PIPSS: Parallel Integrated Planning and Scheduling System

J. Plaza[1], M.D. R-Moreno[1], B. Castaño[2], M. Carbajo[1] and A. Moreno[1]

[1]Departamento de Automática. Universidad de Alcalá.
[2] Departamento de Matemáticas. Universidad de Alcalá.
Ctra. Madrid-Barcelona, Km. 33,6. 28805 Alcalá de Henares (Madrid), España
e-mail: mdolores@aut.uah.es

Abstract

Real problems require the combination of time and resources management. In the last decade, different approaches that integrate planning and scheduling have appeared. In this paper we present PIPSS, a system that combines traditional state space search planning with constraint satisfaction algorithms for scheduling, so that it can also reason about temporal and multicapacity resources problems. PIPSS integrates planning techniques derived from the FF planner and scheduling methods from the OOSCAR scheduler. Its planning and scheduling integration scheme is grounded on a system called IPSS. The most remarkable feature of PIPSS is that it exhibits an open architecture for planning and scheduling integration, based on interfaces, which allows the interoperability of different planning algorithms, scheduling techniques and integration schemes. PIPSS can then be configured to work using any combination of them. Experimental results show that PIPSS outperforms state of the art planner in some of the domains tested.

1 Introduction

Many real planning problems are also scheduling problems because they need to deal with time and resources. Schedulers have the capabilities to deal with time and resources, but it is difficult for them to represent and describe problems. On the other side, planners can give rich problem representations, but find it hard to manage time and resource restrictions. That is why it is interesting to research ways of incorporating scheduling techniques to planning systems, so that they can communicate during the search for solutions.

Some examples of systems that integrate planning and scheduling are EUROPA (Frank and Jónsson 2003) or CRIKEY (Coles et al. 2008). EUROPA uses a constraint-based representation, very different from propositional formalisms such as PDDL (McDermott et al. 1998). CRIKEY, on the other side, is a PDDL planner. It solves durative problems by performing scheduling at each node during search as well as at the end.

Another architecture is shown by IPSS (R-Moreno 2003), which combines a planner called QPRODIGY (Borrajo, Vegas, and Veloso 2001) and some algorithms used in the OOSCAR scheduler (Cesta et al. 2001). Its working model lies on parallelizing each partial plan obtained by the planner and making the scheduler check whether it complies with the time and resource constraints imposed by the problem. If so, the planner keeps on expanding the current search branch; otherwise, the planner prunes that branch and backtracks. It goes on until a solution that meets the goal is obtained which is time and resources consistent.

Among the desired improvements that could be made to IPSS, one of them was to use a state of the art PDDL planner, because QPRODIGY does not understand PDDL. This goal was met by the implementation of the HPP planner (R-Moreno, Camacho, and Moreno 2005).

PIPSS is an integrated planning and scheduling system built from IPSS philosophy. It uses OOSCAR as its scheduler and HPP as its planner instead of QPRODIGY (see Figure 1).

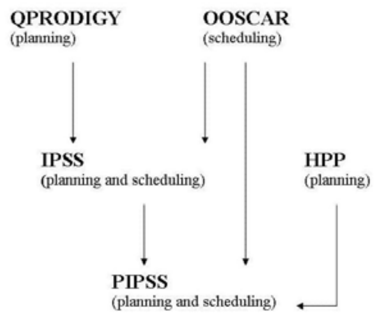

Figure 1: PIPSS Origin

PIPSS stands for "Parallel Integrated Planning and Scheduling System" because it makes possible to launch several parallel search threads until one of them finds a solution. However, the most relevant feature of PIPSS comes from its open planning and scheduling architecture, based on object-oriented interfaces and the definition of standard behaviour patterns for the different most important components of the system: planning searches, scheduling procedures and planning and scheduling integration modules, which makes it possible to treat these as assembling pieces.

The paper is structured as follows. Next section revises the two basic components of PIPSS. Then, PIPSS architecture will be detailed. Afterwards, experimental results will be discussed. Finally, some conclusions and future work will be presented.

2 PIPSS Basic Components

In this section we briefly describe the two main components of the PIPSS system: the HPP planner (R-Moreno, Camacho, and Moreno 2005) and the ISES algorithm (Cesta, Oddi, and Smith 1999).

2.1 HPP: Heuristic Progressive Planner

HPP is a heuristic progressive PDDL planner based on the FF planner (Hoffmann and Nebel 2001). But HPP introduces improvements in the operators instantiation. It includes a new module as shown in Figure 2 called *analysis reachability module* that is able to exclude irrelevant domain-dependent operators for the planning process. The analysis performed by this module avoids the expansion of several parts of the search tree and can obtain better results in the planning process.

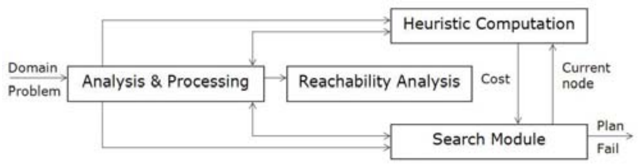

Figure 2: HPP General Structure

In particular, HPP builds three sets of operators called A, B and C *vectors*. A *vector* contains all the possible operators of the instantiation process. B *vector* has fewer operators than A *vector*, which are the result of employing a relaxed graph plan as FF does when it calculates its heuristic (Hoffmann and Nebel 2001). Finally, C *vector* is generated using an additive heuristic h_{add} as the one used in HSP planner (Bonet and Geffner 2001) for computing the heuristic cost.

2.2 ISES Algorithm

OOSCAR (Object-Oriented SCheduling ARchitecture) is a scheduling system that works with time and resources to deal with the RCPSP/max problem (Klein 2000). The RCPSP problem (Resource Constrained Project Scheduling Problem) is a kind of scheduling problem that attempts to look for a way of ordering activities along time, which are restricted by precedence relations (some of them cannot start until some others have finished) and which use renewable resources (resources whose availability is a fixed quantity for every unit of time). The goal is to find initial times for activities so that the makespan is minimized.

The main algorithm used by OOSCAR to find feasible solutions for RCPSP/max problems is ISES (Iterative Sampling Earliest Solutions) (Cesta, Oddi, and Smith 1999). Basically, ISES is a simple optimization algorithm that iterates another algorithm called ESA (Earliest Start Algorithm), which is in charge of returning time and resources consistent solutions. ESA solves temporal restrictions using a temporal network and avoids conflicts due to resources by imposing additional precedence relations between pairs of activities that are responsible for such conflicts. ISES just asks ESA for several solutions to try to find one with a better makespan.

3 PIPSS

PIPSS is a system that integrates planning and scheduling. It incorporates the HPP planner and the ISES scheduling algorithm, using IPSS planning and scheduling integration ideas, but it also goes beyond by providing the resulting system with the capability to treat planning algorithms, scheduling procedures and planning and scheduling integrated search schemes as interchangeable components that can be completely configured at execution time and that follow a standard coding design that uses object-oriented interfaces. This section explains PIPSS interfaces architecture and then it goes over the interfaces themselves.

3.1 PIPSS Architecture

PIPSS defines an interface for each of the most important components of the planning and scheduling integrated search, which are: planning algorithm, scheduling algorithm and planning and scheduling integrated search scheme. This allows the cooperation of different components of these kinds, which can work together regardless of their particularities. Various standard patterns have been defined for each of these components that all algorithms must compulsory comply with. This way, it is possible to connect any planning and scheduling algorithms with any kind of integration scheme without the need to modify or enlarge other parts of the system.

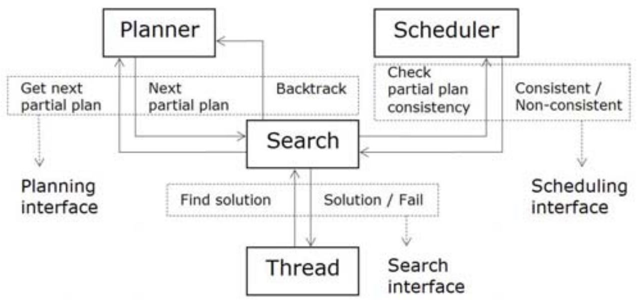

Figure 3: PIPSS Architecture

Figure 3 shows this architecture. A search thread is connected to an instance of a search -here, a search refers to a search scheme that integrates planning and scheduling, i.e., a planning an scheduling integration module-, and they communicate by means of a standard search interface. It does not matter how this particular search instance behaves internally, because the connection point, i.e., the search interface, is always the same. Then, the search instance has to manage the interaction between the planning algorithm and the scheduling technique, so it is connected to an instance of each of these kinds of components through a standard planning interface and a standard scheduling interface respectively. This means that, e.g., if there were 2 different planning and scheduling integrated search schemes, 3 different planning algorithms and 3 different scheduling algorithms, all of them compliant with their respective interfaces, then all these could be combined with each other to look for solutions, having a total number of 18 different possibilities.

Currently, PIPSS has two kinds of planning searches: EHC and greedy best-first search. One type of scheduling (ISES) or the possibility to disable scheduling. And also two types of planning and scheduling integrated search schemes: integrated or sequential. In addition to this, every search thread has to be instantiated with one of the three kinds of operators sets that HPP builds (A, B or C *vectors*). Therefore, there are now 24 different possibilities when PIPSS launches a search thread. PIPSS users can manage these features by editing an xml configuration file in which they can specify the number of execution threads and the combination of algorithms that will be used for each of them. When search threads are already built, they are launched in parallel and every of them performs a search according to its particular characteristics. The search finishes when one thread finds a solution or when all threads exhaust their search spaces.

The two planning and scheduling integrated search schemes that PIPSS offers are called integrated search and sequential search. The integrated search can be seen as a true type of integration between planning and scheduling and that is why it is called so. The second one is called sequential search because it basically asks the planner to find a plan that achieves the goals and then, it asks the scheduler to schedule such plan, so the scheduling process does not affect how the planning search space is examined. The integrated search is the planning and scheduling search scheme that comes from IPSS. This search does not wait for the planner to find a plan that reaches the goals before using the scheduler. For every search tree expansion that the planner performs, the resulting plan is passed to the scheduler. This lets the integrated search tell the planner to make backtracking if the current partial solution is not time and resources consistent, before the planner goes on. This way, the scheduler can influence how to guide the search for a solution. However, PIPSS currently will only be able to take advantage of this feature when the user introduces a maximum makespan value as a parameter, which means that, otherwise, an integrated search will not try to minimize the makespan by itself (this limitation is discussed in the conclusions section).

Of course, other planning and scheduling integrated search schemes could be designed and implemented, following the specifications of the interfaces.

3.2 PIPSS Interfaces

Every interface describes the behaviour of one of the three components of a search thread (planning algorithm, scheduling algorithm and planning and scheduling integrated search scheme). They standardize the services that every component must provide to the outside. This ensures that all components behave uniformly, in spite of the fact that their internal work can be rather different. E.g., when an integrated search instance uses a planning search instance, it will always access the same kind of interface and it will not have to care about whether the planning algorithm is EHC, greedy best-first search, A*, or any other. This will allow the interaction of different components, so that the internal code of every component never has any knowledge about the existence of the internal code of other components.

Next, a brief description of the most important methods of the interfaces is given, which are displayed in Figure 3 too.

Planning interface. PIPSS planning interface consists of the following methods, which every planning search included within PIPSS must correctly implement:

- Get next partial plan: it tells the instantiated planning search to build and return a new partial plan. Successive calls to this method will return different partial plans, which will be the result of visiting the nodes of the search space with the criteria of the particular planning search algorithm. So, this method allows to control the visiting of the tree search on demand. EHC, e.g., will return a partial plan as a result of its last subspace search exploration. A greedy search, on the other side, will return a partial plan as a result of adding the next first node from its open nodes list. This method is useful because an integrated search may need to do some kind of process before the search tree is expanded. For example, it may pass the partial plan to the scheduler to check if the plan is time and resources consistent. If it is not, the integrated search may order the planner to backtrack.

- Backtrack: it closes the current branch of the search space in order to continue the search in an alternative branch. A greedy search, e.g., can make backtracking, because it stores all the open branches of its search space. However, if EHC is asked to perform backtracking, it finishes its search because it does not save any alternative path that it can follow. So, its search space gets exhausted. This method is used when there is a need to force the planner to backtrack due to some sort of criteria external to the planner, such as a particular kind of feedback coming from the scheduler.

Scheduling interface. PIPSS scheduling interface currently consists of several methods, which every scheduling search included within PIPSS must correctly implement. The most important of them is the following:

- Check consistency of a partial plan: it checks the consistency of a partial plan according to the scheduling algorithm criteria. ISES, e.g., will call ESA algorithm so that it tries to find a feasible solution for the partial plan and will return true in case it finds it. However, if scheduling is disabled, it will always return true because no scheduling restrictions will be applied to the partial plan at all.

Search interface. There is a third interface that allows PIPSS to have more than one kind of planning and scheduling integrated search scheme. Search threads directly work with these search schemes or integration modules and, in order to do so, they need a standard interface through which they can call them and always receive the same type of solution from any of them. Threads will simply call a method that tells the search scheme to find and return a solution.

4 Experimental Results

PIPSS has been tested against some state of the art planners. This section is divided into three parts, each of them corresponding to one of the three domains that have been used for experimental purposes.

Two of these domains are standard PDDL durative domains. The first is *Openstacks*, taken from IPC'06 (International Planning Competition 2006). The other domain is called *Hospital*. A third domain has been tested, which exploits PIPSS resources management. It is called *Robocarerc* (*Robocare* with resources), based on the domain described in (Cesta and Pecora 2003). The only planner against which it has been possible to test it is IPSS, because PIPSS deals with resources like IPSS.

Five planners have been tested, though PIPSS and LPG-td (Gerevini, Saetti, and Serina 2004) have been executed using different modalities:

- PIPSS A: PIPSS running one thread with the A *vector*, enforced hill-climbing, ISES and sequential search.
- PIPSS B: PIPSS running one thread with the B *vector*, greedy best-first search, ISES and integrated search.
- PIPSS C: PIPSS running one thread with the C *vector*, greedy best-first search, ISES and sequential search.
- PIPSS ABC: PIPSS running three threads like the previous three configurations.
- IPSS (R-Moreno 2003): Integrated Planning and Scheduling System (this planner has only been used for the Robocarerc domain).
- LPG speed: LPG-td trying to achieve a solution as fast as possible[1].
- LPG quality: LPG-td trying to achieve a solution with the lowest makespan.
- CPT 1.0 (Vidal and Geffner 2006): planning system for optimal temporal planning[2].
- CRIKEY (Coles et al. 2008): temporal planner written in java[3].

There are too many PIPSS variants that can be tested. Currently, there are 12 different possible search threads to solve temporal problems. The ISES algorithm would always have to be used, because it is the only current option to obtain time values for the operators of the solutions. Then, there are 3 different sets of operators that can be selected (A, B or C *vectors*), 2 planning algorithms (EHC or greedy best-first search) and 2 different planning and scheduling integrated search schemes (integrated search and sequential search).

Although PIPSS B launches an integrated search, it will not be able to try to minimize makespan on its own, because it lacks appropriate heuristics or control rules. PIPSS current only option is to try to minimize a maximum makespan provided as a user parameter, but that means that an adequate maximum value should be known in advance for the tests. So, PIPSS B has been executed without such parameter, knowing that it would not obtain better makespans guided by the scheduler. However, it was still interesting to test the overall performance of this combination. Anyway, such drawback is discussed in the conclusions section.

The first three PIPSS modalities have been chosen to try to test as many different options as possible. Each of them has a different set of operators. Then, the first one (PIPSS A) uses EHC and sequential search. EHC is an algorithm that does not allow backtracking, so it cannot benefit from the guidance that could be obtained from the feedback of the scheduler. That is why it only makes sense to first plan with EHC and then schedule with ISES. PIPSS B and PIPSS C both launch a greedy best-first search planning algorithm, which is an algorithm that can backtrack and it

[1] http://prometeo.ing.unibs.it/lpg
[2] http://www.cril.univ-artois.fr/~vidal/#cpt
[3] http://planning.cis.strath.ac.uk/CRIKEY

is possible to mix it both with integrated search (planning guided by the scheduler) or sequential search (scheduling after planning). PIPSS B will run an integrated search and PIPSS C a sequential search.

For each domain, three graphs with accumulated search times, accumulated number of steps and accumulated makespans will show the performance achieved by the planners. It is important to remark that measured times will not take into account other times like, e.g., instantiation time and that problems durations have been truncated because PIPSS currently deals only with integer times. Another issue is that if one or more planners fail to find a solution for a problem, no value will be accumulated for any of the planners at that point. Otherwise, planners that did not solve some problems could show a better performance than their contenders. All executions have had a maximum available time to find a problem of ten minutes and all planners have been launched under the same platform, Windows XP. The computer run with an Intel Core 2 Duo processor (2.33Ghz) and 2Gb of RAM memory.

4.1 Openstacks Domain

In this domain, LPG and CRIKEY solved 90% of the problems, PIPSS A solved 85% and PIPSS ABC solved 80% of them. PIPSS B, PIPSS C and CPT did not find any solution.

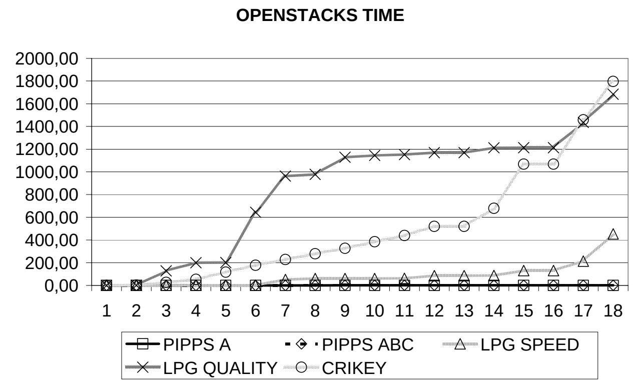

Figure 4: Openstacks Time Graph

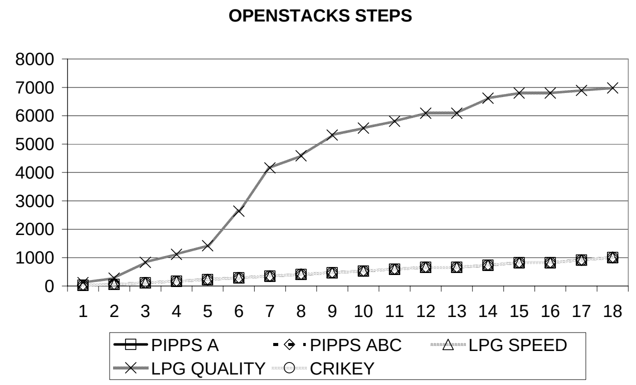

Figure 5: Openstacks Steps Graph

Figure 4 shows the accumulated execution times. PIPSS A and PIPSS ABC are the fastest. LPG speed is the third best, but far from PIPSS. Then, LPG quality and CRIKEY show slow performances in comparison.

Figure 5 shows the accumulated length (number of steps) of the solutions. In this graph, all planners but LPG quality produce the same number of steps for every solution (they overlap in the graph). LPG quality, on the other side, always finds solutions with much more steps.

Figure 6 shows the accumulated makespan of the solutions. LPG quality obtains better makespans than the rest. Then, the other planners do not show great differences, though CRIKEY tends to be a bit better and LPG speed a bit worse. PIPSS A and PIPSS ABC both obtain the same results and they are the third best.

The first remarkable thing is that both PIPSS B and PIPSS C have not been able to find any solution within time, which probably indicates that B and C *vectors* cannot achieve the goals. On the other hand, PIPSS A and PIPSS ABC have the best search times by far. For every problem, the search time is near 0 all the time. This is due to the enforced hill-climbing algorithm. No planner finds a solution to the last two problems and, for the rest, PIPSS only fails to find a solution to problem 13. PIPSS always finds solutions with a minimum number of steps because of EHC. LPG quality plans, on the other hand, are always longer and this probably helps to find better makespans. However, ISES algorithm performs quite well and PIPSS makespans are close to the minimum half of the time, so the relation between time search and quality of the solution that PIPSS gets is still remarkable.

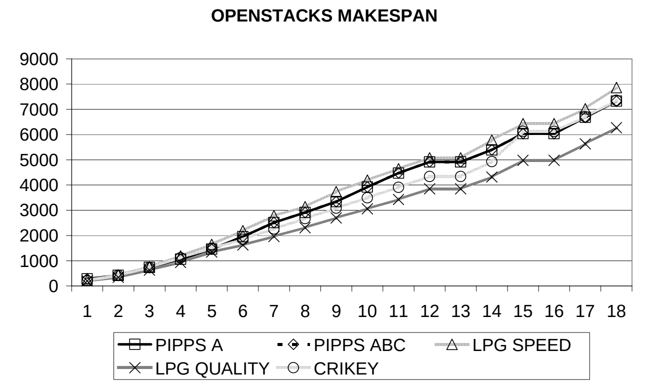

Figure 6: Openstacks Makespan Graph

4.2 Hospital Domain

The *Hospital* domain is a simple path planning domain where there is a map of connected zones and a group of people that want to move from one zone of the map to another. The action of moving between zones is durative and takes one unit of time to be completed. Of course, all people can move independently and in parallel. There are two blocks of problems of ten problems each. In the first block, there is a map formed by a square of zones, with a length of 5 zones per side and they are connected in a such a way that it is possible for any person inside it to move between any two zones needing a maximum of 8 movements, so it can be assured that a solution with this value as its makespan could be found for any problem. The number of people in the first problem is 2 and 2 more people are added to any new problem of the block till there are 26 people in the tenth problem. The second block is similar to the first one, but its sides have a length of 25 zones. In this case, any problem could be completed with a makespan value of less than 49. The number of people in the eleventh problem is 8 and the last problem will have 32 people. All PIPSS modalities and LPG speed found a solution to all the problems, LPG quality solved 75% of them, CPT solved 50% and, finally, CRIKEY solved 45%.

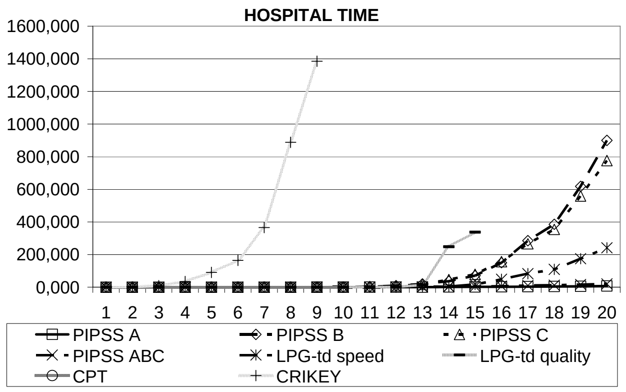

Figure 7: Hospital Time Graph

Figure 7 shows the accumulated execution times. CPT (until problem ten), PIPSS A and PIPSS ABC are the fastest performers, before LPG speed. Then comes PIPSS C, PIPSS B, LPG quality and CRIKEY.

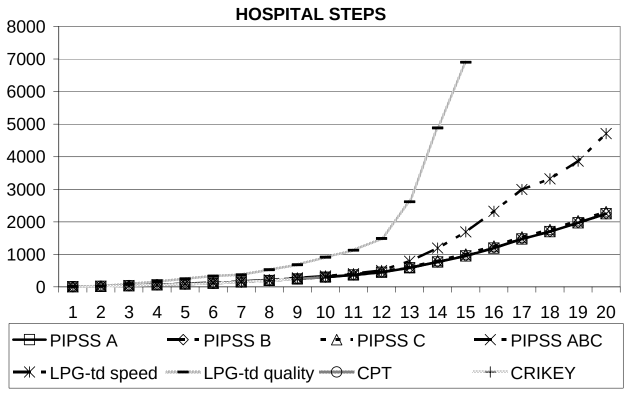

Figure 8: Hospital Steps Graph

Figure 8 shows the accumulated length (number of steps) of the solutions. LPG quality again achieves solutions with more steps than the other planners, which draw nearly the same curve. The only difference here is that LPG speed obtains longer solutions for this domain.

Figure 9 shows the accumulated makespan of the solutions. CPT and CRIKEY do not obtain enough results to be compared to the other planners. In this domain, PIPSS A, PIPSS B and PIPSS ABC are the best. LPG quality is nearly as good as them, but gets stuck in problem fifteen. Then comes PIPSS C, which is near the first four. LPG speed is the last, getting quite big values.

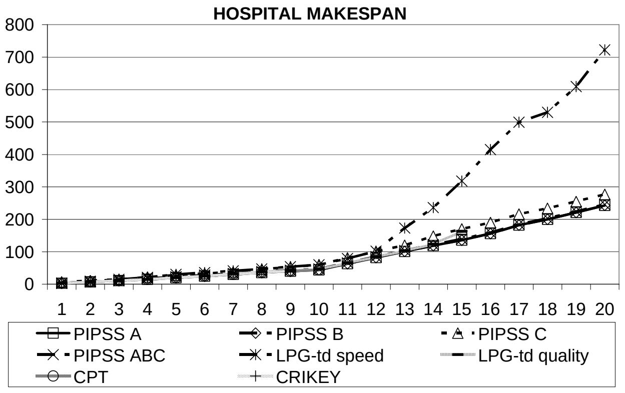

Figure 9: Hospital Makespan Graph

In the *Hospital* domain, PIPSS is the best performer. All PIPSS executions find 100% of the solutions along with LPG speed, but PIPSS makespans are much better by far. LPG quality only solves 75% of the problems and its makespans are never better than those of PIPSS A or PIPSS B. The fastest searches seem to be achieved by PIPSS A and PISS ABC, both launching an EHC thread (however, CPT is a bit faster for the first ten problems, which are the only ones that it solves). The reason why PIPSS obtains the best makespans in this domain is that the EHC implementation that it uses –which comes from HPP– is very good at providing solutions with fewer steps. Since the duration of all the actions of this domain is the same (one unity of time) and since people move independently (so moving a person can be seen as a single subproblem), this means that shorter solutions will result in lower makespans. Of course, in order to do so, it is also important that the scheduler does a good job, which means that ISES performs very well when it is provided with an adequate plan, as in this case.

4.3 Robocarerc Domain

The *Robocarerc* domain is an extension of the *Robocare* domain (Cesta, A., and Pecora, F. 2003) with multicapacity resources. In it, there may be agents (robots) that have to take care of people, by helping them to walk and by clearing and making beds inside rooms. The agents are resources that have a capacity too, which restricts the number of tasks that an agent might perform at the same time. This domain consists of fifty problems that share an initial state, which only differs in the number of agents they have. Problem number one will have one agent; problem number two will have two agents and so on. Furthermore, goals number will be the same as that of the agents, so the harder problem will be the last one with fifty agents and fifty goals. Since this domain deals with multi-capacity resources, PIPSS and IPSS are the only planners capable of being tested in this case. All PIPSS variants and IPSS could solve all the problems, except for PIPSS ABC, which could not solve problem number six.

Figure 10 shows the accumulated execution times. PIPSS A and PIPSS ABC are the best performers. IPSS comes third and finally, PIPSS C and PIPSS B are the slowest.

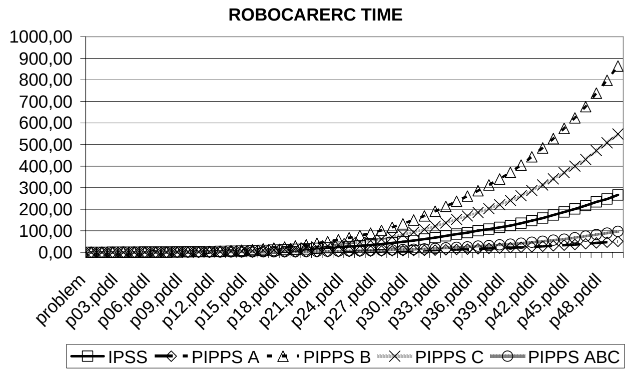

Figure 10: Robocarerc Time Graph

Figure 11 shows the accumulated length (number of steps) of the solutions. IPSS solutions are always longer and all PIPSS variants obtain almost the same results.

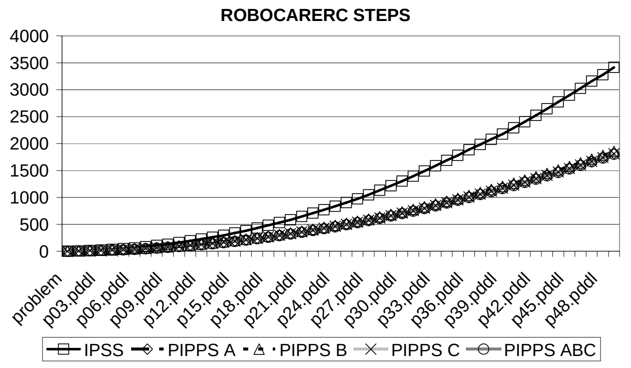

Figure 11: Robocarerc Steps Graph

Figure 12 shows the accumulated makespan of the solutions. Until problem twenty-eight, PIPSS C produces slightly better makespans than the rest but, as problems advance, it is IPSS that has a tendency to be much better than the rest, whose results nearly converge.

Only PIPSS ABC fails to find one solution, so nearly 100% of searches are successful. When it comes to speed, PIPSS A and PIPSS ABC are the fastest of all. This is so for two reasons. Both run an EHC search thread and they only call ISES once they have found a plan. In general, PIPSS A tends to be five times faster than IPSS. PIPSS B and PIPSS C, which run a greedy best-first search thread, are the slowest of all, due to the inefficiency shown by this algorithm. IPSS offers the best makespans as problems get more complex and bigger. Until problem 20, PIPSS finds equal or even better makespans than IPSS, but then IPSS is very regular at minimizing the values of the makespans of the solutions. The difference between PIPSS and IPSS is that IPSS does try to minimize the makespan by means of using a control rule that is based on having into account the resources that are less used (that is why IPSS plans have more steps). PIPSS, on the other hand, does not try to minimize the makespan unless it is given a specific maximum value and, even so, the control rule would probably prove more efficient. This proves that the integrated search, if conveniently used –as in IPSS–, obtains better makespans than a sequential search that schedules after the planning is done.

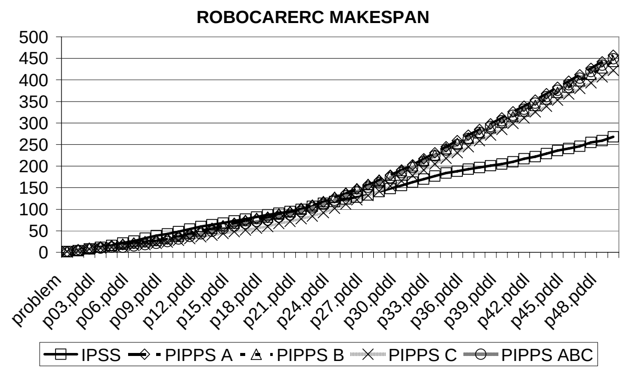

Figure 12: Robocarerc Makespan Graph

5 Conclusions and Future Work

PIPSS was initially thought to be an integrated planning and scheduling system that would use HPP as its core planner and OOSCAR algorithms for its scheduler; they would be integrated following IPSS philosophy. This goal has been achieved and surpassed by providing PIPSS with the capability to dynamically assembly each characteristic of a search at execution time. PIPSS open interfaces architecture allows the system to grow very easily. E.g., if a new scheduling technique is to be added, it can be coded as an independent module, just taking care that it will implement PIPSS scheduling interface. Then, it will automatically be able to work together with any planning search already inside PIPSS. The search may integrate planning and scheduling tightly or it may firstly plan and then schedule (using any of the two present PIPSS integration search schemes).

Four of PIPSS possibilities have been tested against some other planners. The general impression is that it is one of the best at solving most proposed problems (temporal-STRIPS problems) and that, when it runs an enforced hill-climbing search thread, it is the fastest of all. It is not always the one to return the solution with a smaller makespan, but it offers the best speed-makespan relationship. ISES scheduling solutions have a part in this too.

It has also been observed that search threads based on B and C *vectors* are not good alone, because they sometimes lack operators to reach the goals but, when used in parallel searches along with a thread that contains all possible operators, they can help to find a solution faster.

Another kind of search that has not produced good results is that executing greedy best-first search. This planning search has proven to be rather slow. However, the point is that greedy best-first search can backtrack, while EHC cannot. So, EHC can only find a solution and then schedule it. If its makespan is not good enough, it cannot go back and try to find a better one. Greedy best-first search can backtrack and it can find better makespans. But this is only true if the user specifies a maximum makespan before launching the search so, no PIPSS combination is currently able to try to minimize the makespan of solutions unless such maximum value is provided.

In addition to this, it may seem from the results that sequential search (scheduling once a plan is achieved) is better than integrated search. However, IPSS integrated search has demonstrated the opposite by beating PIPSS sequential search at getting good makespans.

There is one IPSS feature that helps it to minimize the makespan of the solutions it returns. When IPSS works with resources, it gives priority to operators that need the resources that have been less used. This control rule allows IPSS to find solutions with smaller makespans. It would be desirable to include this kind of facility into PIPSS and some others for temporal problems that do not use resources. It would also be a good idea to work on a planning search that allowed backtracking and that was faster than greedy best-first search. These two ideas could greatly improve PIPSS overall performance.

PIPSS strength comes from the possibilities that its architecture offers as a framework for integrated planning and scheduling. Its current defficiencies are due to the modules that have been included into PIPSS up to this date. New modules implementing the improvements already exposed will help to make PIPSS better. The good thing about PIPSS is that this can be easily done and that the result can then be compared to its previous state within the same platform.

Acknowledgements

This work has been funded by the Junta de Comunidades de Castilla-La Mancha project PAI07-0054-4397.

References

Bonet, B.; and Geffner, H. 2001. Heuristic Search Planner 2.0. *Artificial Intelligence Magazine* 22: 77-80.

Borrajo, D., Vegas, S., and Veloso, M. 2001. Quality-based Learning for Planning. *In Working notes of the IJCAI'01 Workshop on Planning with Resources*. Seattle, WA.: IJCAI Press.

Cesta, A., Oddi, A., and Smith, S. 1999. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. *In Proceedings of the IJCAI-99*. Stockholm.

Cesta, A., Cortellessa, G., Oddi, A., Policella, N., and Susi, A. 2001. A Constraint-Based Architecture for
Flexible Support to Activity Scheduling. In *Proceedings of 7th Congress of the Italian Association for Artificial Intelligence*. London, UK: Springer-Verlag.

Cesta, A., and Pecora, F. 2003. The RoboCare Project: Multi-Agent Systems for the Care of the Elderly. In *European Research Consortium for Informatics and Mathematics (ERCIM) News* No. 53.

Coles, A. I., Fox, M., Long, D., and Smith, A. J. 2008. Planning with Problems Requiring Temporal Coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 08).*

Frank, J., and Jónsson. 2003. Constraint-Based Attribute and Interval Planning. *Constraints* 8 (4): 339-364.

Gerevini, A., Saetti, A. and Serina, I. 2004 LPG-TD: a Fully Automated Planner for PDDL2.2 Domains (short paper). *In 14th International Conference on Automated Planning and Scheduling (ICAPS-04), booklet of the system demo section*. Whistler, Canada.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. In *Journal of Artificial Intelligence Research.*

Klein, R. 2000. *Scheduling of Resource-Constrained Problems*. Boston: Kluwer Academic Publishers.

McDermott, D., et al. 1998. The PDDL Planning Domain Definition Language. *The AIPS-98 Planning Competition Comitee*.

R-Moreno, M.D. 2003. Representing and Planning tasks with time and resources. Ph.D. diss., Universidad de Alcalá.

R-Moreno, M.D., Camacho, D., and Moreno, A. 2005. HPP: A Heuristic Progressive Planner. In *The 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG05)*. London, UK.

Vidal, V. and Geffner, H. 2006. Branching and pruning: an optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*. 170(3): 298-335.