A CONSTRAINT-POSTING FRAMEWORK FOR SCHEDULING UNDER COMPLEX CONSTRAINTS

Cheng-Chung Cheng and Stephen F. Smith

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890 USA
email: sfs@cs.cmu.edu

Abstract

Scheduling in many practical industrial domains is complicated by the need to account for diverse and idiosyncratic constraints. Quite often these requirements are at odds with the techniques and results produced by the scheduling research community, which has focused in large part on solutions to more idealized, canonical problems. Recent research in temporal reasoning and constraint satisfaction has produced problem solving models that operate with respect to much more general representational assumptions. These frameworks offer possibilities for developing scheduling technologies that naturally extend to accommodate the peculiarities of various application domains. One critical issue, of course, is whether such generality can be obtained without sacrificing scheduling performance.

In this paper, we investigate this issue through application of a previously developed constraint satisfaction problem solving (CSP) model for deadline scheduling to a complicated, multi-product hoist scheduling problem encountered in printed circuit board (PCB) manufacturing. The goal is to maximize throughput of an automated PCB electroplating facility while ensuring feasibility with respect to process, capacity and material movement constraints. Building from a heuristic procedure generically referred to as PCP (precedence constraint posting), which relies on a temporal constraint graph representation of the problem, we straightforwardly define an extended solution procedure for minimizing makespan. In a series of comparative experiments, our procedure is found to significantly outperform previously published procedures for solving this hoist problem across a broad range of input assumptions.

1. Introduction

Constraint satisfaction problem solving (CSP) models and heuristics have increasingly been investigated as a means for solving scheduling problems [6, 13, 14, 17, 19]. One claimed advantage of such approaches is flexibility; representational assumptions are quite general and developed techniques are thus applicable across a broad range of problems. However, much of this work has (naturally) focused on deadline scheduling problems; attempts to extend these approaches to address the optimization issues that dominate most manufacturing scheduling environments are less common. Moreover, analysis of these approaches has tended to concentrate on idealized, canonical benchmark problems which do not stress generality in representational assumptions. There is limited understanding of performance characteristics in domains that require complex representations.

In this paper, we empirically investigate these issues in the context of a specific application domain. We take as our starting point, a previously developed deadline scheduling procedure called PCP (Precedence Constraint Posting)[19, 6]. Somewhat unconventionally from the standpoint of CSP scheduling research, PCP is rooted in a problem representation akin to a disjunctive graph formulation [3], i.e., the problem is formulated as one of sequencing all pairs of operations that are competing for common resources. This formulation allows direct contact to be made with the representational assumptions of contemporary temporal reasoning frameworks, and gives rise to a "constraint-posting" approach to solution development.

We consider the applicability of PCP to the multi-product hoist scheduling problem previously studied in [21]. This problem is challenging in that it requires enforcement of many non-standard constraints, including sequence-dependent setup (travel) times, imprecise operation processing times (expressed in terms of minimum and maximum bounds) and no-delay constraints on the execution of consecutive job steps, while attempting to minimize overall schedule makespan (to maximize facility throughput). We show how this problem can be straightforwardly modeled for solution by PCP, develop an extended procedure for makespan minimization that utilizes PCP, and experimentally demonstrate its comparative performance advantage. We begin by summarizing the basic PCP scheduling framework.

2. The PCP Scheduling Model

2.1. Problem Representation

The PCP modeling framework can be formalized more precisely as a type of *general temporal constraint network (GTCN)* [12]. In brief, a GTCN T consists of a set of variables $\{X_1, ..., X_n\}$ with continuous domains, and a set of unary or binary constraints. Each variable represents a specific temporal object, either a time point (e.g., a start time st_i or an end time et_i) or an interval (e.g. an operation O_i). A constraint C may be qualitative or metric.

A qualitative constraint C is represented by a disjunction $(X_i\, q_1\, X_j) \vee ... \vee (X_i\, q_k\, X_j)$, alternatively expressed as a relation set $X_i\, \{q_1, ..., q_k\}\, X_j$, where q_i represents a *basic*

qualitative constraint. Three types of basic qualitative constraints are allowed:

1. interval to interval constraints - The GTCN definition of [12] includes Allen's 13 basic temporal relations [1]: *before, after, meets, met-by, overlaps, overlapped-by, during, contains, starts, started-by, finishes, finished-by*, and *equal*. For convenience, we also include *before-or-meets* and *after-or-met-by*, which represent the union of relation pairs (*before, meets*) and (*after, met-by*) resp. [4].

2. point to point constraints - The relations identified in [20], denoted by the set $\{<, =, >\}$, are allowable here.

3. point to interval or interval to point constraints - In this case, the 10 relations defined in [10] are specifiable, including *before, starts, during, finishes, after*, and their inverses.

A metric constraint C is represented by a set of intervals $\{I_1, ..., I_k\} = \{[a_1, b_1], ..., [a_k, b_k]\}$. Two types of metric constraints are specifiable. A unary constraint C_i on point X_i restricts X_i's domain to a given set of intervals, i.e. $(X_i \in I_1) \vee ... \vee (X_i \in I_k)$. A binary constraint C_{ij} between points X_i and X_j restricts the feasible values for the distance $X_j - X_i$, i.e., $(X_j - X_i \in I_1) \vee ... \vee (X_j - X_i \in I_k)$. A special time point X_0 can be introduced to represent the "origin". Since all times are relative to X_0, each unary constraint C_i can be treated as a binary constraint C_{0i}.

To support representation of sequence-dependent resource "setup" times, one further augmentation to Mieri's GTCN model is made. Specifically, we extend the representation of qualitative constraints to optionally include a metric quantifier. For our purposes in this paper, it is sufficient to include only the following two extended relations: *before-or-meets[lag]* and *after-or-met-by[lag]*, where $lag \geq 0$ designates a minimum metric separation between the related intervals. Thus, whereas the constraint O_i *before-or-meets* O_j implies $et_i \leq st_j$, the extended constraint O_i *before-or-meets*$[lag_{ij}]$ O_j implies $et_i + lag_{ij} \leq st_j$.

A GTCN forms a *directed constraint graph*, where nodes represent variables, and a edge $i \rightarrow j$ indicates that a constraint C_{ij} between variables X_i and X_j is specified. We say a tuple $X = (x_1, ..., x_n)$ is a *solution* if X satisfies all qualitative and metric constraints. A network is *consistent* if there exists at least one solution.

Figure 1 depicts the constraint graph for a simple 2 job, 2 machine deadline scheduling problem. We are given ready times r_1 and r_2, due dates d_1 and d_2, processing times p_i (for operation O_i), and disjunctive constraints between operation pairs (O_1, O_3) and (O_2, O_4) (dictating exclusive use of machines $M1$ and $M2$ respectively). The objective is to determine if the network is consistent (i.e. admits a feasible solution). Since a GTCN with no disjunctive arcs defines a Simple Temporal Problem[7], which is solvable in $O(n^3)$ time by the Floyd-Warshall's all-pairs shortest-paths algorithm, one simple procedure for determining whether a solution exists is to enumerate all possible labelings of the network. In the problem displayed in Figure 1, there are 4 possible labelings (corresponding to the different sets of sequencing decisions that might be taken).

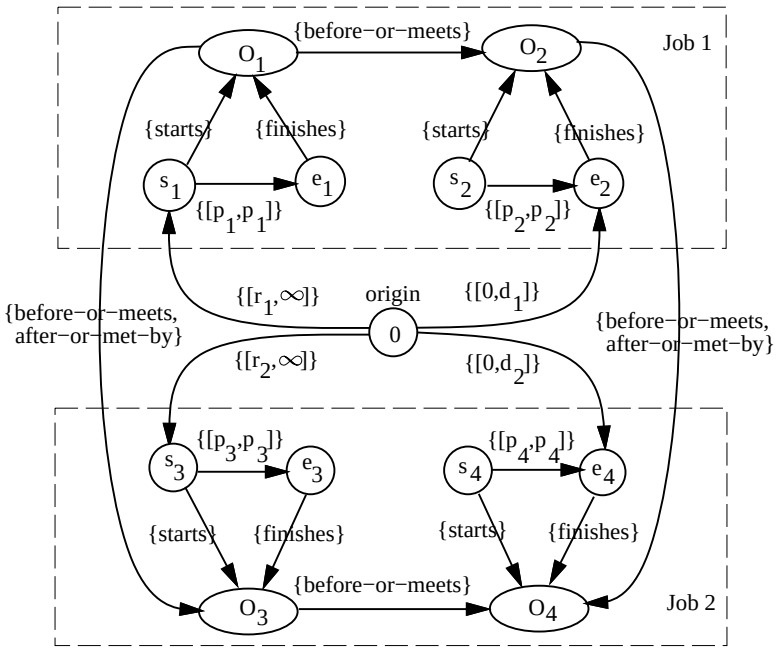

Figure 1. Constraint Graph for simple 2 job, 2 machine problem

2.2. A Heuristic Solution Procedure

In [19, 6], a heuristic procedure, referred to as Precedence Constraint Posting (PCP), is defined for solving deadline scheduling problems formulated as a GTCN. Most generally, PCP is designed as a backtracking search procedure over a *meta-CSP* network, whose variables correspond to disjunctive arcs in the GTCN and whose domains are simply the set of possible disjuncts. In the case of the deadline scheduling problem (as indicated above), this leads to the set of decision variables $V = \{Ordering_{ij}\}$, corresponding to each operation pair (O_i, O_j) that requires the same resource, each with two possible values $O_i \prec O_j$ or $O_j \prec O_i$, corresponding to whether O_i is processed before or after O_j. A feasible solution is incrementally constructed within PCP by repeatedly (1) updating shortest path lengths to verify continued consistency of the current partial solution, (2) applying dominance conditions to identify unconditional decisions and enable early search space pruning , (3) selecting an as yet unassigned variable $Ordering_{ij}$, and (4) posting one of the two possible ordering constraints (values) associated with this variable into the constraint network. PCP gains decision-making leverage from simple, pair-wise analysis of the flexibility associated with each sequencing decision. In step 2, this analysis is used to detect and post any "forced" sequencing decisions, and to detect inconsistent solution states. In steps 3 and 4, it is used respectively as a basis for variable and value ordering (i.e., what decision to make next and what value to assign). These distinguishing aspects of PCP are briefly summarized below.

Specification and use of dominance conditions in PCP derives directly from the concept of *Constraint-Based Analysis* (CBA) [8, 9], developed for solution of classical job shop problems. In [6], these dominance conditions are generalized to account for the wider range of constraints that are specifiable in a GTCN. Suppose $Ordering_{ij}$ is a currently unassigned variable in the meta-CSP network, and consider the cur-

rent partial solution. Let s_i, e_i, s_j, and e_j be the start and end points respectively of operations O_i and O_j, and further assume sp_{ij} is the shortest path length from e_i to s_j and sp_{ji} is the shortest path length from e_j to s_i. Then, four mutually exclusive cases can be identified:

Case 1. If $sp_{ij} \geq 0$ and $sp_{ji} < 0$, then $O_i \prec O_j$ must be selected.

Case 2. If $sp_{ji} \geq 0$ and $sp_{ij} < 0$, then $O_j \prec O_i$ must be selected.

Case 3. If $sp_{ji} < 0$ and $sp_{ij} < 0$, then the partial solution is inconsistent.

Case 4. If $sp_{ji} \geq 0$ and $sp_{ij} \geq 0$, then either ordering relation is still possible.[1]

The second distinguishing aspect of PCP is its use of sequencing flexibility analysis for variable and value ordering, which dictates how the search should proceed in the undecided states (case 4 above). One very simple estimate of the sequencing flexibility associated with a given $Ordering_{ij}$ is the minimum shortest path length, $\omega_{ij} = min(sp_{ij}, sp_{ji})$, which gives rise to a variable ordering heuristic that selects the $Ordering_{ij}$ with the minimum ω_{ij}. This heuristic makes reasonable sense; at each step, the decision which is closest to becoming forced is taken. However, its exclusive reliance on ω_{ij} values can lead to problems (see [19] for details), and PCP consequently bases variable ordering decisions on a slightly more complex notion of *biased* shortest path length. Specifically, $bsp_{ij} = sp_{ij}/\sqrt{S}$ and $bsp_{ji} = sp_{ji}/\sqrt{S}$ are computed, where $S = \frac{\min\{sp_{ij}, sp_{ji}\}}{\max\{sp_{ij}, sp_{ji}\}}$ estimates the degree of similarity between the two values sp_{ij} and sp_{ji}. The sequencing flexibility associated with a given decision $Ordering_{ij}$ is redefined to be $\omega_{ij} = min(bsp_{ij}, bsp_{ji})$, and the decision selected during variable ordering is the decision with the minimum ω_{ij}. The value ordering heuristic utilized in PCP simply selects the ordering relation implied by $max(bsp_{ij}, bsp_{ji})$, i.e. the sequencing constraint that retains the most temporal flexibility is posted.

Finally, to avoid the exponential worst case behavior of a complete backtracking search procedure ($\mathcal{O}(n^3 2^{|V|})$), we introduce a less-costly, backtrack-free version of the basic PCP procedure for later use in this paper. In this case, total reliance is placed on the ability of the search to move directly to a feasible solution; if Case 3 above is ever encountered (i.e., no feasible ordering for a given ordering decision), the search simply terminates in failure (and does not produce a solution). The effectiveness of this partial solution procedure, which we will refer to as "Simple PCP" below, was demonstrated in [19] on a set of previously published CSP scheduling benchmark problems. Its worst case complexity can be seen to be $\mathcal{O}n^3|V|$.

[1] More precisely, these dominance conditions assume selection among {*before-or-meets, after-or-met-by*} relation sets. If a given $Ordering_{ij}$ involves selection among qualitative relations with metric quantifiers (e.g., representing sequence-dependent setup times), then substitute $(sp_{ij} - lag_{i,j})$ for sp_{ij} in all expressions. A similar generalization applies when $Ordering_{ij}$ involves selection among a $before, after$ relation set; in this case $lag_{i,j} = lag_{ji}$ is the smallest possible temporal increment. To simplify the presentation of variable and value ordering heuristics below, we will continue with the assumption of {*before-or-meets, after-or-met-by*} value sets; refer to /citeCheng95 for the general formulation.

3. The Hoist Scheduling Problem

To demonstrate the viability of our CSP scheduling model, we consider its application to a complex, previously studied scheduling problem: the multi-product version of the hoist scheduling problem [21]. The hoist scheduling problem finds its origin in printed circuit board (PCB) electroplating facilities. In brief, a set J of jobs, $J = \{J_1, ..., J_n\}$ each require a sequence of chemical baths, which take place within a set M of m chemical tanks, $M = \{1, ..., m\}$. Execution of a particular chemical bath operation O_i requires exclusive use of tank m_i. The processing time of any O_i required for a job j is not rigidly fixed; instead there is a designated minimum time, p_i^{min}, that j must stay in the tank for the bath to accomplish its intended effect and a maximum time, p_i^{max}, over which product spoilage occurs. All jobs move through the chemical tanks in the same order, though a given job may require only a subset of the baths and thus "skip" processing in one or more tanks along the way. All job movement through the facility is accomplished via a single material handling hoist, H, which is capable of transporting a job initially into the system from the input buffer, from tank to tank, and finally out of the system into the output buffer. H can grip only a single job at a time, moves between any two adjacent stations (input buffer, tanks, or output buffer) at constant speed s, and has constant loading and unloading speeds, L and U, at any tank or buffer. The facility itself has no internal buffering capability; thus jobs must be moved directly from one tank to the next once they have entered the system. The objective is to maximize facility throughput (or equivalently minimize makespan) subject to these process and resource constraints.

Most previous work in hoist scheduling has considered simplified versions of this problem. The single-product, hoist scheduling problem has received the most attention. In this special case, the problem can be reduced to one of finding a minimum length cycle of hoist operations, which can then be repeated over time; several algorithms for generating optimal (or near-optimal) cyclic schedules have been reported [16, 18, 11, 2]. In [23, 22], a hoist scheduling problem involving a multi-product facility is considered, but without permitting variance in job routings (i.e. no tank skipping). To our best knowledge, only [21] has reported procedures for solving the general hoist scheduling problem defined above.

4. Representation as a Constraint Graph

The hoist scheduling problem is straightforwardly modeled in the extended GTCN formalism of Section 2.1. Let's first consider representation of the process constraints of any given job j. For each individual operation O_i in j's process sequence, we define three constraint graph nodes: two time points, representing O_i's start and end points, and an interval, representing O_i itself. A given pair of start and end points are related to its corresponding interval through use of the qualitative constraints *starts* and *finishes* respectively. The values ultimately assigned to the time points of any O_i in the constraint graph will represent O_i's scheduled start and finish times.

The duration of a given operation O_i is modeled by specifying a metric constraint between its start and end points. There are two cases, corresponding to the two types of operations that must be interleaved to process any given job. If O_i is a

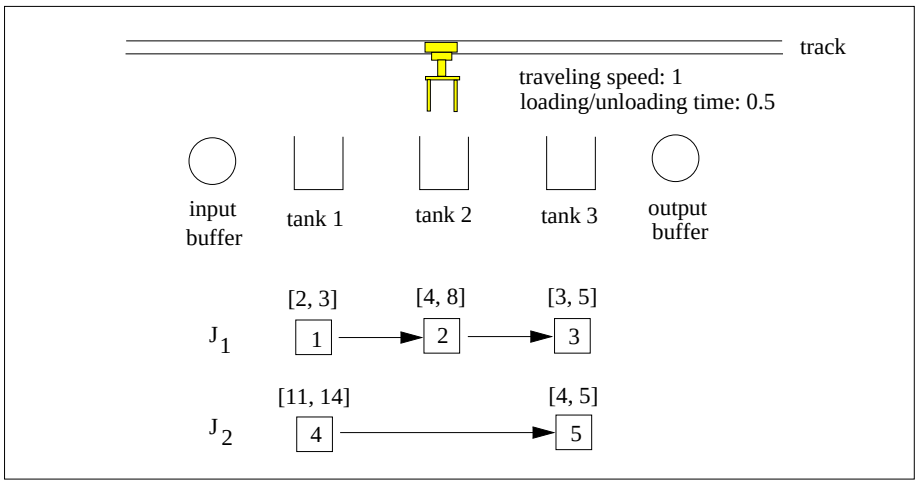

Figure 2. Simple hoist scheduling problem

required tank operation, the constraint $\{[p_i^{min}, p_i^{max}]\}$ is specified to enforce minimum and maximum allowable times in tank m_i. If O_i is a hoist (transport) operation, then the constraint is instead a function of the distance to be traversed and hoist loading, traveling and unloading speeds. The constraint $\{[mt, mt]\}$ is specified in this case, where $mt = L + s * (destination_i - origin_i) + U$. For convenience, we assume a correspondence between a given tank's index and its location, and assign indices 0 and $m + 1$ to the system's input and output buffers respectively to complete this mapping.

The process sequence for any given job j is specified by temporally relating the intervals defined for j's constituent tank and transport operations. Since the end of any given operation O_i must, by definition, coincide with the start of O_{i+1}, the *meets* constraint is used to establish this linkage. O_i *meets* O_{i+1} implies that $et_i = st_{i+1}$. Disjunctive constraints on resource usage are specified as relation sets between operations that require the same resource. Again there are two cases. For tank operations O_i and O_j where $m_i = m_j$, the constraint $\{$*before, after*$\}$ is introduced. Note that the relation set used for this purpose in the canonical job shop problem, $\{$*before-or-meets, after-or-met-by*$\}$, is not correct here, since it is physically impossible to switch between tank operations without intermediate loading and unloading. Synchronization of competing hoist operations is the only remaining issue. In this case, however, basic qualitative relations are insufficient, as they do not allow us to account for the "setup" time that may be required to position the hoist at the loading location. For each pair of hoist operations O_i and O_j belonging to different jobs, we specify the constraint O_i $\{$*before-or-meets*$[h_{ij}]$, *after-or-met-by*$[h_{ji}]\}$ O_j, where $h_{ij} = s * |destination_i - origin_j|$ and $h_{ji} = s * |destination_j - origin_i|$.

To illustrate, Figure 2 shows a simple example problem (taken from [21]) involving 2 jobs to be processed in a 3 tank system. Each operation is displayed below the tank that is required, and gives its minimum and maximum processing times. The corresponding constraint graph model is given in Figure 3 (with qualitative relations abbreviated as follows: starts (s), finishes(f), meets (m), before (b), after (bi), before-or-meets(bm), after-or-met-by (bmi)).

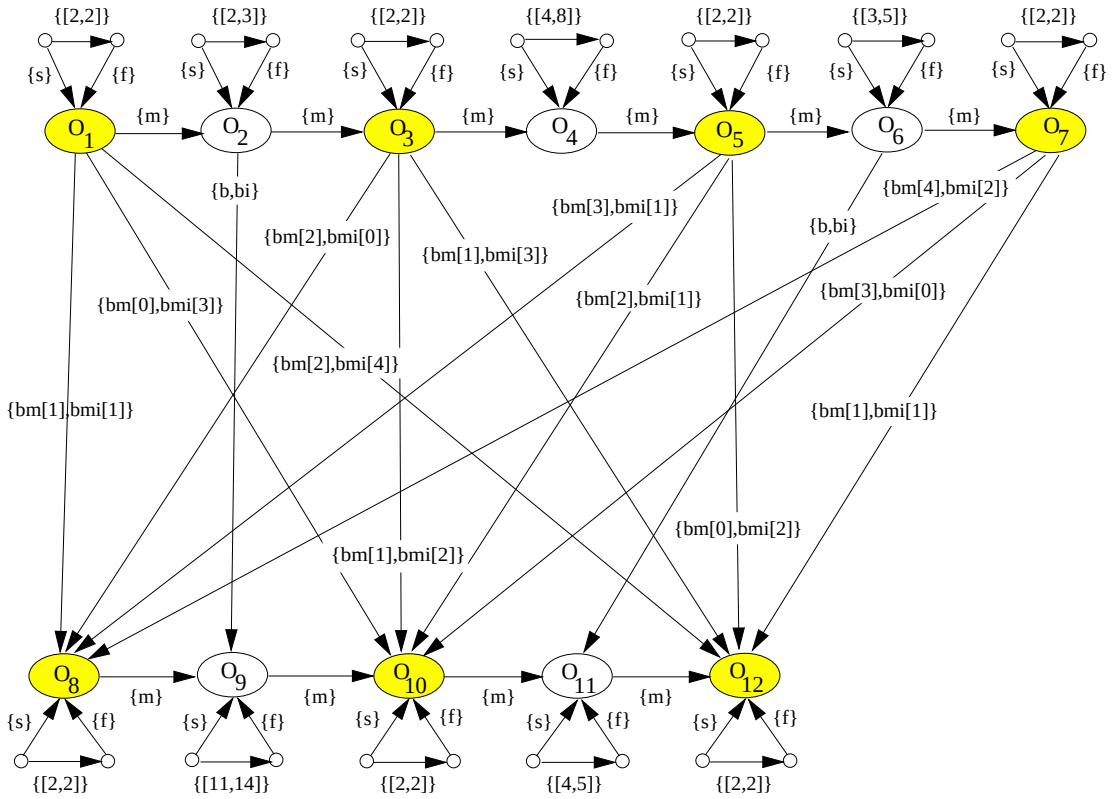

Figure 3. The constraint graph for simple hoist scheduling problem

5. An Extended Procedure for Makespan Minimization

As a deadline scheduling procedure, PCP does not provide a direct basis for minimizing makespan, and at first glance may seem inappropriate for application to the hoist scheduling problem. However, by exploiting the concept of problem duality, we can transform a makespan minimization problem into a series of related deadline scheduling problems and exploit PCP as a subproblem solver. Suppose that we are given an instance of a makespan problem, denoted by $\Pi_M(I)$ where I represents the problem data associated with this problem instance. If we know the minimum makespan for $\Pi_M(I)$ to be C^*_{max}, then we can reduce $\Pi_M(I)$ to a special deadline problem $\Pi_D(I,d)$, where each job is assigned a 0 ready time and a common deadline d, with $d = C^*_{max}$. For any $d \geq C^*_{max}$, we are assured that a feasible solution to $\Pi_D(I,d)$ exists. More important, C^*_{max} defines a unique common deadline such that for $d < C^*_{max}$, $\Pi_D(I,d)$ has no feasible solution. This dual relationship between problems $\Pi_M(I)$ and $\Pi_D(I,d)$ implies that the makespan problem $\Pi_M(I)$ can be reformulated as a problem of finding the smallest common deadline, d_{min}, for which $\Pi_D(I,d)$ has a feasible solution.

Given an optimal algorithm for solving the deadline problem $\Pi_D(I,d)$, it is straightforward to construct an search procedure for determining d_{min} (and its associated schedule). We start with known upper and lower bounds d_U and d_L on the common deadline d_{min}; at each step, we attempt to solve $\Pi_D(I,d)$ for $d = (d_U + d_L)/2$. If a feasible schedule is found, d_U becomes d; otherwise, d_L becomes d. We continue the search until $d_U = d_L$, retaining the schedule with the best makespan as

we go. Unfortunately, d_{min} is not guaranteed if a heuristic deadline procedure (such as the backtrack-free version of PCP discussed in Section 2.2. For this reason, we instead define our extended makespan minimization procedure in terms of a more conventional k-iteration search; the "Simple PCP" procedure is applied k times with different common deadlines evenly distributed between d_L and d_U. We refer to this extended procedure below as Multi-PCP.

6. Performance Summary

To assess performance, we carried out computational study following the same experimental design of [21]. A PCB electroplating facility with 5 chemical tanks was assumed. All problems generated consisted of 100 jobs, each with randomly generated routings and tank processing time constraints, and all assumed to be simultaneously available. Since material flow is uni-directional, differences in job routings correspond to which and how many tanks are skipped. Experiments were conducted to evaluate performance along two dimensions relating to facility constraints and operation: first as function of the relative speed of the hoist to mean tank processing time, and second as a function of the degree of flexibility provided by tank processing time constraints. To calibrate results, problems were also solved using the hoist scheduling procedure previously developed by Yih [21], designated below as the "Yih94 algorithm". Both procedures were implemented in C and run on a Sun SPARC 10 workstation.

In configuring Multi-PCP for these experiments, a simpler, "basic algorithm", also defined in [21] and used there as a baseline for comparison, was incorporated to provide the upper bound d_U on the common deadline interval; d_L was obtained by computing the minimum total required processing time (including hoist operations) for each job and taking the maximum. To provide a more computationally competitive alternative to Yih's "real-time" procedure, a simple problem decomposition method[15] was also employed; the input problem was partitioned into subproblems with equal numbers of jobs (10 for these experiments) and solved independently by Multi-PCP, with the results then randomly combined to produce the overall solution - yielding overall solution times of about 100 seconds.

We present only the results obtained from one of the experiments performed, on problem sets designed to vary the ratio $\gamma = \hat{p}^{min}/s$, where $\hat{p}^{min}$ is the mean minimum processing time of tank operations and s is the speed of the hoist in moving between adjacent system locations. Figure 4 summarizes the the performance of Multi-PCP and Yih94 in this experiment. Values plotted for each γ ratio represent the average % improvement over the basic algorithm on 10 randomly generated problems. [2]

Both procedures are seen to generate the largest improvement for values of γ in the range of [10,25], with improvement rates degrading as γ becomes larger or smaller. In the case of Yih94, no improvement is obtained at either of the extreme points tested. Multi-PCP, alternatively, yields an improvement rate of 8% at the smallest γ value, and as γ becomes increasingly larger, its improvement rate stabilizes at about 15%. Across all experiments, Multi-PCP is seen to produce solutions that, on average, are

[2] Details of the full experimental design and all results obtained are reported in [5], and only strengthen the performance comparison.

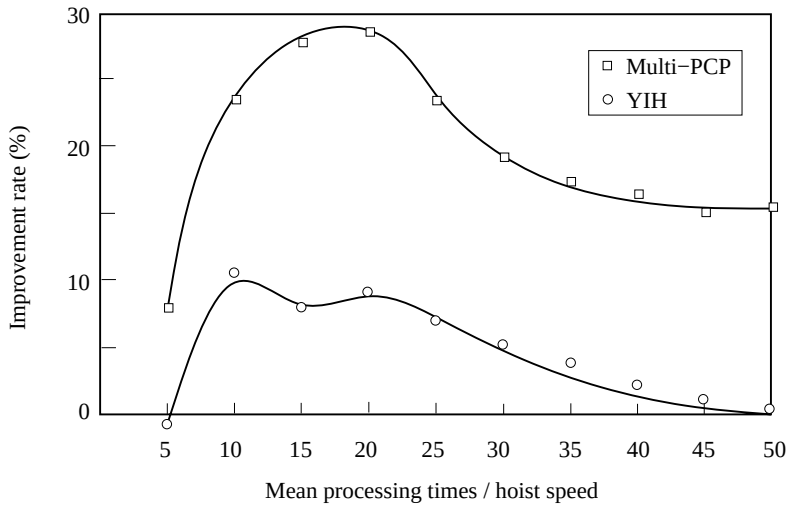

Figure 4. Solution improvement rates for increasing γ ratio

15% better (in relation to the baseline solution) than those obtained with Yih94.

7. Concluding Remarks

The comparative results presented above provide evidence of the viability of CSP scheduling techniques in complex industrial domains. They also illustrate the utility of a disjunctive graph problem formulation, and, conversely, the difficulty of accommodating complex, interacting constraints within scheduling procedures that proceed via explicit commitment to specific start times. In the case of the Yih94 algorithm, scheduling proceeds by incrementally assigning operations to precise execution intervals, and a priori algorithm design decisions dictate when it is productive to extend processing times versus delay job starts and what specific extend or delay decisions should be taken. Most frequently, however, the best decision actually involves some combination of these two alternatives. This tradeoff is not considered by Yih94, and it is not clear how one could extend the approach to generally address this tradeoff in a cost/effective manner. Multi-PCP, alternatively, does not suffer from this limitation. By operating instead in the space of sequencing decisions, this tradeoff is naturally and directly considered. There is no need to design the algorithm to reason explicitly about the types of constraints involved; decision-making can instead be based strictly on their emergent influence on the evolving partial solution.

8. Acknowledgements

This work has been sponsored in part by by the National Aeronautics and Space Administration, under contract NCC 2-531, by the Advanced Research Projects Agency under contract F30602-90-C-0119 and the CMU Robotics Institute.

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 11(26):832–843, 1983.
2. R. Armstrong, L. Lei, and S. Gu. A bounding scheme for deriving the minimal cycle time of a single-transporter n-stage processs with time window constraints. *European Journal of Operational Research*, 78:130–140, 1994.
3. E. Balas. Machine sequencing via disjunctive graphs: An implicit enumerated algorithm. *Operations Research*, 17:941–957, 1969.
4. C. Bell. Maintaining project networks in automated artificial intelligence planning. *Management Science*, 35(10):1192–1214, 1989.
5. C. Cheng and S. Smith. Applying constraint satisfaction techniques to job shop scheduling. Tech. Rep. CMU-RI-TR-95-03, The Robotics Institute, Carnegie Mellon, Jan 1995.
6. C. Cheng and S. F. Smith. Generating feasible schedules under complex metric constraints. In *Proc. 12th National Conf. on Artificial Intelligence*, Seattle, WA., 1994.
7. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
8. J. Erschler, F. Roubellat, and J. P. Vernhes. Finding some essential characteristics of the feasible solutions for a scheduling problem. *Operations Research*, 24:772–782, 1976.
9. J. Erschler, F. Roubellat, and J. P. Vernhes. Characterizing the set of feasible sequences for n jobs to be carried out on a single machine. *European Journal of Operational Research*, 4:189–194, 1980.
10. P. B. Ladkin and R. D. Maddux. On binary constraint networks. Technical report, Kestrel Institute, Palo Alto, CA., 1989.
11. L. Lei and T. Wang. The minimum common-cycle algorithm for cyclic scheduling of two hoists with time window constraints. *Management Science*, 37(12):1629–1639, 1991.
12. I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. In *Proc. 9th National Conf. on Artificial Intelligence*, pages 260–267, Anaheim, CA., 1991.
13. S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.
14. N. Muscettola. Scheduling by iterative partition of bottleneck conflicts. In *Proc. 9th IEEE Conf. on Artificial Intelligence Applications, Orlando, FL.*, 1993.
15. H. N. N. Hirabayashi and N. Nishiyama. A decomposition scheduling method for operating flexible manufacturing systems. *Int. Journal of Prod. Res.*, 32(1):161–178, 1994.
16. L. Phillips and P. Unger. Mathematical programming solution of a hoist scheduling problem. *AIIE Transactions*, 8(2):219–225, 1976.
17. N. Sadeh. Look-ahead techniques for micro-opportunistic job shop scheduling. Tech. Rep. CMU-CS-91-102, School of Computer Science, Carnegie Mellon Univ., 1991.
18. G. Shapiro and H. Nuttle. Hoist scheduling for a pcb electroplating facility. *IIE Transactions*, 20(2):157–167, 1988.
19. S. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction schedul-

ing. In *Proc. 11th National Conf. on Artificial Intelligence, Wash DC.*, pages 139 – 144, 1993.

20. M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. 4th Nat. Conf. on Artificial Intelligence, Philadelphia, PA.*, pages 377–382, 1986.
21. Y. Yih. An algorithm for hoist scheduling problems. *International Journal of Production Research*, 32(3):501–516, 1994.
22. Y. Yih, T. Liang, and H.Moskowitz. Robot scheduling in a circuit board production line. *IIE Transactions*, 25(2):26–33, 1993.
23. Y. Yih and A. Thesen. Semi-markov decision models for real-time scheduling. *International Journal of Production Research*, 29(11):2331–2346, 1991.