Scheduling by Iterative Partition of Bottleneck Conflicts

Nicola Muscettola

CMU-RI-TR-92-05

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

February 1992

This work was sponsored in part by the National Aeronautics and Space Administration under contract # NCC 2-707, the Defense Advanced Research Projects under contract #F30602-88-C-001, and the Robotics Institute.

Abstract

In this paper we describe Conflict Partition Scheduling (CPS), a novel methodology that constructs solutions to scheduling problems by repeatedly identifying bottleneck conflicts and posting constraints to resolve them. The identification of bottleneck conflicts is based on a capacity analysis using a stochastic simulation methodology. Once a conflict is identified, CPS does not attempt to resolve it completely; instead it introduces constraints that merely decrease its criticality. By reducing the amount by which each scheduling decision prunes the search space, CPS tries to to minimize the chance of getting lost in blind alleys. Moreover, the capacity analysis metrics computed at each decision step give an indication of the areas of the search space where pruning is more likely to be effective. CPS effectiveness is demonstrated by the results of an extensive experimental analysis that compares it to two current scheduling methods: micro-opportunistic constraint-directed search and min-conflict iterative repair. CPS is shown to significantly outperform both of them on a standard benchmark of constraint satisfaction scheduling problems.

1 Introduction

A wide array of real-life complex management problems, from space mission planning to large-scale transportation planning to manufacturing production control, requires the solution of large scheduling problems. In a nutshell, scheduling consists of determining an assignment of values for the start and end time variables associated to a set of inter-related activities; such assignments must comply with a set of user requirements (e.g., release dates and due dates), must satisfy resource capacity constraints (e.g., no more than one activity can use a resource at the same time) and, possibly, must optimize some performance criteria (e.g., each activity should occur as soon as possible). Unfortunately, besides being NP-hard even in simple formulations, scheduling has the reputation of being one of the combinatorial problems that most stubbornly resists effective computational solutions.

In this paper we describe Conflict Partition Scheduling (CPS), a novel methodology that constructs solutions to scheduling problems by repeatedly identifying bottleneck conflicts and posting constraints to resolve them. The power of CPS stems from its two main characteristics: the use of constraint-posting as the primary problem solving operator and the use of a stochastic simulation methodology to extract measures of the characteristics of the intermediate partial solution states.

In general, there are two main ways to build a schedule. The first proceeds by generating unique assignments of values to state variables: this category includes simulation-based dispatching scheduling [12], search-based and opportunistic heuristic approaches [5] [15], [14], and iterative schedule repair [9] [16] [3]. The second way is to post additional sequencing constraints (e.g., activity x has to precede activity y) to exclude capacity conflicts, and deduce variable assignments from the resulting network of constraints: besides CPS, this category includes other heuristic [2] and search-based [1] approaches.

Intuitively, one would expect notable advantages in leaving flexibility within a plan/schedule. While a schedule with all the time variables fixed to a value represents a single, completely detailed operating condition, a constraint network can represent an entire set of legal behaviors and allows the problem solver to concentrate on critical conditions, leaving open details that can be easily adjusted at execution time (e.g., the sequencing of two operations that require the same resource if the requests are not likely to be in conflict). This provides more robust and reliable guidance during during activity execution in the presence of unexpected events. In this paper we will concentrate on the advantages arising during schedule generation. In a value committment approach, each time the value of a time variable is fixed, the dimensionality of the remaining search space (i.e., the set of all the possible assignments of start and end times for the remaining unscheduled operations) is collapsed by one. Alternatively, constraint-posting approaches, by only restricting the range the problem's variables without a necessary loss of dimensionality, leave available a higher number of possibilities at each problem solving state. This is an important pre-condition to the development of more global measures of the search space and suggests a lower danger for the scheduling algorithm to get lost in blind alleys.

To guide the introduction of new constraints, we need search space "metrics" that point to where such introduction is most likely to be effective. Ideally one would like to estimate the distribution of solutions over the search space in order to direct problem solving toward areas with high solution density. Unfortunately, this might require to actually generate solutions, which in the case of scheduling problems is extremely hard in and of itself. CPS uses the opposite approach: it looks for areas where there is high likelyhood of not finding a solution and posts constraints to move away from them. Similarly to most of the current bottleneck-guided scheduling approaches [11] [15] [3] [14] [2], CPS computes its focusing metric on a relaxed version of the original problem space. In particular, CPS ignores the capacity limitations of the original scheduling problem, allowing a resource to process more than one operation at a time. The lack of disjunctive capacity constraints in the relaxed space allows a very fast generation of

consistent value assignments. The focusing mechanism of CPS stochastically generates a sample of value assignments in the relaxed space and compiles statistics on the likelihood of contention out of the generated sample.

The results reported in this paper demonstrate the superiority of the constraint posting approach embodied in CPS with respect to value commitment scheduling approaches. In particular, we show through the results of an extensive experimental analysis that CPS outperforms two dominant heuristic scheduling methodologies: micro-opportunistic search [14] and min-conflict iterative repair [9]. CPS is able to reliably solve tougher problems than possible with the two considered value commitment approaches.

In the rest of the paper, section 2 describes the CPS methodology, introducing the underlying representation principles (section 2.1), the details of the capacity analysis (section 2.2) and the basic scheduling cycle (section 2.3). Section 3 reports the results of the experimental analysis. Section 3.1 describes the class of scheduling problem over which the analysis has been conducted. Sections 3.2 and 3.3 briefly describe the two competing methods. Section 3.4 reports the experimental results. Finally, Section 4 reports conclusions and future directions.

2 Conflict Partition Scheduling

2.1 Domain Representation Principles and Notation

CPS is implemented in the HSTS planning and scheduling framework [7]. HSTS addresses problems that involve the determination of sets of behaviors of a dynamical system [10] [4] by explicitly assuming a decomposition of the state of the dynamical system into a vector of state variables, each of which can assume at any point in time one and only one of a given set of possible values. HSTS has shown the ability of addressing complex planning and scheduling problems [7] [6].

Here we restrict our attention to scheduling domains, i.e., domains where each state variable is a **resource** that can only assume only one of two values at a given time: *PROCESSING*(?*task*,?*resource*) and *IDLE*. The problem is defined by a list of available resources and a network of goals. Each goal is a time interval, or **token**, with its duration bounded by an interval $[d, D]$ and labeled with with a value of the kind $PROCESSING(Task_i, Resource_k)$. Goals within the network can be related by metric interval temporal relations: for example , τ_1 *before*$([d, D])$ τ_2 indicates that the start time of τ_2 is constrained to occur after the end time of τ_1, with a delay included in the interval $[d, D]$. To simplify the exposition, in the following we will assume that the initial network contains only *before* temporal relations. Underlying the goal token network there is a network $\langle V_t, C_t \rangle$ of time variables and metric interval distances, that we call **time points network**. Time propagation through this network allows deduction of a range of possible time values for each token start and end time point: in particular, we will identify with $EST(\tau)$ the lower bound of the interval associated to the start time of the token τ and with $LFT(\tau)$ the upper bound of τ's end time. A given a token τ can be allowed to occur (e.g., a task could be allowed to be in process on a resource) at any time $EST(\tau) \leq t < LFT(\tau)$. In the following, we will assume time to be discrete and identify with H the problem horizon, i.e., the segment of time over which we want to generate behaviors. The primitive operation that we will use to refine a goal network into a schedule is the introduction of an additional *before*$([0, +\infty])$ relation between tokens requiring the same resource; a schedule is achieved when enough constraints have been added to insure that no two such tokens can occur at the same time.

2.2 Stochastic Capacity Analysis

CPS computes search space metrics through a stochastic capacity analysis that extends and generalizes the one first proposed in [Muscettola&Smith87]. The logic behind the method is quite simple. Generating consistent schedules from an intermediate problem solving state is complicated by the presence of unresolved disjunctive sequencing constraints (i.e., two tokens using the same resource must necessarily follow one another); alternatively generating assignments that satisfy the constraints that are explicitly represented in the network is very easy, even considering additional preference criteria (e.g., select times as soon as possible). Therefore, the method relaxes the implicit capacity constraints and uses a simple strategy that includes preference information to generate a set of complete variable assignments. Then, it reintroduces the resource capacity constraints and analyzes where the simulation strategy has performed the worse with respect to them (i.e., more than on token requiring the same resource occurs at the same time). The higher the possibility of capacity conflicts for a given set of tokens, the more urgent the need to arbitrate start and end time assignments among them by introducing additional sequencing constraints.

More precisely, given the time points network $\langle V_t, C_t \rangle$, a complete value assignment to the variables in V_t consistent with the constraints in C_t is obtained through a **stochastic simulation** of the network; the following steps are repeatedly applied until all variables in V_t have a value:

1. select a variable $t \in V_t$ according to a predefined *variable selection strategy*;
2. select a value for t within the current range of its possible values, according to a *stochastic value selection rule*;
3. assign the value to t and propagate the consequences throughout the time point network; the results are new ranges of possible values for the variables in V_t;
4. delete t from V_t.

At the end of a stochastic simulation, all the tokens in the goal network have a definite start and end time. It is now possible to evaluate the effect of their planned occurrence with respect to their requests for resource availability. For each token we record the time interval of occurrence; also, for each resource and each instant of time within the scheduling horizon H, we record how many tokens require its use.

After repeating the stochastic simulation cycle N times, we gather statistical metrics to summarize the overall process. In particular, we compute the two following metrics:

- **token demand**: for each token τ and for each time $EST(\tau) \le t_i < LFT(\tau)$, token demand $\Delta(\tau, t_i)$ is equal to n_{t_i}/N, where n_{t_i} is the number of stochastic simulations in which the token's interval of occurrence overlaps t_i.
- **resource contention**: for each resource ρ and for each time t_j within the scheduling horizon H, resource contention $X(\rho, t_j)$ is equal to n_{t_j}/N, where n_{t_j} is the number of stochastic simulations in which ρ is requested by more than one token at time t_j.

Token demand and resource contention represent the effects of the simple value assignment strategy from two distinct points of view. Token demand gives an indication of how much a token relies on the availability of the corresponding resource at a certain time: the higher the demand, the more the assignment strategy relies on the availability of the resource at that time for that token. Resource contention, on the other end, gives an indication of the level of inadequacy of the simple assignment strategy in generating consistent schedules; the higher the resource contention at a given time, the higher the need to introduce additional constraints to

attend to the "assignment congestion" for the tokens that rely on that time.

The stochastic simulation described before is parametric with respect to both variable selection strategy and value selection rule. Several alternatives are possible. A very simple variable selection strategy is *forward temporal dispatching*. A set of time variables, or open set, is initialized to the sources of the time point network, i.e., the time points with no incoming temporal constraints. Variable selection proceeds by first extracting one variable from the open set and then updating the open set for the next iteration. During the update, all the time points that follow the selected time point in a temporal constraint are considered for introduction in the open set; a candidate time point is actually inserted in the open set if all its other incoming temporal constraints originate from time points that have already been assigned a value. Analogously, it is possible to define a *backward temporal dispatching* strategy. More sophisticated strategies may involve the selection of variables according to predefined or dynamically evaluated priorities. The stochastic value selection rule can change depending on the preference that we might want to emphasize during the stochastic simulation. For example, no preference can be reflected by a random value selection according to a uniform distribution. Preference toward finishing as early as possible (as late as possible) can be introduced by executing the random selection with respect to a monotonically decreasing (increasing) distribution.

2.3 The scheduling process

The CPS process is driven by the identification of **bottlenecks**. We use the following formal definition of a bottleneck:

- **Bottleneck**: Given the set of resource contention functions $\{X(\rho, .)\}$ resulting from a capacity analysis, we call bottleneck a pair $\langle \rho_b, t_b \rangle$ such that:

  $$X(\rho_b, t_b) = \max \{X(\rho, t)\}$$

  for any $\rho \in R$ and $t \in H$ such that $\{X(\rho, t) > 0\}$.

A bottleneck identifies a neighborhood in the search space where the time assignment strategy generates a maximum of inconsistency. Since this depends on several tokens requiring the same resource at the same time in most of the value assignments, we need to sort these token requests. However, in order to avoid unduly over-constraining the token network, we do not attempt to completely resolve the conflict (e.g., by generating a complete ordering of the requests); instead we follow a limited commitment approach, introducing constraints that will simply lower the level of contention. In this way we attempt to perturb as the partial solution state as minimally as possible. The rationale behind this is that, since the assignments reflect the preferences injected in the stochastic simulation, we should try to find an intermediate state that is more consistent but still near to the bulk of the stochastic value assignments to maintain a high level of solution goodness. If we were to completely resolve a conflict, we would be forced to make decisions without a precise idea of their impact on the rest of the network, possibly significantly lowering the quality of the solution. From this point of view the method is closer to micro-opportunistic approaches [14] than to macro opportunistic ones [15] [2].

The conflict resolution strategy can be briefly described as follows. First, a conflict is identified in the set of tokens that maximally rely on the bottleneck (i.e., on the availability of ρ_b at time t_b). Then the conflict set is partitioned in two subsets, T_{before} and T_{after}, and every token in T_{before} is constrained to end before the start of any token in T_{after} by the introduction of appropriate *before* ($[0, +\infty]$) constraints.

The basic scheduling cycle is described in Figure 1.

1. **Capacity Analysis**: estimate token demand and resource contention profiles using the stochastic capacity analysis.
2. **Termination Test**: If the resource contention is zero for each resource over the entire scheduling horizon, then exit. The current token network is the solution.
3. **Bottleneck Detection**: Identify resource and time with the highest contention;
4. **Conflict Identification**: Select the tokens that are most likely to participate in the bottleneck conflict.
5. **Conflict Partition**: Sort the set of conflicting tokens by inserting appropriate temporal constraints.
6. **Constraint Propagation**: Propagate the consequences of the introduction of the new constraints to the time point network underlying the token network.
7. **Consistency Test**: If the time point network is inconsistent, exit with failure.
8. Go to 1.

Figure 1: The scheduler's basic cycle

The conflict resolution steps rely on the token demand functions $\Delta(\tau, .)$ constructed during Capacity Analysis. Since scheduling preferences are embedded in the strategy used in the stochastic simulation, we can observe that a value with a higher demand is also likely to have a high level of preference. Therefore token demand can be also considered as a measure of relative value utility on the range of times included in the token's time bound. Whichever interpretation we use, however, we need a means to simplify comparisons among sets of these distributions. The method we use is to summarize the demand function into a single parameter, the *demand centroid*, given by:

$$t_{\Delta}(\tau) = \frac{\sum_{t=EST(\tau)}^{LFT(\tau)-1} t\Delta(\tau, t)}{\sum_{t=EST(\tau)}^{LFT(\tau)-1} \Delta(\tau, t)}$$

We will now discuss in depth the heuristics used during Conflict Identification and Conflict Partition.

- **Conflict Identification.** We want to be sure that the introduction of temporal constraints among the conflicting tokens leaves the time points network consistent. Our Conflict Identification heuristic is designed to insure that no cycle could be possibly introduced, whichever conflict partition is selected during the Conflict Arbitration phase. The heuristic starts by identifying the set of candidate tokens T_{cand} as the set of all tokens insisting on the bottleneck, i.e., all τ that request ρ_b such that $EST(\tau) \le t_b < LFT(\tau)$. Then T_{cand} is sorted according to the distance of the demand centroid from the bottleneck time, with:

  $$\tau_1 < \tau_2 \text{ if } |t_{\Delta}(\tau_1) - t_b| < |t_{\Delta}(\tau_2) - t_b|$$

  We construct the conflict set T_{conf} by cyclically augmenting it with the minimum token in T_{cand} and deleting from T_{cand} the selected token and all the tokens that precede and follow it in the token network. The selection is repeated until T_{cand} is empty. If during the first selection cycle T_{cand} remains empty (i.e., the selected

token precedes or follows in the network all the tokens in T_{cand}), T_{cand} is restored to the state preceding the selection cycle and the selection process resumes after having eliminated the minimum token from T_{cand}.

When applied to a bottleneck, this Conflict Identification heuristic is guaranteed to return a conflict set partitionable in two non-empty sets. In fact the identification of a bottleneck confirms that there are at least two tokens with overlapping resource requests for some legal assignment of times to the time point network. But this is only possible if these two tokens are not constrained to occur sequentially. Therefore there are at least two tokens that will not be eliminated by T_{cand} during the selection process and will therefore be included in T_{conf}.

- **Conflict Partition.** The partition of the conflict set T_{conf} in T_{before} and T_{after} relies on the interpretation of the demand centroid as center of gravity of temporal preferences. This is done by analyzing the mutual position of the demand centroids and favoring the introduction of temporal constraints that respect this mutual relation. In particular, a **conflict centroid** t_{conf} is evaluated for T_{conf} as:

$$t_{conf} = \frac{\sum_{\tau \in T_{conf}} t_{\Delta}(\tau)}{|T_{conf}|}$$

where $|T_{conf}|$ is the cardinality of T_{conf}. For each token in T_{conf} the heuristic decides on its assignment as follows:

  - $\tau \in T_{before}$ if $t_{\Delta}(\tau) - t_{conf} < 0$
  - $\tau \in T_{after}$ if $t_{\Delta}(\tau) - t_{conf} > 0$
  - select randomly $\tau \in T_{before}$ or $\tau \in T_{after}$ if $t_{\Delta}(\tau) = t_{conf}$

Figures 2 and 3 illustrate the effects of the execution of a scheduling cycle on resource contention and token time bounds.* The top graph of figure 2 shows the resource contention over time for the bottleneck resource, while the bottom part shows the time bounds associated with each of the tokens requesting the resource. The solid black segment at the far right of a time bound represents the token duration; each token therefore displays a high degree of slack. The tokens have been partitioned in T_{before} and T_{after} sets as a result of previous processing: in particular, the 5 time bounds at the bottom constitute the T_{before} set while the 5 at the top belong to T_{after}. Bottleneck detection on the next cycle identifies $t_b = 140$ as the new bottleneck time and all the 5 top operations are selected to be part of the new T_{conf}. Figure 3 shows the result of the partition of the new T_{conf}; the level of contention around the bottleneck time has been lowered and the resource contention function presents now three peaks, one for each of the new time bound clusters identifiable in the figure. Notice that the slack associated with each time bound is only slightly reduced by this partitioning.

*The figures were generated by using SAGE, a system for the automation of data presentation [13]

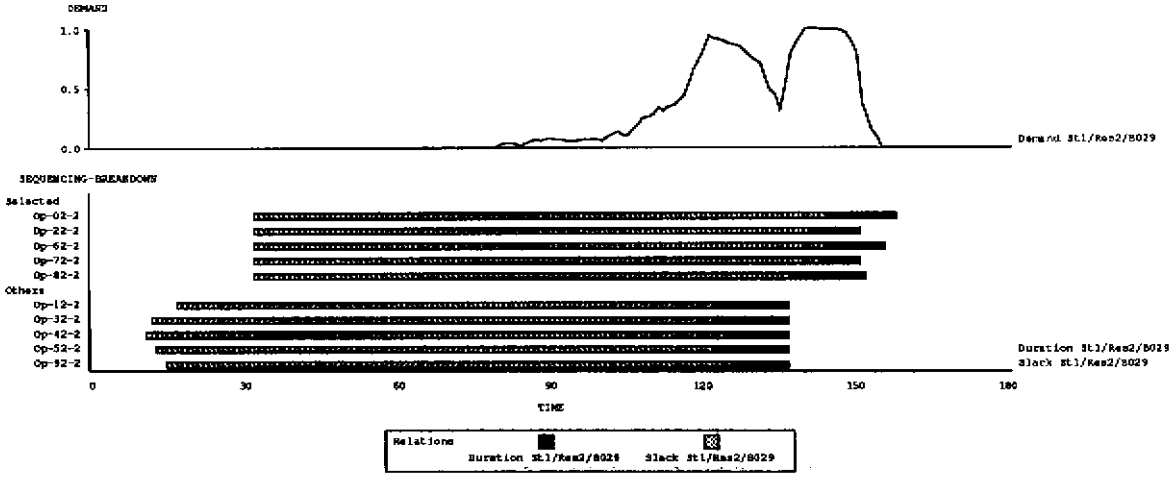

Figure 2: Initial bottleneck resource status

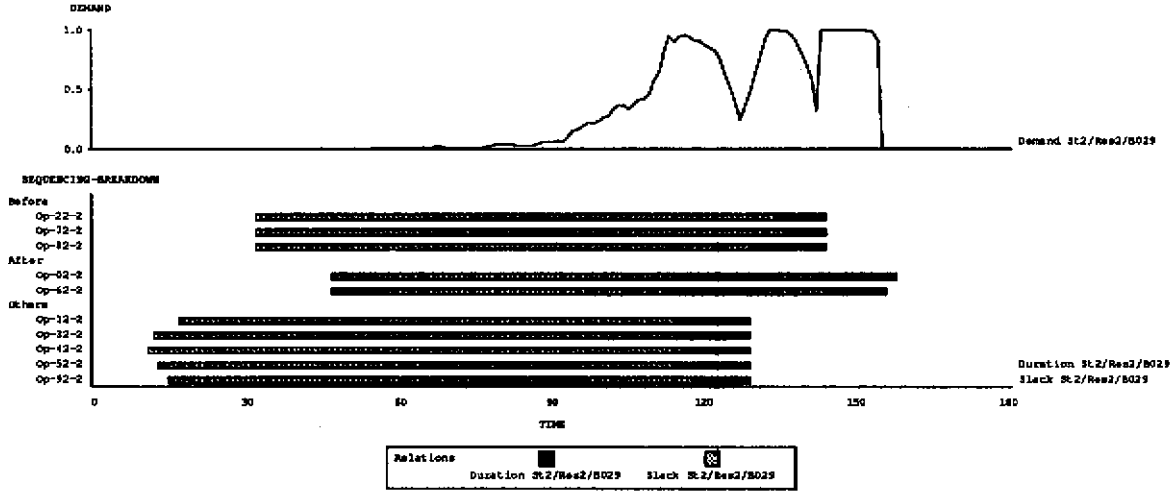

Figure 3: Bottleneck resource status after a scheduling cycle

3 Experimental Analysis on Constraint Satisfaction Scheduling Problems

3.1 The problem

To demonstrate the viability of CPS, we compared its performance with respect to two state-of-the-art heuristic scheduling methods that use the value commitment approach: micro-opportunistic scheduling [Sadeh] and min-conflict iterative repair scheduling [Minton]. The analysis was conducted on a standard constraint satisfaction scheduling benchmark proposed by [Sadeh]. All problems in this benchmark suite satisfy the following formal definition.

The underlying dynamical system consists of m resources ρ_k The initial goals are described by n disjoint linear token sequences, each of length m. Each token has a different fixed duration, so the only variables that need value assignment are the token start times. For each sequence, or **job**, each constituent token requires a different ρ_k than any other token in the sequence. The pattern with which each job visits the m resources is arbitrary. In addition to the sequencing constraints imposed by the jobs and the disjunctive ordering constraints imposed by the resource

capacity limitations (i.e., two tokens using the same resource must necessarily follow one another), there are also release date constraints (i.e., for each job i, its first token must start after a release date r_i) and due date constraints (i.e., for each job i, its last token must end before a due date d_i.

The previous description identifies a job-shop scheduling problem. The main source of complexity stems from the interaction of the temporal limits imposed by release and due dates and the resource capacity limits. We seek a value assignment to the underlying time points network that satisfies the previous constraints; the lack of a goodness criterion to be maximized makes this a constraint satisfaction problem.

3.2 Micro-Opportunistic Scheduling

Micro-opportunistic scheduling [14] adopts a value commitment approach within a constrained heuristic search framework. The basic operator applied at each step of the search, consists of two subsequent decisions: (1) select which start time variable (time point) should be assigned a value; (2) select which value should be assigned to the variable. The method builds a solution by iteratively extending consistent partial value assignments. The search chronologically backtracks when no consistent extension of a partial assignment is possible, i.e., the range of the possible values of some time point not yet committed to has become empty.

As in CPS, the focusing heuristics of micro-opportunistic scheduling are founded on computing measures of resource contention and token demand in each search state. For example, conflicting sets of tokens are selected by identifying a bottleneck with respect to a resource contention measure. In contrast to CPS, different relaxation assumptions are made in different stages of the search to compute different metrics; also, these assumptions are often stronger than those made by CSP. This allows fast computability of the metrics using deterministic methods that start from the underlying probabilistic characteristics of the relaxed space. However, complex constraint interactions are ignored, which, in principle, limits the classes of problems for which the relaxation assumptions are semantically correct. For example, one of the simplifying assumptions for computing the resource contention measure is that all token demands are generated independently one from the other, according to a uniform stochastic value selection rule; this obviously ignores the interaction among the selection of start times for tokens connected by chains of temporal constraints.

Micro-opportunistic scheduling has been successfully applied to the benchmark used in the empirical analysis of this paper. In particular, micro-opportunistic scheduling has significantly outperformed several other heuristic search approaches, both applicable to more general constraint satisfaction and specifically tuned to the solution of scheduling problems.

3.3 Iterative repair scheduling

Min-conflict iterative repair [9] is an member of a family of scheduling methods based on the repeated modification of inconsistent total value assignments. The value assigned to each time point variable has an associated measure of conflict; this is obtained by considering all the constraints included in the original problem definition that involve the time point variable and adding together one inconsistency factor for each constraint. An inconsistent constraint contributes with a +1 factor while a consistent constraint contributes with a 0 factor. The constraints included in the measure of conflict for the scheduling problem are: (1) pairwise constraints enforcing job sequentiality (i.e., the start of a token must follow the end of the token immediately preceding it in the job); (2) pairwise constraints among tokens enforcing resource capacity limitations (i.e., for each pair of tokens requiring the same resource, one token must precede the other, with no preference on their ordering). In our implementation of the method, release and due date constraints are guaranteed to be satisfied, since the the range of values for

each variable is obtained by propagating release and due dates through the job precedence constraints. At each iteration, a variable is selected randomly among those whose values have a positive conflict measure. The conflict measure is evaluated for all the possible values within the range of the selected variable. Then, a value is assigned to the selected variable, randomly selected within the set of values with minimum level of conflicts. The conflict measure for the values of the variables related to the selected variable by some constraint is then updated to reflect the new value assignment. This basic cycle is repeated until the conflict levels of all time point values is equal to 0, at which point a solution to the problem is achieved.

As in CPS, the scheduling algorithm contains an intrinsic factor of randomness. However, differently than in CPS and Micro-Opportunistic scheduling, randomness is not used as a means for extracting search space metrics, but as the fundamental method for wandering around the search space. The conflict metric used by the approach is very local in nature. Therefore, the min-conflict approach would be expected to find difficulty in problems where tight interactions among time variables limit the number of consistent value assignments to set of variables making the generation of an assignment one component at a time difficult without a more global view of the structure of the search space. The min-conflict iterative repair method has been included in the SPIKE scheduling system [8] for the generation of long-term schedules for the Hubble Space Telescope.

3.4 Experimental Comparison

This section reports the results of the experimental analysis. The scheduling problems used are the Constraint Satisfaction Scheduling benchmark used in [14]. The benchmark consists of 6 groups of 10 problems, each problem including 10 jobs and 5 resources. The set of problems differ with respect to: (1) spread of the release and due dates among jobs; (2) number of a-priori bottlenecks. The spread is controlled by varying the amplitude of the intervals within which release and due dates are generated. The benchmark includes three spread levels: wide (w), narrow (n) and null, i.e., both release and due date intervals are collapsed to single points (0). An a-priori bottleneck is introduced by selecting a position in the job's sequence of tokens and forcing all the tokens in that position to use the same resource (e.g., the second token of all jobs requires resource ρ_2). The benchmark includes problems with 1 and 2 a-priori bottlenecks**

Conflict Partition Scheduling (CPS) was run on the benchmark. Since the criterion of success does not include any particular preference on the characteristics of the solution, the stochastic simulation was tuned to obtain maximum performance, using a forward temporal dispatching variable selection strategy and a uniform value selection rule. At each scheduling cycle, capacity analysis was conducted on the basis of $N = 100$ runs of the stochastic simulation. We also ran our implementation of a Min-Conflict Iterative Repair scheduler (MIN CONF). Since the method could cycle indefinitely, we set 5000 as an upper limit on the number of repair cycles, after which the run was considered unsuccessful. The maximum number of cycles was selected so that the maximum processing time spent by MIN CONF on a problem would approximately equal the longest run time of CPS. The performance data concerning the constraint satisfaction method of micro-opportunistic scheduling (MICRO OPP) are taken from [14]). The search was run with an upper limit of 1000 on the number of states generated. Given the intrinsic random nature of both CPS and MIN CONF, these algorithms were each run 5 times for each problem instance in order to test the repeatability of the results. This is not necessary for MICRO OPP, it being a deterministic algorithm.

Table 1 reports the comparative performance results. Each column reports the number of problems in the benchmark that were solved by CPS, MIN CONF and MICRO OPP. The rows

---

**For a more detailed description of the benchmark see [14]

are labeled with the spread and bottleneck parameters of the problems set in the benchmark; for example label w/2 refers to problems with wide spread and two a priori bottlenecks. Each number in parenthesis represents a repeatability measure for the results, i.e., the average fraction of runs that succeeded for a problem in the problem set for which a solution was found. No repeatability measure is needed for MICRO OPP since its scheduling algorithm is deterministic. The last row of the table reports the total number of problems in the benchmark that were solved by the different methods. The results clearly show that CPS outperforms both MIN CONF and MICRO OPP.

	CPS	MIN CONF	MICRO OPP
w/1	10 (0.96)	10 (0.36)	10
w/2	9 (0.89)	3 (0.33)	10
n/1	10 (0.94)	5 (0.44)	8
n/2	10 (0.92)	1 (0.40)	9
0/1	10 (0.82)	4 (0.25)	7
0/2	9 (0.91)	0	8
	58	23	52

Table 1: Comparative results

With respect to MICRO OPP, CPS consistently solved more problems in the toughest problem categories (spread n and 0). It is fair to say that, when both MICRO OPP and CPS succeeded, MICRO OPP's run times were significantly shorter than CPS's; when CPS alone succeeded, instead, the processing time of the unsuccessful MICRO OPP search was comparable or longer than the time taken by CPS to succeed (possibly involving more than one run). Excluding any improvements in CPS's run time due to more efficient implementations, the results still confirm CPS superiority over MICRO OPP on the toughest problems.

The performances of MIN CONF were consistently inferior than both CPS and MICRO OPP[***] On the problems where a solution was found, MIN CONF usually converged relatively quickly; however, the very poor repeatability measures still suggest that, on the average, CPS outperforms MIN CONF also with respect to its efficiency. Notice that together with the performance degradation due to decrease of problem spread, MIN CONF yields its worse performance on the set of problems including two a-priori bottlenecks. The presence of several bottlenecks in a scheduling problem introduces complex interactions that are traditionally difficult to deal in scheduling algorithms. The results confirm the expectation that the local nature of the measure of conflict used in MIN CONF is unable to detect such interactions.

[***] In fact, they also are inferior to any of the heuristic search methods to which MICRO OPP is compared in [14].

4 Conclusions

This paper describes the Conflict Partition Scheduling methodology and gives supports to the claim that CPS is an effective way to solve difficult scheduling problems. The framework presented here is based on a strictly monotonic problem solving approach. While the capacity analysis seems adequate to orient constraint posting, unavoidable imprecision of the method sometimes causes the scheduling process to fail. To avoid this, we need to either flexibly tune the precision of the capacity analysis, (increasing the number of runs of the stochastic simulation), or introduce backtracking and the possibility of (local) search. Future work will also demonstrate the extensibility of the technique to the solution of optimization scheduling problems, where, besides consistency, we must also achieve a high value for a given goodness criterion.

Acknowledgments

The author thanks Stephen Smith for his helpful comments on an earlier draft of this paper.

REFERENCES

[1] Applegate, D., Cook, W.
A Computational Study of Job-Shop Scheduling.
Technical Report CMU-CS-90-145, School of Computer Science, Carnegie Mellon University, 1990.

[2] Adams, J., Balas, E., Zawack, D.
The shifting Bottleneck Procedure for Job Shop Scheduling.
Management Science 34, 1988.

[3] Biefeld, E., Cooper, L.
Bottleneck Identification Using Process Chronologies.
In *Proceedings of the 12th International Joint Conference on Artificial Intelligence.* 1991.

[4] Dean, T., Wellman, M.
Planning and Control.
Morgan Kaufmann, 1991.

[5] Fox, M.S., Smith, S.F.
ISIS: A Knowledge-Based System for Factory Scheduling.
Expert Systems 1(1):25-49, 1984.

[6] Frederking, R.E., Muscettola, N.
Temporal Planning for Transportation Planning and Scheduling.
In *Proceeding of 1992 IEEE International Conference on Robotics and Automation (to appear).* 1992.

[7] Muscettola, N., Smith, S.F., Cesta, A., D'Aloisi, D.
Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture.
IEEE Control Systems Magazine 12(2), 1992.

[8] Johnston, M.D.
SPIKE: AI Scheduling for NASA's Hubble Space Telescope.
In *Proceedings of the 6th Conference on Artificial Intelligence Applications*, pages 184-190. IEEE Computer Society Press, 1990.

[9] Minton, S., Johnston, M. D., Philips, A. B., Laird, P.
Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method.
In *Proceedings of the 8th National Conference on Artificial Intelligence.* 1990.

[10] Muscettola, N.
Planning the Behavior of Dynamical Systems.
Technical Report CMU-RI-TR-90-10, The Robotics Institute, Carnegie Mellon University, 1990.

[11] Muscettola, N., S.F. Smith.
A Probabilistic Framework for Resource-Constrained Multi-Agent Planning.
In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 1063-1066. Morgan Kaufmann, 1987.

[12] Panwalker, S.S., W. Iskander.
A Survey of Scheduling Rules.
Operations Research 25:45-61, 1977.

[13] Roth, S.F., Mattis, J.
Automating the Presentation of Information.
In *Proceedings of the Conference on Artificial Intelligence Applications*. IEEE, Miami Beach, February, 1991.

[14] Sadeh, N.
Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling.
Technical Report CMU-CS-91-102, School of Computer Science, Carnegie Mellon University, 1991.

[15] Smith, S.F., Ow, P.S., Potvin, J.Y., Muscettola, N., and Matthys, D.
An Integrated Framework for Generating and Revising Factory Schedules.
Journal of the Operational Research Society 41(6):539-552, 1990.

[16] Zweben, M., Deale, M., Gargan, R.
Anytime Rescheduling.
In *Proceeding of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*. 1990.