Solving Over-Constrained Temporal Reasoning Problems Using Local Search

Author

Beaumont, M, Thornton, J, Sattar, A, Maher, M

Published

2004

Conference Title

PRICAI 2004: TRENDS IN ARTIFICIAL INTELLIGENCE, PROCEEDINGS

Solving Over-constrained Temporal Reasoning Problems using Local Search *

M. Beaumont[1], J. Thornton[1], A. Sattar[1] and Michael Maher[2]

[1] School of Information Technology,
Griffith University Gold Coast,
Southport, Qld, Australia 4215
{m.beaumont, j.thornton, a.sattar}@griffith.edu.au
[2] Department of Computer Science,
Loyola University,
Chicago, IL 60626, USA
mjm@cs.luc.edu

Abstract. Temporal reasoning is an important task in many areas of computer science including planning, scheduling, temporal databases and instruction optimisation for compilers. Given a knowledge-base consisting of temporal relations, the main reasoning problem is to determine whether the knowledge-base is satisfiable, i.e., is there a scenario which is consistent with the information provided. However, many real world problems are over-constrained (i.e. unsatisfiable). To date, there has been little research aimed at solving over-constrained temporal reasoning problems. Recently, we developed standard backtracking algorithms to compute *partial scenarios*, in the spirit of Freuder and Wallace's notion of *partial satisfaction*. While these algorithms were capable of obtaining optimal partial solutions, they were viable only for small problem sizes.

In this paper, we apply local search methods to overcome the deficiencies of the standard approach to solving over-constrained temporal reasoning problems. Inspired by our recent success in efficiently handling reasonably large satisfiable temporal reasoning problems using local search, we have developed two new local search algorithms using a random restart strategy and a tabu search. Further, we extend our previous constraint weighting algorithm to handle over-constrained problems. An empirical study of these new algorithms was performed using randomly generated under- and over-constrained temporal reasoning problems. We conclude that 1) local search significantly outperforms standard backtracking approaches on over-constrained temporal reasoning problems; and 2) the random restart strategy and tabu search have a superior performance to constraint weighting for the over-constrained problems. We also conjecture that the poorer performance of constraint weighting is due to distortions of non-zero global minima caused by the weighting process.

* The authors gratefully acknowledge the financial support of the Australian Research Council, grant A00000118, in the conduct of this research

1 Introduction

Temporal reasoning plays an important role in many areas of computer science including planning [2], scheduling [7], natural language processing [10], temporal databases and instruction optimisation for compilers. Temporal information can generally be broken up into two categories, quantitative information and qualitative information. Quantitative information is specific numerical information about an event, whereas qualitative information is information about the relationship between events. This study is primarily concerned with qualitative temporal information.

Allen's interval algebra [1] models qualitative information about temporal problems by representing the relation between two events as a disjunction of up to thirteen possible atomic relations. The reasoning problem is then the task of finding a consistent labelling of every relation in the problem with one atomic relation from the disjunctive set of relations available. Traditionally interval algebra (IA) problems have been represented as binary temporal constraint satisfaction problems (TCSP), expressed as constraint networks. where the arcs between nodes represent relations and the nodes represent events.

An over-constrained TCSP is a TCSP that has no solution satisfying all the constraints; to "solve" such problems we look for a labelling that is consistent with a maximal number of constraints [4]. In [3], we developed a traditional backtracking approach to solve over-constrained IA problems. While our algorithm was capable of obtaining optimal solutions, it was only viable on small problem sizes. Even with the use of path consistency, the search space is not reduced sufficiently to find a solution in a practical time frame. To overcome this problem we turned to the local search paradigm.

Local search techniques, while not complete, have been shown to be effective on problems that are often too large for traditional backtracking to solve [8, 9, 11, 13]. Unfortunately, the standard approach of representing an IA problem as a TCSP proved impractical for a local search approach, as to find an accurate cost of a potential solution involves a significant search in its own right [3]. By remodelling the problem as a standard CSP using the *end point ordering* model [13] we were able to obtain the cost of potential solutions accurately without the need of a separate search, thus allowing us to apply a local search algorithm in a straight forward and efficient manner.

In this paper, we apply local search methods to overcome the deficiencies of the standard approach to solving over-constrained temporal reasoning problems. Inspired by our recent success [13] in efficiently handling a reasonably large (under-constrained) temporal reasoning problems using constraint weighting local search, we develop two new algorithms using a random restart strategy and a tabu search. Further, we extend our previous constraint weighting algorithm to handle over-constrained problems and present an empirical evaluation of all three algorithms.

The rest of the paper is organised as follows: Section 2 introduces Interval Algebra (IA). Section 3 describes how local search can be applied to temporal reasoning problems by reformulating them using end-point ordering. Section 4

describes local search algorithms for handling over-constrained temporal reasoning problems. Section 5 presents results and analysis of the empirical study. Finally, we conclude the paper with a few remarks on future work.

2 Interval Algebra

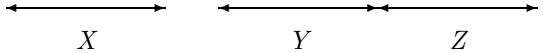
Allen's Interval Algebra (IA) provides a rich formalism for expressing qualitative relations between interval events [1]. In IA, a time interval X is an ordered pair of real-valued time points or *end-points* (X^-, X^+) such that $X^- < X^+$. Allen defined a set **B** of 13 basic interval relations such that any pair of time intervals satisfy exactly one basic relation. These relations capture the *qualitative* aspect of event pairs being before, meeting, overlapping, starting, during, equal or finishing each other. Indefinite information is expressed in IA as a disjunction of basic relations, represented as an *interval formula* of the form: $X\{B_1..B_n\}Y$ where $\{B_1..B_n\} \subseteq \mathbf{B}$. For example, the interval formula $X\{m, o\}Y$ represents the disjunction (X meets Y) or (X overlaps Y).
An IA problem has a solution if there is an assignment of an interval to each interval variable such that all interval relations are satisfied. An I-interpretation [6] maps each interval variable to an interval. It *satisfies* a basic relation $X\{B\}Y$ iff the end-points of the intervals assigned to X and Y satisfy the corresponding end-point constraints (see Table **??**). We say that an IA problem Θ is I-satisfiable iff there exists an I-interpretation such that at least one basic relation in each interval formula is satisfied. ISAT is the problem of deciding whether Θ is I-satisfiable and is one of the basic tasks of temporal reasoning [6]. This problem is known to be NP-complete [14] in general.

3 End Point Ordering

End-point ordering [13] translates the ISAT problem into a standard CSP, taking the end-point relations of interval formulas to be constraints and the time interval end-points to be variables. The main innovation of our approach is that we define the domain value of each time interval end-point to be the integer valued position or rank of that end-point within the *total ordering of all end-points*. For example, consider the following solution S to a hypothetical IA problem:

$$S = X\{b\}Y \wedge Y\{m\}Z \wedge Z\{bi\}X$$

Given the solution is consistent, a set of possible I-interpretations must exist that satisfy S. One member of this set is given by $I_a = (X^- = 12,\ X^+ = 15,\ Y^- = 27,\ Y^+ = 30,\ Z^- = 30,\ Z^+ = 45)$. For each I-interpretation, I_n, there must also exist a *unique* ordering of the time-interval end-points that corresponds to I_n. For example, the ordering of I_a is given by $(X^- < X^+ < Y^- < Y^+ = Z^- < Z^+)$ and is shown in the following diagram:



As any I-interpretation can be translated into a unique end-point ordering, it follows that the search space of all possible end-point orderings will necessarily contain all possible solutions for a particular problem. In addition, since it is the end-point ordering that is key – and not the values assigned to each endpoint, we can choose convenient values for the end-points. For example, we can assign an integer to each of the end-points in a way that respects the ordering (e.g. $X^- = 1$, $X^+ = 2$, $Y^- = 3$, $Y^+ = 4$, $Z^- = 4$, $Z^+ = 5$ for the above ordering).

Ideally, we would use the smallest range of integers necessary to represent the end-point orderings, but this is not practical as it would first involve finding all feasible solutions. However, the total number of end-points is an upper bound on the number of integers necessary. Thus we use the integers $1, 2, \ldots, 2m$ to represent the end-point orderings, where m is the number of interval variables in the IA problem.

4 Local search for over-constrained problems

4.1 Constraint Weighting

The original constraint weighting algorithm [13] works with the certainty that a solution to a problem exists and therefore only tracks the weighted cost (since when this cost is zero the unweighted cost will also be zero). As there are no zero cost solutions in an over-constrained problem, the algorithm will fail to recognise that a new optimum cost solution has been found, and at timeout will simply report failure. To solve over-constrained problems we extend the algorithm by tracking the unweighted cost at every move point in the cost function, shown by the FindBestMoves function in Figure 1, where the global variable *BestRealCost* holds the current optimum cost. The algorithm will still navigate the search space with the weighting heuristic but the best solution found so far in the search will only be recorded and replaced based on the unweighted cost.

4.2 TABU Search

The TABU search is a local search technique that relies on keeping a memory of the recent moves [5]. When a new move is selected, it is compared to the moves currently kept in memory and, if a match is found, this move is rejected as tabu. This prevents the algorithm from cycling back and forth between a few common moves and effectively getting stuck. If the move selected is not tabu and is different from the current value it is replacing, then the current value is made tabu and is replaced by the new move. The length of time that a value remains tabu for plays a vital role, if it is to large, then it becomes possible that all available moves are tabu and, if it is too small, it is possible for the algorithm to fall into a cycle and get stuck.

To improve the performance of our tabu search algorithm we allow it to make aspiration moves [5]. An aspiration occurs when there exists one or more tabu moves that could produce a better cost than the current best cost. In this

```
function FindBestMoves(Constraints, Cost, e_i^-, e_i^+)
  Moves ← ∅, OuterCost ← 0
  OuterConstraints ← all c_i ∈ Constraints involving (e_i^-, e_i^+)
  d_min^- ← min domain value of e_i^-
  while d_min^- ≤ max domain value of e_i^- do
    (TestCost, OuterCost, d_max^-) ← FindCost(e_i^-, d_min^-, OuterConstraints, OuterCost)
    if OuterCost > Cost then d_max^- ← max domain value of e_i^-
    else if TestCost <= Cost then
      InnerCost ← OuterCost, InnerConstraints ← OuterConstraints
      d_min^+ ← d_min^- + 1
      while d_min^+ ≤ max domain value of e_i^+ do
        (TestCost, InnerCost, d_max^+, RealCost)
                              ← FindCost(e_i^+, d_min^+, InnerConstraints, InnerCost)
        if RealCost < BestRealCost then
          BestRealCost ← RealCost
        if TestCost < Cost then
          Cost ← TestCost
          Moves ← ∅
        else if TestCost = Cost then Moves ← Moves ⊕ ((d_min^- ... d_max^-), (d_min^+ ... d_max^+))
        if InnerCost > Cost then d_max^+ ← max domain value of e_i^+ + 1
        d_min^+ ← d_max^+ + 1
      end while
    end if
    d_min^- ← d_max^- + 1
  end while
  return (Moves, Cost)
end
```

Fig. 1. The modified *FindBestMoves* TSAT Move Selection Function

instance the algorithm selects the first such move and instantiates it, ignoring that it is currently tabu. However, if non-tabu best cost improving moves exist, these will be preferred and an aspiration will not occur.

4.3 Random-Restart Search

The Random-Restart technique is a simplistic strategy for escaping a local minima. In the event the algorithm detects a local minimum, all the variables in the problem are randomly re-instantiated, and the search is restarted (the algorithm is deemed to be in a minimum when for a pre-defined number of loops the value for Cost has not changed). The Random-Restart algorithm is presented in Figure 4, using the same FindBestMoves presented in [13]. This is virtually the same function as FindBestMoves in Figure 1, except the code to calculate *BestRealCost* is removed (as Random-Restart does not use a weighted cost function).

5 Empirical Study

In our earlier work [3], we developed two backtracking based algorithms for handling over-constrained temporal reasoning problems. These algorithms guaranteed to find the optimal partial solution of the problem. However, our empirical

```
function FindMoves(Constraints, Cost, e_i^-, e_i^+)
  Moves ← ∅, OuterCost ← 0
  OuterConstraints ← all c_i ∈ Constraints involving (e_i^-, e_i^+)
  d^-_min ← min domain value of e_i^-
  while d^-_min ≤ max domain value of e_i^- do
    (TestCost, OuterCost, d^-_max) ← FindCost(e_i^-, d^-_min, OuterConstraints, OuterCost)
    InnerCost ← OuterCost, InnerConstraints ← OuterConstraints
    d^+_min ← d^-_min + 1
    while d^+_min ≤ max domain value of e_i^+ do
      (TestCost, InnerCost, d^+_max) ← FindCost(e_i^+, d^+_min, InnerConstraints, InnerCost)
      Moves ← Moves ⊕ ((d^-_min ... d^-_max), (d^+_min ... d^+_max), TestCost)
      d^+_min ← d^+_max + 1
    d^-_min ← d^-_max + 1
  end while
  Sort the Moves into ascending order of TestCost
  return (Moves)
end
```

Fig. 2. The Move Function for TABU

study was based on relatively small sized problems (we used problems with 8-10 nodes in the network with varying degrees).

In [13], we studied the application of local search to under-constrained (solvable) temporal reasoning problems. The main purpose of this study was to investigate practical value of local search techniques in this domain, which was largely unexplored. Our results indicated that a portfolio algorithm using TSAT (local search) and heuristic backtracking would be the best solution on the range of the 80 node problems we considered.

5.1 Problem Generation

For this study, we used Nebel's problem generator [6] to randomly generate problems using the $A(n, d, s)$ model, where n is the number of nodes or events, d is the degree size (defining the percentage of all possible arcs that are actually constrained) and s is the label size (defining the number of the thirteen possible atomic relations that are actually assigned to a constrained arc). As the results show, by varying the values of d and s it is possible to generate random problems that are either nearly all over-constrained or nearly all under-constrained.

5.2 Results

The purpose of our empirical study is to evaluate comparative performance of the extended weighting, TABU search and Random-Restart algorithms. We used a randomly generated test set using $n = 40$, $d = 25\%$, 50%, 75%, 100% and $s = 2.5, 9.5$, giving a total of 8 problem sets. To further evaluate the three algorithms, we re-tested the hard solvable problem set for 80 nodes used in our initial study [13]. Each problem set, except the hard set, contains 100 problems, and each problem was solved 10 times with a timeout of 15 seconds. The hard solvable problem set contains 318 problems which were also solved 10 times

```
procedure TABU(Events, Constraints)
  Randomly instantiate every event (e_i^-, e_i^+) ∈ Events
  Cost ← number of unsatisfied constraints ∈ Constraints
  TABULIST ← ∅
  while Cost > 0 do
    for each (e_i^-, e_i^+) ∈ Events do
      Add the range for (e_i^-, e_i^+) to TABULIST
      (Moves) ← FindMoves(Constraints, Cost, e_i^-, e_i^+)
      if the cost of the first Move ∈ Moves < Cost then remove every Move ∈ Moves ≥ Cost
      Aspiration ← first Move ∈ Moves
      while Moves ≠ ∅ do
        Remove the first Move ∈ Moves
        if (randomly selected (d_i^-, d_i^+) ∈ Move) ∉ TABULIST then
          Instantiate (e_i^-, e_i^+) with (d_i^-, d_i^+)
          Moves ← ∅
      if no Move ∈ Moves was instantiated then
        Instantiate (e^-, e^+) with randomly selected (d_i^-, d_i^+) ∈ Aspiration
      if cost of selected Move < Cost then Cost = cost of Move
  end while
end
```

Fig. 3. The TABU Local Search Procedure for Interval Algebra

each with a timeout of 30 seconds. In the results of Table 1, Cost refers to the least number of violated constraints found during a search, and Time and Number of Moves refer to the elapsed time and the number of changes of variable instantiation that had occurred at the point when the least cost solution was found. All experiments were performed on a Intel Celeron 450MHz machine with 160Mb of RAM running FreeBSD 4.2. For Tabu search we set the list length of the $TABULIST$ to be 50, for Random-Restart $RESTART$ was set at 250 and for Weighting MAX_FLATS was set to 4 and $MAX\_WEIGHTS$ was set to 10 (refer to [13] for a complete explanation of these variables).

5.3 Analysis

The experimental results indicate that the problem sets fall into two groups: one where nearly all problems had solutions ($n = 40$ $d = 25$ $s = 9.5$), ($n = 40$ $d = 50$ $s = 9.5$) and the original hard set ($n = 80$ $d = 75$ $s = 9.5$), and the remaining sets where nearly all problems were over-constrained. Looking at the results in Table 1, we can see that random re-start TABU search performs better than Weighting in terms of cost on all over-constrained problem sets. For instance, comparing the mean and min cost columns, Weighting is between 2% to 3% worse for the mean cost and 4% to 20% worse for the min cost (min cost being the minimum cost value found in all runs). In order to more clearly compare the relative performance of the algorithms, we plotted cost descent graphs for each algorithm against time. These graphs record the average best cost achieved at each time point for each problem set. Figure 5 shows a typical over-constrained descent curve (similar shapes were found for all other over-constrained problem sets). Here we see all three algorithms starting in a similar descent, but with Weighting starting to descend at a slower rate well before both TABU and

```
procedure Random-Restart(Events, Constraints)
  Randomly instantiate every event (e_i^-, e_i^+) ∈ Events
  Cost ← number of unsatisfied constraints ∈ Constraints
  RESTART ← 0
  while Cost > 0 do
    StartCost ← Cost
    for each (e_i^-, e_i^+) ∈ Events do
      (Moves, Cost) ← FindBestMoves(Constraints, Cost, e_i^-, e_i^+)
      Instantiate (e_i^-, e_i^+) with randomly selected (d_i^-, d_i^+) ∈ Moves
    if Cost < StartCost then RESTART ← 0
    else if (++RESTART) > MAX_RESTART then
      Randomly instantiate every event (e_i^-, e_i^+) ∈ Events
      TCost ← number of unsatisfied constraints ∈ Constraints
      if TCost < Cost then Cost ← TCost
      RESTART ← 0
  end while
end
```

Fig. 4. The Random-Restart Local Search Procedure for Interval Algebra

Random-Restart. A probable cause for the poorer performance of Weighting on the over-constrained problems is that by adding weight to unsatisfied constraints, a weighting algorithm distorts the original cost surface (i.e. by changing the relative cost of the constraints). In an under-constrained problem this will not change the relative cost of a solution, as this is always zero. However, in an over-constrained problem, the weighting process can *disguise* an optimal minimum cost solution by adding weights to the constraints that are violated in that solution. In that case, the search may be guided away from potentially optimal regions of the search space. As both TABU and Random-Restart are guided by the true unweighted cost, they are not subject to such misguidance.

Conversely, on all the under-constrained problem sets, Weighting has a distinct advantage, as shown in the results table and in the graph of Figure 6. This performance is paralleled in other studies that have shown weighting to outperform standard local search heuristics on a range of difficult constraint satisfaction

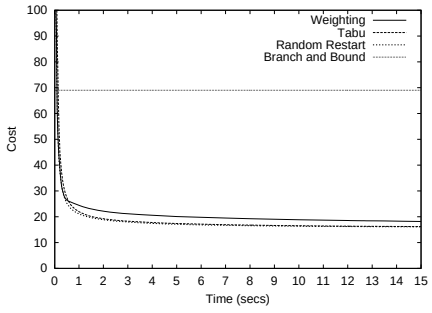


Fig. 5. Over-constrained descent graph for $n = 40$, $d = 75$, $s = 9.5$

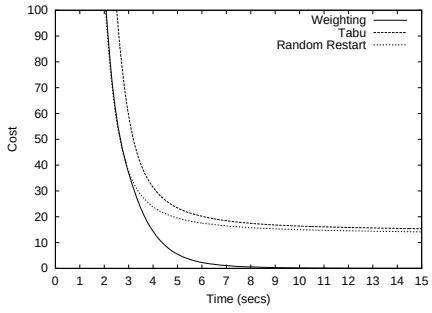


Fig. 6. Under-constrained descent graph for $n = 80$, $d = 75$, $s = 9.5$

		Solved	Cost				Number of Moves			Time
Problem	Method	%	Mean	Std Dev	Max	Min	Mean	Median	Std Dev	Mean
$n = 40$	*TABU*	0.00	61	8.69	91	37	16203	15838	4304	3.37
$d = 25$	*Random-Restart*	0.00	61	8.39	86	38	4166	4182	605	1.14
$s = 2.5$	*Weighting*	0.00	63	8.75	89	40	3234	3175	432	5.22
$n = 40$	*TABU*	72.00	0	0.59	2	0	1712	38	4111	0.06
$d = 25$	*Random-Restart*	91.00	0	0.31	2	0	83	39	128	0.03
$s = 9.5$	*Weighting*	100.00	0	0.00	0	0	30	30	4	0.03
$n = 40$	*TABU*	0.00	179	9.69	211	151	3598	3548	798	5.58
$d = 50$	*Random-Restart*	0.00	179	9.53	210	153	3339	3341	554	4.81
$s = 2.5$	*Weighting*	0.00	185	9.44	219	160	1756	1750	107	6.23
$n = 40$	*TABU*	0.60	3	1.59	13	0	3894	3977	2543	1.17
$d = 50$	*Random-Restart*	2.70	3	1.77	10	0	1264	1228	525	0.85
$s = 9.5$	*Weighting*	96.10	0	0.31	3	0	478	188	870	1.45
$n = 40$	*TABU*	0.00	310	10.65	341	282	1510	1469	297	7.16
$d = 75$	*Random-Restart*	0.00	310	10.53	338	280	1705	1663	346	7.32
$s = 2.5$	*Weighting*	0.00	318	10.49	351	290	1426	1415	96	6.99
$n = 40$	*TABU*	0.00	16	3.30	28	5	3328	3272	714	6.33
$d = 75$	*Random-Restart*	0.00	16	3.20	26	7	3218	3207	512	5.52
$s = 9.5$	*Weighting*	0.00	18	3.70	31	6	2952	2913	402	7.17
$n = 40$	*TABU*	0.00	433	7.72	454	410	905	892	172	8.38
$d = 100$	*Random-Restart*	0.00	433	7.51	454	405	1004	988	185	7.90
$s = 2.5$	*Weighting*	0.00	443	6.36	460	424	1252	1243	89	6.68
$n = 40$	*TABU*	0.00	37	4.70	50	24	1945	1905	391	8.18
$d = 100$	*Random-Restart*	0.00	36	4.70	55	25	2158	2109	413	8.31
$s = 9.5$	*Weighting*	0.00	45	4.65	58	29	2107	2075	287	6.61
$n = 80$	*TABU*	0.60	4	2.54	19	0	2092	2125	1081	8.33
$d = 75$	*Random-Restart*	3.18	4	2.60	19	0	1717	1666	710	7.41
$s = 9.5$	*Weighting*	99.97	0	0.02	1	0	215	200	69	4.80

Table 1. Experimental Results

and satisfiability problems [12]. The results and graphs also show there is little difference between the long-term performance of TABU and Random-Restart. This is somewhat surprising, as we would expect TABU to have an advantage over a simple restart (i.e. if TABU provides good guidance in escaping a local minimum this should lead us more efficiently to a more promising solution than randomly restarting the algorithm and losing all current instantiations). Random-restart is generally effective on cost surfaces where local minima occur discontinuously, i.e. where they occur singly and are fairly distant from each other. Our results may imply such a cost surface, or alternatively there may be more work needed in optimising TABU's performance (i.e, by experimenting with different heuristics and tabu list lengths).

To obtain a clearer picture of the advantages of local search in the over-constrained domain, we ran an existing branch and bound algorithm (known as Method 1 in [3]) on a range of the over-constrained problems. The graph in Figure 5 shows the descent curve of this algorithm on the ($n = 40$ $d = 75$ $s = 9.5$) problems in comparison to our three local search techniques (similar curves were obtained across the range of our over-constrained problem sets). These results showed branch and bound was unable to make any significant cost descent within a 100 second cut-off period.

6 Conclusion

We have demonstrated that a local search approach to solving over-constrained temporal reasoning problems is both practical and efficient. While we do not have an absolute measure of optimum cost for our problem sets (as no known complete algorithm is able to solve them), our 40 node graphs show that a local search is able to reach a flat area on a descent curve within a few seconds. This should be compared to the performance of existing backtracking techniques, which have trouble finding solutions for over-constrained random problems of greater than ten nodes [3]. We have also introduced and compared three new local search algorithms for over-constrained temporal reasoning. Our results indicate that the existing Weighting algorithm does not compare well to the relatively simple TABU and Random-Restart local search heuristics on over-constrained problems, but is still superior in the under-constrained domain.

Our work opens up several possibilities for further research. Firstly, existing work on constraint weighting has shown that hybrid constraint weighting and tabu search algorithms perform well on over-constrained problems with hard and soft constraints [12]. Hence, it would be interesting to explore such hybrid algorithms in the temporal reasoning domain. Additionally, as many real world problems resolve into hard (mandatory) and soft (desirable) constraints, it would be useful to extend our work to look at such realistic problems.

References

1. J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. J. Allen and J. Koomen. Planning using a temporal world model. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 741–747, Karlsruhe, W.Germany, 1983.
3. M. Beaumont, A. Sattar, M. Maher, and J. Thornton. Solving over-constrained temporal reasoning problems. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI 01)*, pages 37–49, 2001.
4. E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1):21–70, 1992.
5. F. Glover. Tabu search: Part 1. *ORSA Journal on Computing*, 1(3):190–206, 1989.
6. B. Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. *Constraints*, 1:175–190, 1997.
7. M. Poesio and R. Brachman. Metric constraints for maintaining appointments: Dates and repeated activities. In *Proceedings of the 9th National Conference of the American Association for Artificial Intelligence (AAAI-91)*, pages 253–259, 1991.
8. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.
9. Y. Shang and B. Wah. A discrete Lagrangian-based global search method for solving satisfiability problems. *J. Global Optimization*, 12:61–99, 1998.
10. F. Song and R. Cohen. The interpretation of temporal relations in narrative. In *Proceedings of the 7th National Conference of the American Association for Artificial Intelligence (AAAI-88)*, pages 745–750, Saint Paul, MI, 1988.

11. J. Thornton. *Constraint Weighting Local Search for Constraint Satisfaction*. PhD thesis, School of Information Technology, Griffith University Gold Coast, Australia, January 2000.
12. J. Thornton, S. Bain, A. Sattar, and D. Pham. A two level local search for MAX-SAT problems with hard and soft constraints. In *Proceedings of the Fifteenth Australian Joint Conference on Artificial Intelligence (AI 2002)*, pages 603–614, 2002.
13. J. Thornton, M. Beaumont, A. Sattar, and M. Maher. Applying local search to temporal reasoning. In *Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME-02)*, pages 94–99, 2002.
14. M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 377–382, 1986.