IPSS: A problem solver that integrates P&S

M. D. R-Moreno [1], D. Borrajo [2] A. Oddi [3], A. Cesta [3] and D. Meziat [1]

[1] Departamento de Automática. Universidad de Alcalá.
Ctra. Madrid-Barcelona, Km. 33,6. 28871 Alcalá de Henares (Madrid), Spain.
{mdolores, meziat}@aut.uah.es
[2] Departamento de Informática. Universidad Carlos III de Madrid.
Avda. de la Universidad, 30. 28911 Leganés (Madrid), Spain.
dborrajo@ia.uc3m.es
[3] ISTC-CNR-Italian National Research Council
Viale Marx 15, I-00137 Rome, Italy.
{a.oddi, a.cesta}@istc.cnr.it

Abstract. Recently the fields of AI planning and scheduling have witnessed a big interest on the integration of techniques from both areas in order to solve complex problems. These problems require the reasoning on which actions to be performed as well as their precedence constrains (planning) in combination with the reasoning with respect to the time at which those actions should be executed and the resources they use (scheduling). In this paper we describe IPSS (Integrated Planning and Scheduling System) a domain independent solver that integrates an AI heuristic planner, that synthesizes courses of actions, with constraint satisfaction (CS) techniques that reason about time and resources. IPSS is able to reason about precedence constraints, time (deadline, time windows, etc) and binary resource usage/consumption. Experimental results show that the contextual reasoning of the planner with the CSP solver allows to improve the total makespan on a set of problems characterized by multiple agents. The results show that the time and resource reasoning also allow to regain plan parallelism that is not preserved by the planner.

1 Introduction

AI Planning and Scheduling (P&S) have always been two very related fields that have evolved separately. But nowadays, the AI community shows an increasingly interest to integrate both fields due to real application needs. Planning generates a plan (sequence or parallelization of activities) such that it achieves a set of goals given an initial state and satisfying a set of domain constraints represented in operators schemas. Planning researchers have focused, among other research directions, on the development of model languages that allow a rich representation of the domains and problems [10]. However, this type of representation has problems dealing with variables with infinite values such as information about time and resources. Although the time representation (durative actions) has been one of the main objectives in the language for the IPC'02 competition, resources are not modelled and handled differently than the rest of knowledge. The second problem of planners for dealing with time and/or resources relates to

the fact that usually when solving time-resource dependent tasks, some measure of optimality is pursued, such as makespan, and there are very few planners that can handle optimality criteria (though more and more planners are incorporating now some way of handling this, such as, [12, 13, 22]).

Scheduling systems can perfectly handle temporal reasoning and resource consumption, together with some quality criteria (usually centered around time or resource consumption) but they cannot produce the needed activities and their precedence relations given that they lack an expressive language to represent the activities. From this perspective, integration of P&S systems allow to improve things.

This paper describes a novel framework, IPSS, that incorporates the ideas mentioned above to solve planning problems with time and resource constraints. One of its advantages is that, in addition to integrate planning and scheduling by focusing each type of technique in the corresponding subproblem, it is also able to cope with different quality criteria, control knowledge to reduce the search, and learning tools. In this paper we will describe the integration of planning and scheduling, as well as the use of some kind of control knowledge to improve the resource reasoning.

The paper is organised as follows: Section 2 motivates the main choices within IPSS. Section 3 presents the IPSS architecture and its main algorithm. Then, Section 4 presents an experimental setting and discuss the experimental results. Finally, conclusions and future work are outlined.

2 Motivation

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 makespans) and resources (as agents or fuel). Most of these planners consider discrete resources like robots or trucks as logical predicates. This causes the search to become intractable when the number of resources increases as shown in [22]. In general, we can solve this class of problems with a *planning algorithm* or with the integration of P&S. The latter approach has evolved along two lines:

- Uniform integration: in which P&S are carried out within a uniform representation as in [14].
- Weak integration: in which a clear separation divide the Planning and Scheduling phases as in [22].

Our starting point was the QPRODIGY planner [5] and a Constraint Based Scheduler (e.g., [21]). We have followed an approach that somehow is intermediate between the two mentioned above because we integrate two heterogeneous components: a Total Order planner (TOP) like QPRODIGY and a Time-Resource reasoner based on CSP for the scheduling components. We have built a hybrid software architecture with two parallel processes reasoning and collaborating on the same problem using different representations. So there is some heterogeneity of representation, but reasoners work contextually not following separation of phases.

For the scheduling component we are inspired by early work from Rutten and Hertzberg [20] that integrates a Partial Order Planner (POP) with a time map manager,

Cesta and Stella [7] that study an external module for Time and Resource reasoning to be integrated with a planner, and from precedence constraint posting scheduling algorithms like the one proposed in Smith and Cheng [21] that we ended up using in this work.

Our attention is guided toward domains in which there is a high degree of parallelism, like a multi-agent planning domain, that, in broad terms, is suitable for planners based on GRAPHPLAN [3], that maintain parallelism in the output plan, rather than heuristic planners, that tend to produce more sequential plans.

3 IPSS: An integrated system

IPSS [19, 18] is a hybrid software architecture with two *parallel* processes (planning and scheduling) "collaborating" to solve the same problem relying on different representations. It is mainly subdivided in two systems (see Figure 1): IPSS-P that corresponds to the planning reasoner and IPSS-S that corresponds to the scheduler. On one hand, the planner focuses on the actions selection and generates a Total Ordered (TO) plan.

On the other, the scheduler focuses on the time and resource constraints for generating a consistent schedule of the plan's activities. Since the maximization of plan parallelism – with a consequent minimization of its makespan – is a key issue in this work, IPSS integrates a fundamental extra functionality that transforms a TO plan (that IPSS-P is generating), into a partial ordered (PO) plan that the scheduler needs for generating a better solution. We could have used a PO planner instead. But, as described in [11], their system performance is low. The least commitment approach lets all possibilities be open, what is translated into unnecessary backtracking in the temporal network, and thus a higher solving time.

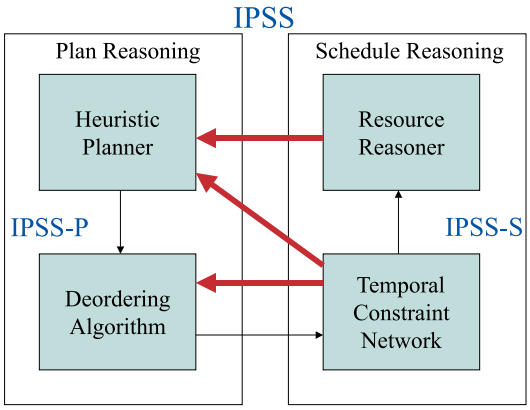

Fig. 1. IPSS architecture.

3.1 The Plan Reasoner

The Plan Reasoner (IPSS-P) is composed of the planner and the Deordering algorithm. The planner performs a bi-directional search: it begins by performing backward search

from the goals, selecting which goals to achieve, which operator to use to achieve the corresponding goal, and which bindings to use for its variables. As soon as it can apply any selected operator, it decides whether applying it or continue subgoaling. If, at any decision point, it applies one operator, the planner consults the scheduler for the time and resource consistency of the partial plan generated so far. If the resource-time reasoner finds the plan inconsistent, then the planner backtracks. If not, the operator gets applied, the current state is modified, and search continues. The planner also allows us to choose actions under a pre-defined metric: *actions that consume less quantity of a given resource*, *actions that take less time in being executed*, *actions that maximise the capacity*, etc. The scheduler also helps using other optimisation criteria during the solving process as minimising the makespan or maximising resource usage.

The plans generated by QPRODIGY are not appropriate if we are looking for time and resource optimisation. Some orderings can be removed from the incomplete TO plan generated by the planner, allowing parallel executions of operators. This is referred as *deordering* of the solution, and IPSS has a module that transforms the incomplete TO plan into an incomplete PO plan. Given that calculate the possible deordering can be generally NP-hard [2], we have followed a greedy incremental heuristic approach to avoid searching in the space of all possible deorderings. The *Minimal Link Deordered* heuristic tries to place the operators as near to the origin and between them as possible, – that is to minimise the makespan. This heuristic makes the *Deordering algorithm* be not complete, but we obtain good results as the experimental section shows.

Once the new links are computed, this algorithm provides the new links to the schedule reasoner. Given that there could be new precedence constraints computed by the scheduler in case of resource conflicts, the scheduler could add new links.

3.2 The Schedule Reasoner

The scheduling problem is represented as a Constraint Satisfaction Problem (CSP) partitioned in two sub-problems.

A basic *Ground*-CSP to reason on temporal constraints and a *Meta*-CSP to reason on resource constraints (see the right part of Figure 1).

The Ground-CSP layer represents the set of temporal constraints as a Simple Temporal Problem [8]. In particular, significant events, as start/end time of operators are represented as temporal variable tp_i called time points. And each temporal constraint has the form a $\leq \text{tp}_i\text{-tp}_j \leq$ b, where tp_i - tp_j are time points and a and b are constants. For example, let tp_s and tp_e be respectively the start and end time of an operator p, its duration constraints can be represented as d $\leq \text{tp}_e$ - $\text{tp}_s \leq$ 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 Temporal Constraint Network ($\mathcal{TCN}$). The time complexity for add (included the consistency checking) or remove a constraints is O(ne), where n is the current number of time points and e is the number of constraints. In fact we use a Bellman-Ford based implementation of these kind of algorithm in line with [6].

For solving the *Meta-CSP* sub-problem we use the algorithm for reasoning on binary capacity resources proposed in [21]. Under this representation model, resource conflicts are computable in polynomial time because conflicts are represented as pairs

of temporally overlapping activities that require the same resource. The algorithm iteratively imposes ordering constraints for solving resource conflicts between activities that require the same resource. When there is more than one possible order it chooses as heuristic the link following the planner logical order. Again, in case of an unsolvable resource conflict (no precedence posting can solve the contention) a failure for resource inconsistency is sent to the planner that backtracks the last decision.

4 Experimental setting and results

In this section we describe some features of temporal and resources domains, then we describe a domain that we have chosen for comparison, the two versions of the IPSS architecture that we have used, and the comparison with current state-of-the-art techniques to solve integrated planning and scheduling problems.

A lot of effort has been recently done for extending planning languages with time. For example, PDDL2.1, supported by almost all state of the art planners, uses a discrete model of time in the version used in the 2002 competition. It also supports the specification of minimising the makespan, but it does not currently support specifying a makespan bound nor the definition of distances between activities or time windows. In IPSS this feature is easily managed thanks to the $\mathcal{TCN}$ representation.

4.1 Temporal and Resources Domains: The ROBOCARE Domain

The ROBOCARE domain [17] describes a multi-agent system which generates user services for human assistance. The system is to be implemented on a distributed and heterogeneous platform, consisting of hardware (robots) and software. It intends to give a user service in a closed environment such as a health-care institution or a domestic environment. It has some features that make it specially fit to separate the resource reasoning from the causal reasoning. All the operations in this domain need to be executed by agents (*robots*), being all agents equal. The tasks that robots can accomplish are: cleaning beds, making beds, serve meals and accompany persons to specific rooms. There are some restrictions to consider: one agent cannot perform two operations at the same time; or for making the bed, they should be cleaned beforehand. We have also defined a fixed duration for each operator.

The ROBOCARE domain is one of these domains that allows to consider the object *robot* as a resource of binary capacity and treat it separately from the causal reasoning. The modelling 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 NON TO planners would not be valid. They can generate solutions in which the same agent performs different actions in the same time step (in parallel), which is impossible, but preconditions of operators do not represent this fact. Planners that generate sequences of instantiated operations (total order) solve this problem by not allowing the parallel execution of actions.

In order to compare our approach against NON TO planners, we have also coded this domain forcing each action to require the agent not to be busy on performing some other action. After the execution of the action, the agent (robot) 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. We have called this domain the ROBOBUSY domain.

4.2 IPSS configurations

We have used two configurations of IPSS for the experiments. IPSS works as it has been described before. IPSS-R is equal to the previous one, but includes the so-called *load-balancing* heuristic: the resource reasoning component (*Meta*-CSP) "tells" the planner what are the less used resources, and the planner selects these when assigning resources to operators. This feedback has been implemented in the form of *domain dependent* control rules allowing to choose different resource bindings that minimise the makespan and at the same time avoid resource conflicts.

Figure 2 shows an example of a control rule for an operator in the ROBOCARE domain. The **resource-less-used-p** meta-predicate receives the information from the *Meta*-CSP layer, binding the agent that should make the bed with the one that the *Meta*-CSP layer has decided is less used. As the results shows, these control rules can greatly improve the makespan.

```
(Control-Rule Bindings_Make-Bed
 (IF (and (current-goal (unmade <bed>))
          (current-operator Make-Bed)
          (resource-less-used-p <agent>)))
 (THEN select bindings ((<a0> . <agent>)
                        (<b0> . <bed>))))
```

Fig. 2. A ROBOCARE control rule to bind resources.

4.3 Results

From all the planners of the competition that we can compare to, we have chosen METRIC-FF [13] for its very good performance. With respect to the type of the domains of the competition, IPSS can solve Strips, Time and Complex problem types. The planners that can solve these types of problems are: LPG [12], MIPS [9], SHOP2 [16], TALplanner [15], TLplan [1] and TP4 [4]. 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 domain specific search control information to control the search. But any of the other mentioned planners could have been used. We have decided to use LPG, MIPS and METRIC-FF.

We have randomly generated 396 problems, increasing the number of agents (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50). For each number of agents we generated 3 problems with an increasing number of goals (1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50). That is, for each number of agents we used 33 problems. The total time given to solve each problem (time bound) was 120 seconds.

Given that LPG bases its search in a non-deterministic local search, we have run five times each problem, and considered the best solution, the third best solution found

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. Table 1 and 2 show the number of solved problems by each planner considering the number of robots [1] and goals [2]. As it can be seen, FF and LPG are fast and efficients planners that solves all problems. IPSS-R (note that it has bad performance when there is one agent because any of the added algorithms cannot help in a TO solution) and IPSS almost solve all the problems and the worst performance is by the MIPS planner.

Table 1. Number of problems solved by each planner in the ROBOCARE domain from a total of 396 problems given 120 seconds considering the n. of agents.

System	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A20	A50	Solved	%
LPG	33	33	33	33	33	33	33	33	33	33	33	33	396	100%
METRIC-FF	33	33	33	33	33	33	33	33	33	33	33	33	396	100%
IPSS-R	27	33	33	33	33	33	33	33	33	33	33	33	390	98%
IPSS	27	28	32	33	33	33	33	33	33	33	33	33	384	96%
MIPS	4	14	25	25	28	29	28	26	27	25	27	24	282	71%

Table 2. Number of problems solved by each planner in the ROBOCARE domain from a total of 396 problems given 120 seconds considering the n. of goals.

System	G1	G2	G3	G4	G5	G10	G15	G20	G30	G40	G50	Solved	%
LPG	36	36	36	36	36	36	36	36	36	36	36	396	100%
METRIC-FF	36	36	36	36	36	36	36	36	36	36	36	396	100%
IPSS-R	36	36	36	36	36	36	36	36	36	33	33	390	98%
IPSS	36	36	36	36	36	36	36	36	36	31	29	384	96%
MIPS	34	32	31	34	30	30	33	23	18	13	4	282	71 %

Our main concern in this paper was the ability to improve the quality of the solutions not the solvability horizon. Here, we used the makespan as the quality measure to compare the generated plans. Table 3 shows the makespan for all problems with the same number of agents for each planner. We only considered the makespan of problems solved by all configurations.

The results show the superiority of the integration of the planning and scheduling approach in combination with the load balancing heuristic. With the IPSS-R configuration, the makespan is lower than the one of IPSS and LPG although the time to solve the problems is relatively higher than LPG or FF but lower than in MIPS as Table 4 shows.

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

[1] A1 stands for problems with one agent, A2 with two agents and so on.

[2] G1 stands for problems with one goal, G2 with two goals and so on.

Table 3. Sum of the makespans for the problems in the ROBOCARE with the same number of agents given 120 seconds.

System	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A20	A50	Makespan
IPSS-R	104	170	298	168	282	246	154	121	119	89	100	64	2005
IPSS	104	173	376	291	366	383	271	174	177	125	140	110	2690
LPG-Min	104	208	398	270	387	353	273	206	202	133	105	61	2700
LPG-Med	104	240	478	338	457	441	342	242	267	183	137	90	3319
LPG-Max	114	285	563	424	596	520	439	308	345	236	175	115	4120
MIPS	104	289	667	508	496	602	462	353	355	250	180	79	4347
METRIC-FF	102	326	838	670	1131	1133	932	708	790	511	538	281	7960

Table 4. Total time in cpu-time to solve all the problems by each planner in the ROBOCARE domain.

Planners	FF	LPG-Min	LPG-Med	LPG-Max	IPSS-R	IPSS	MIPS
Seconds	231,91	167,49	174,31	184,91	356,15	731,78	4323,68

give them the same amount of time to generate better solutions to the problems. We will call this mode the -Q mode. The total time given to both of them is 120 seconds.

Table 5 shows that IPSS-R-Q finds globally as good solutions as LPG-Q configurations. Note that the sum of the makespan values are higher than in Table 3 because we are now considering more solved problems than in the previous table as MIPS solved just the 71% of them (as mentioned before we have only considered problems solved by all the configurations).

Again shows that the the planning and scheduling approach can outperforms one of the best IPC'02 planners.

Table 5. Makespan by each planner in the ROBOCARE domain considering the n. of agents and letting the planners find solution during 120 seconds.

System	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A20	A50	Makespan
LPG-Min-Q	941	511	625	525	429	382	340	310	284	255	158	90	4850
IPSS-R-Q	953	431	569	500	399	394	251	223	201	225	118	91	4985
LPG-Med-Q	949	513	672	541	460	406	351	333	308	275	170	99	5077
LPG-Max-Q	957	529	700	569	482	423	368	350	316	297	180	106	5277
IPSS-Q	953	536	703	571	492	396	369	303	308	292	242	270	5435

The ROBOCARE domain is naturally decomposable in a set of sub-problems, each one is a scheduling problem for each robot (or agent). However, we have to consider that the set of agents are not completely independent; in same cases we have to consider some synchronization. Within our framework we are able to capture this decomposition.

The fact that we are able to show good results with respect to the makespan means that the deordering and the $\mathcal{TCN}$ reasoning give an improvement with respect to the sequential choices of the TOP. Additionally the planner takes advantage of the feedback

from the resource reasoner through the control rule, allowing to maximise the use of resources and then, reducing the makespan.

An important feature of our planner is the possibility of specifying the goal of minimising the makespan given an initial bound. However, we have not tested it here since the rest of state of the art planners do not incorporate it. From the results, we can draw the following conclusions:

- We can increase the performance of a planner/scheduler by considering resources separatedly from the rest of logical predicates and trying to maximise its use, therefore minimising the makespan.
- We have modified the default behaviour of IPSS when load balancing of resources is taken into account. Instead of searching for plans with less number of operators (and then, maximising the use of few resources), we maximise the use of all the resources available (then, minimising the makespan).
- The ability to use information from the resource reasoning allows to solve more problems and minimise the makespan.

5 Conclusions

In this paper we have described the IPSS architecture composed of two systems executing in parallel; a planner and a time-resource reasoner. Information is continuously exchanged between these two representations, and, in particular, it is used to influence the planner search to prune choices that lead to either temporal or resource inconsistencies. The most important features of IPSS are:

- The degree of reasoning that can be given to each level can be varied. Since our planner allows to use some type of temporal-resource usage reasoning together with some optimisation capabilities, we can control/vary the amount of reasoning that the planner/scheduler performs.
- It is easy to add control knowledge, metrics and learning to the system.
- It is implemented in a modular way so we can change any of its components and the system would still work just changing the interface to introduce the knowledge that each part needs.

In the next future, we want to test the next IPSS version in complex domains with multicapacity resources that allows to exploit the power of the algorithms used.

A further activity would be dedicated to taking advantage of several tools that have been developed for QPRODIGY to learn control rules for the correct decisions made while the solving problems.

Acknowlegment

This work was partially funded by the CICYT and MCyT projects TAP1999-0535-C02-02 and TIC2002-04146-C05-05 and a bilateral coordinated project funded by Spanish and Italian Foreign Affairs Departments. Cesta and Oddi work is partially supported by ASI (Italian Space Agency) project ARISCOM.

References

1. BACCHUS, F., AND KABANZA, F. Using Temporal Logics to Express Search Control knowledge for planning. *Artificial Intelligence 16* (2000), 123–191.
2. BÄCKSTRÖM, C. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research 9* (1998), 99–137.
3. BLUM, A., AND FURST, M. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence 90* (1997), 281–300.
4. BONET, B., AND GEFFNER, H. Planning as Heuristic Search. *Artificial Intelligence 129*, 1-2 (2001), 5–33.
5. BORRAJO, D., VEGAS, S., AND VELOSO, M. Quality-Based Learning for Planning. In *Working notes of the IJCAI'01 Workshop on Planning with Resources.* (2001).
6. CESTA, A., AND ODDI, A. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *Procs. of the Third International Conference on Temporal Representation and Reasoning (TIME-96)* (1996), IEEE Computer Society Press.
7. CESTA, A., AND STELLA, C. A Time and Resource Problem for Planning Architectures. In *Procs. of the 4th European Conference on Planning, ECP'97.* (1997), pp. 117–129.
8. DECHTER, R., MEIRI, I., AND PEARL, J. Temporal Constraint Networks. *Artificial Intelligence 49* (1991), 61–95.
9. EDELKAMP, S., AND HELMERT, M. On the Implementation of MIPS. In *AIPS Workshop on Model-Theoretic Approaches to Planning.* (2000).
10. FOX, M., AND LONG, D. *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. University of Durham, Durham (UK), February 2002.
11. GARRIDO, A., AND BARBER, F. Integrating Planning and Scheduling. *Applied Artificial Intelligence* (2001).
12. GEREVINI, A., SAETTI, A., AND SERINA, I. Planning through Stochastic Local Search and Temporal Action Graphs. *Jounal of Artificial Intelligence Research 20* (2003), 239–290.
13. HOFFMANN, J. Where Ignoring Delete Lists Works: Local Search Topology in Planning Benchmarks. Tech. Rep. Technical Report No. 185, Institut fr Informatik, 2003.
14. JONSSON, A., MORRIS, P., MUSCETTOLA, N., RAJAN, K., AND SMITH, B. Planning in Interplanetary Space: Theory and Practice. In *International Procs. of the Conference on Artificial Intelligence Planning Systems* (2000), pp. 177–186.
15. MAGNUSSON, M. *Domain Knowledge in TALplanner.* PhD thesis, LiTH-IDA-Ex-02/104, 2003.
16. NAU, D., MUOZ-AVILA, H., CAO, Y., LOTEM, A., AND MITCHELL, S. Total-Order Planning with Partially Ordered Subtasks. In *Procs. of the IJCAI' 2001.* (2001).
17. PECORA, F., AND CESTA, A. Planning and Scheduling Ingredients for a Multi-Agent System. In *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands* (2002).
18. R-MORENO, M. D. *Representing and Planning tasks with time and resources*. PhD thesis, Universidad de Alcal, 2003.
19. R-MORENO, M. D., ODDI, A., BORRAJO, D., CESTA, A., AND MEZIAT, D. IPSS: Integrating Hybrid Reasoners for Planning and Scheduling. In *Procs. of the 16th European Conference on Artificial Intelligence, ECAI04* (2004).
20. RUTTEN, E., AND HERTZBERG, J. Temporal Planner = Nonlinear Planner + Time Map Manager. *AI Comuunication 6*, 1 (1993), 18–26.
21. SMITH, S., AND CHENG, C. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Procs. of the 11th National Conference on AI (AAAI-93)* (1993).
22. SRIVASTAVA, B., KAMBHAMPATI, R., AND DO, M. B. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *Artificial Intelligence 131* (2001), 73–134.