Annals of Mathematics and Artificial Intelligence **40:** 283–301, 2004.
© 2004 *Kluwer Academic Publishers. Printed in the Netherlands.*

Solving hierarchical constraints over finite domains with local search

Martin Henz [a], Roland H.C. Yap [a], Yun Fong Lim [b], Seet Chong Lua [a], J. Paul Walser [c] and Xiao Ping Shi [d]

[a] *School of Computing, National University of Singapore*
E-mail: {henz, ryap, luasc}@comp.nus.edu.sg
[b] *School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205, USA*
E-mail: yflim@isye.gatech.edu
[c] *i2 Technologies, Munich, Germany*
E-mail: walser@i2.com
[d] *Motorola, Singapore*
E-mail: xiaoping.shi@motorola.com

Many real world problems have requirements and constraints which conflict with each other. One approach for dealing with such over-constrained problems is with constraint hierarchies. In the constraint hierarchy framework, constraints are classified into ranks, and appropriate solutions are selected using a comparator which takes into account the constraints and their ranks. In this paper, we present a local search solution to solving hierarchical constraint problems over finite domains (HCPs). This is an extension of local search for over-constrained integer programs WSAT(OIP) to constraint hierarchies and general finite domain constraints.

The motivation for this work arose from solving large airport gate allocation problems. We show how gate allocation problems can be formulated as HCPs using typical gate allocation constraints. Using the gate allocation benchmarks, we investigate how constraint heirarchy selection strategies and the problem formulation using two models: a 0–1 linear constraint hierarchy model and a nonlinear finite domain constraint hierarchy model.

Keywords: hierarchical constraints, finite domain constraints, over-constrained problems, airport gate allocation

1. Introduction

The goal in solving constraint satisfaction problems (CSPs) is to find a solution which satisfies all given constraints. Approaches to CSPs such as constraint programming have proven successful for a wide range of problems. However, many real world problems cannot be represented directly as CSPs, because there may either be conflicting constraints, or there may be difficulties in defining the problem constraints precisely to allow direct solution. Problems with such features are typically over-constrained, and hence by definition it is not possible to satisfy all given constraints.

There are two general approaches to dealing with over-constrained problems. Constraint hierarchies [2] (HCLP [14] exemplifies this approach) addresses the over-constrainedness by resolving the conflict using preferences on the importance of some constraints and particular solutions. The other approach exemplified by PCSP (Partial CSP) [4] is to relax the problem definition so that it is consistent. Some even more general approaches are semiring-based CSPs and valued CSPs [1] which are beyond the scope of this paper. A good collection of papers on approaches for over-constrained problems can be found in [7].

The work here was motivated by a project on airport passenger gate allocation, where we were confronted with user-specified constraint ranks as the underlying intuitive model used by the human experts. Thus we focused on solving the gate allocation problem using constraint hierarchies.

Section 2 presents the framework for constraint hierarchies that is used throughout this paper. The size of the gate allocation problems makes it impossible to reach globally optimal solutions using complete search techniques (both finite domain search as well as integer optimization are intractable, see section 4 for a discussion). Thus we worked on local search as a more computationaly feasible approach. The local search algorithm WSAT(OIP) [11,13] is a walk search algorithm designed for solving over-constrained linear integer programs and provides a good starting point for this work. In section 3, we extend WSAT(OIP) in two directions. Firstly, we allow arbitrary finite domain constraints as opposed to linear ones. Secondly, we exploit the structure of constraint hierarchies during the search. We propose several variants of the algorithm to handle constraint hierarchies. Section 4 describes the airport gate allocation problem. This is the experimental benchmark which we use to investigate hierarchical local search. We evaluate the performance of the local search algorithm on two models for gate allocation: a 0/1 model that uses only linear constraints and a finite domain model that in addition to arithmetic constraints some other symbolic constraints. We then investigate several variations of the local search algorithm.

2. Constraint hierarchies

In this section, we describe the framework of constraint hierarchies following [2].

Let $\mathcal{X}$ be a set of variables. Each variable $x \in \mathcal{X}$ ranges over a set $\mathcal{D}_x$ which denotes the finite set of values that x can take. A k-ary constraint c over variables $x_1, \ldots, x_k$ is a relation over $\mathcal{D}_{x_1} \times \cdots \times \mathcal{D}_{x_k}$. The constraints are organized in a vector $\mathcal{C}_H$ of the form $\langle C_0, C_1, \ldots, C_n\rangle$, where for each i, $0 \leqslant i \leqslant n$, C_i is a multiset containing constraints of rank i.

Each constraint rank represents the importance of that multiset of constraints, C_i. Constraints in rank 0, C_0, are distinguished as they denote *required constraints* (or *hard constraints*), which *must* be satisfied. The constraints in $C_1, C_2, \ldots, C_n$ denote *preferential constraints* (or *soft constraints*), which need not necessarily be all satisfied. These soft constraints range from the strongest rank C_1 to the weakest rank C_n.

A *valuation* θ is a function that maps the all variables in $\mathcal{X}$ to elements in the domain $\mathcal{D}$. The set of solutions $S_0 = \{\theta \mid \forall c \in C_0,\ c\theta \text{ holds}\}$ are those valuations that satisfy the required constraints, i.e. for every constraint c over $x_1, \ldots, x_k$, $(x_1\theta, \ldots, x_k\theta) \in c$. In order to take into account the other constraints in the hierarchy, a partial ordering *better*, which is called a *comparator*, is used. The comparator $\mathit{better}(\sigma, \theta, \mathcal{C}_H)$ is true when valuation σ is preferred to θ in the context of the constraint hierarchy $\mathcal{C}_H$. The *solution set* for the constraint hierarchy, $\mathcal{C}_H$, which is denoted as S_{better} is defined as those solutions that are optimal respect to *better* as follows:

$$S_{better} = \big\{\theta \in S_0 \mid \forall\sigma \in S_0.\neg \mathit{better}(\sigma, \theta, \mathcal{C}_H)\big\}.$$

There are many suitable choices for comparators in a constraint hierarchy (see [2]). In this paper, we concentrate on the following weighted-sum-better comparator which is the most relevant to the gate allocation problem described in section 4. However, the local search techniques here should be applicable to other comparators.

Since a constraint need not be satisfied, the result of a valuation $c\theta$ can be describe in terms of an *error function* $e(c, \theta)$ which returns a non-negative real number indicating the degree of violation of constraint c under the valuation θ. We require that the error function e has the property that $e(c, \theta) = 0$ iff c, θ holds, and $e(c, \theta) > 0$ otherwise. Where it is clear, we will sometimes denote the error function simply as $e(c)$. A *trivial error function* returns 0 when θ satisfies c and 1 if not. The weighted-sum-better comparator can be defined as follows:

$$\begin{aligned}
&\mathit{weighted\text{-}sum\text{-}better}(\theta, \sigma) \equiv \exists k.\ 1 \leqslant k \leqslant n \text{ such that}\\
&\quad \forall i \in \{1, \ldots, k-1\}.\ \mathit{weighted\text{-}sum}(\theta, C_i) = \mathit{weighted\text{-}sum}(\sigma, C_i)\\
&\quad \wedge \mathit{weighted\text{-}sum}(\theta, C_k) < \mathit{weighted\text{-}sum}(\sigma, C_k).
\end{aligned}$$

The function *weighted-sum* requires for each constraint c the definition of a weight $w(c)$, a positive real number. Using weights for constraints, the function *weighted-sum* combines the error values of constraints in a given rank into a single number as follows.

$$\mathit{weighted\text{-}sum}(\theta, C_i) \equiv \sum_{c \in C_i} w(c)e(c\theta).$$

Given a constraint hierarchy defined by the tuple $\langle \mathcal{X}, \mathcal{C}_H, e, w\rangle$, the goal is to find solutions in $S_{weighted\text{-}sum\text{-}better}$, where *weighted-sum-better* uses the given error function e and weight function w. We will mostly be using nontrivial error functions in the benchmark applications.

3. Hierarchical local search

While constraint hierarchies provide an expressive framework for defining over-constrained problems, applying it directly to large combinatorial problems would be computationally very expensive. Preliminary experiments with finite domain solvers indicated that for the airport gate allocation problem that it was only feasible to solve small problems. We are also not aware of any application where constraint heirarchies are used to solve large combinatorial problems. Thus we turned to investigate local search. Local search techniques such as randomized search, simulated annealing, genetic algorithms, artificial neural networks, etc. have shown to be quite effective in solving large combinatorial problems.

One local search technique which has shown to be effective for finite domain problems is WSAT(OIP) [11,13]. OIP here refers to *overconstrained integer programs*. WSAT(OIP) focuses on OIPs consisting of hard and soft linear inequality constraints which can be defined as

$$Ax \geqslant b, \qquad Dx \leqslant e(soft), \quad x \in \mathcal{D},$$

where A and D are $m \times n$ matrices, b and e are m vectors and $x = (x_1, \ldots, x_n)$ is the variable vector and each x_i ranges over a positive finite domain $\mathcal{D}_i$. The OIP problem is to minimize those constraints using the following objective function:

$$\min\bigl\{\|Dx - e\|\colon Ax \geqslant b, x \in \mathcal{D}\bigr\}, \quad \text{where } \|v\| = \sum_i \max(0, v_i).$$

WSAT(OIP) uses a Walksat like strategy [8,10] to solve such OIP minimization problems.

Here we extend WSAT(OIP) in two ways. The first is to extend soft constraints to hierarchies. The second is to allow arbitrary finite domain constraints, not just arithmetic ones.

3.1. The WalkSearch algorithm

We assume that the constraint hierarchy problem is specified as a tuple of the form $\langle \mathcal{X}, \mathcal{C}_H, e, w\rangle$. The WalkSearch algorithm is given in figure 1 as a generalization of WSAT(OIP) so that the form of constraints is not specified. The algorithm is parameterized by *Max_moves*, *Max_tries* and various probabilities. WalkSearch always works with a full assignment for all the variables $\mathcal{X}$. It begins with an initial not necessarily feasible solution, for example, a random assignment. The inner loop starts a local search by selecting a constraint c with *select-unsatisfied-constraint*. The current assignment is then perturbed using *select-partial-repair* which changes the value of one variable in c. Local search continues until *Max_moves* local repairs have been made or some solution stopping criteria is met. To escape from being stuck in local minima, the outer loop restarts the local search *Max_tries* times. WalkSearch returns the best solution found, and this is determined using *improve*$(\theta, \theta_{\text{best}}, \mathcal{C}_H)$. With a hierarchy this is simply the *better* comparator which is used to determine whether solution θ is preferred over the previously found θ_{best}.

```
proc WalkSearch($\mathcal{C}_H, \mathcal{X}, Max\_moves, Max\_tries$)
  for $i := 1$ to $Max\_tries$ do
    $\theta :=$ an initial assignment;
    $\theta\_best := \theta$;
    for $j := 1$ to $Max\_moves$ do
      if $\theta$ meets solution stopping condition
        then return $\theta$;
      if $\theta$ is feasible $\wedge$ $improve(\theta, \theta_{best}, \mathcal{C}_H)$ then
        $\theta_{best} := \theta$;
      $c :=$ select-unsatisfied-constraint($\mathcal{C}_H, \mathcal{X}, \theta$);
      $\langle x_k, v\rangle :=$ select-partial-repair($\mathcal{C}_H, \mathcal{X}, c, \theta$);
      $\theta := \theta[x_k \rightarrow v]$;
    end
  end
  return $\theta\_best$;
end
```

Figure 1. Basic WalkSearch algorithm.

WalkSearch leaves unspecified the three procedures *improve*, *select-unsatisfied-constraint* and *select-partial-repair*. We now look at the extensions which exploit the constraint hierarchies.

3.2. *Constraint selection schemes*

With WSAT(OIP) there were only two kinds of constraints: hard and soft. Thus constraint selection is to decide only whether a hard or soft constraint is chosen. A hierarchy provides for more classes of constraints. We present four reasonable variations for selecting a violated constraint (i.e. $e(c) > 0$) with *select-unsatisfied-constraint*$(\mathcal{C}_H, \mathcal{X}, \theta)$ from the different hierarchy ranks. The difference between the various strategies lies in the emphasis placed on differentiating constraints between ranks, and how greedy is the selection with respect to the hierarchy. These selection schemes are evaluated experimentally in section 4.

HardOrSoft Constraint Selection If all hard constraints are satisfied, randomly select a violated soft constraint from $C_1, \ldots, C_n$. Otherwise, choose a violated hard constraint from C_0 with probability P_{hard}, and with probability $1 - P_{\text{hard}}$ a soft constraint from $C_1, \ldots, C_n$. This selection scheme is a direct extension of that used in WSAT(OIP) to hierarchies. HardOrSoft does not distinguish between soft constraints in different ranks.

TopOrRest Constraint Selection This is similar to HardOrSoft, however instead of the choice between hard and soft, the choice is between the top most unsatisfied constraint rank and the rest of the ranks. Choose the smallest i such that C_i contains unsatisfied constraints, call this rank *Top*. The unsatisfied constraints in the remaining ranks

$C_{i+1}, \ldots, C_n$ are denoted by *Rest*. If *Rest* is empty, choose a constraint randomly from *Top*, otherwise with probability P_{Top}, choose a constraint randomly from *Top* and with probability $1 - P_{Top}$ from *Rest*.

RankProb Constraint Selection RankProb chooses the violated constraint based on its rank. Each constraint rank C_i, where $i \in \{0, \ldots, n\}$, is associated with a probability P_i. First, a rank i is selected with probability P_i. Then a violated constraints in rank C_i are randomly selected. If there is no violated constraint in C_i, then randomly choose a violated constraint in rank $C_{i+1}, \ldots, C_n$. If there are no violated constraints in rank $C_{i+1}, \ldots, C_n$, then randomly choose a violated constraint in rank $C_0, \ldots, C_{i-1}$.

ConsProb Constraint Selection Using RankProb, a probability is associated with each rank. In constrast to RankProb, the probability of a constraint in a rank to be selected by ConsProb is influenced dynamically by the number of unsatisfied constraints at each rank. Each constraint rank C_i, where $i \in \{0, \ldots, n\}$, is associated with a probability P_i. The probability (which changes dynamically) of selecting a rank C_i at runtime is defined as

$$\frac{P_i |C_i|_{\text{violated}}}{\sum_{j \in \{0,\ldots,n\}} P_j |C_j|_{\text{violated}}},$$

where $|C_i|_{\text{violated}}$ is the number of violated constraints in C_i. Thus, the probability of choosing a particular rank changes depending on the number of violated constraints as the local search progresses.

3.3. *Partial repair and heuristics*

After *select-unsatisfied-constraint* picks a violated constraint c, *select-partial-repair*$(\mathcal{C}, \mathcal{X}, c, \theta)$ returns the choice of change in the valuation with a variable value pair to modify. In WSAT which deals with SAT problems in clausal form, making a single change by flipping a single variable will change the clause/constraint to being satisfied. In the more general case, of any finite domain constraint, it is less clear what strategy should be used. We adopt the same strategy as WSAT(OIP), namely, first try to make the constraint c more satisfied, that is compute the set of variable value pairs which differ by one value

$$\mathit{Improve} = \{(x, v) \mid \forall x \in \mathcal{X}, \forall v \in \mathcal{D}_x.\ e(c, \theta \backslash x \mapsto v) < e(c, \theta)\}.$$

The notation $\theta \backslash x \mapsto v$ denotes the valuation θ is used but the substitution for x is v instead of its value in θ.

The hierarchy is then taken into account using the *better* comparator. Select one variable value pair to return from the set $\{(x, v) \mid \neg \mathit{better}(\theta \backslash x' \mapsto v', \theta \backslash x \mapsto v, \mathcal{C}_H)$ such that $(x, v), (x', v') \in \mathit{Improve}\}$ of best local moves with respect to the comparator. It may be possible that no such pair exists, in which case the local move is not made and some variable value pair from θ is returned to give a null move. This constraint repair strategy is based on the expectation that checking a solution in the whole constraint hierarchy is much more expensive than checking it for that single violated constraint alone. Similarly it is cheaper to find the set of local moves which makes one constraint less violated than to do it for the whole constraint hierarchy. Other strategies are also possible, for example, one greedy strategy is optimize first to minimize local error and then optimize for the hierarchy.

Further heuristics from WSAT(OIP) are tabu list with aspiration which allows tabu moves if the solution is improved and a noise factor to allow moves which decrease the quality of the solution.

4. Airport gate allocation

The problem of allocating gates to arriving and departing aircraft is an integral aspect of airport operations. It can have a decisive impact the quality of service of an airport. Most work in operations research (see [5] for further references) concentrates on the minimization of passenger walking distance, one particular aspect of quality of service. Yu Cheng [3] addresses a more general problem using knowledge representation techniques and simulation. Optimal gate allocation is a hard problem. Even with rigorous simplification of the problem, only unrealistically small problems can be solved optimally with reasonable computational effort [3,5]. Thus heuristic solutions such as local search on hierarchical constraints are indispensable.

4.1. Gate allocation problems

In practice, gate allocation is subject to numerous operational constraints. Natural hard constraints include, for example:

- No two aircraft can be allocated to the same gate simultaneously.
- Particular gates can be restricted to admit only certain aircraft types.
- An aircraft leaving a gate (“push-back”) will restrict other operations in close temporal or spatial vicinity.

Typical soft constraints include:

- Airlines and ground handlers prefer to use particular gates or terminals.
- Passengers prefer to walk short distances to reach the exit or their connecting gate.
- Passengers prefer gates connected to terminal buildings rather than remotely located gates.

The users find it hard to quantify the cost of violations of soft constraints. They prefer instead to state that some soft constraints are absolutely more important compared to the others. Hence, the use of constraint hierarchies to group the constraints into ranks of importance is a natural way to express the quality of solutions. In collaboration with the Civil Aviation Authority of Singapore, we identified 25 classes of constraints, which we organized in a constraint hierarchy with four ranks, according to their relative importance as judged by the users. The highest rank is reserved for hard constraints which are the operational requirements.

4.2. Gate allocation models

The main operational parameters for a gate allocation problem are: (i) the number of aircraft arriving within a time interval; and (ii) the number of available gates during that time interval. In what follows, we consider problems consisting of m aircraft and n gates.

We experiment with the following two gate allocation models:

- A 0/1 model with only binary variables, the decision variables are $m \cdot n$ variables denoted as y_{ik} where $1 \leqslant i \leqslant m$ and $1 \leqslant k \leqslant n$. These variables express whether aircraft i uses gate k.
- A finite domain model where variables range from 1 to m. The value of variable x_i gives the gate used by aircraft i.

In the 0/1 model, all 25 constraint classes are expressed in the form of linear inequalities. For the finite domain model, it was necessary to introduce three new symbolic constraints besides the usual linear constraints in order to express the 25 constraint classes in this model. These are *alldiff* for the non-overlapping ground time constraint, airline preference constraint and walking distance constraint.

4.2.1. Constraint formulations

The formulation of some of the constraints used in both the finite domain and 0/1 hierarchical constraint models are now described. The full details and formulation of the gate allocation problem is beyond the scope of this paper. Here we are concerned with investigating the behavior of the hierarchical local search solver.

In the sample constraints below, the non-overlapping ground time constraint is a hard constraint. The other two, airline preference and walking distance, are soft constraints which are placed at a user specified level in the constraint hierarchy.

Non-overlapping ground time constraint. The non-overlap constraint states that no two aircraft with overlapping ground times can be allocated to the same gate. In the 0/1 model, it is formulated as follows: for each gate k and each pair of aircraft i and j with overlapping ground times, the constraint $y_{i,k} + y_{j,k} \leqslant 1$ must hold. In the finite domain model, the formulation is the following: for each maximal set of aircraft S whose ground time overlaps, we introduce a constraint *alldiff* (S). The *alldiff* constraint is the usual all-different constraint, as in many finite domain solvers, which constrains the variables in S to have different values.

Airline preference constraint. This preference constraint states the preferences for certain airlines to have aircraft be parked at particular sets of gates. A preference value $AP_{a,k}$ is assigned for each airline a and gate k which is a natural number giving the degree of in-compatibility of the airline a for the gate k. Bigger values indicate increasing gate incompatability and the value zero indicates perfect match.

In the 0/1 model, the constraints are formulated as follows: for each aircraft i and each gate k, a constraint $AP_{a,k} \cdot Y_{i,k} \leqslant 0$ is added as a soft constraint at some particular level. In the finite domanin model, the formulation is the following: for each aircraft i, the soft unary constraint $AP_a(x_i)$ is used. The error function for $AP_a(x_i)$ is defined as $e(AP_a(x_i)) = AP_{a,x_i}$. Thus the value of x_i is used as the index for AP.

Walking distance constraint. This preference constraint aims to minimize the total walking distance for transfering passenger. Let $P_{i,j}$ be the number of passengers that transfer from flight i to flight j and $D_{k,l}$ is the walking distance between gate k and gate l. In the 0/1 model, the constraints are formulated as follows: for each pair of passenger transfers (i, j), for each pair of gates (k, l), a soft constraint $P_{i,j} \cdot D_{k,l} \cdot (x_{i,k} + x_{j,l}) \leqslant 1$ is added.

In the finite domain model, the formulation is the following: for each pair of passenger transfers (i, j), a soft binary constraint $WD(x_i, x_j)$ is created. The error function for for $WD(x_i, x_j)$ is defined as

$$e\big(WD(x_i, x_j)\big) = P_{i,j} \cdot D_{x_i,x_j}.$$

Thus both AP and WD are symbolic finite domain constraints rather than the usual arithmetic ones.

4.3. *Benchmark data*

We focus on the following goals for this experimental study:

– evaluation of hierarchical local search for realistic gate allocation problems,
– a comparison of the performance of the two models, and
– a comparison the performance of the four constraint selection schemes.

The dataset used for gate allocation consists of historical flight data set spanning 24 hours at Changi Airport with 257 flights and 104 gates available in total. This dataset was chosen to be fairly typical of daily operations.

We used a number of different problem sizes where for a given number of flights, the problem difficulty in varied by changing the number of gates, table 1 lists the problems and their sizes which is defined by the number of flights and gates. Problem sets P1 to P6 are small problems with up to 30 flights. The other problem sets P7 to P15 are larger data sets going all the way up to the maximum of 257 flights. For each problem, we created two models, a finite domain (FD) model and a 0/1 model. Table 2 gives the resulting hierarchical constraint problems together with the number of variables and constraints in each hierarchy level for each model. The FD model being more expressive has much less variables and constraints than the 0/1 model. For the smaller benchmarks P1 to P4, the optimal solutions are known and were obtained by running the corresponding integer program.

Table 1
Specifications of the data sets.

Problem	No. of flights	No. of gates
P1	10	10
P2	20	20
P3	20	19
P4	20	17
P5	30	30
P6	30	26
P7	50	50
P8	50	46
P9	50	40
P10	100	100
P11	100	90
P12	100	80
P13	257	104
P14	257	94
P15	257	84

4.4. Experimental setup

All experiments were run on a Pentium II PC running Linux. The performance of the local search algorithm depends very much on the noise level and on the probabilities used in constraint selection. The best setting of these parameters depends on the chosen model, selection scheme and constraints. In order to gauge the performance of the models and selection schemes, we tried different parameter settings for each benchmark problem.

For all combinations of problem models, problem sets and constraint selection schemes, we used 11 different noise probabilities $(0.0, 0.1, \ldots, 1.0)$. In most of our benchmarks, a small noise level (about 0.1) works best for the 0/1 model, while a higher noise level (about 0.4) works best for the FD model. One explanation is that the FD model has more possible values to try hence a higher noise level can be used to achieve better coverage of non-improving moves.

For the best constraint selection schemes found, ConsProb and RankProb, we tried five probability distributions. As expected, the probability distributions that emphasize the more important ranks work best. In our benchmarks, we found that the probability ratio of $1000:100:10:1$ worked for most of the test cases. However, sometimes inversions of probability ratios such as $8:0.5:0.5:1$ did perform better.

In all figures given in the next section, the best noise level and probability distribution for each model, problem set and constraint selection scheme was used, in order to ensure a fair comparison.

For the smaller problems (P1 through P4) it was possible using the 0/1 model to obtain the optimal solutions using integer programming with the CPLEX solver. The conversion from the 0/1 model to an integer program with linear optimization function is adapted from [12]. Hierarchies are expressed by multiplying each error value with a

Table 2
Specifications of the data sets.

Problem	Model	Variables	$\|C_0\|$	$\|C_1\|$	$\|C_2\|$	$\|C_3\|$
P1	FD	10	52	8	15	11
	01	42	86	6	5	11
P2	FD	20	95	17	40	19
	01	137	223	17	27	19
P3	FD	20	95	17	40	19
	01	132	216	17	27	19
P4	FD	20	95	17	40	19
	01	121	202	17	27	19
P5	FD	30	140	24	69	27
	01	380	702	24	192	27
P6	FD	30	140	24	69	27
	01	313	594	24	169	27
P7	FD	50	231	41	194	49
	01	1097	2038	41	1545	1293
P8	FD	50	231	41	194	49
	01	1021	1892	41	1488	1155
P9	FD	50	231	41	194	49
	01	885	1646	41	1274	864
P10	FD	100	461	82	571	114
	01	4426	7887	82	16022	15183
P11	FD	100	361	82	571	114
	01	4023	7320	82	11739	12091
P12	FD	100	361	82	571	114
	01	3533	6514	82	9628	9049
P13	FD	257	953	217	2057	303
	01	12956	25926	217	65523	45576
P14	FD	257	1210	217	2057	303
	01	10827	21866	217	44676	27655
P15	FD	257	953	217	2057	303
	01	9709	19626	217	36688	21805

factor computed using the weight of the constraint, its rank and the number of constraints in the rank. Using the CPLEX solver on the converted 0/1 model, optimal solutions for problem sizes of up to around 25 flights could be found. This was useful since it allows us to judge the performance of local search on the smaller problems.

4.5. Experimental results

For the largest problem, the 24 hour P15 problem, local search was able to find good solutions within 6 minutes of cpu time using either model. Here "good" solutions means those solutions which significantly improve the (also computer-generated) schedules currently used by the airport. This comparison is with respect to the same *weighted-sum-better* hierarchy.

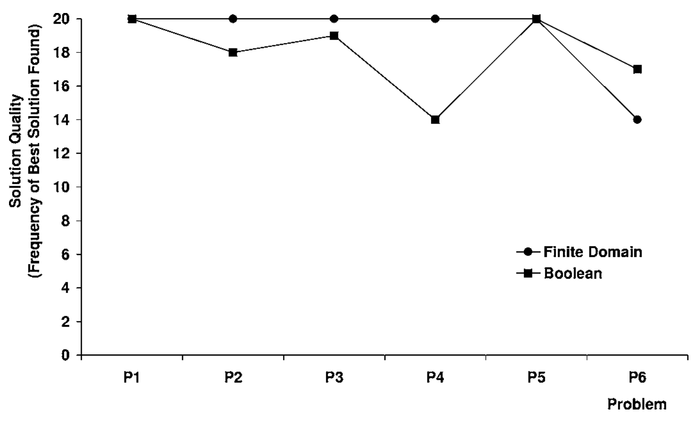

Figure 2. Performance of finite domain vs. 0/1 model using best constraint selection scheme.

We now examine the solution quality of the two models. Figure 2 gives the number of times the best solution found was obtained using the best constraint selection scheme for both models. For the smaller problems (P1 through P4), where we know the optimum, local search also succeeds in finding optimum solutions. For the other two problems P5 and P6, both models found the same best solutions with different random seeds.

In the small problems, the best solution is found more than once. As such, we use the frequency of the best solution reached among 20 runs to compare the performance the two problem models. We found that the finite domain model obtains the best solution more often than the 0/1 model.

In the larger problems P7 to P15, the true optimum solution is not known and thus we do not have an absolute measure of comparision. Hence, we only compare the solutions found against each other. With the larger problems, within the time limits usually only one best solution is found for one model. The comparison of solution quality then for problems P7 to P15 is which model found the better solution. In all cases, with the exception of P7 and P9, the finite domain model was better. Thus overall it appears that a more general finite domain constraint model is better suited for this problem. It is a little surprising that the finite domain model does so much better than the 0/1 model since being a more compact model there is less opportunities for local search to exploit randomness when exploring the solution neighbourhood. One possible explaination is that the finite domain constraint has more information about the application constraint and thus can be more goal directed. For example, consider an airline preference constraint on aircraft i. In the finite domain constraint model, this becomes one constraint $AP_a(x_i)$. While in the 0/1 model, it is expressed as n constraints of the form $AP_{a,k} \cdot y_{i,k} \leqslant 0$ where $k \in \{1, \dots, n\}$. So the 0/1 model allows for some kind of partial satisfaction of the airline preference for one aircraft since not all the n constraints need be satisfied or unsatisfied. This possibility is disallowed by the symbolic finite domain constraint AP since there is only one value for x_i. For gate allocation, we believe that stronger constraints are more important for solution quality.

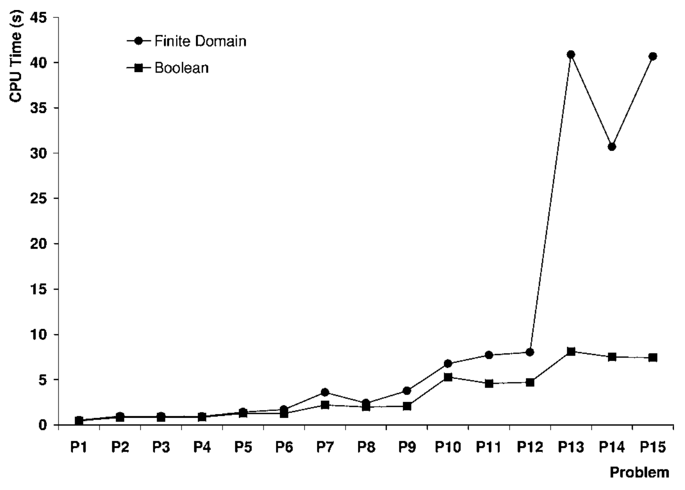

Figure 3. Comparison of solving time.

Next we look at the time required for the best solution for each model which is given in figure 3. Here both models behave similarly for the small problems. When the problems start becoming large, the finite domain model is significantly more expensive. Our implementation does not have a highly optimized solver for the finite domain case. The 0/1 model thus being simpler is significantly cheaper since the *improve* and *select-partial-repair* operations are much more efficient with a binary choice. Furthermore as our implementation is unoptimized, the runtime memory requirements for problems P13 to P15 was observed to be very large in the finite domain model, and the sharp knee is also caused by increased memory paging.

We will now compare the performance of the various proposed constraint selection schemes. Figure 4 shows the quality of the best solution found using the four constraint selection schemes using the finite domain model for small problems and figure 5 does the same for the larger problems. Similarly for the 0/1 model, figure 6 is for the small problems and figure 7 is for the large problems. The graphs show that the performance of the different schemes do vary significantly and it is not so clear which schemes are clearly better. The results show that in most cases, choosing either RankProb or ConsProb, both of which make explicit use of the hierarchy in constraint selection, is better than the other two strategies. Thus exploiting the hierarchy does seem to be effective to a limited extent but the actual performance of the various constraint schemes is complicated without a clear winner.

In all the gate allocation benchmarks, the aspect of finding an optimum solution dominanted the aspect of finding a solution satisfying the hard constraints. Here, it was almost always easy to find feasible solutions and most of the computation time was spent on optimizing the fulfillment of soft constraints.

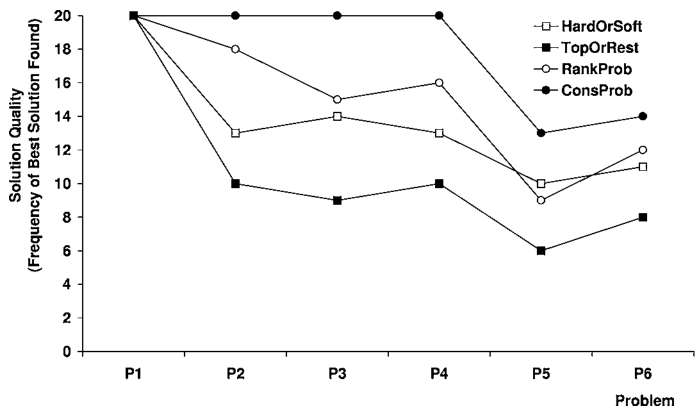

Figure 4. Performance of different constraint selection scheme on FD model.

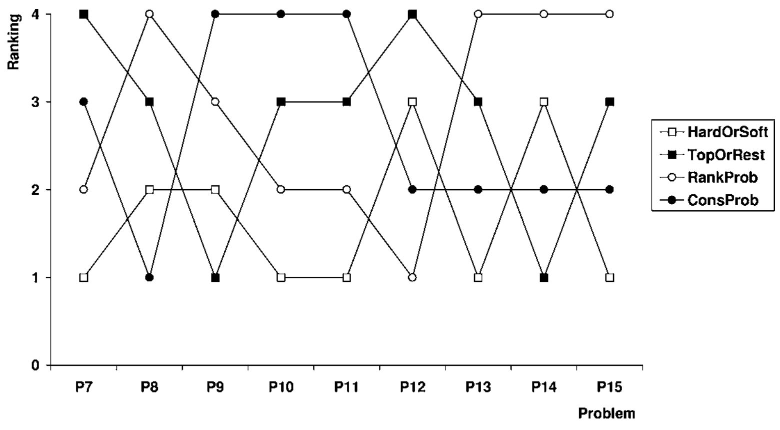

Figure 5. Performance of different constraint selection scheme on FD model for bigger test cases.

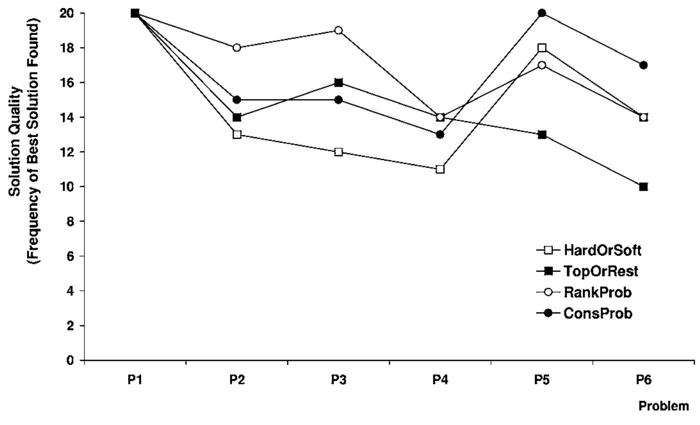

Figure 6. Performance of different constraint selection scheme on 0/1 model.

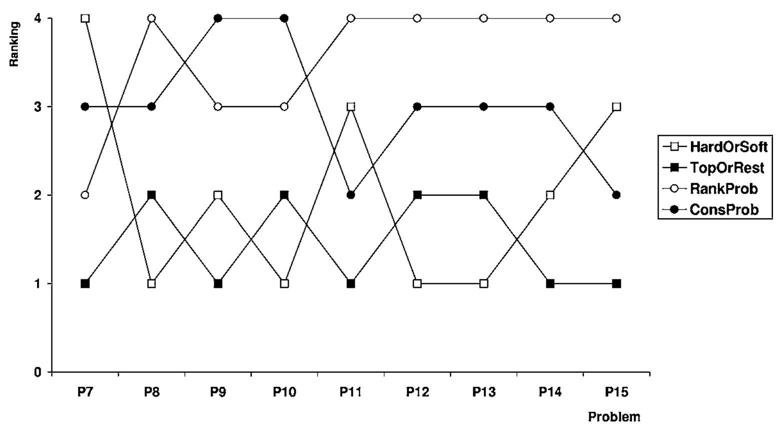

Figure 7. Performance of different constraint selection scheme on 0/1 model for bigger test cases.

5. Nonhierarchical solver performance

The final experimental result is to evaluate the performance of our solver simply as a local search 0/1 solver. For this purpose, we choose the sports scheduling of the Atlantic Coast Competition in basketball (ACC benchmarks). The ACC benchmarks which has been solved using a mix of integer programming and explicit enumeration by Nemhauser and Trick [9], finite domain constraint programming by Henz [6] and WSAT(OIP) by Walser [13].

We choose as a reference benchmark the WSAT(OIP) solver which has been shown to be effective for this problem and also many other problems. The same constraint model was used for both solvers – this is the ACC problem description written in AMPL [13]. The solver parameters and heuristics were chosen to be as similar as possible. However it is not possible to reproduce exact behavior simply because of the random nature of local search given the different implementations. The purpose of this benchmark is simply to compare our hierarchical local solver which supports general finite domain constraints against the optimized WSAT(OIP) implementation which is specifically optimized for 0/1 OIPs.

Figures 8 and 9 compare the number of moves needed to reach a solution given various random seeds for our solver and WSAT(OIP). Where no solution is found within 20,000,000 moves, it is indicated as a zero height bar. We see here that the pattern for the two solvers is not dissimilar but our solver seems to be a little more effective here in that the optimum is found more often.

To compare solver efficiency, figures 10 and 11 give the runtimes for the same runs. Our solver is not as efficient as WSAT(OIP) and is several times slower. However given that it is not an optimized implementation, the purpose of this benchmark is to show that our solver for hierarchical general finite domain constraints remains effective as a nonhierarchical solver for OIP.

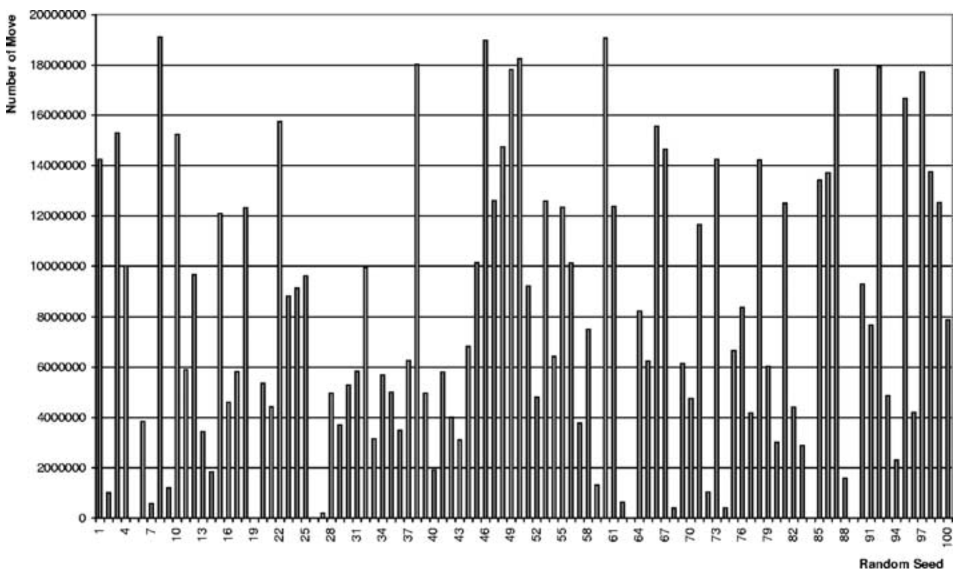

Figure 8. Number of moves to solve ACC using 0/1 solver.

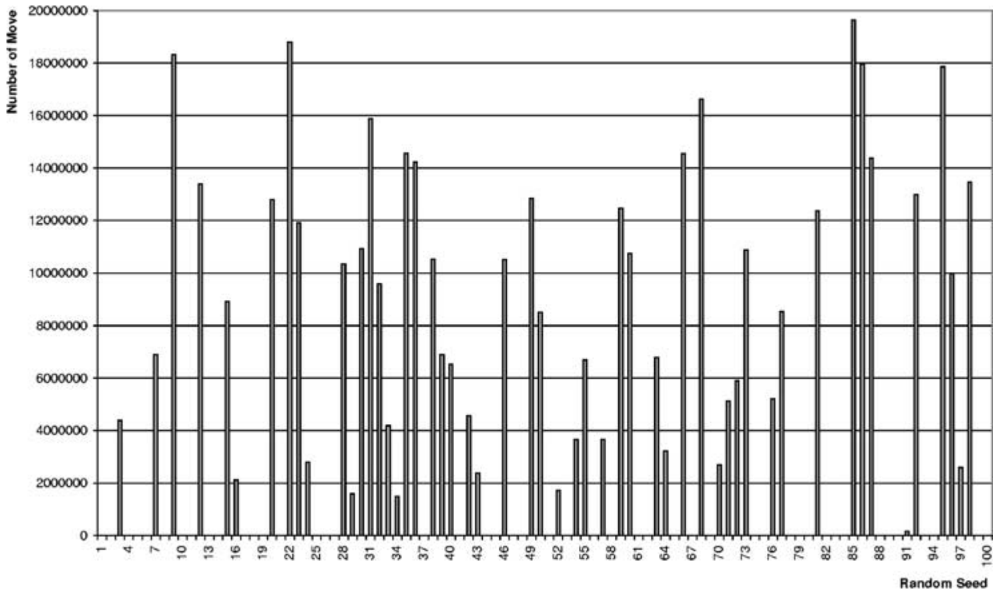

Figure 9. Number of moves to solve ACC using WSAT(OIP).

6. Conclusion

In this paper we present a local search solver based on a Walksat strategy which solves finite domain constraints in a constraint hierarchy setting. We believe that many real world problems are applicable in such a framework since there are usually many

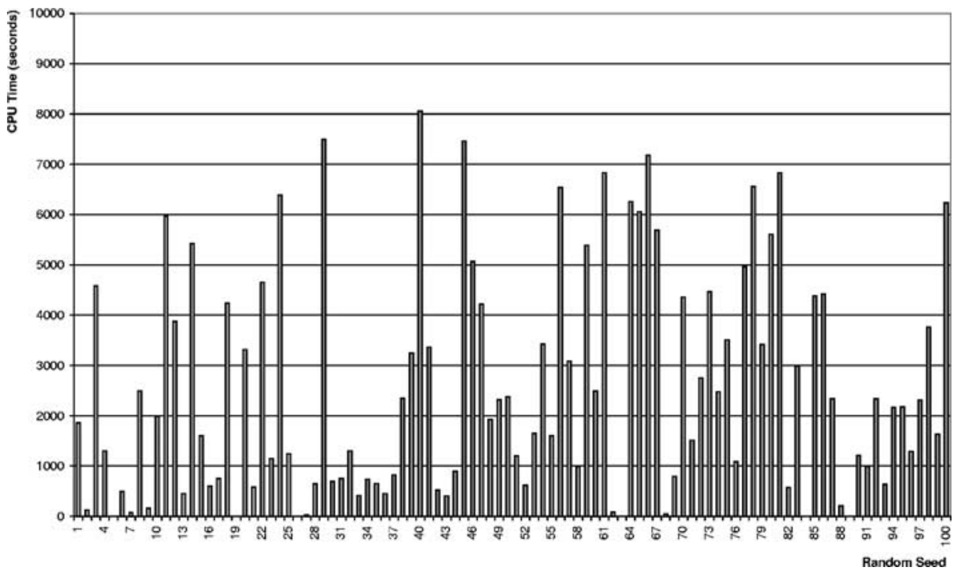

Figure 10. CPU time taken to solve ACC problem using 0/1 solver.

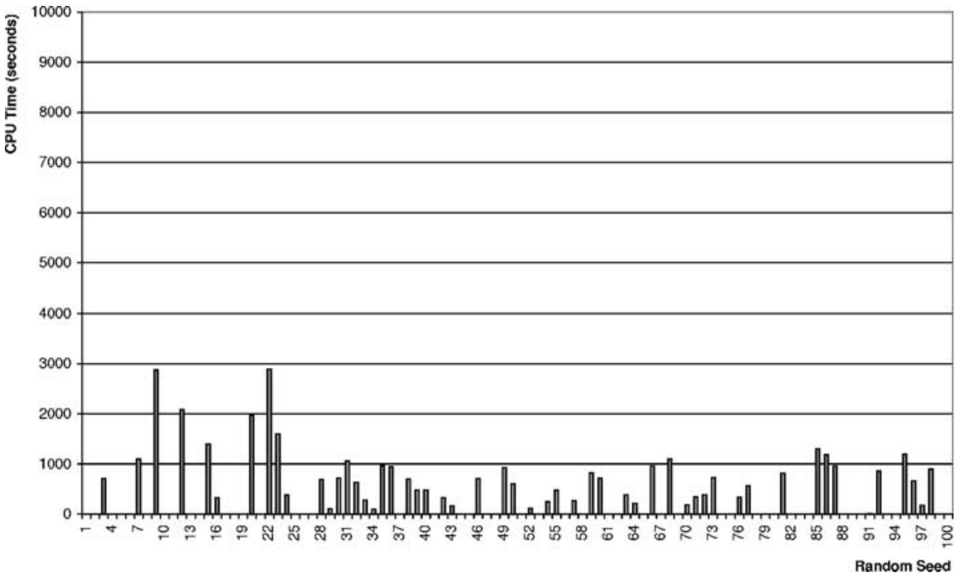

Figure 11. CPU time taken to solve ACC problem using WSAT(OIP).

conflicting constraints and it is easy for users to state the importance of various constraints in terms of a hierarchy. Furthermore, many of these problems share a similar characteristic to our driving airport gate allocation application in that obtaining satisfying solutions is easy and the main computation problem is in optimizing the conflicting constraints.

We show that it is easy to incorporate general hierarchical finite domain constraints into a Walksat based local search solver. Various strategies for constraint selection and solution repair which take into account constraint hierarchies are presented.

As our motivation for this work came from the need to solve a difficult gate allocation problem expressed in terms of hierarchies, this was a natural benchmark to evaluate our solver. There are also no other benchmark of substantial difficulty where finite domain constraints and hierarchies are used, hence this is a key benchmark. We go into some problem detail of the gate allocation problem. We compare a 0/1 linear constraint model with a finite domain one and show that the richer models available with general finite domain constraints produces better solutions for gate allocation with local search. The choice of model and the increased possibilities offered by symbolic finite domain constraints with local search is an interesting avenue for further work. While there is substantial work with comparing different models in the linear/integer programming literature and with finite domain based consistency solvers together with search, this is not the case for local search techniques.

The local search solver for gate allocation gives optimum solutions for small problems and solutions better than the existing constraint programming based application used for gate allocation. Among the various strategies for constraint selection, we found that for gate allocation, ConsProb and RankProb which take advantage of the hierarchy performed better. Finally we show that our solver is also competitive as a non-hierarchical solver.

Acknowledgements

This work was supported by the Singapore Government through a grant from the National Science and Technology Board. We thank the Civil Aviation Authority of Singapore and the Kent Ridge Digital Laboratories for providing documents and test data sets on the gate allocation of Changi airport.

References

[1] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schies and G. Verfaille, Semiring-based CSPs and valued CSPs: Basic properties and comparison, in: [7] pp. 111–150.

[2] A. Borning, B. Freeman-Benson and M. Wilson, Constraint hierarchies, Lisp and Symbolic Computation 5 (1992) 223–270.

[3] Y. Cheng, A rule-based reactive model for the simulation of aircraft on airport gates, Knowledge-Based Systems 10(4) (1998) 225–236.

[4] E. Freuder and R. Wallace, Partial constraint satisfaction, in: [7] pp. 63–110.

[5] A. Haghani and M.-C. Chen, Optimizing gate assignments at airport terminals, Transportation Research 32(6) (1998) 437–454.

[6] M. Henz, Scheduling a major college basketball conference (revisited), Operations Research 49(1) (2001).

[7] M. Jampel, E. Freuder and M. Maher (eds.), *Over-Constrained Systems*, Lecture Notes in Computer Science, Vol. 1106 (Springer, 1996).

[8] D. McAllester, B. Selman and H. Kautz, Evidence for invariants in local search, in: *Proceedings Fourteenth National Conference on Artificial Intelligence (AAAI-97)* (1997).

[9] G.L. Nemhauser and M.A. Trick, Scheduling a major college basketball conference, Operations Research 46(1) (1998) 1–8.

[10] B. Selman, H. Kautz and B. Cohen, Noise strategies for improving local search, in: *Proceedings of AAAI-94* (1994) pp. 337–343.

[11] J.P. Walser, Solving linear pseudo-Boolean constraint problems with local search, in: *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)* (AAAI Press, Providence, RI, 1997) pp. 269–274.

[12] J.P. Walser, Domain-independent local search for linear integer optimization, Ph.D. Thesis, Universität des Saarlandes, Saarbrücken, Germany (1998).

[13] J.P. Walser, *Integer Optimization by Local Search, a Domain-Independent Approach*, Lecture Notes in Artificial Intelligence, Vol. 1637 (Springer, 1999).

[14] M. Wilson and A. Borning, Hierarchical constraint logic programming, Journal of Logic Programming 16(3/4) (1993) 277–319.