Minimal Perturbation in Dynamic Scheduling

Hani El Sakkout, Thomas Richards, and **Mark Wallace**[1]

Abstract.
This paper describes an algorithm, *unimodular probing*, conceived to optimally reconfigure schedules in response to a changing environment. In the problems studied, resources may become unavailable, and scheduled activities may change. The total shift in the start and end times of activities should be kept to a minimum. This requirement is captured in terms of a linear optimization function over linear constraints. However, the disjunctive nature of many scheduling problems impedes traditional mathematical programming approaches. The *unimodular probing* algorithm interleaves constraint programming and linear programming. The linear programming solver is applied to a dynamically controlled subset of the problem constraints, to guarantee that the values returned are discrete. Using a repair strategy, these values are naturally integrated into the constraint programming search. We explore why the algorithm is effective and discuss its applicability to a wider class of problems. It appears that other problems comprising disjunctive constraints and a linear optimization function may be suited to the algorithm. *Unimodular probing* outperforms alternative algorithms on randomly generated benchmarks, and on a major airline application.

1 INTRODUCTION

In this paper we tackle a dynamic variant of the classic scheduling problem. The aim is to restore consistency in a schedule which has been disrupted due to resource or activity changes. To minimize disruption, the new schedule should differ minimally from the old one.

Effective solutions to this NP-hard problem are theoretically and commercially significant. Resource-constrained scheduling problems give rise to the NP-complete resource feasibility problem of planning (RFP) [3]. A predecessor of our algorithm is a constraint satisfaction algorithm proposed to solve the RFP. We are currently exploring the utility of using unimodular probing to guide the introduction of actions in an incremental planner.

However, work on the algorithm was originally driven by the commercial task of improving resource utilization. In resource utilization problems the number of activities is held constant while resources are removed to increase schedule efficiency. The task of the rescheduler is then to produce a consistent, minimally perturbed schedule. In this paper we define the features of the resource utilization problem, and use it to illustrate the deficiencies of constraint programming and mathematical programming approaches. Unimodular probing is empirically compared with two other commonly used techniques on randomly generated benchmark tests.

Section 2 gives initial definitions, and a model for minimal perturbation variants of the RFP. The following section outlines a constraint model for this variant. Section 4 gives details of the unimodular probing algorithm tailored for solving this dynamic RFP variant and others. The algorithm is compared empirically with a traditional constraint programming algorithm, and with mixed integer programming search in Sect. 5. Finally, Sect. 6 gives pointers to related work, and Sect. 7, discusses why unimodular probing leads to gains in performance, and suggests the class of problems to which it is suited.

2 PROBLEM DEFINITIONS

This section describes the resource utilization variant of the RFP. The minimal perturbation problem and the RFP are defined first.

The minimal perturbation problem is an extension of the *constraint satisfaction problem* (CSP). A CSP Θ is a 3-tuple (V, D, C) where V is a set of n variables $\{v_1, .., v_n\}$; D is a set of n corresponding domains $\{d_1, .., d_n\}$; and C a set of constraints (relations) on subsets of the variables in V. Let α be a complete assignment, mapping each variable $v_i \in V$ to a value in its corresponding domain $d_i \in D$. α is a solution to Θ iff each constraint in C holds under α.

A *dynamic CSP* (DCSP) is a sequence of CSPs, where each one differs from its predecessor by constraint additions or deletions [2]. Consider the case of a CSP Θ_i about to be transformed into a new CSP Θ_{i+1}. The transformation is captured in terms of two sets C_{del} and C_{add}, corresponding to the set of constraints about to be removed from, and added to the constraints C_i of Θ_i.

A solution to the *minimal perturbation problem* is a solution to the altered CSP with the additional requirement that it is "minimally different" to a previous solution.

Def. 1 : Minimal Perturbation Problem
A minimal perturbation problem Π is defined to be a 5-tuple $(\Theta_i, \alpha_i, C_{\text{del}}, C_{\text{add}}, \delta)$ where:

- Θ_i is a CSP;
- α_i is a solution to Θ_i;
- C_{del}, C_{add} are constraint removal and addition sets;
- δ is a function that measures the distance between two complete assignments. It is used to encapsulate perturbation.

A complete assignment α_{i+1} is a solution to Π iff it is a solution to $\Theta_{i+1} = (V, D, C_{i+1})$ and $\delta(\alpha_{i+1}, \alpha_i)$ is minimal, where C_{i+1} is given by $(C_i \setminus C_{\text{del}}) \cup C_{\text{add}}$.
In the *resource feasibility problem* (RFP), the goal is to fix the start and end times of activities such that resource capacities are not exceeded. For simplicity, we restrict here the general RFP model described in [3] such that there is only one resource type, and activities require only one resource. However, unimodular probing extends naturally to problems with multiple resource types [5].

[1] IC-Parc, Imperial College, London SW7 2AZ, UK. Email: {hhe,etr,mgw}@icparc.ic.ac.uk

© 1998 H. El Sakkout, T. Richards, and M. Wallace
ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

Def. 2 : Resource Feasibility Problem (RFP)
In the following, $\mathbb{N}$ represents the set of natural numbers, and # the set cardinality function. An RFP is composed of:

- a set A of possibly variable duration activities;
- for each $a_i \in A$, temporal start and end variables s_i, e_i;
- a resource usage variable r bounded by a capacity constraint $r \leq B \in \mathbb{N}$, and a constraint equating it to the maximum resource overlap:
  $r = \operatorname*{MAX}_{t \in \mathbb{N}} \left\{ \#\big(\{a_i : s_i \leq t < e_i\}\big) \right\}$
- a set L of linear inequalities that bound the temporal variables and impose distance constraints between them. Bounding constraints are of the form u R c, while distance constraints are of the form u R $v \pm$ c, where:
  R $\in \{ =, <, >, \leq, \geq \}, u, v \in \bigcup_{a_i \in A} \{s_i, e_i\}$ and c $\in \mathbb{N}$.

A solution to the RFP is an assignment to the start and end variables satisfying the problem's temporal and resource constraints.

The algorithms compared in this paper are applied to the *resource utilization problem*. It is defined as follows :

Def. 3 : Resource Utilization Problem
A resource utilization problem is a minimal perturbation problem $(\Theta_i, \alpha_i, C_{\text{del}}, C_{\text{add}}, \delta)$ where:

- Θ_i is a CSP capturing the constraints of an RFP with an activity set A and a resource bound B_i;
- α_i is a solution to Θ_i;
- $C_{\text{del}} = \phi, C_{\text{add}} = \{r \leq B_{i+1}\}$, such that $B_{i+1} < B_i$
- $\delta(\alpha_{i+1}, \alpha_i) = \sum_{u \in \bigcup_{a_k \in A} \{s_k, e_k\}} |\alpha_{i+1}(u) - \alpha_i(u)|$

This final problem is a good representative of dynamic scheduling problems with a linear perturbation cost function, because in non-trivial problems of this type, the required changes will lead to resource contention, as discussed in the following section.

3 CONSTRAINT MODEL

The RFP resource usage variable r corresponds to the number of resources required (the maximum resource overlap over the temporal horizon). In fact, it is sufficient to count resource overlap at the start times of activities, since these are the points where increases in resource usage occur. For each activity a_i, a variable r_{a_i} is introduced to count the number of overlapping resources at its start time s_i.

r_{a_i} variables are defined in terms of Booleans. A Boolean B_{ij} is introduced for each pair of activities a_i and a_j. It is set when activity a_j overlaps with the start of activity a_i.

$$\forall i, j.\ i, j \in \{1..|A|\} : B_{ij} = \left\{ \begin{array}{l} 1 \text{ iff } s_j \leq s_i \wedge s_i < e_j \\ 0 \text{ otherwise} \end{array} \right\} \quad (1)$$

When linked with the corresponding r_{a_i}, these Boolean variables link temporal and resource reasoning:

$$r_{a_i} = \sum_{j : 1 \leq j \leq |A|} B_{ij} \quad (2)$$

Each r_{a_i} is bounded by the maximum resource capacity B, via the constraints $r_{a_i} \leq r$ and $r \leq B$. Initially, the start and time variables are not fixed and this is reflected in the range of values r_{a_i} may take. By analyzing equation 2, a search procedure may determine potential contention points and prioritize them. Once one is chosen, contention is relieved by forcing apart temporal variables (i.e. imposing new distance constraints). Under certain conditions, this can cause the local consistency procedure to infer further distance constraints.

Unimodular probing handles dynamic scheduling problems in a uniform manner using the above contention point strategy; temporal inconsistencies (generated by imposing new temporal constraints) are transformed into resource contention [5]. Resource utilization problems are thus representative of other dynamic scheduling problems since they introduce resource contention directly.

4 UNIMODULAR PROBING

The resource utilization problem shows up the deficiencies of constraint programming (CP) and mathematical programming. CP algorithms are suited to exploring the disjunctive scheduling component, but local heuristics and consistency techniques are often not effective at global optimization. Conversely, in mathematical programming the optimization function is usually central to the search, but disjunctive constraints are hard to satisfy.

As a general strategy, unimodular probing counters these weaknesses on specific problem classes. Mathematical programming, in the form of linear programming (LP), is integrated into the CP search. By contrast with previous algorithms combining CP with LP, the algorithm passes only a restricted subset of the linear constraints to the LP solver. This is to ensure that the solver - Primal or Dual Simplex - works on constraints that have a property known as *total unimodularity* (TU) [7]. As detailed in Sect. 4.1 this property guarantees that the values returned by the LP algorithm are discrete. Instead of returning consistent solutions to a relaxed, non-discrete version of the discrete problem, LP is made to return values that are discrete but only partially consistent. In combination with CP, consistency is incrementally restored.

Applied to a sub-problem with TU, LP becomes a meaningful tool for solving *discrete* problems. In reality, resource scheduling problems are often discrete since they have temporal granularity.[2]

The unimodular probing algorithm should be considered for constraint optimization problems with the following features:

- A constraint subset with the TU property (the term *easy set* is later used to describe this subset);
- A linear optimization function on some or all of the variables in the easy set;
- Problem variables with discrete domain values.

4.1 Total Unimodularity

TU problems are well known in the mathematical programming community, and are components of many combinatorial optimization problems. Ordinary network flow problems, such as transportation, assignment and minimum cost network flow, have TU. The traveling salesman problem (TSP) is an example of a complex problem with an underlying "easy" network flow component. The temporal constraints of the RFP also have TU.

In LP, non-unary constraints are represented by means of a constraint matrix. The unary (bounding) constraints are handled separately, and do not violate TU when the bounds are integer. The necessary conditions for this property to hold in the matrix are not given here, but a more practical set of sufficient conditions are relayed [7].

[2] For example, the aircraft utilization problem driving our work required flights to be fixed with a 5 minute resolution. Rounding non-integer LP solutions can lead to inconsistency or sub-optimality.

A constraint matrix may be configured in two ways. In the first, rows represent constraints, and columns represent variables, with the element e_{ij} of the matrix denoting the coefficient of a variable j in constraint i. In the dual configuration, the reverse holds, with rows corresponding to variables and columns to constraints. Brackets are used to distinguish the dual and primal versions of the matrix.

Sufficient conditions:

1. All variable coefficients are 0, 1, or -1, and all constants are integer.
2. Two or less nonzero coefficients appear in each row (column).
3. The columns (rows) of the matrix can be partitioned into two subsets S_1 and S_2 such that
   (a) If a row (column) contains two nonzero coefficients with the same sign, one element is in each of the subsets.
   (b) If a row (column) contains two nonzero elements of opposite sign, both elements are in the same subset.

4.2 The Abstract Algorithm

An abstract version of the unimodular probing algorithm is given before describing its application to the RFP. In unimodular probing, local consistency techniques and a repair algorithm work on the *full set* of problem constraints, while LP handles only a restricted but dynamically changing easy set. The local consistency techniques applied depend on the specifics of the problem addressed by the algorithm. Section 4.3 gives the consistency methods selected for an algorithm designed to solve the RFP.

At each search step the unimodular probing strategy first applies chosen local consistency procedures to the full set and then LP to the easy, TU set. The discrete solutions returned by LP, called *unimodular probes*, satisfy the constraints in the easy set and are optimal with respect to the objective function. The algorithm identifies the constraints in the full set violated by the latest unimodular probe. If there are no violated constraints then the unimodular probe is returned as the optimal solution. Otherwise, the algorithm selects a violated constraint, and imposes an appropriate, new easy constraint. The new easy constraint corresponds to a repair, or a search decision that does not allow the variables of the violated constraint to hold the same values in subsequent probes.[3] A number of alternative repairs (new easy constraints) that reduce violation generally exist. The algorithm may backtrack through these choices in its search for optimality. Once a repair is selected, local consistency methods derive further easy constraints that follow from this choice. In this way, as repair choices are made, easy constraints are incrementally added to the TU set until either the LP optimum satisfies all the constraints of the full set, or inconsistency is proven. Backtracking is initiated on inconsistency as in conventional depth-first search.

Figure 1 gives the abstract search procedure. The search procedure utilizes four procedures, push_cp, pop_cp, push_lp and pop_lp to add and remove constraints from the local consistency (CP) and LP constraint stores, implementing a depth-first search. It is assumed that before search all the problem constraints have been added to the CP constraint store (including the Branch & Bound optimization cost bound), while initial and derived easy constraints have been added to LP's. unimodular_probing_search is then called with TRUE as the initial value for its argument C_{new}, representing the first (dummy) search decision.

push_cp pushes the constraint onto the CP constraint store, applying local consistency procedures and returning either newly derived constraints c_{cpnew}, or FALSE if inconsistency is proved. obtain_easy_constraints filters the new constraints $c_{\mathrm{new}} \cup c_{\mathrm{cpnew}}$ returning new easy constraints c_{lpnew} for addition to the LP constraint store. push_lp adds the constraints and invokes the LP solver, returning the LP assignment s_{lp} (the unimodular probe), or FALSE if no LP solution exists. The set of constraints violated under the LP assignment (c_{conf}) is obtained by violated_constraints. select_repair_choice selects one repair from the set of alternative repairs c_{repair}. The procedure recurses with this repair constraint, attempting to impose its negation on backtracking to search the alternative search branch. Appropriate constraint store removal routines implement backtracking on failure.

```
begin unimodular_probing_search($C_{new}$)
    $C_{cpnew}$:= push_cp($C_{new}$);
    if ($C_{cpnew} \neq$ FALSE) then {
      $C_{lpnew}$ := obtain_easy_constraints($C_{new} \cup C_{cpnew}$);
      $S_{lp}$:= push_lp($C_{lpnew}$);
      if ($S_{lp} \neq$ FALSE) then {
        $C_{conf}$:= violated_constraints($S_{lp}$);
        if ($C_{conf} = \emptyset$) then
          return $S_{lp}$
        else {
          $C_{repair}$:= select_repair_choice($C_{conf}$, $S_{lp}$);
          $S_{lp}$ := unimodular_probing_search($C_{repair}$);
          if ($S_{lp} \neq$ FALSE) then
            return $S_{lp}$
            else $S_{lp}$ := unimodular_probing_search(NOT($C_{repair}$));
          if ($S_{lp} \neq$ FALSE) then
            return $S_{lp}$;
        };
      };
      pop_lp;
    };
    pop_cp;
    return FALSE;
end unimodular_probing_search
```

Figure 1. Unimodular Probing Search

Thus LP extends backtrack search with a propagation method (it may detect failure), as well as the basis for variable and value selection heuristics (it suggests a solution that needs repair). In addition, the repair strategy limits the number of search nodes in a complete search by allowing termination when the conflict set is empty. Orthogonal local consistency techniques handle the disjunctive constraints that are out of LP's scope, and infer new LP constraints.

4.3 Unimodular Probing for the RFP

The creation of a unimodular probing algorithm involves three steps. First, a suitable constraint subset and the constraints linking it to the linear optimization function are proven to have TU. Second, the local consistency and heuristics are decided. Third, a procedure for extending the TU set is chosen.

4.3.1 The TU set and the optimization function

In the following we sketch a proof that the temporal constraints of the RFP, and the constraints linking the RFP's objective function to its temporal variables, constitute a TU set.

Lemma 1: *The temporal constraints of the RFP constitute a TU set.* All the non-unary temporal constraints (matrix rows) involve two nonzero coefficients of opposite sign. The variables (matrix columns) can be partitioned into a subset S_1 containing all columns, and another S_2 containing none. Thus the temporal constraints of the RFP satisfy the sufficient conditions as outlined in Sect. 4.1.

Equation 2 defined a δ function that measured perturbation in terms of a sum, over the temporal variables, of change from the previous solution. In LP, this function is modeled by introducing a variable

[3] The form of easy constraint imposed is dependent on the problem class addressed. In the simplest case it is a unary constraint bounding or fixing a variable.

d_x for each temporal variable x. x ranges over all temporal variables s_i and e_i (activity durations may not be fixed in the RFP). The absolute change d_x in a temporal variable x and is given by $d_x = |x - \mathrm{c}|$, where c represents the value given to x in the previous solution. This expression is captured by placing d_x in the optimization function and adding the following linear delta constraints:

$$d_x \geq x - \mathrm{c} \tag{3}$$

$$d_x \geq \mathrm{c} - x \tag{4}$$

Equation 4 does not satisfy the sufficient conditions for a TU set. The following lemma outlines a proof that Eqns. 3 & 4 do not destroy the TU property when added to a TU set.

Lemma 2: *If E is a set of constraints having TU then $E \cup \{d_x \geq x - \mathrm{c}, d_x \geq \mathrm{c} - x\}$ has TU, where d_x is unconstrained in E.* The Primal and Dual Simplex algorithms work by obtaining solutions at the vertices of the solution polyhedron, as defined by the linear inequalities of the problem. The TU property guarantees that the solutions at the vertices are discrete. The aim of the proof is to show that the new vertices created by the addition are also discrete.

Let E be a TU set. Let E be extended to E' by adding a new variable d_x and the two delta constraints 3 and 4. d_x adds a new dimension to the polyhedron, and cuts it along the hyperplanes $d_x = x - \mathrm{c}$ and $d_x = \mathrm{c} - x$. All existing vertices are projected in the d_x dimension onto these two hyperplanes. At the projected vertices d_x takes the value of either $x - \mathrm{c}$ or $\mathrm{c} - x$. Both values are discrete because c is an integer constant and x is discrete at the projected vertex since E has TU. In addition, new vertices are added where the two new hyperplanes intersect at $x = \mathrm{c}$. But the hyperplane $x = \mathrm{c}$ is a unary bounding constraint with an integer constant and as such guarantees the new vertices are discrete. Thus the extended set E' also has TU.

Theorem: *The temporal constraints of the RFP and the delta constraints of its objective function constitute a TU set.*

Proof: By induction over delta constraint pair addition. Base Case: The temporal constraints have TU. Inductive Step: If E has TU then $E \cup \{d_x \geq x - \mathrm{c}, d_x \geq \mathrm{c} - x\}$ has TU, where d_x is unconstrained in E.

4.3.2 *The local consistency and the search heuristics*

The algorithm applies arc-B consistency propagation on the problem's constraints [10]. An additional form of lookahead is applied after each decision; a minimum resource usage profile is built for affected portions of the schedule. Backtracking is initiated when the minimal usage exceeds the resource bound.

Resource contention constraints are ordered according to the degree to which they are violated. The most violated constraint is chosen for repair, and a local heuristic is used to select the least constraining temporal distance constraint viz. the one which yields the smallest reduction in the domains of its variables. This approach increases the chances of finding a good solution early.

The heuristics and lookahead check are based on an earlier algorithm for solving the RFP [12].

4.3.3 *Extending the TU set*

Since search decisions are made through the imposition of a new temporal distance constraint, that means at every search node a at least one new temporal constraint preserving TU is added to the set. As noted in Sect. 3, however, other temporal constraints may be derived from the subsequent constraint propagation sequence. The new temporal constraints are added to the TU set after each decision.

5 PERFORMANCE COMPARISONS

In our experience, the unimodular probing algorithm has been found to be substantially more effective than both an extended constraint programming algorithm and a mathematical programming approach – mixed integer programming (MIP) – on a set of commercial aircraft utilization problems. However, the results given here are for approximately 1200 randomly generated resource utilization problems. In these, the density of temporal constraints was varied as well as the required resource reduction. Details of these and other benchmarks have been made available on the Internet for future comparisons.[4]

Unimodular probing is compared with two algorithms. The first is an extension of a pure CP algorithm detailed in [12]. This algorithm, designated CP0, conducts the search in two search phases. In phase 1, CP0 reduces resource conflict by forcing apart temporal variables until resource conflicts are no longer possible. In phase 2, CP0 fixes the temporal variables to the values returned by an LP solver which are optimal given the choices made in the first phase. The search is complete because the phase 1 choices may be backtracked. The second algorithm is a commercial MIP package, CPLEX. The default search settings are used, with only the resource Booleans B_{ij} declared as integer (see Sect. 3). Other settings were attempted on a representative sample of 200 problems for both the node and variable selection strategies with minor or no improvement (3% in the best case). This confirms advice received from MIP practitioners that the default CPLEX settings are among the best for MIP search.

Table 1. Reduction in CP0 Solution Quality versus Density

Density	Quality of Best Solution Found (Avg. % worse than Optimal)
0	0.0
0.002	1.0
0.005	3.5

The results for CP0 are interesting. Most importantly, the algorithm rarely completes the search within the timeout interval. However, we note that for many problems it generates good solutions close to the optimal during the search, as shown in Table 1. The table's initial results demonstrate the sophistication of the heuristics used for search ordering, but show that the quality of the best obtained solution degrades as the number of distance constraints between tasks increases. This is because the local heuristics do not take into account the global dependencies that result from distance constraints. We believe that a bigger sample set on higher densities will reduce the quality of the best solution found by the algorithm even further. This would be consistent with our experience on the commercial problem.

It is possible to make a deeper comparison between unimodular probing and MIP search. In the first place, both searches complete the search. Secondly at every search node visited, both algorithms call the LP solver once. Figure 2 shows the substantial reductions in the number of search nodes that is achieved by unimodular probing. Table 2 shows that our relatively inefficient integration, which involves traversing all the problem variables twice at each search node, still yields substantial benefits in terms of CPU time. However, the CPLEX package's much tighter integration of Simplex reduces the CPU advantage by enabling more than ten times as many LP calls per second on average.

Note that the improvements of unimodular probing are despite a "flattening effect" whereby MIP gains an advantage because its more frequent timeouts are not penalized.

[4] http://www.icparc.ic.ac.uk/ ∼hhe/rfp_benchmarks.html

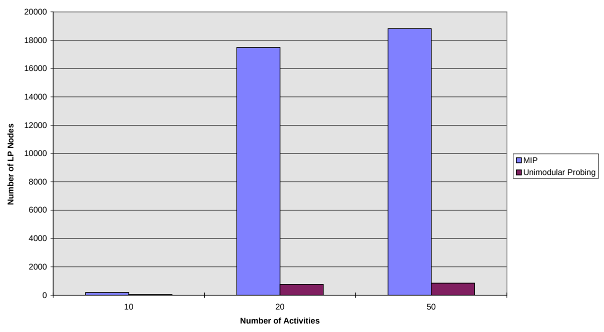

Figure 2. No. of LP Nodes versus No. of Activities

Table 2. CPU Seconds versus No. of Activities

No. Activities	UP Avg CPU/sec	MIP Avg CPU/sec
10	2	2
20	65	110
50	247	398

6 RELATED WORK

Many integrations of CP and LP are in existence [1, 9]. A general framework for the integration of LP with logical methods is presented in [8]. Unimodular probing may be viewed as a restriction of these frameworks since LP is limited to easy constraints. This is a natural progression from work on adaptive propagation [6]. By avoiding rounding or semantically weak interpretations of non-integer LP solutions in discrete problems, the restriction maintains algorithm soundness and completeness, and simplifies the interface.

In the CP community, the idea of using a "probe" assignment, or a set of temporary values to improve the effectiveness of backtrack search is increasingly utilized [13, 11]. The constraint model and a heuristic for solving the RFP were described in [3, 12]. [14] gives a scheduling strategy that bears similarities to the RFP unimodular probing algorithm; it resolves bottlenecks using probe assignments that are optimal with respect to criteria such as tardiness and inventory costs. The RFP algorithm described here differs because it minimizes perturbation, deploys a temporally consistent probe assignment, and searches a richer scheduling problem with variable durations and flexible temporal constraints.

Finally, an early version of this paper was presented in [4] and a deeper analysis and empirical study is given in [5].

7 DISCUSSION

The results of Sect. 5 stated the algorithm proposed is capable of proving optimality for many problems where a CP-based algorithm fails. The dramatic reduction in the number of LP solver calls by unimodular probing as opposed to MIP are also encouraging. This confirms our experience on a large scale real-world problem. Two questions arise from the results. Firstly, why do we see performance improvements, and secondly, for what class of problems do these improvements hold.

Unimodular probing extends CP backtrack search by focusing on the optimization function. Traditional approaches use the old solution as a static guide to search. The unimodular probes provide more useful information, because each probe satisfies the dynamically changing easy constraints while remaining optimal.

When comparing with MIP search there are clear advantages. First, it is possible to build good heuristics based on semantically meaningful information. Non-discrete values have a limited meaning in a discrete context, and MIP heuristics based on them may not lead to well informed search choices. Second, MIP will see a large number of violations of the discreteness requirement, in comparison with the relatively small set of resource constraint violations seen by unimodular probing. Hence the space of repairs is larger for MIP in the scheduling problems addressed. Third, local consistency methods are applied in unimodular probing achieving a significant reduction in the search space.

As noted, unimodular probing should be applied to discrete problems, with an optimization criterion that can be captured using a linear function, and a TU subset that includes the optimization variables. A feature of minimal perturbation problems is that the values of the infeasible optimal – the old solution – tend to already satisfy some of the problem's constraints. This additional factor may mean that minimal perturbation problems are especially suited to solution by unimodular probing.

ACKNOWLEDGEMENTS

We thank Euro-Decision for advice on MIP search, reviewer A and Claude Le Pape for valuable comments, and Stefano Novello and Joachim Schimpf for extending ECL^{i}PS^e.

REFERENCES

[1] H. Beringer and B. de Backer, 'Satisfiability of boolean formulas over linear constraints', in *IJCAI-93*, pp. 296–301, (1993).

[2] R. Dechter and A. Dechter, 'Belief maintenance in dynamic constraint networks', in *Proc. of AAAI-88*, pp. 37–42, (1988).

[3] A. El-Kholy and B. Richards, 'Temporal and resource reasoning in planning: The *parc*PLAN approach', in *Proc. of ECAI-96*, pp. 614–618, (1996).

[4] H. El Sakkout, T. Richards, and M. Wallace, 'Unimodular probing for minimal perturbance in dynamic resource feasibility problems', in *Proc. of CP97 workshop on Dynamic Constraint Satisfaction*, (1997).

[5] H. El Sakkout and M. Wallace, 'A strategy for minimal temporal perturbation in dynamic scheduling', *Constraints*, (1998). Submitted April.

[6] H. El Sakkout, M. Wallace, and B. Richards, 'An instance of adaptive constraint propagation', in *Proc. of CP96*, pp. 164–178. Springer Verlag, (1996).

[7] R. Garfinkel and G. Nemhauser, *Integer Programming*, John Wiley & Sons, 1972.

[8] J. Hooker and M. Osorio, 'Mixed logical/linear programming', *Discrete Applied Mathematics (to appear)*, (1998).

[9] J. Jaffar and J.-L. Lassez, 'Constraint logic programming', in *Proc. of the 14th ACM Symp. on Princ. of Prog. Langs.*, pp. 111–119, (1987).

[10] O. Lhomme, 'Consistency techniques for numeric csps', in *Proc. of IJCAI-93*, pp. 232–238, (1993).

[11] S. Minton, M. Johnston, A. Philips, and P. Laird, 'Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems', *Artificial Intelligence*, **58**, 161–205, (1992).

[12] D. Pothos, 'A constraint-based approach to the british airways schedule re-timing problem', Technical Report 97/04-01, IC-Parc, (1997).

[13] P. Purdom, Jr. and G. Haven, 'Probe order backtracking', *Siam Journal of Computing*, **26**, 456–483, (1997).

[14] N. Sadeh, 'Micro-opportunistic scheduling: The micro-boss factory scheduler', in *Intelligent Scheduling*, eds., M. Zweben and M. Fox, chapter 4, 99–136, Morgan Kaufman, (1994).