A Constraint Satisfaction Approach to Makespan Scheduling*

Cheng-Chung Cheng and Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
sfs@cs.cmu.edu

Abstract

In this paper, we consider the application a constraint satisfaction problem solving (CSP) framework recently developed for deadline scheduling to more commonly studied problems of schedule optimization. Our hypothesis is two-fold: (1) that CSP scheduling techniques can provide a basis for developing high-performance approximate solution procedures in optimization contexts, and (2) that the representational assumptions underlying CSP models allow these procedures to naturally accommodate the idiosyncratic constraints that complicate most real-world applications. We focus specifically on the objective criterion of makespan minimization, which has received the most attention within the job shop scheduling literature. We define an extended solution procedure somewhat unconventionally by reformulating the makespan problem as one of solving a series of different but related deadline scheduling problems, and embedding a simple CSP procedure as the subproblem solver. We summarize results of an empirical evaluation of our procedure performed on a range of previously studied benchmark problems. Our procedure is found to provide strong cost/performance, producing solutions competitive with those obtained using recently reported shifting bottleneck search procedures at reduced computational expense. To demonstrate generality, we also consider application of our procedure to a more complicated, multi-product hoist scheduling problem. With only minor adjustments, our procedure is found to significantly outperform previously published procedures for solving this problem across a range of input assumptions.

Introduction

Constraint satisfaction problem solving (CSP) models and heuristics have increasingly been investigated as a means for solving scheduling problems (Cheng & Smith 1994; Minton *et al.* 1992; Muscettola 1993; Sadeh 1991; Smith & Cheng 1993). Much of this work has focused on variations of the job shop deadline problem, and a number of effective procedures have been developed in this context. A job shop deadline problem involves synchronization of the production of n jobs in a facility with m machines, where (1) each job j requires execution of a sequence of operations within a time interval specified by its ready time r_j and deadline d_j, and (2) each operation O_i requires exclusive use of a designated machine M_i for a specified amount of processing time p_i. The objective is to determine a schedule for production that satisfies all temporal and resource capacity constraints.

Our interest in this paper is in investigating the applicability of CSP procedures for deadline scheduling to problems of schedule optimization. We focus specifically on the problem of minimizing schedule makespan, which has received the most attention in the Operations Research (OR) scheduling literature. In this problem, jobs do not have individual deadlines; the objective is instead to complete all jobs in the least amount of time possible. Thus, the problem cannot be formulated strictly as a CSP.

We define a procedure for makespan minimization by reformulating the problem as a series of distinct but related deadline scheduling subproblems, and searching for the smallest feasible "common deadline" for all jobs. This allows us to take direct advantage of a previously developed deadline scheduling procedure as the core subproblem solver. In the sections below, we first summarize the embedded CSP scheduling procedure (called PCP), then define the extended procedure (Multi-PCP) for finding the minimum common deadline, and finally discuss two computational studies.

PCP

We take as our starting point a state-of-the-art heuristic procedure for deadline scheduling called Precedence Constraint Posting (PCP) (Smith & Cheng 1993; Cheng & Smith 1994). There are different ways to formulate the deadline scheduling problem as a CSP. In PCP, the problem is formulated as one of establishing sequencing constraints between those operations contending for the same resource. In CSP terms, we have a set of decision variables $Ordering_{ij}$ for each (O_i, O_j)

*This research has been sponsored in part by the National Aeronautics and Space Administration, under contract NCC 2-531, by the Advanced Research Projects Agency under contracts F30602-90-C-0119 and F30602-95-1-0018, and the CMU Robotics Institute.

such that $M_i = M_j$, which can take on two possible values: $O_i \prec O_j$ or $O_j \prec O_i$.

Problem Representation

The PCP scheduling model can be formalized more precisely as a type of *general temporal constraint network (GTCN)* (Meiri 1991). 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 (Meiri 1991) includes Allen's 13 basic temporal relations (Allen 1983): *before, after, meets, met-by, overlaps, overlapped-by, during, contains, starts, started-by, finishes, finished-by,* and *equal.* For convenience, we additionally include the relations *before-or-meets* and *after-or-met-by*, which represent the union of relation pairs (*before, meets*) and (*after, met-by*) respectively (Bell 1989).
2. point to point constraints - The relations identified in (Vilain & Kautz 1986), denoted by the set $\{<, =, >\}$, are allowable here.
3. point to interval or interval to point constraints - In this case, the 10 relations defined in (Ladkin & Maddux 1989) 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}.

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.

An enumerative scheme for solving a GTCN is given in (Meiri 1991). Let a *labeling* of a general temporal constraint network, T, be a selection of a single disjunct (relation or interval) from each constraint specified in T. In the graph of Figure 1 there are 4 possible labelings, owing to the $\{$*before-or-meets, after-or-met-by*$\}$ relation sets introduced to avoid resource contention between operation pairs (O_1, O_2) and (O_3, O_4). Since any basic qualitative constraint can be translated into at most four metric constraints (Kautz & Ladkin 1991) (e.g., O_i *before-or-meets* O_j translates to $et_i \leq st_j$), any labeling of T defines a Simple Temporal Problem (STP) network - a metric network containing only single interval constraints (Dechter, Meiri, & Pearl 1991). T will be consistent if and only if there exists a labeling whose associated STP is consistent.

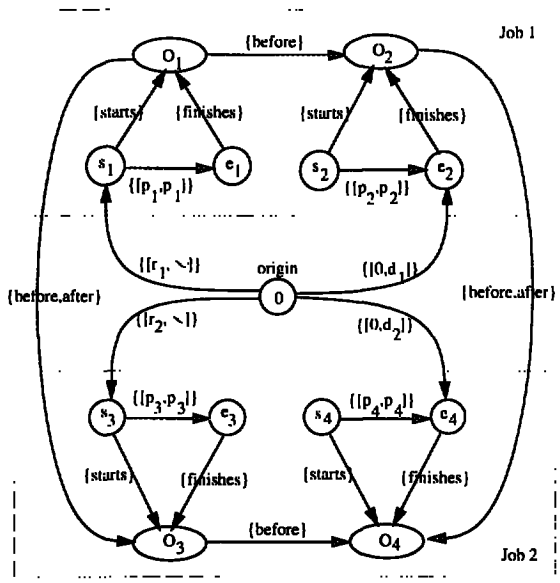

Figure 1: Constraint Graph for simple 2 job, 2 machine problem

For any STP network, we can define a directed edge-weighted graph of time points, G_d, called a *distance graph*. An STP is consistent if and only if the corresponding distance-graph G_d has no negative weight cycles. The *minimal network* of the STP can be specified by a complete directed graph, called the *d-graph*, where each edge, $i \rightarrow j$, is labeled by the shortest path length, sp_{ij}, from point i to point j in G_d (Dechter, Meiri, & Pearl 1991). An STP network can be solved in $O(n^3)$ time by the Floyd-Warshall's all-pairs shortest-paths algorithm, where n is the number of variables in the STP network.

Thus, a simple, complete procedure for solving a GTCN is to enumerate all labelings, solve each corresponding STP and combine results. We can increase the efficiency of this enumeration procedure by running a backtracking search over a *meta-CSP* network, whose variables correspond to arcs in the GTCN that can be labeled in more than one way and whose domains are simply the set of possible labelings. In the case of the deadline scheduling problem, this leads to the set of

decision variables $V = \{Ordering_{ij}\}$ previously identified, and a worst case complexity of $\mathcal{O}(n^3 2^{|V|})$.

Basic Solution Procedure

The PCP scheduling model (Smith & Cheng 1993; Cheng & Smith 1994) augments this basic backtracking search procedure to incorporate simple analysis of the temporal flexibility associated with each sequencing decision that must be made. This analysis is utilized in two ways: (1) to specify dominance conditions that allow identification of unconditional decisions and early search space pruning, and (2) to provide heuristic guidance for variable and value ordering (i.e., decisions as to what variable to assign next and what value to assign).

Specification and use of dominance conditions in PCP derives directly from the concept of *Constraint-Based Analysis* (CBA) originally developed in (Erschler, Roubellat, & Vernhes 1976; 1980). This work utilized calculations of the temporal slack associated with an unordered operation pair to distinguish among cases where neither ordering alternative, just one ordering alternative, or either alternative remains feasible. For example, if $slack(O_i \prec O_j) = lft_j - est_i - (p_i + p_j) < 0$ then O_i cannot be sequenced before O_j. These conditions are applied to detect and post any "forced" sequencing constraints at each step of the search, and to detect inconsistent solution states.

In (Cheng & Smith 1994), 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 d-graph associated with the current 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.

We note that the "slack-based" dominance conditions of (Erschler, Roubellat, & Vernhes 1976) represent a special case of the above conditions; under classical job shop scheduling assumptions (i.e., fixed processing times, simple job precedence constraints) $slack(O_i \prec O_j) = sp_{ij}$. However, many practical scheduling problems require satisfaction of more complex temporal constraints (e.g., bounded delays between job steps, minimum and maximum processing time constraints, inter-job synchronization). Under such more complex modeling assumptions, shortest path information provides stronger dominance criteria.

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). Intuitively, in situations where several $Ordering_{ij}$ decisions remain to be made, each with both possibilities still open, we would like to focus attention on the decision that has the least amount of sequencing flexibility. Conversely, in making the selected ordering decision, we intuitively prefer the ordering relation that leaves the search with the most degrees of freedom.

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. Consider two ordering decisions $Ordering_{ij}$ with associated shortest path lengths $sp_{ij} = 3$ and $sp_{ji} = 100$, and $Ordering_{kl}$ with $sp_{kl} = 4$ and $sp_{lk} = 4$. In this case, there are only limited possibilities for feasibly resolving $Ordering_{kl}$ and deferring this decision may well eliminate them, while a feasible assignment to $Ordering_{ij}$ is not really in any jeopardy.

To hedge against these situations, PCP instead 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 $S = \min\{sp_{ij}, sp_{ji}\}/\max\{sp_{ij}, sp_{ji}\}$ 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.

A More Efficient, Approximate Procedure

The dominance conditions and variable/value ordering heuristics that distinguish the basic PCP procedure do not, of course, change the exponential worst case behavior of the backtracking search required to guarantee completeness. To provide a more computationally efficient procedure for embedded use in solving makespan minimization problems, we introduce a backtrack-free variant of the basic PCP procedure. This approximate solution procedure, referred to as "One-Pass PCP", is defined as follows. Whenever an ordering decision is recognized as Case 3 (i.e., no feasible ordering), the search does not backtrack, but instead the unresolvable decision is set aside, and the search is allowed to

proceed with other, still resolvable ordering decisions. Once all feasibly resolvable decisions have been made, the set U of unresolvable (Case 3) decisions is then reconsidered. For each $Ordering_{ij}$ in U, deadlines d_i and d_j are relaxed (increased) by $|max(sp_{ij}, sp_{ji})|$ and the corresponding precedence relation (which is now feasible) is posted. This procedure thus always produces a solution, albeit one that may not satisfy all original problem constraints. Its worst-case time complexity is $\mathcal{O}(n^3|V|)$, where $|V|$ is the number of ordering decisions that must be made. In the next section, we define an extended procedure for makespan minimization that incorporates One-Pass PCP as its subproblem solver.

MULTI-PCP

Our approach to designing an extended procedure for makespan minimization is motivated by the concept of problem duality exploited in the MULTIFIT algorithm (Coffman, Garey, & Johnson 1978) in the context of multiprocessor scheduling. 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 algorithm for optimally 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.

There is a complication, however, in utilizing this binary search procedure in conjunction with a heuristic deadline scheduling procedure like "One-Pass PCP". The search may fail to yield the best solution if the deadline scheduling procedure does not ensure *monotonicity* in solution results across an interval of common deadlines. This property implies that if a feasible solution cannot be found for a given common deadline d_1, then a solution will also not be found for any common deadline $d_2 < d_1$, and likewise if a solution is found for a given d_1, then a solution will also be found for any $d_2 > d_1$. It is not difficult to construct examples which demonstrate that One-Pass PCP does not possess this property, and consequently the assumptions underlying use of binary search are no longer valid. For this reason, we instead define our extended makespan minimization procedure, which we will refer to as Multi-PCP, in terms of a more conventional k-iteration search; One-Pass PCP is applied k times with different common deadlines evenly distributed between d_L and d_U. While k-iteration search obviously also provides no guarantee of finding the optimal solution, empirical analysis has indicated that, with proper selection of k, use of k-iteration search leads to consistently better makespan minimization performance.

The only remaining issue concerns initial establishment of upper and lower bounds on d_{min}. A lower bound d_L is provided by the procedure originally described in (Florian, Trepant, & McMahon 1971), where each machine is sequenced independently in order of earliest operation start times and the maximum job completion time is then selected. An upper bound d_U can be obtained through application of one or more priority dispatch rules. In the experiments reported below, a set of six well-known priority rules - SPT, LPT, LFT, EFT, MOR, and LOR - were applied, taking the best makespan generated as d_U [Please refer to (Panwalker & Iskander 1977) for details concerning these priority rules.] For all runs, the bound k on the number of iterations performed was set to 8.

Benchmark Problem Results

We applied the above defined Multi-PCP procedure to two sets of previously studied benchmark problems within the Operations Research (OR) literature. The first ("small") problem set consists of 39 job shop problems with sizes varying from 6-job by 6-machine to 15-job by 15-machine. The first three problems, Mt06, Mt10, and Mt20, are the long standing problems of (Fisher & Thompson 1963). The remainder are taken from the 40 problems originally created by (Lawrence 1984); of these 40 problems, we include only the 36 problems for which optimal solutions have been obtained. The second ("large") set of benchmark problems, are the problems more recently defined by (Taillard 1993). This set consists of 80 larger job shop problems with sizes ranging from 15-job by 15-machine to 100-job by 20-machine. For each problem in this set, Taillard reported the "best solution" obtained with a tabu search procedure that was run for extended time intervals.

We take as a principal comparative base, the shifting bottleneck family of procedures (Adams, Balas, & Zawack 1988; Balas, Lenstra, & Vazacopoulos 1993), which has produced performance comparison standards within the OR makespan scheduling literature. The shifting bottleneck procedures - SB1, SB3 and SB4 - provide a series of increasingly more accurate approximate procedures for makespan minimization at increasingly greater computational expense. We com-

Table 1: Mean % deviation from optimal solution for Multi-PCP, SB1, SB3, and SB4 across small benchmark problem categories

Job x Machine	Multi-PCP	SB1	SB3	SB4
mt06	0.00	7.27	0.00	0.00
mt10	2.04	2.37	5.48	1.08
mt20	2.32	5.41	2.92	2.92
10 x 5	1.47	1.59	1.44	1.44
15 x 5	0.00	0.00	0.00	0.00
20 x 5	0.00	0.00	0.00	0.00
10 x 10	2.44	4.94	3.16	2.25
15 x 10	3.78	6.34	2.72	2.72
20 x 10	3.12	6.57	1.37	0.96
30 x 10	0.30	0.00	0.00	0.00
15 x 15	4.15	6.16	3.09	3.01
All	1.72	3.01	1.51	1.24

Table 2: Mean CPU time (in seconds) for Multi-PCP, SB1, SB3, and SB4 across small benchmark problem categories

Job x Machine	Multi-PCP	SB1	SB3	SB4
mt06	0.05	0.12	0.78	1.45
mt10	0.38	0.72	2.21	7.76
mt20	1.38	0.28	1.99	3.62
10 x 5	0.13	0.12	0.40	0.56
15 x 5	0.22	0.15	0.12	0.12
20 x 5	0.07	0.16	0.14	0.14
10 x 10	0.26	0.67	1.61	3.20
15 x 10	1.04	1.20	3.41	5.74
20 x 10	2.51	1.61	3.86	7.50
30 x 10	4.85	2.58	4.13	4.13
15 x 15	1.29	4.55	13.34	26.79
Average	1.19	1.21	2.96	5.29

pare the performance of each of these procedures and Multi-PCP in terms of two measures: % deviation from the optimal solution (or % deviation from the best tabu search solution in the case of the large problem set) and amount of computation time required. Results for SB1 were obtained on a Sun SPARC 10 workstation using an implementation kindly provided to us by Applegate and Cook (for detail please see (Applegate & Cook 1991)). Results for SB3 and SB4 were taken from (Balas, Lenstra, & Vazacopoulos 1993), with the reported Sun SPARC 330 computation times translated to reflect expected performance on a SPARC 10. Multi-PCP computation times were also obtained on a SPARC 10. All procedures considered were implemented in C. Since SB3 and SB4 results have not been reported for the large problem set, comparison here is restricted to Multi-PCP and SB1.

Table 1 summarizes the performance results obtained on the small benchmark problem set (with results aggregated according to problem size for the Lawrence problems). Associated computation times are given in Table 2. Computation times were found to be identical for both Multi-PCP configurations at the level of precision reported and are thus listed only once. [Detailed results for each individual problem for both problem sets are reported in (Cheng & Smith 1995).]

The makespan minimization performance of Multi-PCP on the small problem set falls within the performance continuum defined by the shifting bottleneck procedures. On average, Multi-PCP is seen to perform better than SB1 and very close to SB3, with SB4 yielding the best overall makespan performance. Relative performance was found to vary across different problem subsets. On the three classic Fisher and Thompson problems, Multi-PCP found equivalent or better solutions than both SB1 and SB3 in all cases, and failed to match the performance of SB4 in just one case. There is little difference in performance on the very small, 6-machine problems; all procedures produce optimal or near optimal solutions in these problem categories. The results on the larger, 10-machine problem categories reveal perhaps the most significant comparative performance trend. For problems with low ratios of number of jobs to number of machines, Multi-PCP exhibits its strongest comparative performance. In the case of the 10x10 problem category, Multi-PCP performed better on average than both SB1 and SB3, and very close to SB4. Conversely, Multi-PCP was found to be less effective (comparatively) on problems with high job to machine ratios. On the 30x10 problems (which turn out to be the easiest 10-machine problems for all procedures), all three shifting bottleneck procedures were able to obtain optimal solutions, whereas Multi-PCP failed to find the optimum for 2 of the 5 problems in this category. Computationally, Multi-PCP's solution times on this problem set are seen to be comparable overall to those of SB1.

Table 3 extends the performance comparison of Multi-PCP and SB1 to the larger problem set of Taillard. Corresponding average computation times by problem category are given in Table 4. Ignoring the scalability problems encountered with the tested SB1 implementation (it couldn't solve some problems due to memory problems), the results at larger problem sizes make much more explicit the comparative performance trends observed at the 10-machine problem level. Multi-PCP is seen to consistently outperform SB1 at low job-to-machine ratios, while the inverse is true at high job-to-machine ratios. Both procedures achieve increasingly better solutions at higher job-machine ratios (consistent with Taillard's observation that these problems are easier), but in no cases does either Multi-PCP or SB1 achieve the best solutions generated by extended Tabu search.

Examination of relative computational costs indi-

Table 3: % deviation from the best solution for Multi-PCP and SB1 across large benchmark problem categories

Job x Machine	Multi-PCP		Multi-PCP w/ Relax		SB1	
	mean	σ	mean	σ	mean	σ
15 x 15	5.74	0.74	5.57	0.78	9.00	2.05
20 x 15	7.52	1.82	7.27	1.49	10.15	2.14
30 x 15	9.88	2.57	9.65	2.47	8.38	3.06
50 x 15*	7.39	2.31	7.21	2.26	2.66†	1.57
20 x 20	7.60	1.54	7.33	1.33	9.98	2.29
30 x 20	11.76	2.43	11.64	2.37	13.05	2.55
50 x 20*	8.77	0.96	8.34	1.06	5.33†	1.82
100 x 20	4.88	1.41	4.88	1.41	- ‡	-
All*	8.38	1.72	8.14	1.65	8.36	2.21

† SB1 able to solve nine out of ten problems.
‡ SB1 unable to solve any of the 100x20 problems.
* Average performance is measured with respect to problems solved by both procedures.

Table 4: CPU seconds for Multi-PCP and SB1 on the large benchmark problems

Job x Machine	Multi-PCP		SB1	
	mean	σ	mean	σ
15 x 15	1.20	0.08	5.10	1.39
20 x 15	3.42	0.33	7.63	1.06
30 x 15	11.90	0.85	14.68	1.72
50 x 15	68.11	7.12	141.33	104.84
20 x 20	3.73	0.32	15.64	2.91
30 x 20	15.51	0.77	31.58	4.03
50 x 20	94.90	6.28	165.83	94.75
100 x 20	857.36	38.43	-	-

cates some additional scalability trends and tradeoffs. Multi-PCP was found to consistently produce solutions in less computation time than SB1; the largest differential (roughly 4 times as fast) was observed in the problem categories with the smallest job-to-machine ratios, and, on average, Multi-PCP obtained solutions in about half as much CPU time as SB1. Multi-PCP was also found to be much more predictable with respect to computational cost. The variance in Multi-PCP solution times across all problem categories was extremely low in comparison to SB1.

The Hoist Scheduling Problem

The above study relates the performance of Multi-PCP to state-of-the-art makespan minimization procedures; perhaps somewhat surprising, it shows that Multi-PCP's use of a CSP scheduling model in conjunction with fairly simple search control heuristics yields respectable performance (although certainly not outperforming all previously reported results). A complementary consideration is its broader applicability to more idiosyncratic problem formulations.

To demonstrate generality, we consider application of Multi-PCP to a less-structured makespan minimization problem: the multi-product version of the hoist scheduling problem(Yih 1994). The 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 (e.g., single product only, multiple products but no tank skipping). To our best knowledge, only (Yih 1994) has reported procedures for solving the general hoist scheduling problem defined above.

Extensions

The GTCN formalism introduced in Section requires only slight extension to model the hoist scheduling problem. The only constraints that are not directly formulatable are those relating to synchronization of competing hoist (or material movement) operations; in this case, basic qualitative relations are insufficient, as they do not allow accounting of the "setup" time that may be required to position the hoist at the loading location. To overcome this limitation, we extend our representation of qualitative constraints to optionally include a metric quantifier. For purposes here, it is sufficient to include only the following two extended relations: *before-or-meets[lagtime]* and *after-or-met-by[lagtime]*, where *lagtime* ≥ 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*$[h_{ij}]$ O_j implies $et_i + h_{ij} \leq st_j$. 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 $h_{ji} = s*|destination_j - origin_i|$. [see (Cheng & Smith 1995) for a complete discussion of how other aspects of the hoist problem are modeled and a simple example].

The presence of sequence-dependent setups also impacts the variable and value ordering heuristics utilized with the base PCP procedure. Recall from Section , that these heuristics rely on shortest path lengths as a basic indicator of sequencing flexibility. In essence, the shortest path from et_i to st_j for operations O_i and O_j, designated sp_{ij}, indicates the current maximum feasible separation between these two points. However, shortest path lengths provide only a partial (distorted) view of maximum separation if sequence-dependent setup delays are required. To sharpen the heuristics, we generalize the basic measure of flexibility in PCP to incorporate sequence-dependent lag times. Assume h_{ij} to be the lag time required if O_i is processed before O_j, and h_{ji} be the lag time required if O_j is processed before O_i (i.e., the constraint specified in the network is O_i {*before-or-meets*$[h_{ij}]$, *after-or-met-by*$[h_{ji}]$} O_j). We revise the dominance conditions and search control heuristics specified in Section by simply substituting the extended calculation $(sp_{ij} - h_{ij})$ for sp_{ij} and, likewise, substituting $(sp_{ji} - h_{ji})$ for sp_{ji}. Note that these revised definitions continue to accommodate the basic {*before, after*} relation (in which case, h_{ij} and h_{ji} are both set to the smallest possible temporal increment), as well as the basic {*before-or-meets, after-or-met-by*} relation set (where $h_{ij}, h_{ji} = 0$).

Results

To assess performance, we carried out computational study following the same experimental design of (Yih 1994). 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 unidirectional, 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 (Yih 1994), 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", used in (Yih 1994) 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(N. Hirabayashi & Nishiyama 1994) 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.

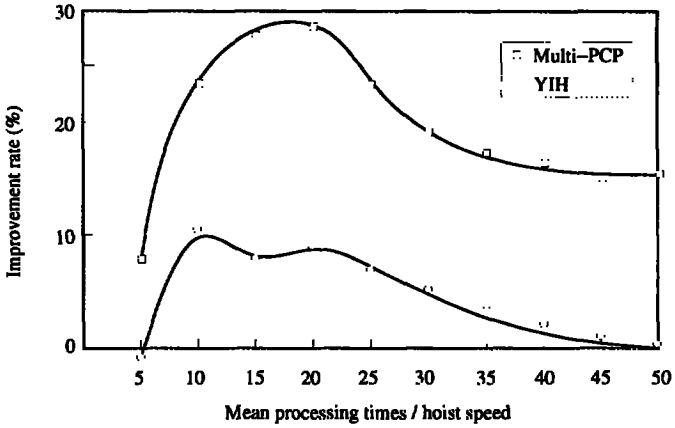

Figure 2: Solution improvement for increasing ratio of mean processing time / hoist speed

We present only the results obtained from one of the experiments performed, on problem sets designed to vary the ratio $\gamma = \bar{p}^{min}/s$, where $\bar{p}^{min}$ is the mean minimum processing time of tank operations and s is the speed of the hoist in moving between adjacent system locations. Figure 2 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.

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

Details of the full experimental design and all results

obtained are reported in (Cheng & Smith 1995), and only strengthen the performance comparison.

Concluding Remarks

We have described a procedure for makespan scheduling based on formulation of the problem as a series of CSPs and iterative application of a CSP scheduling procedure with fairly simple search control heuristics. It was shown to produce strong performance in relation to shifting bottleneck procedures on benchmark scheduling problems. Perhaps more significant however, is the procedure's generality. Real-world applications are often complicated by additional temporal synchronization and resource usage constraints, and solution procedures which rely on problem structure that is peculiar to the canonical job shop problem formulation are of little use in such contexts. CSP scheduling models like Multi-PCP, alternatively, are based on very general representational assumptions and naturally extend to accommodate richer problem formulations.

References

Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391 - 401.

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 11(26):832–843.

Applegate, D., and Cook, W. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing* 3(2):149 - 156.

Balas, E.; Lenstra, J. K.; and Vazacopoulos, A. 1993. The one machine problem with delayed precedence constraints and its use in job shop scheduling. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, #MSRR-589(R).

Bell, C. 1989. Maintaining project networks in automated artificial intelligence planning. *Management Science* 35(10):1192-1214.

Cheng, C., and Smith, S. F. 1994. Generating feasible schedules under complex metric constraints. In *Proc. 12th Nat. Conf. on AI (AAAI-94), Seattle, WA.*

Cheng, C., and Smith, S. 1995. Applying constraint satisfaction techniques to job shop scheduling. Technical Report CMU-RI-TR-95-03, The Robotics Institute, Carnegie Mellon University.

Coffman, E. G.; Garey, M. R.; and Johnson, D. S. 1978. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.* 7:1–17.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Erschler, J.; Roubellat, F.; and Vernhes, J. P. 1976. Finding some essential characteristics of the feasible solutions for a scheduling problem. *Operations Research* 24:772–782.

Erschler, J.; Roubellat, F.; and Vernhes, J. P. 1980. 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.

Fisher, H., and Thompson, G. L. 1963. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling.* J.F. Muth, G. Thompson (eds), Prentice-Hall, Englewood Cliffs, NJ.

Florian, M.; Trepant, P.; and McMahon, G. B. 1971. An implicit enumeration algorithm for the machine sequencing problem. *Management Science* 17(12):B-782-B-792.

Kautz, H., and Ladkin, P. B. 1991. Integrating metric and qualitative temporal reasoning. In *Proc. 9th Nat. Conf. on AI (AAAI-91), Anaheim, CA.*, 241-246.

Ladkin, P. B., and Maddux, R. D. 1989. On binary constraint networks. Technical report, Kestrel Institute, Palo Alto, CA.

Lawrence, S. 1984. Resource constraint project scheduling: An experimental investigation of heuristic scheduling techniques. Technical report, Graduate School of Ind. Admin., Carnegie Mellon University.

Meiri, I. 1991. Combining qualitative and quantitative constraints in temporal reasoning. In *Proc. 9th Nat. Conf. on AI (AAAI-91), Anaheim*, 260-267.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58:161–205.

Muscettola, N. 1993. Scheduling by iterative partition of bottleneck conflicts. In *Proc. 9th IEEE Conf. on AI Applications, Orlando, FL.*

N. Hirabayashi, H. N., and Nishiyama, N. 1994. A decomposition scheduling method for operating flexible manufacturing systems. *International Journal of Production Research* 32(1):161–178.

Panwalker, S. S., and Iskander, W. 1977. A survey of scheduling rules. *Operations Research* 25:45 61.

Sadeh, N. 1991. Look-ahead techniques for micro-opportunistic job shop scheduling. Technical report, CMU-CS-91-102, School of Computer Science, Carnegie Mellon University.

Smith, F. S., and Cheng, C. 1993. Slack-based heuristics for constraint satisfaction scheduling. In *Proc. 11th Nat. Conf. on AI (AAAI-93), Wash DC*, 139 - 144.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64:278 - 285.

Vilain, M., and Kautz, H. 1986. Constraint propagation algorithms for temporal reasoning. In *Proc. 4th Nat. Conf. on AI (AAAI-86), Phila, PA.*, 377–382.

Yih, Y. 1994. An algorithm for hoist scheduling problems. *International Journal of Production Research* 32(3):501-516.