IPSS: A Hybrid Reasoner for Planning and Scheduling

M^a Dolores R-Moreno[1] and **Angelo Oddi**[2] and **Daniel Borrajo**[3] and **Amedeo Cesta**[2] and **Daniel Meziat**[1]

Abstract. 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 a constraint-based scheduler for reasoning about time and resources. IPSS is able to solve planning problems with time (precedence constraints, deadline, time windows, etc) and binary resource usage/consumption. Experimental results show that the contextual reasoning of the planner with the constraint-based solver allows to improve the total makespan on a set of problems characterized by multiple agents.

1 IPSS: AN INTEGRATED SYSTEM

IPSS 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 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 consistent solution.

IPSS-P. As a planner we are currently using PRODIGY [7] that performs a bi-directional search: it begins by performing backward search from the goals, selecting which goal to achieve, which operator to use to achieve the chosen 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. 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. Because the planner generates total ordered (TO) plans, this type of plans is not appropriate if we are looking for makespan minimisation. Some orderings of the TO plan can be eliminated through the so-called *deordering* of the solution. IPSS has a module that transforms the current TO plan into a PO plan. Given that finding a deordering with minimum makespan is in general an NP-HARD problem [1], we follow 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 among them as possible – that is, it tries to minimise the makespan. This heuristic makes the deordering algorithm be not complete. However, as the experimental section shows, it is an effective heuristic for makespan minimisation.

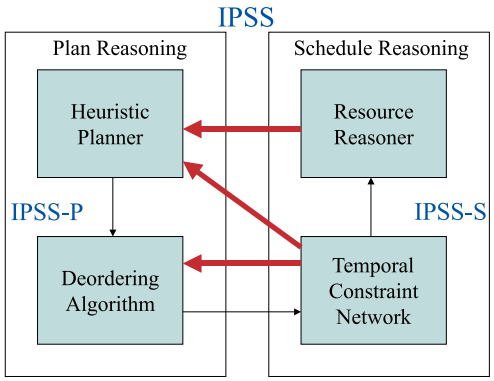

Figure 1. The IPSS architecture.

IPSS-S. 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). Temporal constraints are represented as a Simple Temporal Problem [2], such that significant events, 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 a and b are constants. Hence, the temporal constraints of the generated PO plan (the output from the deordering algorithm) are represented in a Simple Temporal Network that is checked for consistency. In case of inconsistency, feedback could be given first to the deordering algorithm, to ask for a different deordering and next to the planner for rejecting the last choices and performing a higher level backtracking step.

For solving the *Meta-CSP* sub-problem we use the algorithm for reasoning on binary capacity resources proposed in [5]. 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 (e.g., no precedence posting can solve the contention) a failure for resource inconsistency is sent to the planner that backtracks the last decision.

[1] Departamento de Automática. Universidad de Alcalá. Ctra Madrid-Barcelona, Km. 33,6. 28871 Alcalá de Henares (Madrid), Spain
[2] ISTC-CNR-Italian National Research Council. Viale Marx 15, I-00137 Rome, Italy.
[3] Departamento de Informática. Universidad Carlos III de Madrid. Avda. de la Universidad, 30. 28911 Leganés (Madrid), Spain.

2 EXPERIMENTAL EVALUATION

For the experimental settings we have used the ROBOCARE domain [4] that models a multi-agent system which generates user services for human assistance. All the operations in this domain need to be executed by an agent (*robot*), being all agents equal. The tasks that robots can accomplish are: cleaning and making beds, serve meals and accompany persons to specific rooms. In particular, each robot is represented as a resource of unary capacity, hence one robot cannot perform two or more operations at the same time. Also, the robot has the skill of moving among the different rooms. So there is a path planning problem inside the domain.

We have used two IPSS configurations. 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.

In order to compare IPSS against state of the art planners, we analysed characteristics of several planners. We wanted to compare it against domain independent planners, so we discarded domain dependent ones. Also, since we wanted to measure makespan, and this domain is highly parallel, we also discarded TO-based planners, given that they provide low quality solutions in this type of domains. So, finally, we selected LPG [3] as one of the best competitors given its good behaviour in the domain independent part of the planning competition, as well as its capability of generating good quality parallel solutions.

We have randomly generated 363 problems, increasing the number of agents (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20). 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 40 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 (median) and the worst solution. This is represented by LPG-Min, LPG-Med (it will be considered the baseline to compare the results) and LPG-Max respectively.

The number of problems solved by each planner is as follows: LPG the 100%, IPSS the 90% and IPSS-R the 85%. However, our main concern in this paper was the ability to improve the quality of the solutions with a reasonable solvability horizon. Here, we used the makespan as the quality measure to compare the generated plans. Figure 2 shows the cumulative makespan for all problems with the same number of agents for each planner and solved by all of them. These results are particularly interesting if we consider that the increasing of the number of resources (robots in this case) severely curtails the scale-up of existing planners [6]. In particular, the results show the superiority of the integration of the planning and scheduling approach in combination with the load balancing heuristic. The makespan of the IPSS-R configuration is lower than the one of IPSS and LPG, although the time to solve the problems is relatively higher than LPG.

3 CONCLUSIONS

This paper introduces the IPSS architecture composed of two solvers executing in *parallel*: a planner and a time-resource reasoner. The experimental results show that we can increase the performance of a planner/scheduler by considering resources separatedly from the rest of logical predicates and trying to maximise their use, or minimise the makespan. In addition, we have modified the default behaviour of IPSS when load balancing among 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). Therefore, the ability to use information from the resource reasoning allows improvements in system performance. In the next future, we want to test the next IPSS version in complex domains with multicapacity resources that allows to further exploit the capabilities of the integrated algorithms.

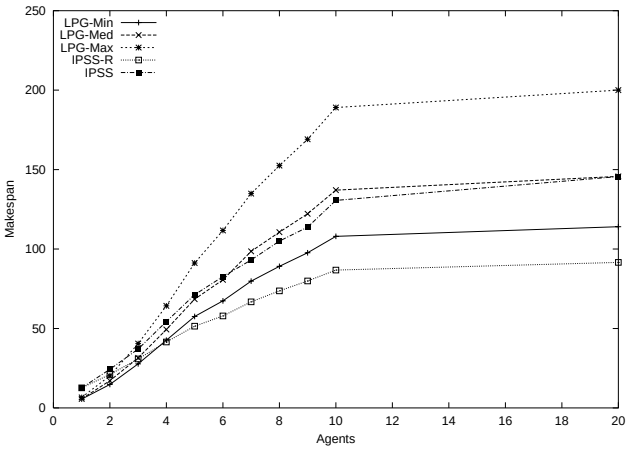

Figure 2. Average makespan increasing the number of agents.

ACKNOWLEDGEMENTS

The first author wants to thank the ISTC-CNR group for their help during her visit to CNR. This work was partially funded by the CICYT and MCyT projects 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] C. Bäckström, 'Computational aspects of reordering plans', *Journal of Artificial Intelligence Research*, **9**, 99–137, (1998).
[2] R. Dechter, I. Meiri, and J. Pearl, 'Temporal Constraint Networks', *Artificial Intelligence*, **49**, 61–95, (1991).
[3] A. Gerevini, A. Saetti, and I. Serina, 'Planning through Stochastic Local Search and Temporal Action Graphs', *Jounal of Artificial Intelligence Research*, **20**, 239–290, (2003).
[4] F. Pecora and A. Cesta, 'Planning and Scheduling Ingredients for a Multi-Agent System', in *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands*, (2002).
[5] S.F. Smith and C. Cheng, 'Slack-Based Heuristics for Constraint Satisfaction Scheduling', in *Procs. of the 11th National Conference on AI (AAAI-93)*, (1993).
[6] B. Srivastava, R. Kambhampati, and M. B. Do, 'Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan', *Artificial Intelligence*, **131**, 73–134, (2001).
[7] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe, 'Integrating Planning and Learning: The PRODIGY Architecture.', *Journal of Experimental and Theoretical AI*, **7**, 81–120, (1995).