Applied Partial Constraint Satisfaction using Weighted Iterative Repair

JohnThornton[1] and Abdul Sattar[2]

[1] School of Information Technology, Griffith University Gold Coast,
Parklands Drive, Southport, Qld 4215
e-mail: j.thornton@eas.gu.edu.au

[2] School of Computing and Information Technology, Griffith University,
Kessels Road, Nathan, Qld, 4111
email: sattar@cit.gu.edu.au

Abstract. Many real-world constraint satisfaction problems (CSPs) can be over-constrained or too large to solve using a standard constructive/backtracking approach. Instead, faster heuristic techniques have been proposed that perform a partial search of all possible solutions using an iterative repair or hill-climbing approach. The main problem with such approaches is that they can become stuck in local minima. Consequently, various strategies or *meta-heuristics* have been developed to escape from local minima. This paper investigates the application of one such meta-heuristic, *weighted iterative repair*, to solving a real-world problem of scheduling nurses at an Australian hospital. Weighted iterative repair has already proved successful in solving various binary CSPs. The current research extends this work by looking at a non-binary problem formulation, and partial constraint satisfaction involving hard and soft constraints. This has lead to the development of a *soft constraint heuristic* to improve the level of soft constraint optimisation and an extension of the original weighted iterative repair that avoids certain forms of cyclic behaviour. It is also demonstrated that weighted iterative repair can learn from repeatedly solving the same problem. and that restarting the algorithm on the same problem can result in faster execution times. The overall results show that weighted iterative repair finds better quality solutions than a standard iterative repair, whilst approaching near optimal solutions in less time than an alternative integer programming approach.

1 Introduction

A constraint satisfaction problem (CSP) consists of a set of variables, $v_i \in V$, each with a domain of possible values D_i, and a set of constraints between variables. A CSP solution is an assignment of variable values which satisfy *all* the problem constraints [12]. However, many real-world CSPs are either too large to be solved completely, or are over-constrained so that a solution that satisfies all constraints is not possible. Such problems can be treated as *partial constraint satisfaction problems* (PCSPs), as only a partial subset of constraints can be or are satisfied in a solution [5]. The objective of partial constraint satisfaction is to solve a problem that is *as close as*

possible to the original CSP. The distance between problems can be measured using a *constraint hierarchy* [2], which defines weights or costs for constraint violations and categorises constraints as either *hard* (having to be satisfied) or *soft* (can be violated).

The presence of hard and soft constraints and of various levels of constraint weighting is typical of many large, over-constrained real-world scheduling problems. Examples include job-shop scheduling [4], university timetabling [6], telescope scheduling [8] and nurse rostering [3]. Such problems can be solved using standard constructive/backtracking techniques, if selected constraints are first relaxed so the problem is no longer over-constrained [4,3]. The difficulty here is recognising which constraints to relax (as some may be relaxed unnecessarily). Freuder and Wallace [5] propose a branch and bound algorithm for PCSPs which parallels the constructive approach but is able to find an optimum level of constraint relaxation. Alternatively, PCSPs can be expressed as optimisation problems and solved using integer and goal programming techniques [13]. However, in the worst case, all the above techniques use exponential time algorithms, and as problem size increases their use becomes more impractical.

Recent studies have indicated that heuristic algorithms can efficiently find optimal solutions to PCSPs [12,8]. Interest has focused on iterative repair techniques, which start with a complete but inconsistent solution and attempt to improve or repair the solution by changing individual variable assignments. The main problem with iterative repair is that it can become `stuck' on a local optimum solution and fail to find the global optimum. For this reason several *meta-heuristic* schemes have been proposed to guide iterative repair beyond local minimum solutions. These include randomising variable assignments [1], restarting the repair from different positions [9], and adding weights to violated constraints until a local minima is exceeded [9,10]. Of these techniques, constraint weighting has had the greater success in solving various classical binary constraint problems, and particularly in solving Conjunctive Normal Form (CNF) satisfiability problems (see [9] and [10]).

The current research introduces an independently developed constraint weighting algorithm (*weighted iterative repair*) that parallels the work of Morris [9] and Selman and Kautz [10]. The paper extends earlier research by looking at a real-world over-constrained problem with non-binary constraints, and introduces a *soft constraint* heuristic to handle hard and soft constraints, and a modified weighting heuristic that avoids certain types of cyclic behaviour. Experimentation has also revealed the heuristic has a limited learning ability which can be exploited by restarting a problem using previously learned weightings. The weighted iterative repair is tested against a standard iterative repair algorithm using a set of real-world nurse rostering problems. In addition, an integer programming application is used to *optimally* solve the rostering problems, and so provide an absolute standard from which the quality of other solutions can be evaluated.

The next section gives an introduction to the algorithms developed in the study. Section 3 describes the nurse rostering problem and details the experimental methods used to evaluate the algorithms. Section 4 gives the results of comparisons between algorithms and section 5 presents a discussion of these results. Finally, section 6 presents the conclusions.

2 Algorithms

2.1 Basic Iterative Repair

The approach of iterative repair is to improve on an existing inconsistent solution by selectively changing variable assignments until a consistent solution or a terminating condition is reached. In the basic iterative repair used in the study, each variable $v_i \in V$ is sequentially selected and all elements x in the domain D_i of v_i are tried in the solution S. The cost of S for each new x value is calculated by summing the current constraint violations, with the lowest cost x finally being assigned to v_i. If there is more than one lowest cost value then x is chosen randomly from the candidates. The domain values for the next variable are then tried and the process continues until no further improvement in S is possible. If S has not reached the desired cost, the algorithm is restarted with a different variable ordering, until a maximum number of restarts have occurred or a desired cost solution is reached.

The basic iterative repair algorithm can be categorised as a *local search* or a *cyclic descent* algorithm [7], and is similar to Johnson and Minton's min-conflicts heuristic [8], except that *hard* and *soft non-binary* constraints are considered. The hard and soft constraints are initially distinguished by being given different weights or penalties when calculating the cost of S. Weights are set so that the sum of all possible soft constraint violations cannot exceed the cost of a single hard constraint violation. This reflects the priority that the hard constraints *must* be satisfied.

2.2 Weighted Iterative Repair

The fundamental problem with a standard iterative repair is that it tends to get `stuck' in non-optimal solutions. This occurs when a superior solution can only be reached through a series of moves that involve at least one *cost-increasing* move. One strategy to exceed a local minima is to restart the algorithm at a new position each time it gets stuck (as in section 2.1). However, given a complicated cost surface, this can be a slow process and information from previous searches is lost (see [9]). Other strategies proposed to escape local minima include randomised move selection (as in simulated annealing [1]) and the selection of the best move that does not repeat a previous move (as in tabu search [6]). Both these approaches are complete, but they are also relatively slow. A more recently proposed strategy is to change the shape of the cost surface by dynamically increasing the weights of any violated constraints [9]. Whilst not complete, this approach has had promising results on various classical and random binary CSP's. The weighted iterative repair used in the current paper is an independently developed adaptation of earlier weighting algorithms, specifically designed to solve partial constraint satisfaction problems with hard and soft constraints. The basic iterative repair described in section 2.1 is adapted so that each time a local minima is encountered, a fixed quantity is added to the weight of each violated constraint (see fig 1). This changes the cost of the current solution until another *neighbouring* solution becomes preferred and the algorithm moves on. In effect, by recording weights against constraints the algorithm is *learning* which constraints are hardest to satisfy. These difficult constraints tend to be become

satisfied so the algorithm finds spaces where there is more freedom of movement. If a particular search space is unpromising, the combined weights of the newly violated constraints will eventually cause a previously "difficult" constraint to become violated and a new area will be explored.

The Soft Constraint Heuristic. As weighted iterative repair is continually changing constraint weights, any initial weight settings will soon become blurred and disappear. This can result in soft constraints acquiring equal or greater weighting than hard constraints and the unnecessary exploration of infeasible solutions. One answer is to only increment the weights of hard constraints, but this can mean little or no attempt is made to optimise the soft constraints. This is especially the case in problems where feasible solutions are difficult to find and little movement is possible to optimise soft constraints. To remedy this behaviour a *soft constraint heuristic* is proposed (see fig 1). The heuristic causes an iterative repair to consider all the moves that *reduce* the hard constraint cost and to accept the move that has the best soft constraint cost. This means the best overall cost assignment is not necessarily chosen. Instead, assignments are selected that improve hard constraint violations at the *least cost* to the soft constraints.

```
procedure WeightedIterativeRepair
begin
    StuckCounter ← 0, set variables to initial assignments
    while weighted cost of current state > DesiredCost do
        if current state is not a local minima then
            Improve ← False
            for each variable
                Moves ← select moves that reduce the hard constraint cost
                SoftMoves ← select moves that do not increase the total cost
                if Moves not empty then
                    perform move from Moves that has least soft constraint cost
                    Improve ← True
                else perform move from SoftMoves that has the least total cost
            end for
        else if Improve = True then
            increase weights of all violated hard constraints
            StuckCounter ← StuckCounter + 1
        else
            increase weight of hard constraint with greatest weighted violation
            StuckCounter ← StuckCounter + 1
        if StuckCounter > MaxStucks then
            reset variables to initial assignments
    end while
end
```

Fig. 1. The Weighted Iterative Repair Algorithm

Avoiding Cyclic Weight Escalation. Morris [9] demonstrated that cyclic behaviour is theoretically possible in weighted iterative repair giving an example of a Boolean Satisfiability problem with four variables, w, x, y, z, and the clause $w \vee x \vee y \vee z$ together with the 12 clauses:

$$\neg w \vee x, \neg w \vee y, \neg w \vee z, \neg x \vee w \neg x \vee y, \neg x \vee z,$$
$$\neg y \vee w, \neg y \vee x, \neg y \vee z, \neg z \vee w, \neg z \vee x, \neg z \vee y$$

The clauses have a single solution where all variables are *true*. Given all variables start as *false*, a weighted iterative repair will continually try each variable as *true* but will never arrive at a situation where two or more variables are *true* because two *true* instantiations are always more expensive than one. In more general problems, a group of interrelated constraints can have their weights repeatedly incremented before a better solution is found, where the weighting of a *single* constraint would have broken the deadlock immediately. A heuristic which solves both of these problems is to increment the weights of all violated constraints the *first time* a local minima is encountered (see fig 1). Then, if no reduced cost move is found only the highest weighted constraint is incremented (breaking ties randomly).

3 Experimental Study

The algorithms described in the previous section were evaluated on a real-world problem of rostering nurses in an Australian public hospital. The problem was chosen firstly because it represents a practical, complex, over-constrained problem, and secondly because an optimal integer programming algorithm already exists for the problem [13]. The complexity of the problem means a simplistic heuristic approach is inadequate, while the availability of optimal problem solutions provides a standard by which more sophisticated heuristics can be evaluated.

3.1 Nurse Rostering

Nurse rostering can be considered as a *partial constraint satisfaction problem*. The task is to find a consistent allocation of shift values, for a group of nurses, over a fixed period of time, that satisfy *as many as possible* of a set of rostering constraints. There are two basic types of constraints: (i) *schedule constraints* defining acceptable shift combinations for each nurse and (ii) *staff constraints* defining acceptable overall staffing levels for each day in the roster. Research into nurse rostering has focused on optimising integer programming solutions [13], and non-optimal heuristic solutions [7]. More recent attention has been placed on constraint programming with selective constraint relaxation [3]. The iterative repair approach developed in the current study most closely resembles the heuristic *coordinate cyclic descent algorithm* for nurse rostering first proposed by Miller *et al* [7]. For this reason it was decided to follow Miller's modelling of the problem. This involves considering each nurse as a variable, with the variable domain being the set of schedules that the nurse is able to work

[13,7]. The task is then to select a schedule from the set of all possible schedules for each nurse such that the best possible roster is generated (fig 2).

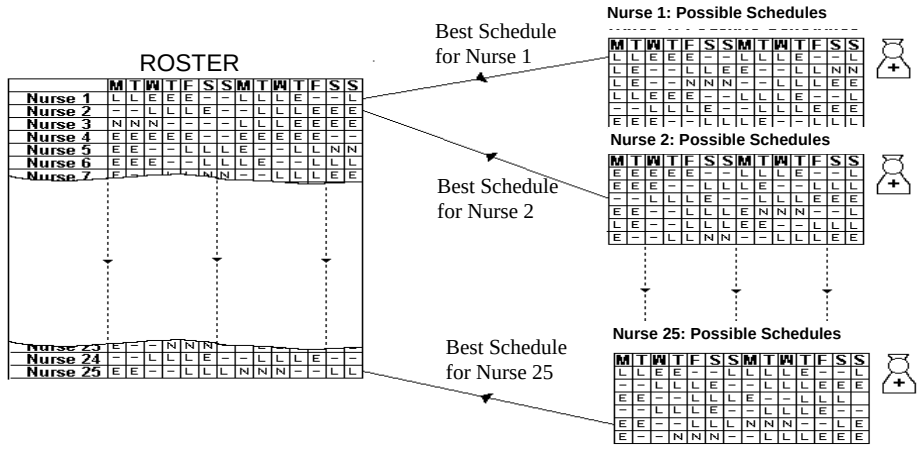

Fig. 2. The Possible Schedule Selection Approach to Rostering

In the current problem, a 30 bed medical ward is considered, employing between 25 to 37 nurses in each roster. There are three shift values covering a 24-hour period (an early, late and night shift) and each roster lasts 14 days. Nurses are allowed to request particular shifts or days off, with part-time *and* full-time staff included in the roster. Staff are further divided into five levels of seniority with maximum, minimum and desired staff constraints defined for each day of the roster and for each seniority level of nurse. Hard schedule constraints are defined specifying the type of schedule each nurse can work. These constraints are then used in a separate *schedule generation program* which builds all possible schedules for each nurse.

In addition, soft schedule constraints are defined to express which schedules a nurse would *prefer* to work. Based on interviews with nursing staff, weights were defined for each level of deviation for each constraint, and two roster quality measures were developed. The first equals the sum of all hard staff constraint violations for a roster, and the second equals the sum of all soft schedule constraint violations for a roster.

3.2 Experimental Methods

The study evaluates the two algorithms described in section 2 with a comparison of results generated from solving a test bed of 25 rostering problems. These problems were reconstructed from rosters actually worked on a hospital ward during 1993. The results were further compared with the original manually generated solutions and with *optimal* solutions generated using an Integer Programming (IP) package [13]. The soft constraint heuristic was tested by running the weighted iterative repair over the same problems twice, firstly with the heuristic activated and secondly by only selecting the

lowest overall cost moves (in neither case are the soft constraint weights incremented). In addition the weighted iterative repair algorithm was used to repeatedly solve identical roster problems, with the weight settings from one solution acting as the starting weights for the next solution. This was intended to test whether the algorithm can *learn* from past weightings and so find solutions more quickly.

4 Results

Table 1 summarises the experimental results. The iterative repair algorithms were set to stop at the first minima that satisfies all hard constraints, and in the event of not finding a minima, the basic iterative repair was set to stop after 50 restarts. Feasible solutions are defined as solutions that match the level of hard constraint satisfaction obtained from the integer programming algorithm. One integer programming solution was discarded because the execution time was in excess of 4 hours (execution times represent processor time in seconds for a 486 DX50 PC running linux).

	Weight + Soft	Weight - Soft	Basic	Integer	Manual
Mean Execution Time (secs)	93.8	81.24	305.9	576.7	n/a
Mean Soft Constraint Cost	390.7	399.1	390.2	340.5	519.2
% Feasible Solutions Found	100.0	100.0	60.0	96.0	48.0

Weight = weighted iterative repair, Soft = soft constraint heuristic, Basic = basic iterative repair
Integer = integer programming, Manual = manually solved rosters

Table 1. Mean scores for each roster generation method

The graph in figure 3 shows the results of using weighted iterative repair to solve each roster problem 20 times in succession with the weights from one solution acting as the start weights for the next. The normalised time is found by transforming the solution times for each of the 20 runs for a problem onto a scale of 0 to 1000 (0 for the fastest, 1000 for the slowest). Then solution 1 normalised time is the mean of the transformed first solution times for all 25 roster problems, and solution 2 normalised time is the mean of the transformed second solution times for all 25 roster problems, and so on.

5 Discussion

5.1 Execution Time

The experimental results show that weighted iterative repair is significantly faster than the other methods considered (this was confirmed with a further statistical analysis). Weighted iterative repair was also the only method that could reliably solve all 25 problems within a reasonable time frame (basic iterative repair was unable to solve 10 of the problems even after allowing an additional 50 restarts or approximately 2 hours of execution time per problem). In an attempt to improve the basic iterative repair, the algorithm was allowed greater freedom to search *sideways*

by making moves of equal cost (as with GSAT [10]), but no improvement in the search was observed.

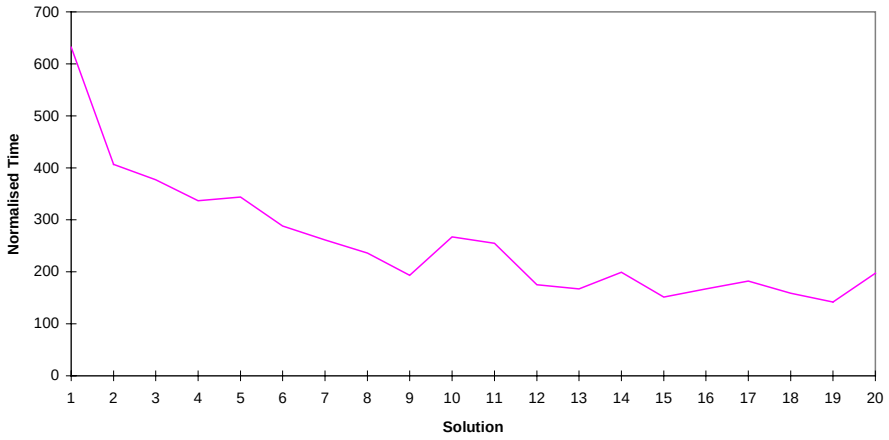

Fig. 3. Normalised times for repeated problem solving with weight feedback

5.2 Soft Constraint Cost

All the algorithmic techniques were able to convincingly improve on the soft constraint costs for the manual solutions (see table 1). This was practically confirmed by the use of an operational iterative repair system on the hospital ward during 1994-5. In addition, the soft constraint heuristic was found to make a small improvement in the soft constraint cost, without causing a large increase in execution time (this was verified by repeated solving of the test problems). The iterative repair algorithms were also able to *approach* optimal levels of soft constraint satisfaction, although the integer programming costs were significantly better.

5.3 Learning

The graph in figure 3 demonstrates that feeding back constraint weights from a solution into a repeated attempt to solve the same problem does cause execution times to reduce. This indicates the constraint weights do *learn* which constraints are hardest to satisfy, and that this information is useful in guiding a new search. The weighted iterative repair algorithm exploits this information by restarting after a fixed number of minima, whilst maintaining the current weightings (see fig 1). This differs from a normal restart strategy because information about the previous search is passed on to the next search. The restart strategy was developed because in 4 of the test problems a non-restarting algorithm occasionally became ‘lost’ and seemed unable to find a solution even after several hours of execution. After adding a restart strategy this

behaviour was eliminated. These results suggest the learning property of the weighted iterative repair could also be exploited to solve a series of very similar problems[1].

5.4 Overall

As a tool to solve nurse rostering problems, weighted iterative repair has proved most useful in practice. This is because rostering problems are generally solved repeatedly, due to initial infeasibilities and last minute changes in staff availabilities. In such circumstances, a fast response time and an ability to find the best *infeasible* solution are important. A capability to learn from a previous similar problem is also useful.

Overall, the application of weighted iterative repair to an over-constrained non-binary CSP has proved successful. However, results from using the soft constraint heuristic are inconclusive. A small improvement was found, but at the expense of a slightly reduced execution time.

6 Conclusions

The study of real-world over-constrained problems is important to extend the theoretical work already published on iterative repair and binary CSPs. The current work shows that weighted iterative repair has practical value in efficiently solving a real-world scheduling problem. The idea of constraint weighting has been extended to handle a non-binary system with hard and soft constraints, and the weighting algorithm has been extended to include a restart facility, a cycle avoidance strategy and a soft constraint heuristic.

Promising areas for further research include the exploitation and refinement of the learning ability of weighted iterative repair and the further exploration of heuristics to optimise soft constraint satisfaction.

References

1. D. Abramson. A very high speed architecture for simulated annealing. *IEEE Comp*., May:27-36, 1992.
2. A. Borning, B. Freeman-Benson and M. Wilson. Constraint hierarchies. In M. Jampel, E. Freuder and M. Maher, editors, *Over-Constrained Systems*, pages 23-62. Springer-Verlag, 1996.
3. B. Cheng, J. Lee and J. Wu. A constraint-based nurse rostering system using a redundant modeling approach. In *Proc. of the 8th IEEE International Conference on Tools with AI*, 1996.
4. M. S. Fox. ISIS: A Retrospective. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 3-28. Morgan Kaufman, 1994.

[1]Paul Morris came to a similar conclusion in looking at the Zebra problem (personal communication).

5. E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artif. Intell*., 58(1-3):21-70, 1992.
6. F. Glover. Tabu search - part 1. *ORSA Journal on Computing*, 1(3):190-206, 1989.
7. H. E. Miller, W. P. Pierskalla and G. J. Rath. Nurse scheduling using mathematical programming. *Ops. Res*., 24(5):857-870, 1976.
8. S. Minton, M. D. Johnston, A. B. Philips and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artif. Intell.*, 58:161-205, 1992.
9. P. Morris. The breakout method for escaping local from minima. In *Proc.of AAAI'93*, pages 40-45, 1993.
10. B. Selman and H. Kautz. Domain independent extensions to GSAT: Solving large structured satisfiability problems. In *Proc.of IJCAI'93*, pages 290-295, 1993.
11. J. R. Thornton and A. Sattar. An integer programming-based nurse rostering system. In *Proc. of ASIAN '96*, pages 357-358, Singapore, 1996.
12. R. J. Wallace and E. C. Freuder. Heuristic methods for over-constrained constraint satisfaction problems. In M. Jampel, E. Freuder and M. Maher, editors, *Over-Constrained Systems*, pages 207-216. Springer-Verlag, 1996.
13. D. M. Warner. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Ops. Res*. 24(5):842-856, 1976.