Biasing the Structure of Scheduling Problems Through Classical Planners

Amedeo Cesta, Federico Pecora, Riccardo Rasconi
{a.cesta, fpecora, rasconi}@istc.cnr.it
Institute for Cognitive Science and Technology
Italian National Research Council
Viale Marx 15, I-00137 Rome, Italy

Abstract

Since Planning and Scheduling address complementary aspects of Problem Solving, much attention is recently being paid to the possibility of mutually exchanging the information yielded during the planning and the scheduling search procedures. In this context, we investigate a loosely coupled approach, 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.

Introduction

In their most general form, scheduling tools are designed to resolve resource contention on a set of predefined tasks, employing heuristics which are capable of optimizing with respect to some criteria on the duration of the tasks and/or on resource usage. One of the most common uses of schedulers is to generate schedules while minimizing the makespan objective function.

Our work stems from the general question of how to empower a makespan-optimizing scheduling tool with causal reasoning capabilities. While a reasonably large amount of research has been dedicated to integrating planning and scheduling (Ghallab & Laruelle 1994; Muscettola 1994; Srivastava 2000), few analyses have focused on the nature of the information which is mutually shared between the two solving components. In this context, we report an investigation into the type of information which can be contributed to scheduling by planning, focusing on the structure of the causal knowledge that STRIPS-based reasoners can contribute to a scheduling tool.

In order to focus on the nature of the shared information rather than the mechanism with which this information is exchanged, we employ 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 (Srivastava 2000)).

The aim of this analysis is to explore some basic properties of integrating planning into scheduling. First, we investigate the causal solving strategies which are best suited for makespan-optimizing schedulers, showing that the least-commitment property of Planning Graph (PG) based planners is related to critical path optimization. Second, we analyze the structure induced by the planning problem on the underlying scheduling sub-problem. This analysis reveals some properties which explain why the scheduling problem yielded by a causal reasoner is often not very challenging, and exposes the dependency of the distribution of a scheduling problem's makespans on the presence of threads of execution and strong-coupling constraints among them.

A Naïve Component-Based Approach

The general schema we will use in this investigation is depicted in figure 1. In this framework, a causal model of the environment is given as input to the 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 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 and/or renewable resources that ensure its executability. Given a Partial-Order Plan (POP) produced by the planning phase, i.e. a partially ordered set of activities $\{A_1, \ldots A_n\}$, these time and resource constraints must be accommodated into a new problem specification for the scheduling phase, yielding a complete representation of the problem instance which can be reasoned upon by the scheduler. For clarity, we will refer to the actions synthesized by the planner as *activities* (A_i), and the process of integrating these activities with time and resource related information yields *tasks* (T_i).

It is observed in (Do & Kambhampati 2003) that classical planners produce *position constrained* plans, that is plans in which the execution time of each activity is fixed to a specific time point, while planners such as HSTS (Muscettola 1994) and IxTeT (Ghallab & Laruelle 1994) produce *order constrained* plans, i.e. in which only the precedence constraints of the activities are maintained. In the light of this general classification, we can state that a POP as we intend it here is a position constrained plan for a relaxed problem

instance in which there are no resources and all activities have unitary duration. This plan is then transformed into an order constrained plan for the complete problem specification by means of the AdaptPlan procedure. As opposed to what is reported in (Do & Kambhampati 2003), there are no computational hazards in this transformation, since the position constrained plans we deal with do not contemplate metric and temporal constraints, which are added separately by the AdaptPlan procedure. Lastly, the scheduling phase computes a makespan-optimized schedule, which corresponds to a position constrained plan.

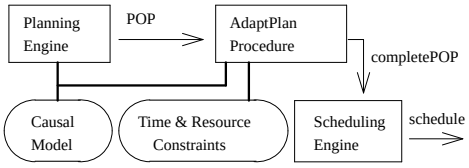

Figure 1: A losely-coupled integrated P&S achitecture.

It should be clear that this architecture is certainly not the best way to build an integrated reasoner, since its efficiency relies very strongly on how separable the two aspects of the problem are (hence the term 'naïve'). This requirement is often not realistic, since the degree of inter-dependency between causal and time/resource-related constraints in the problem is much higher. Nonetheless, this approach is well-suited for our purposes because it focuses our attention on the "intermediate" product of the combined reasoning, namely the structural properties of the scheduling problem which are enforced by causal reasoning.

Building a Scheduling Problem

In order to produce a problem specification which can be reasoned upon by the scheduling procedure, the POP is integrated with time and resource related information by means of an AdaptPlan procedure which will be shown shortly. We call the object produced by the AdaptPlan procedure a completePOP, and it can be defined as a POP augmented with the time and/or resource information associated to the problem at hand. 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 non-empty 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) (Brucker *et al.* 1999). This scheduling problem, which is well known in the OR and AI scheduling literature (Bartusch, Möhring, & Radermacher 1988), derives from a project management environment in which activities represent steps that must be performed to achieve completion of a project and are subject to partial order constraints that are dependencies on project progression.

A completePOP can be visualized in the form of a precedence graph, as shown in figure 2. 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.

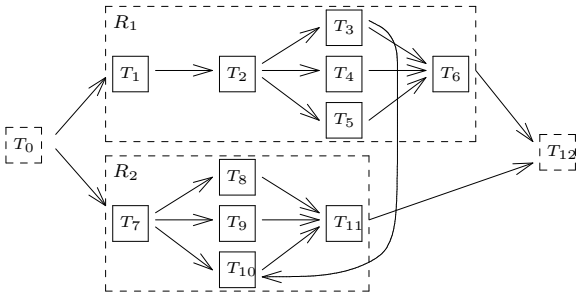

Figure 2: A completePOP with 11 tasks in which $\mathcal{R} = \{R_1, R_2\}$, $[T_1] = [T_2] = [T_3] = [T_4] = [T_5] = [T_6] = \{R_1\}$, and $[T_7] = [T_8] = [T_9] = [T_{10}] = [T_{11}] = \{R_2\}$. The source and sink tasks are T_0 and T_{12}.

The AdaptPlan algorithm is basically composed of two parts: in the first one, every activity belonging to the initial POP is integrated with information regarding its duration ($\text{Dur}(A_i)$) as well as the resource instances ($\text{Res}(A_i)$) which are necessary for its execution. Thus, the first phase of the AdaptPlan procedure yields a set of tasks, each of which represents one activity in the POP and is characterized by duration and resource usage parameters.

Given the POP which is produced by the planner, phase two of the AdaptPlan procedure adds a precedence constraint between every pair (T_i, T_j) of tasks if and only if $\text{Pre}(A_j) \subseteq \text{Eff}(A_i)$ or the link resolves a threat. It is easy to see that this constructive constraint-generating procedure is equivalent to the opposite approach, better known as *deordering*, in which a total-order plan is progressively "stripped" of its precedence constraints so long as the plan is still valid. This form of deordering is known as *minimal-constrained deordering*, and is extensively described in (Bäckström 1998). Such deordering has been proved to be achievable in polynomial time by means of a simple algorithm (MLCD) if it is possible to decide in polynomial time whether removing a precedence constraint from a plan causes it to be invalid. Incidentally, it is interesting to notice that the AdaptPlan algorithm achieves a minimal-constrained deordering of the input plan, and that the operation of checking whether a precedence constraint can be removed in the original MLCD procedure described in (Bäckström 1998) corresponds to the (polynomial) oper-

Algorithm 1 completePOP : AdaptPlan(POP)

for all $A_i \in$ POP **do**
  $\mathcal{T} \leftarrow T_i = A_i$
  $\mathcal{D} \leftarrow D_i =$ Dur(A_i)
  if Res$(A_i) \notin \mathcal{R}$ **then**
    $\mathcal{R} \leftarrow [T_i] =$ Res(A_i)
    $\mathcal{C} \leftarrow C_i =$ Cap(Res(A_i))
  end if
end for
{*Step 2: calculate causal links*}
for all $T_i \in \mathcal{T}$ **do**
  for all $T_j \in \mathcal{T}$ **do**
    if $\exists p$ s.t. $p \in$ Pre$(A_j) \wedge p \in$ Eff(A_i) **then**
      $\mathcal{P} \leftarrow \{T_i \prec T_j\}$
    end if
    if $\exists p$ s.t. $p \in$ Eff$(A_j) \wedge \neg p \in$ Pre(A_i) **then**
      $\mathcal{P} \leftarrow \{T_i \prec T_j\}$
    end if
  end for
end for
return new completePOP$(\mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{D})$

ation of checking whether a link should be added (step 2).

Planning Strategies

By observing the precedence graph produced by the AdaptPlan procedure, one extremely important characteristic for our purposes can be recognized as the *degree of concurrency* (or *parallelism*). Depending on many factors, such as the planner used and/or the global properties of the causal problem instance, the produced graph will exhibit a different extent of concurrency of the tasks.

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 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 [1]. 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. This characteristic would indeed maximize the performance of any makespan-optimizing scheduler employed in the integration.

Let us now analyze how different planning paradigms, namely Heuristic Search (HS) planning and Planning Graph (PG) based planning, behave in the loosely coupled framework. PG-based planners work by alternating one step of graph expansion to a search on the planning graph [2] 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 (Kambhampati, Parker, & Lambrecht 1997), 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 (Blum & Furst 1997).

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 parallel 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, let us proceed by contradiction. 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.

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 deordering of the POP. □

It is interesting to notice how the AdaptPlan procedure we have shown earlier *ignores* the partial ordering information of the initial solution produced by the planner. Rather, it "relies" on the fact that the plan is optimal with respect to the number of parallel steps. In other words, whereas any type of planner may be used to produce the initial POP, the quality of the final solution would indeed be negatively affected by the non-optimality (with respect to the number of parallel steps) of the generic planner.

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*. From a loosely-coupled point of view, this means that

[1] This assumption is realistic when there are no tasks in the problem specification whose duration is dominating.

[2] 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.

PG-based planners are capable of providing a causal network of tasks which is more suited for makespan-optimizing scheduling.

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 modeling multi-agent problems. Before analyzing the results, let us focus on this general multi-agent domain structure.

Construct	Where
`:capacity`	`:objects` definitions in the problem specification
`:uses`	`:action` definitions in the domain
`:duration`	`:action` definitions in the domain

Figure 3: Simple extensions to the PDDL language.

The causal specification of the domain encapsulates the notion of *executing agent*. The idea is to generate completePOPs which contain *threads* of execution, where a thread is a sequence of tasks which are executed by a single agent. Concurrency is obtained by inducing the presence of multiple threads (i.e. multiple agents) in the POP. Having augmented the PDDL language with the simple extensions shown in figure 3 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 4, 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.

We have chosen to impose some simplifying restrictions on the completePOPs:

- $C_i = 1,\ \forall C_i \in \mathcal{C}$ — the capacity of the resources is restricted to 1, i.e. there can be only binary resources;
- $|[T_i]| = 1,\ \forall T_i \in \mathcal{T}$ — all tasks use exactly one resource.

As a consequence, `CAP` 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. This

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

is enforced during scheduling, since the planning phase ignores the capacity constraints on the resources, while subsequent scheduling of the completePOP produces a schedule in which all resource conflicts have been resolved. For instance, a PG-based planner would certainly allow the execution of N instantiations of an operator on the same agent in the same step, while the scheduling phase would impose the resolution of the resource conflicts by serializing the N activities. More formally (Cesta, Oddi, & Smith 2002):

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* $\mathcal{P}$.

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). Indeed, not only would a PG-based planner allow the execution of one agent's tasks in a single step, but it would maximize this feature, as shown in theorem 1. In other words, since PG-based planners optimize with respect to the critical path, they tend to *maximize the size of the contention peaks*. This behavior explains why PG-based planners are particularly suited for loosely coupled P&S integration: contention peaks are precisely what any profile-based (Cesta, Oddi, & Smith 1998) 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.

Definition 4. *A* minimal critical set *is a contention peak*

such that any proper subset of its activities has a combined resource requirement $< C_k$.

Clearly, in the case of single-capacity completePOPs, all minimal critical sets contain only two tasks. In the example in figure 2, $\{T_3, T_4, T_5\}$ and $\{T_8, T_9, T_{10}\}$ are contention peaks, and $\{T_3, T_4\}$, $\{T_3, T_5\}$ and $\{T_4, T_5\}$ are minimal critical sets of $\{T_3, T_4, T_5\}$. In this type of domain, the two contention peaks $\{T_3, T_4, T_5\}$ and $\{T_8, T_9, T_{10}\}$ would represent precisely tasks which can be concurrent from the causal point of view, but not from the point of view of resource usage.

Experimental Results

Figure 5 shows the makespan-optimizing performance obtained with two different component based implementations of the loosely coupled framework: one employs BLACKBOX (Kautz & Selman 1999), 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 (Hoffmann & Nebel 2001), a heuristic search planner which was Top Performer in the Strips Track of the 3^{rd} International Planning Competition.

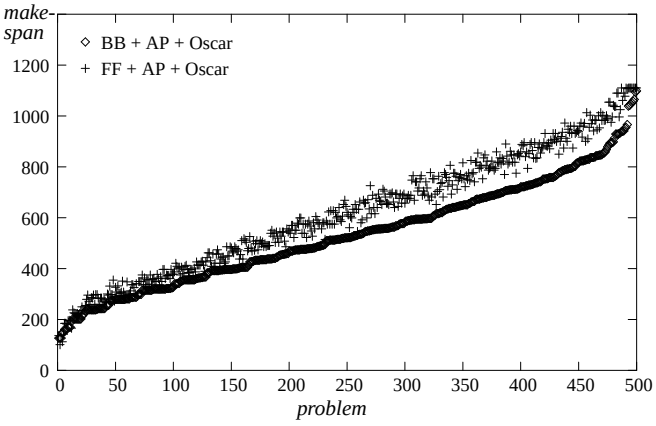

Figure 5: Comparing the makespan obtained by loosely coupling BLACKBOX and FF with O-OSCAR on 500 randomly generated multi-agent problems.

Both instantiations of the framework employ the O-OSCAR scheduler (Cesta *et al.* 2001), a versatile, general purpose CSP solver which implements the ISES algorithm (Cesta, Oddi, & Smith 2002). 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 complete, 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.

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 the SchedLoop procedure shown in algorithm 2: after performing *MAX-RESTART* attempts to produce a feasible schedule, if one of these attempts improves the best makespan found in the previous run, another run is performed by imposing the new best makespan as the horizon.

Algorithm 2 schedule : SchedLoop(completePOP)

```
boolean run ← ⊤
schedule best-sol ← ⊘
integer horizon ← upper-bound
integer best-ms ← upper-bound
while run do
  run ← ⊥
  for i = 1 to MAX-RESTART do
    schedule cur-sol ← compute-solution(horizon)
    if horizon > cur-sol.makespan() then
      best-ms ← cur-sol.makespan()
      best-sol ← cur-sol
      run ← ⊤
    end if
  end for
  horizon ← best-ms
end while
return best-sol
```

A makespan improvement in the first run induces the execution of a second run. If no further improvement is found, the whole procedure will be iterated only twice; as better makespans are progressively found, the procedure continues until no further improvement is possible. 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.

Weak and Strong Coupling

Let us start with the example shown in figure 2. 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 6 shows two solutions for this problem in the case that $D_i = 1,\ \forall D_i \in \mathcal{D}$.

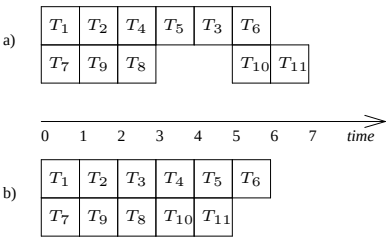

Figure 6: Two admissible solutions (schedules) for the completePOP shown in figure 2 for $D_i = 1,\ \forall D_i \in \mathcal{D}$: solution *a* has makespan 7, while solution *b* can be executed in 6 time units.

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 ($SOL^*(P) \equiv SOL(P)$). 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 belogn to contention peaks.*

In more simple terms, weakly coupled threads can be connected only by constraints whose tasks do not belong to contention peaks. In the example in figure 2, the constraint $T_3 \prec T_{10}$ makes Φ_1 and Φ_2 strongly coupled. If that constraint were instead between tasks T_2 and T_7, for instance, then Φ_1 and Φ_2 would have been only weakly coupled. 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 which connect tasks belonging 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.

The Face of Strong Coupling

In order to state a necessary and sufficient condition for a completePOP to be makespan-optimizable (i.e. not trivial), we would have to (1) enforce the presence of inter-thread constraints which originate from contention peaks, and (2) impose that the target of at least one of these constraints be on the critical path in at least one solution.

Nonetheless, 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 7 shows the density of strong-coupling inter-thread links $|\mathcal{P}_s| \,/\, |\mathcal{P}|$ in the 500 completePOPs generated by the AdaptPlan procedure for the problems shown earlier.

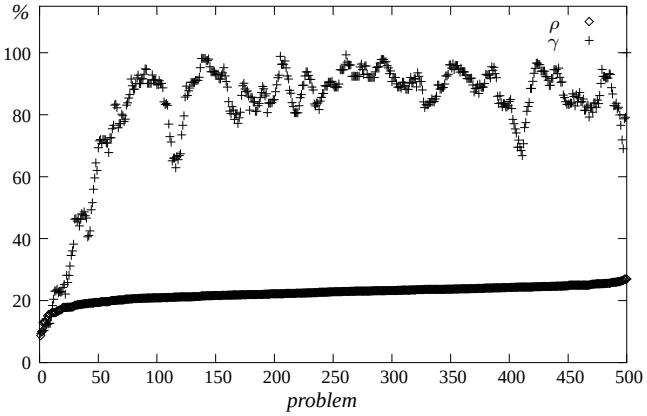

Figure 7: The concentration of strong-coupling inter-thread precedence constraints $\rho = |\mathcal{P}_s| \,/\, |\mathcal{P}|$, is directly responsible for the degree of optimizability of a completePOP $\gamma = \mathcal{S}^+/\mathcal{S}$.

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.

In order to make the γ curve more readable, the data was filtered with a sliding window average of width ten [3]. 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.

We can now go to the root of the thread-coupling structural property of completePOPs. The question we want to answer is the following: what is the causal characteristic of the planning problem which induces high strong-coupling inter-thread constraint densities in the completePOP? In our agentified domain, two tasks which belong to different threads (which are thus executed by different agents) 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. As a consequence, the ρ constraint density measure performed on the scheduling problem is also a measure of the extent to which the agents cooperate in the planner's solution. 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. Indeed, this was done in the experiments shown here, where two types of agents were specified in the domain definition, and increasing degrees of $|\mathcal{P}_s| \,/\, |\mathcal{P}|$ were obtained by specifying goals which induced the use of an increasing number of operators.

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 8 shows the strong-coupling constraint densities for the 500 problems used in the experimental evaluation of the planning strategies shown in figure 5.

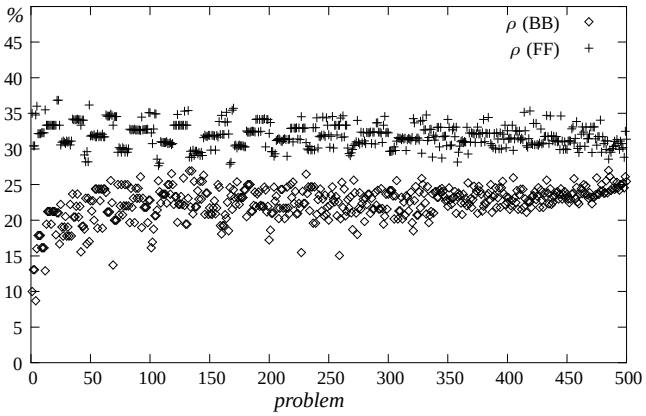

Figure 8: The higher density of strong-coupling inter-thread precedence constraints ρ makes FF-derived completePOPs more difficult for a makespan-optimizing scheduler.

Since higher densities of strong-coupling inter-thread prece-

[3] This corresponds to a low-pass filter: instead of taking $\gamma_i = \mathcal{S}_i^+/\mathcal{S}_i$ for every problem i, we take $\sum_{j=-5}^{5} \gamma_{i+j}$, the average of the ten problems around problem i.

dence 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* 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.

Conclusions

In this article we have explored two basic aspects related to integrating planning into scheduling by analyzing a general loosely coupled framework composed of a planner and a scheduler:

- given a problem specification which requires both causal and time/resource reasoning, the scheduling subproblems derived from PG-based planners have shorter optimal makespans than those obtained with HS-based components;
- the heuristic choices of HS-based planners reflect negatively also on the causal structure of the scheduling subproblem by imposing constraints which make them less prone to makespan-optimization.

The first point emerges from the observation that a desired property of the causal reasoning subsystem is to minimize the critical path through the causal network of tasks (completePOP) which is reasoned upon by the scheduler, and that the optimality with respect to the number of parallel steps of PG-based causal reasoning corresponds to a strong optimization with respect to the critical path in the completePOP. The better performance of PG-based planning with respect to makespan is confirmed experimentally by comparing the solutions to 500 problems obtained with the BLACKBOX+O-OSCAR and FF+O-OSCAR instantiations of the loosely-coupled framework.

The second point is entailed by the formal definitions of thread and thread coupling constraints. Thanks to the proposed formalization, we conclude that increasing densities of strong-coupling inter-thread constraints determine "wider" makespan distributions among the solutions of a completePOP. This provides a structural explanation of the fact that FF-derived plans not only yield longer makespans, but are also more challenging from a makespan-optimization point of view. Also this result is confirmed experimentally by comparing the two instantiations of the loosely-coupled framework.

Future work will strive to (1) look at how the distribution of makespans is affected by relaxing the single-capacity assumption made in this paper, and (2) explore the structure of scheduling problems which are generated planners which employ different heuristics.

Acknowledgments

This research is partially supported by MIUR (Italian Ministry of Education, University and Research) under project ROBOCARE (A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components) and the Italian Space Agency (ASI). The Authors are part of the Planning and Scheduling Team [PST] at ISTC-CNR and would like to thank Angelo Oddi and the other members of the team for their continuous support.

References

Bäckström, C. 1998. Computational Aspects of Reordering Plans. *Journal of Artificial Intelligence Research* 9:99–137.

Bartusch, M.; Möhring, R. H.; and Radermacher, F. J. 1988. Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Operations Research* 16:201–240.

Blum, A., and Furst, M. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 281–300.

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research* 112:3–41.

Cesta, A.; Cortellessa, G.; Oddi, A.; Policella, N.; and Susi, A. 2001. A Constraint-Based Architecture for Flexible Support to Activity Scheduling. In *LNAI 2175*.

Cesta, A.; Oddi, A.; and Smith, S. 1998. Profile-Based Algorithms to Solve Multi-Capacitated Metric Scheduling Problems. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems*.

Cesta, A.; Oddi, A.; and Smith, S. 2002. A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics* 8(1):109–135.

Do, M., and Kambhampati, S. 2003. Improving the Temporal Flexibility of Position Constrained Metric Temporal Planning. In *Proc. of the International Conference on AI Planning and Scheduling (ICAPS)*.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Kambhampati, S.; Parker, E.; and Lambrecht, E. 1997. Understanding and Extending Graphplan. In *Proceedings of ECP '97*, 260–272.

Kautz, H., and Selman, B. 1999. Unifying SAT-Based and Graph-Based Planning. In Minker, J., ed., *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. College Park, Maryland: Computer Science Department, University of Maryland.

Muscettola, N. 1994. HSTS: Integrating planning and scheduling. *In M.Zweben and M.S.Fox (Ed.)* Intelligent Scheduling, *Morgan Kauffmann*.

Srivastava, B. 2000. RealPlan: Decoupling Causal and Resource Reasoning in Planning. In *AAAI/IAAI*, 812–818.