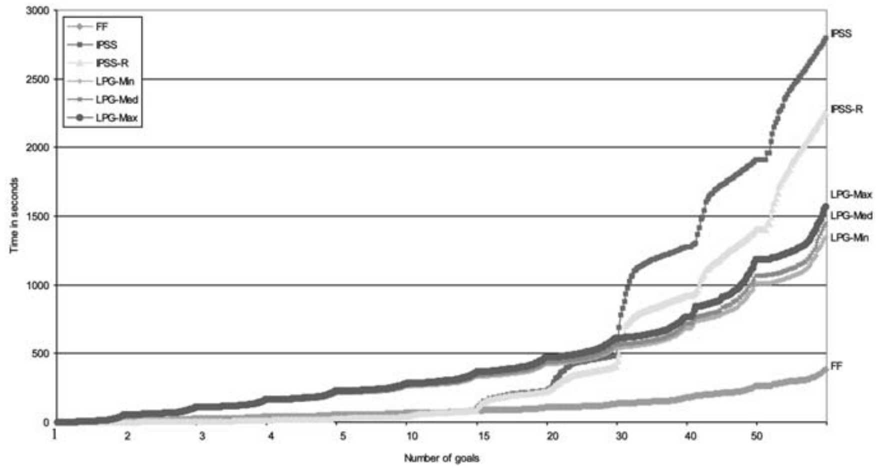


Fig. 20. Total time to solve the problems in the robocare domain.

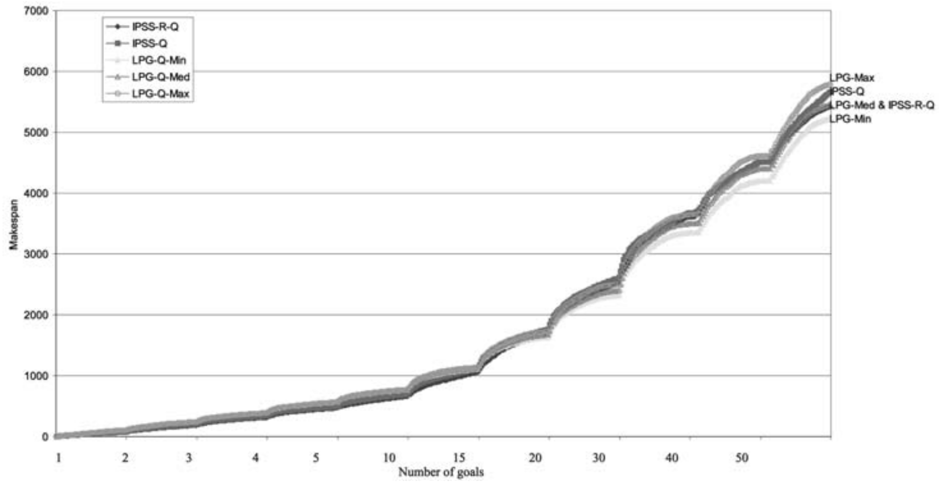


Fig. 21. Makespan for all the systems in the robocare domain running each problem during 120 seconds.

second set of experiments in which we give them the same amount of time to generate better solutions. We have called this the Q mode. The total time given to both is 120 seconds.

Fig. 21 shows that IPSS R Q globally finds solutions as good as LPG Q configurations. Again, it shows that the planning and scheduling approach can obtain equivalent or better results (with respect to some criteria) than the ones obtained by one of the best planners in the International Planning Competition.

The fact that we are able to show good results with respect to the makespan means that the deordering and the temporal reasoning provide an improvement with respect to the sequential choices of the Total Order planner. Additionally,

TABLE 3
Number of Problems Solved by Each Planner in the TI Domain from a Total of 429 Problems Given 120 Seconds Considering the Number of Workers

System	W2	W3	W4	W40	W50	Solved	%
LPG	33	33	33	33	33	429	100%
FF	33	33	33	33	33	429	100%
IPSS-R	33	33	33	33	33	429	100%
IPSS	29	32	33	33	33	424	98%
MIPS	24	27	26	0	0	232	54 %

TABLE 4
Number of Problems Solved by Each Planner in the TI Domain from a Total of 429 Problems Given 120 Seconds Considering the Number of Goals

System	G1	G2	G30	G40	G50	Solved	%
LPG	39	39	39	39	39	429	100%
FF	39	39	39	39	39	429	100%
IPSS-R	39	39	39	39	39	429	100%
IPSS	39	39	39	39	35	424	98%
MIPS	30	29	10	0	0	232	54 %

the planner takes advantage of the feedback from the resource reasoner through the control rule, improving the use of resources, and then reducing the makespan.

4.3 Results for the TI Domain

For the second domain, we have randomly generated 11 sets of problems, with three problems in each set, increasing the number of goals (1, 2, 3, 4, 5, 10, 15, 20, 30, 40, and 50). Then, for each set we have increased the number of workers (2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, and 50). The total time given to solve each problem (time bound) was 120 seconds.

The set of problems for the TI OCCU domain is exactly the same as in the TI domain with the condition that, in all the problems, the availability of the worker must be explicitly set to *not-occupied* (that is, in the initial conditions, we add the predicate *(not-occupied <worker>)* for each <worker> in the problem), so we have added an extra operator to free the worker with duration 0. Tables 3 and 4 show the number of solved problems by each planner considering the number of workers.

Fig. 22 shows the makespan obtained for all the planners in problems solved by all. We have not included MIPS in the graphs due to the low percentage of problems solved, but, if considered, it achieves the worst results in makespan and time, although it generates plans with less number of operators than LPG. The best results are obtained by IPSS R and LPG Min. Then, LPG Med, LPG Max, IPSS, and the worst makespan is, as expected, by FF because it is a TO planner.

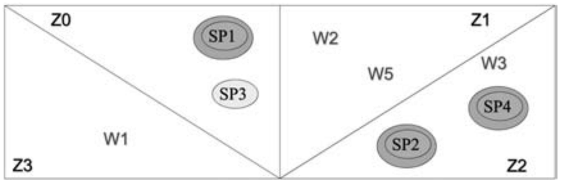


Fig. 23. Initial conditions for the temporal window example.

An important feature of our system that the rest of the IPC state-of-the-art planners do not incorporate is the ability to impose temporal windows on the goals. This is a very critical problem in *workflow* and other real systems because, in most cases, the activities that have to be performed will not be valid if they are not executed in a specific temporal horizon. For example, it is common in telecommunication companies to agree to install a telephone line in two days, so plans whose execution takes more time are not valid.

To test this type of problem, we have not generated any sets, but we show how IPSS works through an example. Because we can set the temporal windows in the operators or in the goals, we have decided to impose temporal constraints on the goals for clarity.

Let us consider a problem with five workers $(W1, W2, \ldots, W5)$ who have to perform five goals: checking five spare line cards $(SP0, \ldots, SP4)$. The spare lines represented with a double circle also need to set the spare pair of wires. Once they are set, the worker can check the line. Fig. 23 shows the disposition of the spare lines, the workers, and the communication among the different zones.

For simplicity, all the activities have a unit duration. We will impose temporal windows on three of the four goals, as Fig. 24 shows. The first goal has to start before or at three time units, the second goal should start at or after one time unit, the third goal can start at any time, and the fourth goal should be contained in the interval (0 3).

The solution that IPSS finds to the problem of Fig. 23 is shown in Fig. 25. It achieves a solution fulfilling all goal constraints.

5 Conclusions and Future Work

In the AI Planning field, two different approaches have been used to solve problems with time and resources. The first, more *traditional*, approach uses predicates for the

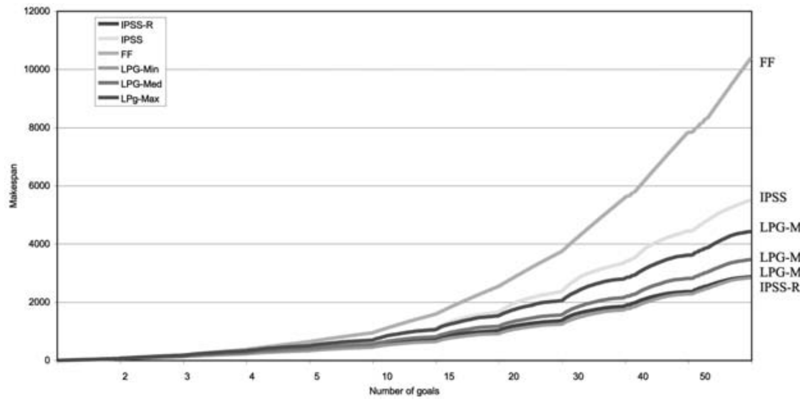
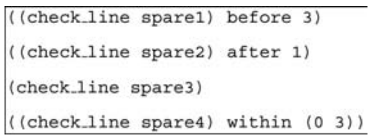
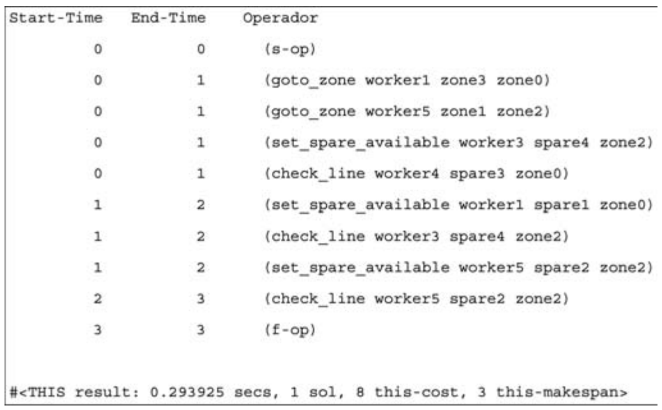


Fig. 22. Makespan for FF, LPG, IPSS, and IPSS R in TI.



```
((check_line spare1) before 3)
((check_line spare2) after 1)
(check_line spare3)
((check_line spare4) within (0 3))
```

Fig. 24. Temporal windows imposed on the goals.



```
Start-Time   End-Time   Operador
       0          0       (s-op)
       0          1       (goto_zone worker1 zone3 zone0)
       0          1       (goto_zone worker5 zone1 zone2)
       0          1       (set_spare_available worker3 spare4 zone2)
       0          1       (check_line worker4 spare3 zone0)
       1          2       (set_spare_available worker1 spare1 zone0)
       1          2       (check_line worker3 spare4 zone2)
       1          2       (set_spare_available worker5 spare2 zone2)
       2          3       (check_line worker5 spare2 zone2)
       3          3       (f-op)

#<THIS result: 0.293925 secs, 1 sol, 8 this-cost, 3 this-makespan>
```

Fig. 25. The solution to the temporal windows example.

action duration and resource availability. The second approach consists of the integration in the planning module of structures and algorithms to manage time and resource availability. All classical planners use the first option, but the search space grows with problem complexity. In this paper, we have described IPSS that is based on a classical planner that uses constraint-based reasoning and extends the reasoning capabilities with a hybrid integration. The main idea is to integrate planning and scheduling having two systems which incrementally build a solution *running* in parallel, a planner and a time-resource reasoner. Information is "continuously" exchanged between these two systems such that temporal and/or resource inconsistencies are discovered as soon as they occur and inform the planner search when to prune choices that lead to either temporal or resource inconsistencies.

We have modified the default search strategy of the planner that searches for optimized Total Order plans to a strategy which searches Partial Order plans under time and resource constraints with the goal of minimizing the makespan of the final solution. The experiments described in Section 4 show improvements with respect to some state-of-the-art planners with a good balance between efficiency and quality. The most important features of IPSS are:

- An integration mechanism such that the planning (scheduling) search process can be incrementally driven by information discovered during the scheduling (planning) search.
- It is implemented in a modular way, so we can easily change any of its components. In particular, when a component is targeted for solving a particular subproblem, it is possible to change the degree of reasoning that can be given to each subproblem.
- It is easy to add control knowledge, metrics, and learning to the system. IPSS, as a QPRODIGY descendant, can use any of the modules implemented for improving its quality or any other learning technique. For example, we have modified the default behavior of IPSS for load balancing of resources.
- We can delegate the planner as just the solution of the temporal aspects of the problem, leaving the resource assignments to the scheduler, or let the scheduler solve both the time and resource assignments.
- The proposed representation and the associated solver are also able to cope with different quality criteria, control knowledge to reduce the search, and learning tools.

The results in this paper pave the way for further development of hybrid integration. In particular, we are interested in understanding how different information that is exchanged between two modules can reflect on the performance of different solvers that use this schema. Additionally, we are interested in using some learning tools to generate control knowledge for the correct decisions made during the problem solving and, again, generating different solvers with additional capabilities.

ACKNOWLEDGMENTS

Maria Dolores Rodriguez-Moreno thanks the ISTC-CNR group for their help during the visit to CNR and Minna Vallentine for proofreading this paper. This work was partially funded by the Universidad de Alcalá project UAH PI2005/084, the CICYT projects TAP1999-0535-C02-02 and TIC2002-04146-C05-05, and a bilateral coordinated project funded by Spanish and Italian Foreign Affairs Departments. Angelo Oddi and Amedo Cesta's work has been partially supported by ASI (Italian Space Agency) under project ARISCOM and is currently supported by MIUR (Italian Ministery for Education, University, and Research) under project ROBOCARE.

REFERENCES

[1] R. Aler and D. Borrajo, *Learning Single Criteria Control Knowledge for Multi Criteria Planning,* pp. 35 40, 2002.
[2] F. Bacchus and F. Kabanza, "Using Temporal Logics to Express Search Control Knowledge for Planning," *Artificial Intelligence,* vol. 16, pp. 123 191, 2000.
[3] C. Bäckström, "Computational Aspects of Reordering Plans," *J. Artificial Intelligence Research,* vol. 9, pp. 99 137, 1998.
[4] A. Blum and M. Furst, "Fast Planning through Planning Graph Analysis," *Artificial Intelligence,* vol. 90, pp. 281 300, 1997.
[5] B. Bonet and H. Geffner, "Planning as Heuristic Search," *Artificial Intelligence,* vol. 129, nos. 1 2, pp. 5 33, 2001.
[6] D. Borrajo, S. Vegas, and M. Veloso, "Quality Based Learning for Planning," *Working Notes of the IJCAI'01 Workshop on Planning with Resources,* 2001.
[7] R. Cervoni, A. Cesta, and A. Oddi, "Managing Dynamic Temporal Constraint Networks," *Proc. Second Int'l Conf. Artificial Intelligence Planning Systems (AIPS '94),* 1994.
[8] A. Cesta and A. Oddi, "Gaining Efficiency and Flexibility in the Simple Temporal Problem," *Proc. Third Int'l Conf. Temporal Representation and Reasoning (TIME '96),* 1996.
[9] A. Cesta, A. Oddi, and S.F. Smith, "Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems," *Proc. Fourth Int'l Conf. Artificial Intelligence Planning Systems (AIPS '98),* 1998.
[10] A. Cesta, A. Oddi, and S.F. Smith, "A Constrained Based Method for Project Scheduling with Time Windows," *J. Heuristics,* vol. 8, pp. 109 136, 2002.
[11] A. Cesta and F. Pecora, "The RoboCare Project: Multi Agent Systems for the Care of the Elderly," *European Research Consortium for Informatics and Math. (ERCIM) News No. 53,* 2003.
[12] A. Cesta, F. Pecora, and R. Rasconi, "Biasing the Structure of Scheduling Problems through Classical Planners," *Proc. Workshop Integrating Planning into Scheduling,* 2004.
[13] R. Dechter, I. Meiri, and J. Pearl, "Temporal Constraint Net works," *Artificial Intelligence,* vol. 49, pp. 61 95, 1991.
[14] M.B. Do and S. Kambhampati, *J. Artificial Intelligence Research,* vol. 20, pp. 155 194, 2003.

[15] S. Edelkamp and M. Helmert, "On the Implementation of MIPS," *Proc. AIPS Workshop Model Theoretic Approaches to Planning,* 2000.

[16] R. Fikes and N. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence,* vol. 2, pp. 189 208, 1971.

[17] E. Fink and J. Blythe, "Prodigy Bidirectional Planning," *J. Experimental & Theoretical Artificial Intelligence,* vol. 17, no. 3, pp. 161 200, Sept. 2005.

[18] A. Garrido, E. Onaindía, and F. Barber, "Time Optimal Planning in Temporal Problems," *Proc. European Conf. Planning (ECP '01),* pp. 397 402, 2001.

[19] A. Gerevini and D. Long, "Plan Constraints and Preferences in PDDL3. The Language of the Fifth International Planning Competition," technical report, Dept. of Electronics for Automa tion, Univ. of Brescia, Italy, 2005.

[20] A. Gerevini, A. Saetti, and I. Serina, "Planning through Stochastic Local Search and Temporal Action Graphs," *J. Artificial Intelligence Research,* vol. 20, pp. 239 290, 2003.

[21] M. Ghallab and H. Laruelle, "Representation and Control in IxTeT, a Temporal Planner," *Proc. Second Int'l Conf. AI Planning Systems (AIPS '94),* 1994.

[22] K. Halsey, D. Long, and M. Fox, "CRIKEY A Temporal Planner Looking at the Integration of Scheduling and Planning," *Proc. Workshop Integrating Planning into Scheduling,* 2004.

[23] J. Hoffmann, "Where Ignoring Delete Lists Works: Local Search Topology in Planning Benchmarks," Technical Report No. 185, Institut für Informatik, 2003.

[24] P. Laborie and M. Ghallab, "Planning with Sharable Resource Constraints," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI '95),* 1995.

[25] M. Magnusson, *Domain Knowledge in TALplanner.* PhD thesis, LiTH IDA Ex 02/104, 2003.

[26] U. Montanari, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Information Sciences,* vol. 7, pp. 95 132, 1974.

[27] N. Muscettola, S.F. Smith, A. Cesta, and D. D'Aloisi, "Coordinat ing Space Telescope Operations in an Integrated Planning and Scheduling Architecture," *IEEE Control Systems,* vol. 12, pp. 28 37, 1992.

[28] D. Nau, H. Muñoz Avila, Y. Cao, A. Lotem, and S. Mitchell, "Total Order Planning with Partially Ordered Subtasks," *Proc. Iint'l Joint Conf. Artificial Intelligence (IJCAI '01),* 2001.

[29] J.S. Penberthy and D.S. Weld, "UCPOP: A Sound, Complete, Partial Order Planner for ADL," *Proc. Int'l Conf. Principles of Knowledge Representation and Reasoning (KR '92),* pp. 103 114, 1992.

[30] M.E. Pollack, D. Joslin, and M. Paolucci, "Flaw Selection Strategies for Partial Order Planning," *J. Artificial Intelligence Research,* vol. 6, pp. 223 262, 1997.

[31] M.D. R. Moreno, "Representing and Planning Tasks with Time and Resources," PhD thesis, Universidad de Alcalá, 2003.

[32] M.D. R. Moreno and P. Kearney, "Integrating AI Planning with Workflow Management System," *Int'l J. Knowledge Based Systems.* vol. 15, pp. 285 291, 2002.

[33] M.D. R. Moreno, A. Oddi, D. Borrajo, A. Cesta, and D. Meziat, "Integrating Hybrid Reasoners for Planning and Scheduling," *Proc. 21st Workshop UK Planning and Scheduling: PLANSIG2002,* 2002.

[34] M.D. R. Moreno, A. Oddi, D. Borrajo, A. Cesta, and D. Meziat, "IPSS: Integrating Hybrid Reasoners for Planning and Schedul ing," *Proc. 16th European Conf. Artificial Intelligence (ECAI '04),* 2004.

[35] S.F. Smith and C. Cheng, "Slack Based Heuristics for Constraint Satisfaction Scheduling," *Proc. 11th Nat'l Conf. AI (AAAI '93),* 1993.

[36] B. Srivastava, R. Kambhampati, and M.B. Do, "Planning the Project Management Way: Efficient Planning by Effective Integra tion of Causal and Resource Reasoning in RealPlan," *Artificial Intelligence,* vol. 131, pp. 73 134, 2001.

[37] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe, "Integrating Planning and Learning: The PRODIGY Architecture," *J. Experimental and Theoretical AI,* vol. 7, pp. 81 120, 1995.

[38] M. Veloso, A. Pérez, and J. Carbonell, "Nonlinear Planning with Parallel Resource Allocation," *Proc. DARPA Workshop Innovative Approaches to Planning, Scheduling, and Control,* pp. 207 212, 1990.

[39] H.L. Younes and R.G. Simmons, "On the Role of Ground Actions in Refinement Planning," *Proc. Sixth Int'l Conf. Artificial Intelligence Planning and Scheduling Systems,* pp. 54 61, 2002.

María Dolores Rodriguez-Moreno received the PhD degree in computer sciences from the Universidad de Alcalá (UAH) in Madrid, Spain, in 2004 with the distinction of the European Doctorate. She has participated in several research projects funded by the European Union and the Spanish government. She has worked at different international centers such as the British Telecom Adastral Park in the United Kingdom, CNR in Rome, and NASA Ames Research Centre in the US, where she is actually developing her postdoctoral research. Her main topics are AI planning and scheduling, execution, and monitoring in real systems as satellite domains, workflow systems, or the Web. Finally, she has more than 50 publications in international conferences, books, and journals in the computer sciences and engineering areas.

Angelo Oddi received the master's degree in electronic engineering from the University of Rome "La Sapienza" in 1993 and the PhD degree in computer science from the same university in 1997. He is a research scientist in the Institute of Cognitive Science and Technology at the Italian National Research Council (ISTC-CNR). He was a visiting scholar at the Intelligent Coordination and Logistics Laboratory at the Robotics Institute at Carnegie Mellon University in 1995-1996. His work focuses on the application of Artificial Intelligence techniques for scheduling, automated planning, and temporal reasoning. Regarding his professional activities, he has published more than 40 papers, both in journals and in proceedings of international conferences in the field. He has vast experience in the design of intelligent systems for real-world applications. In particular, he has been involved in several projects financed by the Italian and European Space Agencies (ASI/ESA) concerning the development of intelligent mission planning support software. He is a member of the IEEE.

Daniel Borrajo received the BS and PhD degrees in computer science in 1990, both from the Universidad Politécnica de Madrid. He has published more than 110 journal and conference papers, mainly in the fields of problem solving methods (planning and game playing) and machine learning.

Amedeo Cesta received the master's degree ("Laurea") in electronic engineering and the PhD degree ("Dottorato di Ricerca") in computer science from the University of Rome "La Sapienza" in 1983 and 1992, respectively. He is a senior research scientist in the Institute of Cognitive Science and Technology at the Italian National Research Council (ISTC-CNR), where he has founded and currently leads the Planning and Scheduling Team (PST, http://pst.istc.cnr. it). His work focuses on the integration of planning and scheduling in software architectures, the use of specialized constraint languages, the synthesis of scheduling heuristics, and on the interactive solution of complex planning and scheduling problems. He is also interested in investigating the gap between theory and practice in planning and scheduling. His research interests also cover multiagent systems and human-computer interaction. He has several publications on all these topics and wide experience in research projects in the area of Artificial Intelligence.