Solving Hierarchical Constraints over Finite Domains

Martin Henz* Lim Yun Fong† Lua Seet Chong* Shi Xiao Ping†
J. Paul Walser‡ Roland H. C. Yap*

December 27, 1999

Abstract

Many real world problems have requirements and constraints which conflict with each other and are not well defined. One framework for dealing with such over-constrained/fuzzy problems is provided by constraint hierarchies, where constraints are divided into ranks, and where a comparator selects preferred solutions over others. In this paper, we present a framework for formulating hierarchical constraint problems over finite domains (HCPs). We show, how the recent framework of over-constrained integer programs (OIPs) can be extended to handle non-linear constraints and to exploit constraint hierarchies.

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 gate allocation benchmarks with varying problem characteristics, we compare local search on the given HCPs with local search and integer programming on linear reformulations of the HCPs.

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. 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 [10] exemplifies this approach) addresses the over-constrainedness by resolving the conflict with preferences on the importance of constraints and 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. These and other papers can be found in the collection in [6].

Since we were confronted with user-specified constraint ranks in an project on gate allocation, we focus on 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 integer optimization (see Section 4 for a discussion). Thus we resort to local search. The local search algorithm WSAT(OIP) [7, 9] is a walk search algorithm designed for solving over-constrained linear integer programs and provides a good starting point for

*National University of Singapore,
email: {henz,luasc,ryap}@comp.nus.edu.sg
†Kent Ridge Digital Laboratories,
email: yflim@krdl.org.sg
‡i2 Technologies, email: j.paul.walser@i2.com

this work. In Section 3, we extend WSAT(OIP) in two directions. Firstly, we allow arbitrary constraints as opposed to linear ones, and 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 airport gate allocation and reports on experiments on solving them with hierarchical local search. We compare 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 uses symbolic constraints. Furthermore, we evaluate the variants of the local search algorithm on gate allocation data sets.

2 Constraint Hierarchies

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

Let $\mathcal{X}$ be a set of variables. To each variable $x \in \mathcal{X}$ belongs a set $\mathcal{D}_x$ denoting 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, $0 \leq i \leq n$, C_i is a multiset constraints of rank i.

The constraints in C_0 denote *required constraints* (or *hard constraints*), which *must* be satisfied. The constraints in $C_1, C_2, ..., C_n$ denote *preferential constraints* (or *soft constraints*), which must not necessarily be all satified and which range from the strongest rank C_1 to the weakest rank C_n.

A *valuation* θ is a function that maps the variables in $\mathcal{X}$ to elements in the domain $\mathcal{D}$. The set of solutions $S_0 = \{\theta | \forall c \in C_0,\ c\theta\ holds\}$ contains those valuations that satisfy the required constraints, i.e. for every constraint c over $x_1, \ldots, x_k$, $(x_1\theta, \ldots, x_k\theta) \in c$.

A partial ordering *better* over solutions describes their quality and is called a *comparator*. The *solution set* S_{better} contains only those solutions that are optimal respect to *better*:

$$S_{better} = \{\theta \in S_0 | \forall \sigma \in S_0.\ \neg better(\sigma, \theta)\}$$

There are many suitable choices for comparators in a constraint hierarchy (see [2]). We found the following weighted-sum-better comparator most useful in our application and thus concentrate on it in this paper.

Weighted-sum-better uses an *error function* $e(c, \theta)$ which returns a non-negative real number indicating the degree of violation of constraint c with 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. A simple error function returns 0 when θ satisfies c and 1 if not.

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

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.

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

In summary, a constraint hierarchy is defined—for the purpose of this paper—by the tuple $\langle \mathcal{X}, \mathcal{C}_H, e, w \rangle$. The goal is to find solutions in $S_{weighted-sum-better}$, where *weighted-sum-better* uses the given error function e and weight function w.

3 Hierarchical 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. We extend WSAT(OIP) [7, 9], which has been shown to be effective for a range of application areas, such that it can handle arbitrary (not necessarily linear) constraints and exploit constraint hierarchies during search.

3.1 The WalkSearch Algorithm

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 or some solution stopping criteria are met. To escape being stuck in local minima, the outer loop restarts local search Max_tries times. WalkSearch returns the best solution found, and this is determined using *improve*$(\theta, \theta_{best}, \mathcal{C})$ which compares whether solution θ is better than the saved θ_{best}.

WalkSearch leaves unspecified the three procedures *improve*, *select-unsatisfied-constraint* and *select-partial-repair*. WSAT(OIP) tailors these procedures for over-constrained integer programs. We use the global comparator *weighted-sum-better* for *improve*. The function *select-partial-repair* is adapted from [9]. Value changes of variables are selected according to *weighted-sum-better*, subject to history and tabu mechanisms and a noise factor.

For the performance of hierarchical local search, it is crucial to exploit the constraint hierarchy for constraint selection

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

Figure 1: Basic WalkSearch algorithm

3.2 Constraint Selection Schemes

We present four reasonable variations for selecting a violated constraint 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 $1 - P_{hard}$ a soft constraint from $C_1, \ldots, C_n$. This selection scheme is similar to that used in WSAT(OIP) [9]. 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 $i \in \{0..n\}$, is associated with probability P_i. First, a rank i is selected with probability P_i. Then a violated constraints in rank C_i is randomly selected. If there is no violated constraint in C_i, then randomly choose a violated constraint in rank $C_{i+1}, ..., C_n$. If there are no violated constraints in rank $C_{i+1}, ..., C_n$, then randomly choose a violated constraint in rank $C_0, ..., C_{i-1}$.

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 $i \in \{0..n\}$, is associated with probability P_i. The dynamic probability of selecting a rank C_i is defined as

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

where $|C_i|_{violated}$ is the number of violated constraints in C_i.

4 Airport Gate Allocation

The problem of allocating gates to arriving and departing aircrafts is an integral aspect of airport operations and can have a decisive impact the quality of service of an airport. Most work in Operations Research (see [5] for further references) concentrated on the minimization of passenger walking distance, a 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 [5, 3], and thus heuristic solutions such as local search are indispensable.

4.1 Gate Allocation Problems

In practice, gate allocation is subject to numerous operational constraints. Naturally 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 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.

4.2 Models

Let us assume a gate allocation problem with m aircrafts and n gates.

- In a 0/1 model, $m * n$ 0/1 variables Y_{ij} are introduced where $1 \leq i \leq m$ and $1 \leq j \leq n$, which express whether aircraft i uses gate j.

- In a finite domain model, for each aircraft i, a variable X_i ranging from 1 to m indicates which gate aircraft i uses.

In the 0/1 model, all 25 constraint classes can be expressed as linear inequality constraints. Thus integer programming and a hierarchical extension of WSAT(OIP) can be used for solving this model. For example, the constraint that no two aircraft can be allocated to the same gate can be expressed by one constraint for each gate j and aircrafts i and k with overlapping ground time of the form $Y_{ij} + Y_{kj} \leq 1$.

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. The first symbolic constraint expresses that all variables among a given set are assigned to different integers. This constraint allows a natural encoding of the non-overlap requirement for aircraft. For each maximal set of aircraft S whose ground time overlaps, we introduce a constraint *alldiff* S.

The second symbolic constraint is used for concise encoding of unary constraints. This constraint maps the possible values that a variable can take to an integer value that reflects a degree of violation. For example, airline preferences for particular gates can be encoded using this constraint by giving low integer values for preferred gates. The third symbolic constraint extends this concept to the binary case and is used for minimization of passenger walking distance.

4.3 Benchmark Data

We focus on the following goals for this experimental study.

- evaluate hierarchical local search for realistic gate allocation problems,
- compare the performance of the two models, and
- compare the performance of the clause selection schemes.

In order to gauge the peformance of the different algorithms for gate allocation, we use a historic data set of 24 hours from Changi Airport, containing 257 flights and using 104 gates.

Problems	P2		P5		P12	
# Flights	20		50		257	
# Gates	20		50		104	
Model	FD	0/1	FD	0/1	FD	0/1
# Variables	20	137	30	380	257	12956
$\|C_0\|$	95	233	140	702	953	25926
$\|C_1\|$	17	17	24	24	217	217
$\|C_2\|$	40	27	69	169	2057	65523
$\|C_3\|$	19	19	27	27	303	45576

Table 1: Specifications of the Selected Data Sets

We study the performance of the variants of local search using the full 24 hour data (P12) and smaller benchmark sets generated from the test data (P1 to P11). Table 1 indicates the size of three of the twelve benchmark problems in terms of the number of variables and the number of constraints in each hierarchy. The numbers of variables and constraints indicate that the finite domain (FD) model is a more compact representation than the 0/1 model.

The smaller benchmarks (P1 - P4) are included because optimal solutions for them are known. This allows a more precise method of comparison of the different heuristics (see next section).

4.4 Setup of Experiments

The performance of the local search algorithm depends on the noise level and on the selection probabilities of constraint selection schemes. The best setting of these parameters depends on the chosen model, selection scheme and benchmark data. In order to gauge the performance of the models and selection schemes, we tried different parameter settings for each benchmark data.

For all combinations of problem model, problem set and constraint selection scheme, we tried 6 different noise levels (0.0, 0.1, ..., 0.5). In most of our benchmarks, a lower noise level (about 0.1) works best for for the 0/1 model, whereas a higher noise level (about 0.4) works best for the FD model.

For the constraint selection schemes ConsProb and RankProb, we tried five probability distributions. As expected, probability distributions that emphasize the higher ranks work best. In our benchmarks, we found that the probability ratio of 1000:100:10:1 works fine for most of the test cases. However, sometimes inversions of probability ratios such as 8:0.5:0.5:1 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) we used the 0/1 model for finding optimal solutions using integer programming with CPLEX. This allowed us to use a more accurate comparison technique of models and selection schemes (see next section). The conversion from the 0/1 model to an integer program with linear optimization function is adapted from [8]. Hierarchies are expressed by multiplying each error value with a factor computed using the weight of the constraint, its rank and the number of constraints in the rank.

The given runtimes were achieved on a Sun Ultra 30 workstation.

4.5 Experimental Results

The first observation is that integer programming using the tool CPLEX on the 0/1 model allows to solve only problem cases of about 25 flights However, the optimal solution found by integer programming allows us to judge the performance of different variants and models of local search on the smaller benchmark problems.

For the full 24 hour problem, local search is able to find good solutions within 6 minutes cpu time using either one of the given models. Here "good" solutions means solutions that significantly improve over the (also computer-generated) schedules currently used by the airport, with respect to *weighted-sum-better* using a user-defined constraint

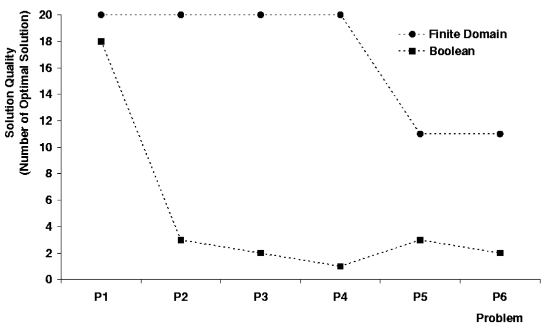

Figure 2: Performance of Finite Domain vs 0/1 Model using Best Constraint Selection Scheme

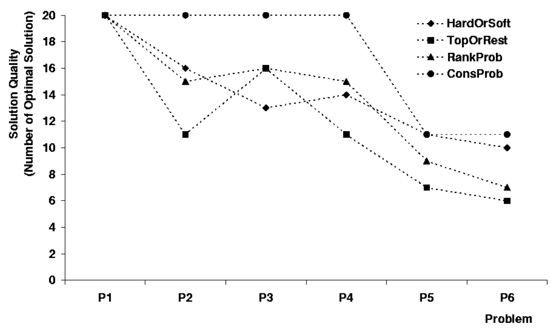

Figure 3: Performance of Different Constraint Selection Scheme on FD Model

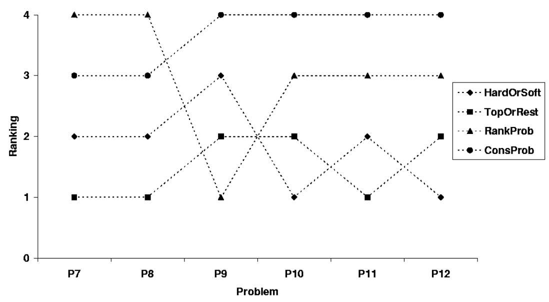

Figure 4: Performance of Different Constraint Selection Scheme on FD Model for bigger test cases

hierarchy.

For the smaller problems (P1 through P4), we can use a comparison technique that relies on known optimal solutions (obtained using integer programming, see previous section). In this comparison technique, we count how often a run of local search encounters an optimal solution. The optimality of the solutions to problems P1 to P4 is shown using CPLEX.

The first set of experiments is directed to evaluating the relative performance of the 0/1 model vs the finite domain model. The vertical axis of Figure 3 shows the number of times the optimal solution is found (for P5 and P6, optimality is not proven, but conjectured) among 20 tries with different random seeds, using the best constraint selection scheme (which was always ConsProb) for the 0/1 model and the finite domain model. Different models and selection schemes require different times per move. Therefore, we allocated a fixed cpu time to each try. This time is proportional to the number of flights in the benchmark), ranging from 24 seconds to 6 minutes. This timing corresponds to about 10000 moves in the 0/1 model and 3000 moves in the finite domain model.

Figure 2 shows that the finite domain model performs better for all of the smaller benchmarks. The more concise representation of the problem seems to allow to explore more fruitful local neighbourhoods. Our experiments show that this continues to be the case for the large benchmark problems.

We use the same technique to judge the performance of the proposed constraint selection schemes. Figure 3 shows that as the problem size and difficulty increases, the ConsProb scheme is superior to the other schemes. Figure 4 gives a comparison of the performance of different constraint selection scheme over the bigger problems. Due to the problem size, an optimal solution cannot be found using integer programming. Hence for every problem, we can only rank the constraint selection schemes relative to each other according to their performance. The figure shows that ConsProb is still the best scheme, when the problem size increases.

In all described benchmarks, the optimization aspect dominated over the satisfaction of hard constraints; it was almost always easy to find feasible solutions and most computation time was spent on optimizing the fulfillment of soft constraints.

4.6 Summary

Our experiments indicate that hierarchical local search is a promising technique for solving large gate allocation problems. Local search on the finite domain model is superior to local search on the 0/1 model. The constraint selection scheme ConsProb is superior to the proposed alternatives.

Acknowledgments

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 Jampel et al. [6], pages 111–150.

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

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

[4] E. Freuder and R. Wallace. Partial constraint satisfaction. In Jampel et al. [6], pages 63–110.

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

[6] M. Jampel, E. Freuder, and M. Maher, editors. *Over-Constrained Systems*. Lecture Notes in Computer Science 1106. Springer-Verlag, 1996.

[7] 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)*, pages 269–274, Providence, RI, USA, July 1997. AAAI Press.

[8] J. P. Walser. *Domain-Independent Local Search for Linear Integer Optimization*. PhD thesis, Universität des Saarlandes, D 66041 Saarbrücken, Germany, Aug. 1998.

[9] J. P. Walser. *Integer Optimization by Local Search, A Domain-Independent Approach*. Lecture Notes in Artificial Intelligence 1637. Springer-Verlag, 1999.

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