Assessing the Bias of Classical Planning Strategies on Makespan-Optimizing Scheduling

Federico Pecora [1,2] and **Riccardo Rasconi** [1,3] and **Amedeo Cesta** [1]

Abstract. This paper investigates a loosely coupled approach to planning and scheduling integration, which consists of cascading a planner and a scheduler. While other implementations of this framework have already been reported, our work aims at analyzing the structural properties of the scheduling problem which results from the planning component, focusing on the bias produced by different planning approaches in the light of makespan-optimizing scheduling.

1 Introduction

Research in scheduling has reached a level of maturity which has enabled it to effectively leap into the industrial realm. Today, scheduling components are employed to solve real-world problems. Yet it is interesting to notice that in most of these applications the tasks to be scheduled and the causal constraints among them are basically predetermined. This trend is also present in the research arena, where scheduling problems have traditionally been generated for performance testing. This article is motivated by our belief that many interesting applications will require the automated generation of the causal structure of scheduling problems. This can be straightforwardly recognized as a form of planning. As a consequence, the first step in this direction is to employ a general purpose planner to do the job.

We address this issue by analyzing the properties of loosely-coupled, component-based planning and scheduling (P&S) integration, with a strongly separating assumption: the causal constraints of the scheduling problem are irrevocably decided by the planner, and the scheduler is responsible for producing an optimized instantiation in time of the tasks. This assumption on one hand limits the mechanism by which domain knowledge is shared between the two reasoning components to one instance of information flow, and on the other partitions the competences of the two reasoners. Given this "one-pass" assumption, we evaluate how the quality of the output is affected by the choice of the planner, where the makespan is a measure of schedule quality, as is common practice in scheduling.

2 Loosely-Coupled Planning & Scheduling Integration

In an integrated P&S context, time and resource constraints as well as causal dependencies are contemplated in the initial problem definition. Every operator is inherently associated to a time duration and requires a certain quantity of consumable multi-capacitated resources that ensure its executability.

The general schema we will use in this investigation is as follows. A causal model of the environment is given as input to a planner, i.e. the domain representation and the problem definition, both expressed in a STRIPS-like formalism. This model does not contemplate time and resource related constraints, which are accommodated after the planning procedure has taken place. In order to produce a problem specification which can be reasoned upon by the scheduling procedure, the Partial Order Plan (POP) produced by the planner is integrated with time and resource related information by means of a plan adaptation procedure. This procedure produces a minimal-constrained de-ordering [1] of the POP and integrates it with time and resource related information to produce what we shall call a completePOP. In more formal terms:

Definition 1. *A* completePOP P *is a quintuple* $\langle \mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{D} \rangle$ *where*

- $\mathcal{T} = \{T_1, \ldots, T_n\} \cup \{T_0, T_{n+1}\}$ *is the set of* tasks *which correspond to the* n *activities in the POP produced by the planner;* T_0 *and* T_{n+1} *are called* source *and* sink *activities;*
- $\mathcal{P}$ *is a set of* precedence constraints *between the tasks, where* $T_i \prec T_j$ *means that task* T_i *must be completed before the execution of task* T_j *can begin;*
- $\mathcal{R} = \{R_1, \ldots, R_m\}$ *is a set of* resources*; each task* $T_i \in \mathcal{T}$ uses *a set* $R \subseteq \mathcal{R}$ *of resources, which is expressed with the notation* $[T_i] = R$*.*
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ *are the* capacities *of the resource;*
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ *is the set of task* durations*.*

According to the definition above, a completePOP coincides with a Resource Constrained Project Scheduling Problem (RCPSP) [3] and can be visualized in the form of a precedence graph, as shown in the example in figure 1, where edges are simple precedence constraints. This particular problem contains 11 tasks (plus source and sink), for which a partial order is specified by means of precedence constraints. The problem contains two binary resources, R_1 and R_2.

3 Planning Strategies

By observing the precedence graph produced by the plan adaptation procedure, one extremely important characteristic for our purposes can be recognized as the *degree of concurrency* (or *parallelism*). The relationship between this characteristic of the precedence graph and the quality (in terms of makespan) of the final schedule is quite straightforward. In fact, the makespan of the schedule is related to the *critical path* through the causal network of tasks, where by critical path we intend the sequence of tasks which determines the shortest makespan of the schedule, i.e. the path that runs from the source

[1] Institute for Cognitive Science and Technology – Italian National Research Council – Viale Marx 15, I-00137 Rome, Italy – email: {fpecora, rasconi, a.cesta}@istc.cnr.it

[2] PhD student in Computer Science Engineering at the University of Rome "La Sapienza".

[3] PhD student in Computer Science Engineering at the University of Genoa.

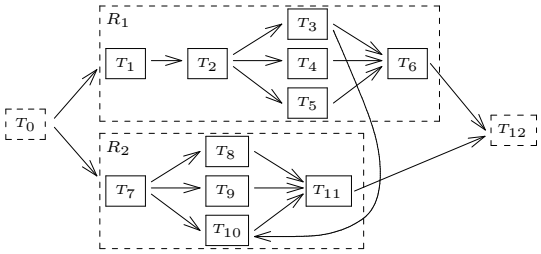

Figure 1. A completePOP with 11 tasks in which $\mathcal{R} = \{R_1, R_2\}$, $[T_1] = [T_2] = [T_3] = [T_4] = [T_5] = [T_6] = \{R_1\}$, and $[T_7] = [T_8] = [T_9] = [T_{10}] = [T_{11}] = \{R_2\}$. The source and sink tasks are T_0 and T_{12}.

to the sink node such that if any activity on the path is delayed by an amount t, then the makespan of the entire schedule increases by t. Obviously, if the durations of the tasks were all the same, then the critical path would coincide with the longest path through the graph. This is in general not true since the path which determines the makespan of the entire schedule may be shorter than the longest path through the graph. Nevertheless, we can assume that the critical path is usually one of the longest paths through the graph [4]. Given our main goal, which is to obtain a schedule with a short makespan, the previous considerations naturally lead us to prefer, among the existing planners, those which are more likely to produce plans which minimize the longest path through the graph. The aim of our analysis is to reveal how different planning paradigms, namely Heuristic Search (HS) planning and Planning Graph (PG) based planning, behave in this loosely coupled framework.

PG-based planners work by alternating one step of graph expansion to a search on the planning graph [5] for a valid plan, the search occurring at every level of expansion (starting when the goals appear non-mutex for the first time). This, together with the disjunctive nature of the search space [10], makes PG-based planners optimal with respect to the number of execution steps. This guarantees that these planners find the shortest plan among those in which independent actions may take place at the same logical step [2].

The optimality of PG-based planners is precisely what makes them best suited with respect to the criteria described above for loosely-coupled P&S integrations. In fact, the length of a POP in terms of execution steps is related with the critical path of the corresponding completePOP as follows:

Theorem 1. *The number of execution steps in a plan produced by a PG-based planner coincides with the length of the longest path through the relative completePOP.*

Proof. The proof of this theorem equates to proving that if the number of steps in the POP is s, then the length of the longest path is both *at least* and *at most* s.

To prove that the length l of the longest path cannot be greater than s, it suffices to observe that if $l > s$, then there would be at least one sequence of $s + m$ actions in the plan which belong to different logical steps, which in turn would mean that no valid plan of length s exists. This contrasts with the validity of the POP generated by the planner, since the shortest possible solution would be longer than s.

[4] This assumption is realistic when there are no tasks in the problem specification whose duration is dominating, i.e. if the durations of all tasks are comparable.

[5] This category includes all those planners which maintain a planning graph representation of the search space, and does not refer to the particular solution extraction algorithm they employ (exploring the planning graph, casting it as a SAT problem and so on). The details of the search do not affect the generality of the observations we make here.

The fact that the longest path cannot be shorter that the length of the POP can be deduced by observing that if indeed $l < s$, then it would be possible to eliminate at least one extra precedence constraint in the completePOP, which contrasts with the fact that the completePOP is a minimal-constrained de-ordering of the POP. □

In one statement, what we have said shows that PG-based planners, by minimizing the critical path, in fact maximize concurrency *with respect to the causal model of the problem*. Indeed, whereas any type of planner may be used to produce the initial POP, the quality of the final solution is negatively affected by the non-optimality (with respect to the number of execution steps) of the generic planner.

3.1 A General Example

In order to confirm the previous statement experimentally, we have created a benchmark PDDL domain which produces completePOPs with high levels of concurrency by encapsulating the notion of *executing agent* and *thread of execution*. Concurrency is obtained by inducing the presence of multiple threads (i.e. multiple agents) in the POP. Having augmented the PDDL language with some simple extensions to allow the specification of durations and resource usage for the operators (these directives are ignored by the planner and integrated into the POP to form the completePOP), the general structure of a problem which leads to threaded completePOPs is as shown in figure 2, where `:capacity 0` denotes an object which is not a resource. Given this structure, the number of threads in the completePOP is determined by the number of objects with non-zero capacity, i.e. $|\{\texttt{A1}, \ldots, \texttt{An}\}|$, and the resource usage of each task is given by the `:uses` clause in the abstract operator specification.

```
(define (domain ... ) ...
  (:action op
     :parameters (?a - agent ... )
     :precondition ( ... )
     :effect ( ... )
     :uses (?a USAGE)
     :duration DUR) ... )
```

(a)

```
(define (problem ... ) (:domain ... )
  (:objects
    A1, ..., An - agent :capacity CAP
    B1, ..., Bm - type :capacity 0 ... )
  (:init ... )
  (:goal ... ))
```

(b)

Figure 2. General structure of augmented PDDL domain (a) and problem specifications (b) which lead to completePOPs with multiple threads.

We have chosen to impose some simplifying restrictions on the completePOPs: first, we deal only with binary resources ($C_i = 1,\ \forall C_i \in \mathcal{C}$); second, all tasks use exactly one resource ($|[T_i]| = 1,\ \forall T_i \in \mathcal{T}$). As a consequence, the value of `CAP` in figure 2(b) must be `1` or `0` for all objects, and each operator must have at least and not more than one `:uses` clause. In terms of the resulting completePOP, this leads to the following formal definition of thread:

Definition 2. *A* thread Φ_k *of a completePOP* $P = \langle \mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{D} \rangle$ *is a set of tasks* $\{T_1, \ldots, T_l\} \subseteq \mathcal{T}$ *such that* $[T_i] = \{R_k\} \in \mathcal{R},\ \forall T_i \in \Phi_k$.

The threaded structure of the completePOP is achieved by instantiating the `Ai` objects as agents, and specifying that all operators are parametric with respect to the executing agent. Given the resource capacity constraint in the completePOP, each agent can carry out only one action at a time (which is enforced during scheduling). More formally [6]:

Definition 3. *A set of tasks $S \subseteq \Phi_k$ is a* contention peak *iff $\sum_{T_i \in S} |[T_i]| > C_k \ \wedge \ \nexists \, T_i, T_j \in S$ s.t. $\{T_i \prec T_j\} \in \overline{\mathcal{P}}$, where $\overline{\mathcal{P}}$ is the transitive closure of $\mathcal{P}$.*

Definition 4. *A* minimal critical set *is a contention peak such that any proper subset of its activities has a combined resource requirement $< C_k$.*

In more simple terms, a contention peak is a set of activities which simultaneously require a resource (in order for the tasks to be potentially simultaneous there must not be any precedence constraints among them in the transitive closure of the precedence graph). A PG-based planner not only allows the execution of one agent's tasks in a single step, but it maximizes this feature (by maximizing the size of the contention peaks), as shown in theorem 1. This behavior explains why PG-based planners are particularly suited for loosely coupled P&S integration: contention peaks are precisely what any profile-based makespan-optimizing scheduler works on in order to obtain shorter makespans — in this respect, PG-based planners never impose over-committing constraints, the insertion of which is delegated to the resource reasoning module.

3.2 Experimental Results

Figure 3 shows the makespan-optimizing performance obtained with two different component based implementations of the loosely coupled framework: one employs BLACKBOX [11], a PG-based planner which combines the efficiency of planning graphs with the power of SAT solution extraction algorithms, while the other uses the FF planning system [9], a heuristic search planner which was Top Performer in the Strips Track of the 3^{rd} International Planning Competition. Both instantiations of the framework employ O-OSCAR [4],

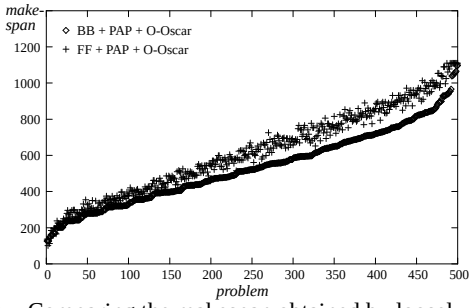

Figure 3. Comparing the makespan obtained by loosely coupling BLACKBOX and FF with O-OSCAR on 500 randomly generated multi-agent problems (PAP = plan adaptation procedure).

a profile-based [5] CSP scheduler which implements an iterative improvement algorithm [6]. The benchmark used for these tests consists of 500 randomly generated problems in an "artificial" multi-agent domain which adheres to the general PDDL structure shown above.

While both planners obtain similar results, as expected the BLACKBOX-based integration yields shorter makespans on the overwhelming majority of instances. Since the O-OSCAR scheduler is not systematic, the computed makespans are not necessarily the shortest possible.

It is even more interesting to notice, though, that in *none* of the instances do FF-derived completePOPs yield shorter makespans than the BLACKBOX-derived problem instances. This is somewhat surprising given the non-complete nature of the scheduling algorithm. While it is true that the *optimal* makespan of a completePOP obtained through BLACKBOX is shorter or equal to that of the equivalent completePOP computed by FF, it is not so obvious that the strongly non-systematic sampling strategy employed by O-OSCAR *never* "stumbles upon" a more optimized solution when solving an FF-derived scheduling problem. Indeed, this is a planner which, according to the HS planning paradigm, employs powerful heuristics to drastically prune the search space. The resulting net effect is a different causal structure of the scheduling problem instance. As we will see, the difference between the two types of completePOPs plays a major role in the distribution of the solutions' makespans, a characteristic which is strongly related to how easy it is to perform makespan optimization on a completePOP. In order to explain this concept fully, we must first focus on the structure of completePOPs and its role in makespan optimization, which is the topic of the following section. The issues we have put forth here will be further analyzed at the end of the next section.

4 The Role of Threads in Makespan Optimization

Generally speaking, a makespan-optimizing scheduler like the one used in the experiments shown above works according to an iterative procedure. The solving core is run according to makespan-optimization criteria so as to eventually obtain multiple, increasingly better solutions. Some degree of randomization is injected in the iteration to retain the ability to restart the search in the event that an unresolvable conflict is encountered, without incurring into the combinatorial overhead of a conventional backtracking search.

In the experiments shown in the previous section, some problems required very few iterations for the solution to converge to a short makespan, while others induced the scheduler to perform many runs before terminating. From a scheduling point of view, it is clear that the most challenging problems are prone to high degrees of optimization, a characteristic which is intrinsic to the structure of the completePOPs. In our simplified context with the single resource usage constraint for each task and the binary nature of the resources, these structural properties can be captured quite easily. As we will show in the remainder of this article, it is possible to identify the parameters which determine how trivial a problem is from the makespan optimization point of view.

4.1 Weak and Strong Coupling

Let us start with the example shown in figure 1. A solution to this scheduling problem can be obtained by sequencing all those tasks which produce resource contention: the tasks $\{T_3, T_4, T_5\}$ cannot be executed together because they all use R_1 (which has capacity 1), and the same holds for tasks $\{T_8, T_9, T_{10}\}$, which use resource R_2. Therefore, any instantiation in time in which these tasks are overlapping violates the resource capacity constraints, and is thus not an admissible solution. Figure 4 shows two solutions for this problem in the case that $D_i = 1, \ \forall D_i \in \mathcal{D}$.

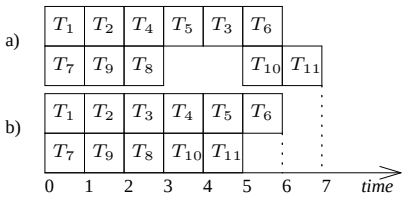

Figure 4. Two admissible solutions (schedules) for the completePOP shown in figure 1 for $D_i = 1, \ \forall D_i \in \mathcal{D}$: solution *a* has makespan 7, while solution *b* can be executed in 6 time units.

The fact that the problem admits solutions with different makespans is due to the precedence constraint $T_3 \prec T_{10}$. Indeed, if this constraint were not present, the problem could be decomposed into two disjoint scheduling problems, and the global solution

could be obtained by simply executing the two solutions simultaneously. In turn, the two scheduling problems so obtained would not be *makespan-optimizable*, since all the tasks use the same binary resource (R_1 or R_2) and thus must be serialized in the solution. In cases like this, we say that the scheduling problem is composed of *disjoint threads*. More precisely:

Definition 5. *Two threads Φ_k and $\Phi_{l\neq k}$ are* coupled *iff* $\exists\ T_i \in \Phi_k, T_j \in \Phi_l$ *s.t.* $T_i \prec T_j$*; if Φ_k and Φ_l are not coupled, they are said to be* disjoint*.*

It is rather intuitive that if all the threads in a given single-capacity completePOP P are disjoint, then all its solutions (schedules) will have the same makespan ($SOL^*(P) \equiv SOL(P)$, i.e. the set of optimal solutions coincides with the set of all solutions). Indeed, a problem in which the resource conflict resolution strategy can yield more or less optimized solutions ($SOL^*(P) \subset SOL(P)$) can be achieved only if there are precedence constraints which *couple* the threads, making the execution of one agent's tasks dependent on the behavior of another agent. More formally:

Definition 6. *Two threads Φ_k and $\Phi_{l\neq k}$ are* strongly coupled *if* $\exists\ T_i \in S \subseteq \Phi_k$ *s.t.* $\{T_i \prec T_j\} \vee \{T_j \prec T_i\} \in \mathcal{P} \ \wedge\ T_j \in \Phi_l$*, where S is a contention peak.*

Definition 7. *Two threads Φ_k and $\Phi_{l\neq k}$ are* weakly coupled *if all inter-thread constraints are bewteen tasks which do not belong to contention peaks.*

In more simple terms, weakly coupled threads can be connected only by constraints between tasks which do not belong to contention peaks. It is easy to show that the absence of strongly coupled threads is the structural trademark of trivial problems from a makespan optimization point of view. In fact:

Theorem 2. *A completePOP $P = \langle \mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{D} \rangle$ whose threads are at most weakly coupled is not makespan-optimizable.*

Proof. As we have already said, if all threads are disjoint, then all solutions are optimal since the problem can be decomposed into $|\mathcal{R}|$ problems whose solutions are all completely sequential.

Let us now suppose that P contains weakly disjoint threads, and that it is also makespan-optimizable, i.e. $SOL^*(P) \subset SOL(P)$. As a consequence, there exists a minimal critical set $S = \{T_i, T_j\} \subseteq \Phi_k$ whose possible serializations have different effects on the total makespan of the solution. Let $T_i \prec T_j$ lead to a solution SOL_1 with makespan m_1, $T_j \prec T_i$ lead to SOL_2 with makespan m_2, and let us suppose that $m_1 < m_2$. As a consequence, T_i must belong to the critical path in SOL_1 and T_j obviously does not belong to the critical path in SOL_1 (as this would make $m_1 = m_2$). Let the critical path for SOL_1 be $T_i \prec T_h \prec \ldots \prec T_{n+1}$. Since T_h may occur concurrently with T_j, T_h and T_j must belong to different threads, hence the necessary presence of the inter-thread constraint $T_i \prec T_h \in \Phi_{l\neq k}$, which contrasts with the hypothesis that all threads are at most weakly coupled. □

The previous theorem states that if in a completePOP there are no inter-thread precedence constraints between tasks which belong to contention peaks then the problem is not makespan-optimizable. This constitutes only a necessary condition for makespan-optimizability. In fact, even if a completePOP does indeed contain such constraints, it still may be non-optimizable.

4.2 The Face of Strong Coupling

It is experimentally verifiable that the presence of strong-coupling inter-thread constraints not only makes it possible for a completePOP to be makespan-optimizable, but the degree of optimizability of the completePOP depends rather strongly on the density of these constraints. Given a completePOP $P = \langle \mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{D} \rangle$, let the set of strong-coupling inter-thread constraints be $\mathcal{P}_s \subseteq \mathcal{P}$. The ρ curve in figure 5 shows the density of strong-coupling inter-thread links $|\mathcal{P}_s| \,/\, |\mathcal{P}|$ in the 500 completePOPs generated for the problems shown earlier.

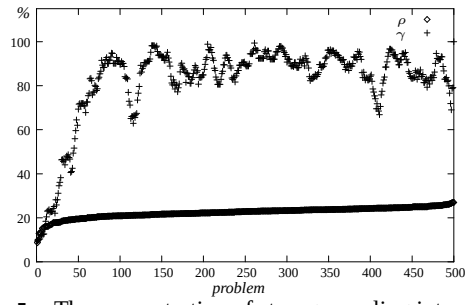

Figure 5. The concentration of strong-coupling inter-thread precedence constraints $\rho = |\mathcal{P}_s| \,/\, |\mathcal{P}|$, is directly responsible for the degree of optimizability of a completePOP $\gamma = \mathcal{S}^+/\mathcal{S}$.

In general, an optimizable scheduling problem admits solutions with a makespan in a bounded interval [*lb*, *ub*]. The width of this interval tells us *how much a completePOP can benefit from makespan optimization during scheduling*: on one hand, we have completePOPs which admit only one makespan and thus *cannot be optimized* (see theorem 2); on the other hand, a very different category of completePOP is one which *retains a higher degree of optimizability*, thus admitting a range of possible schedules with different makespans.

Enumerating all the solutions for a given completePOP is clearly unfeasible. Nonetheless, an estimate of how the makespans of the solutions are distributed can be obtained by sampling a set of solution extraction attempts on the completePOP and calculating the percentage of solutions with different makespans the scheduler is capable of finding. This estimate can be obtained as follows: each completePOP is solved by an optimizing scheduler, and the number of solutions with different makespans $\mathcal{S}^+$ is normalized over the total number of solutions found for that problem $\mathcal{S}$, yielding the $\gamma = \mathcal{S}^+/\mathcal{S}$ curve shown in the plot [6].

The next question we want to answer is the following: what is the causal characteristic of the planning problem which induces high densities of strong-coupling inter-thread constraints in the completePOP? In our agentified domain, two tasks which belong to different threads are linked if and only if the preconditions of one task depend on the effects of the other, which can be semantically interpreted as the enactment of a certain degree of *cooperation* among the agents, which is in turn strongly connected to the ρ constraint density measure. A high value of this density is indeed what makes a scheduling problem challenging for a makespan-optimizing scheduler. This confirms the rather intuitive fact that the role of optimizing scheduling algorithms acquires importance as the degree of agent interaction in the problem increases.

It is interesting to notice that one of the ways we can enforce a high degree of strong thread coupling in the completePOP is to *specialize* the roles of the agents in the causal problem specification.

[6] In order to make the γ curve more readable, the data was filtered with a sliding window average of width ten, which corresponds to a low-pass filter. The low accuracy of the estimate beyond $\rho \approx 21.3\%$ (beyond the first 100 completePOPs) is due to the fact that higher values of strong inter-thread constraint density require larger samples. In fact, as the problems get more challenging from a makespan-optimization point of view, more solution extraction attempts are necessary to derive useful statistics on the distribution of makespans in the [*lb*, *ub*] interval.

4.3 Planning Strategies and Challenging Scheduling Problems

As we saw earlier, the strong heuristic choices performed by FF affect the structure of the resulting scheduling problem. Figure 6 shows the strong-coupling constraint densities for the 500 problems used in the experimental evaluation of the planning strategies shown in figure 3.

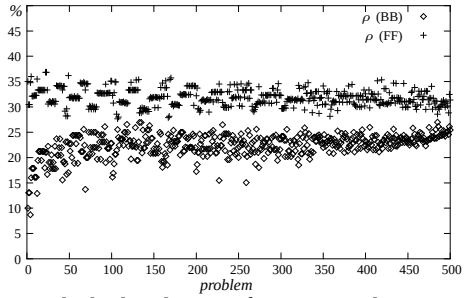

Figure 6. The higher density of strong-coupling inter-thread precedence constraints ρ makes FF-derived completePOPs more difficult for a makespan-optimizing scheduler.

Since higher densities of strong-coupling inter-thread precedence constraints determine a "wider" distribution of the makespans of the solutions, the consequence of FF's heuristics are twofold: first, the optimal makespan of the completePOP is greater or equal to that of a PG-derived completePOP which solves the same planning problem, and second, these completePOPs are actually more difficult to optimize. On the other hand, PG-based planners have the opposite effect, yielding causal structures which produce better makespans and also facilitate the performance of an optimizing scheduler. This effect is obtained thanks to the characteristic of the PG paradigm which tends to maximize the *size* of the contention peaks, as opposed to the performance-oriented HS strategies such as those employed by FF. Our analysis has shown that, at least in the case of FF, these heuristics reflect negatively on makespan-optimizing scheduling components.

5 Conclusions

This investigation stems from the general question of how to empower makespan-optimizing schedulers with causal reasoning capabilities. While a reasonably large amount of research has been dedicated to integrating P&S [8, 12, 13, 7], few analyses have focused on the nature of the information which is mutually shared between the two solving components. In this context, we have shown an investigation into the type of information which can be contributed to scheduling by planning, focusing on the structure of the causal knowledge that a scheduling tool can inherit from STRIPS-based reasoners.

In order to focus on the nature of the shared information rather than the mechanism with which this information is exchanged, we have employed a framework in which an explicit distinction between the causal and time/resource aspects of the problem is maintained (an architecture which is similar to REALPLAN-MS [13]). The aim of this analysis yields two results.

First, we show that HS planners are liable to produce scheduling problems with longer optimal makespans than those of the PG-derived problems. In this context, the over-committing nature of HS appears to be counter productive in makespan-optimization, while PG-based planners have the nice property of never committing to resource-leveling decisions, thus never invading the decision space of the scheduler. In fact, "blind" performance-oriented choices made by HS planners correspond to unilateral resource peak leveling decisions, the uninformed nature of which compromises the optimal makespan of the scheduling problem.

These uninformed heuristics also have another effect on the scheduling problem: by introducing the concepts of weak and strong inter-thread coupling, we show how increasing densities of these constraints not only determine "wider" makespan distributions among the schedules, but also make scheduling problems more challenging from a makespan-optimization point of view.

Future work will investigate how these results can be generalized by relaxing the single-resource, binary-capacity and non-dominating duration assumptions made herein. Also, an interesting future development could be to study on one hand how to empower heuristics to overcome their inefficiency in loosely-coupled makespan optimizing P&S, and on the other to analyse the behavior of planners which can deal with action durations in the loosely-coupled framework.

As a concluding remark, it is worth noticing that this investigation points to the interesting issue of using a classical planner to produce RCPSP benchmark problems, a direction we will be pursuing further in future work.

Acknowledgments

This research is partially supported by MIUR under project ROBOCARE, and by the Italian Space Agency (ASI). The Authors thank Angelo Oddi and the other members of the PST. Federico Pecora is also grateful to ECCAI for supporting his attendance to ECAI 2004.

REFERENCES

[1] C. Bäckström, 'Computational Aspects of Reordering Plans', *Journal of Artificial Intelligence Research*, **9**, 99–137, (1998).
[2] A.L. Blum and M.L. Furst, 'Fast Planning Through Planning Graph Analysis', *Artificial Intelligence*, 281–300, (1997).
[3] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, 'Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods', *European Journal of Operations Research*, **112**, 3–41, (1999).
[4] A. Cesta, G. Cortellessa, A. Oddi, N. Policella, and A. Susi, 'A Constraint-Based Architecture for Flexible Support to Activity Scheduling', in *LNAI 2175*, (2001).
[5] A. Cesta, A. Oddi, and S. Smith, 'Profile-Based Algorithms to Solve Multi-Capacitated Metric Scheduling Problems', in *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems*, (June, 1998).
[6] A. Cesta, A. Oddi, and S.F. Smith, 'A Constrained-Based Method for Project Scheduling with Time Windows', *Journal of Heuristics*, **8**(1), 109–135, (2002).
[7] M.B. Do and S. Kambhampati, 'Improving the Temporal Flexibility of Position Constrained Metric Temporal Planning', in *Proc. of the International Conference on AI Planning and Scheduling (ICAPS)*, (2003).
[8] M. Ghallab and H. Laruelle, 'Representation and Control in IxTeT, a Temporal Planner', in *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, (1994).
[9] J. Hoffmann and B. Nebel, 'The FF Planning System: Fast Plan Generation Through Heuristic Search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).
[10] S. Kambhampati, E. Parker, and E. Lambrecht, 'Understanding and Extending Graphplan', in *Proceedings of ECP '97*, pp. 260–272, (1997).
[11] H. Kautz and B. Selman, 'Unifying SAT-Based and Graph-Based Planning', in *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, ed., Jack Minker, College Park, Maryland, (1999). Computer Science Department, University of Maryland.
[12] N. Muscettola, 'HSTS: Integrating planning and scheduling', *In M.Zweben and M.S.Fox (Ed.)* Intelligent Scheduling, *Morgan Kauffmann*, (1994).
[13] B. Srivastava, 'RealPlan: Decoupling Causal and Resource Reasoning in Planning', in *AAAI/IAAI*, pp. 812–818, (2000).