Applying Local Search to Temporal Reasoning

J. Thornton, M. Beaumont and A. Sattar
School of Information Technology,
Griffith University Gold Coast,
Southport, Qld, Australia 4215
{j.thornton, m.beaumont, a.sattar}@mailbox.gu.edu.au

Michael Maher
Dept. of Math. and CS,
Loyola University,
Chicago, IL 60626, USA
mjm@cs.luc.edu

Abstract

Local search techniques have attracted considerable interest in the Artificial Intelligence (AI) community since the development of GSAT [9] and the min-conflicts heuristic [5] for solving large propositional satisfiability (SAT) problems and binary Constraint Satisfaction Problems (CSPs) respectively. Newer SAT techniques, such as the Discrete Langrangian Method (DLM) [10], have significantly improved on GSAT and can also be applied to general constraint satisfaction and optimisation. However, local search has yet to be successfully employed in solving Temporal Constraint Satisfaction Problems (TCSPs).
In this paper we argue that current formalisms for representing TCSPs are inappropriate for a local search approach, and we propose an alternative CSP-based end-point ordering *model for temporal reasoning. In particular we look at modelling and solving problems formulated using Allen's interval algebra (IA) [1] and propose a new constraint weighting algorithm derived from DLM. Using a set of randomly generated IA problems, we show that our local search outperforms Nebel's backtracking algorithm [6] on larger and more difficult consistent problems.*

1. Introduction

Representing and reasoning with temporal information is a basic requirement for many AI applications, such as scheduling, planning and natural language processing [6]. In these domains temporal information can be *qualitative* as well as quantitative. For instance, an event may need to be before or during another event, but we may not be concerned with actual durations, start times or end times. Such information is not handled well using a simple linear time-stamping model, and requires more expressive constructs to capture the notion of events and the constraints between them. To answer this need, various approaches have been developed in the constraint satisfaction community under the heading of *Temporal Constraint Satisfaction*.
A Temporal Constraint Satisfaction Problem (TCSP) shares the basic features of a standard CSP, i.e. variables with domains and constraints that define the possible domain values that can be assigned to each variable [4]. However, in a TCSP constraints are modelled as *intensional* disjunctions of temporal relations [8] rather than as extensions of allowable domain value combinations. Finding a consistent scenario is then a matter of searching for a consistent set of temporal relations for each constraint. For harder problems this usually means using a combination of backtracking and a constraint propagation technique such as path-consistency [8].
In this paper we look at applying local search to solving TCSPs. Local search techniques such as GSAT [9] and the min-conflicts heuristic [5] have already proved effective both for propositional satisfiability (SAT) and in the general CSP domain, particularly on problems beyond the reach of standard constructive search methods. However, when applied to a TCSP, a local search is unable to exploit the constraint-propagation approach used with backtracking, as it is an incomplete method that *necessarily* moves through inconsistent scenarios. Further, local search requires *exact* cost feedback when deciding between candidate moves. In a binary CSP, a move changes a variable instantiation and cost feedback is obtained from a simple count of violated constraints. However, in a TCSP, a move consists of instantiating a set of temporal relations for a particular constraint. As no variables are actually instantiated, finding an exact move cost becomes a significant search problem in its own right [2].
Given these difficulties, our approach has been to reformulate temporal reasoning as a more standard CSP, i.e. searching by instantiating variables with domain values rather than instantiating constraints with temporal relations. Once in this form, a local search can be applied in a straightforward manner. The main task has been to develop a representation that does not cause an excessive increase in problem size. Our work has resulted in the *end-point ordering* model

for temporal reasoning, described in Section 3.3. To evaluate the model we have used Allen's Interval Algebra (IA) [1] and have developed an efficient *temporal tree constraint* representation to capture the full set of IA relations. Additionally, we propose a new constraint weighting local search algorithm for temporal reasoning, derived from a state-of-the-art SAT technique (the Discrete Lagrangian Method or DLM [10]). In Section 4.3 we give an empirical comparison of this approach with Nebel's backtracking algorithm and finally discuss the future direction of our work.

2. Interval Algebra

Allen's Interval Algebra (IA) provides a rich formalism for expressing quantitative and *qualitative* relations between interval events [1]. Additionally, reasoning with the full set of IA relations is known to be NP-complete [12]. For both these reasons (expressivity and difficulty) IA appeared ideal for the implementation of our local search approach.
In IA, a time interval X is an ordered pair of real-valued time points or *end-points* (X^-, X^+) such that $X^- < X^+$. Hence we can map actual time values to (X^-, X^+) that satisfy $X^- < X^+$. Such mappings are called interval- or I-interpretations [6]. Allen further defined a set **B** of 13 basic interval relations such that for any pair of time intervals the combined I-interpretation can be described by exactly one basic relation. These relations capture the *qualitative* aspect of event pairs being before, meeting, overlapping, starting, during, equal or finishing each other. As shown in Table 1, each relation can be defined in terms of constraints on the end-points of the constituent time intervals X and Y. Hence we can say that an I-interpretation *satisfies* a relation iff it satisfies these corresponding end-point constraints.
Indefinite information is expressed in IA as a disjunction of basic relations, known as an *interval formula*: $X\{B_1..B_n\}Y$ where $B_i \in$ **B**. For example, the interval formula $X\{m, o\}Y$ represents the disjunction (X meets Y) or (X overlaps Y). Using this notation, an IA problem can be simply represented as a finite set of interval formulas Θ. Further, we can say that 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 satisfiable and is one of the basic tasks of temporal reasoning [6].

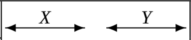
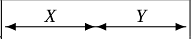
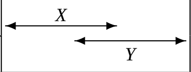
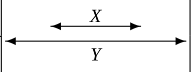
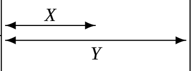
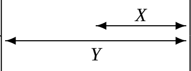
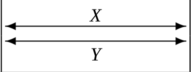
Basic Relation	Explanation	End-point Relations
b :*X* before *Y*		$(X^- < Y^-) \wedge (X^- < Y^+) \wedge$
bi :*Y* after *X*		$(X^+ < Y^-) \wedge (X^+ < Y^+)$
m :*X* meets *Y*		$(X^- < Y^-) \wedge (X^- < Y^+) \wedge$
mi :*Y* met by *X*		$(X^+ = Y^-) \wedge (X^+ < Y^+)$
o :*X* overlaps *Y*		$(X^- < Y^-) \wedge (X^- < Y^+) \wedge$
oi :*Y* olapped by *X*		$(X^+ > Y^-) \wedge (X^+ < Y^+)$
d :*X* during *Y*		$(X^- > Y^-) \wedge (X^- < Y^+) \wedge$
di :*Y* includes *X*		$(X^+ > Y^-) \wedge (X^+ < Y^+)$
s :*X* starts *Y*		$(X^- = Y^-) \wedge (X^- < Y^+) \wedge$
si :*Y* started by *X*		$(X^+ > Y^-) \wedge (X^+ < Y^+)$
f :*X* finishes *Y*		$(X^- > Y^-) \wedge (X^- < Y^+) \wedge$
fi :*Y* finished by *X*		$(X^+ > Y^-) \wedge (X^+ = Y^+)$
eq :*X* equals *Y*		$(X^- = Y^-) \wedge (X^- < Y^+) \wedge$
		$(X^+ > Y^-) \wedge (X^+ = Y^+)$

Table 1. The 13 basic interval relations (note: the relations $(X^- < X^+) \wedge (Y^- < Y^+)$ are implicitly assumed in each end-point relation)

3. Representing ISAT for Local Search

3.1. Current TCSP Approaches to ISAT

Current techniques for solving the ISAT problem follow the general TCSP approach outlined in the introduction [6, 11], using a combination of specialised path-consistency and backtracking algorithms. These techniques search for a consistent solution by *eliminating* basic relations from each disjunctive constraint (or interval formula). A significant group of tractable sub-classes of IA have been identified for which finding a path-consistent scenario is sufficient to guarantee full consistency [7]. These sub-classes are subsets of the 2^{13} possible interval formulas allowed in the full IA. IA algorithms exploit this information by searching for path-consistent scenarios that only contain formulas from a given tractable subset. This is more efficient than searching for a single basic relation from each formula. In addition, specialised ordering heuristics have been developed that further improve the performance of backtracking on full IA [11].

3.2. Local Search and TCSPs

Unfortunately, little of this work is of direct relevance in applying local search to IA. The basic principle behind a hill-climbing local search is to find the set of local moves (changes of instantiation) that most improve the overall solution cost [5]. In a TCSP approach to IA, a change of instantiation means changing the status of a basic relation contained in an interval formula: either it has currently been removed from the formula, hence it can be re-included, or it is currently included and can be removed. In either case we need to calculate the change in the overall solution cost.

The standard CSP approach would be to treat each interval formula as a constraint and to measure cost in terms of unsatisfied constraints. However, in a TCSP, the time interval end-points are not instantiated and so we cannot obtain a direct measure of the number of unsatisfied constraints. In fact, unless we infer information about end-point values, we can only measure the *consistency* of a solution and so can only distinguish between instantiations on the basis of consistency. This means a local search will need to test for the level of consistency of each competing instantiation to obtain the cost guidance needed to select a move. As such consistency checking would, at best, be equivalent to solving a problem using existing consistency-enforcing techniques [8], we can conclude that a local search of this sort will not achieve any benefits over existing approaches.

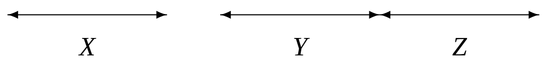
3.3. End-Point Ordering

As Θ can easily be expressed in conjunctive normal form (CNF), an obvious alternative for representing the ISAT problem is to translate Θ into a propositional satisfiability (SAT) formula. This would enable the application of existing SAT local search techniques without modification. However, as Nebel has already pointed out [6], expressing the implicit dependencies between time interval end-points in CNF produces a cubic increase in problem size, making it unlikely that a SAT approach will yield significant benefits. Consequently, our work has focussed on finding a more compact representation of the ISAT problem that still captures end-point dependencies. This has resulted in the *end-point ordering* model:

End-point ordering translates the ISAT problem into a standard CSP, taking the IA 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:



From this we can assign an integer ordering to each of the end-points, i.e. ($X^- = 1$, $X^+ = 2$, $Y^- = 3$, $Y^+ = 4$, $Z^- = 4$, $Z^+ = 5$). 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. The advantage of using end-point ordering is that we can now directly determine the truth or falsity of any interval formula whose end-points have been instantiated. For example, consider the interval formula $X\{m, o\}Y$ and the instantiation ($X^- = 2, X^+ = 4, Y^- = 3, Y^+ = 7$). From Table 1 it follows that $X\{m, o\}Y$ can be expanded to:

$$((X^- < Y^-) \wedge (X^- < Y^+) \wedge (X^+ = Y^-) \wedge (X^+ < Y^+)) \vee$$
$$((X^- < Y^-) \wedge (X^- < Y^+) \wedge (X^+ > Y^-) \wedge (X^+ < Y^+))$$

and substituting in the end-point order values gives:

$$((2 < 3) \wedge (2 < 7) \wedge (4 = 3) \wedge (4 < 7)) \vee$$
$$((2 < 3) \wedge (2 < 7) \wedge (4 > 3) \wedge (4 < 7))$$

resulting in $X\{m, o\}Y$ evaluating to true (note, this representation causes several redundant comparisons which are eliminated using the *temporal tree constraint* representation described in Section 4.2).

As an interval formula or constraint only contains ordering comparisons of the form $X\{<, =, >\}Y$, it follows that an end-point ordering is the *minimal* amount of information required to evaluate such a constraint. This further implies that an integer domain consisting of all feasible order values for a particular end-point is the smallest possible domain size for that variable that allows unambiguous constraint evaluation. Going back to our discussion in relation to local search, this is exactly what we required, i.e. the most compact model that also allows us to evaluate solution cost in terms of violated constraints.

In general, it is not practical to find the smallest possible domain size for each variable, as this would first involve finding all feasible solutions. However, we can set an upper bound E to the domain size, equal to the total number of end-points in a problem, such that each end-point domain is of the form $(1, 2, .., E)$. Depending on the method of constraint representation (i.e. binary or non-binary) we can then further prune domains using standard pre-processing techniques such as arc- and path-consistency.

In summary, the end-point ordering model expresses ISAT as the problem of ordering the end-points of each time-interval such that all the interval formulas in the problem are satisfied. We define the problem of deciding whether such an ordering exists as the OSAT problem and an O-interpretation as a mapping of time order positions onto time interval end-points. As every I-interpretation has a corresponding O-interpretation, it follows that iff a solution exists to the OSAT problem (i.e. it is O-satisfiable) there necessarily exists an I-interpretation that satisfies Θ (i.e. it is also I-satisfiable).

4. Solving OSAT using Local Search

4.1. Constraint Weighting Local Search

A local search differs from a constructive technique (such as backtracking) as the search begins with a complete, but inconsistent, instantiation of variables. It then proceeds to repair the solution by making a series of local moves that minimise the overall cost [5]. The crucial questions for a local search are: how to measure solution cost, choosing a local move operator and what to do when no local move exists that can reduce the overall cost. For OSAT the solution cost has already been defined, i.e. it is a count of the number of false interval formulas for a given variable instantiation. However, the question of defining a local move is still open: In a standard binary CSP, a move involves changing values for a single variable. When applied to end-point ordering this approach would search by changing single end-points. Alternatively we can define a move in terms of intervals and search by simultaneously changing the interval start and end points. This *interval domain* approach tries every possible position for a given interval, ensuring that the best domain value pairs are found, but also performing a greater number of comparisons. In preliminary tests the improved guidance of the interval domain outweighed the comparison cost and so we continued with this approach in our final algorithm.
To deal with situations where no improving move exists we have adopted the general DLM SAT trap escaping strategy proposed in [10]. We chose DLM as it represents the current state-of-the-art for SAT problems and can be simply adapted to the general CSP domain. DLM escapes traps by adding weight to all currently violated constraints. Cost is measured as the sum of weights on violated constraints, hence adding weight changes the *cost surface* of the problem, producing alternative cost reducing moves. In addition, DLM periodically reduces constraint weights to avoid losing sensitivity to local search conditions. The TSAT algorithm (see Figure 1) applies the basic DLM heuristics to the temporal reasoning domain, and is controlled by three parameters: MAX_FLATS (set to 4) which specifies how many consecutive non-improving (flat) moves can be taken before constraint weights are increased, MAX_WEIGHTS (set to 10) which specifies how many constraint weight increases can occur before the weights are reduced and MAX_FLAT_WEIGHTS (set to 50) which specifies how many consecutive weight increases can occur without an improving move before the search is randomly restarted [1].

4.2. Temporal Tree Constraints

Although there are 2^{13} possible disjunctions of the 13 basic IA relations, evaluating these disjunctions as interval end-point constraints is relatively easy. This is because all constraints involve four basic evaluations:

$$((X^-\{r\}Y^-), (X^-\{r\}Y^+), (X^+\{r\}Y^-), (X^+\{r\}Y^+))$$

where $r = \{<, =, >\}$ and any fully instantiated pair of intervals *must* satisfy a single basic relation [6]. This is illustrated in the comparison tree of Figure 2: here all constraints that evaluate *true* follow a single path from root to leaf, skipping the bracketed comparisons (as these are implied by $X^- < X^+$ or $Y^- < Y^+$). For example, the shortest path to *b* (assuming the best ordering) is given by:

$$(X^- < Y^-) \wedge (X^+ < Y^-)$$

as $(X^- < Y^-) \rightarrow (X^- < Y^+)$ and $(X^+ < Y^-) \rightarrow (X^+ < Y^+)$. Similarly, the longest path to *oi* (assuming the worst ordering) is given by:

$$\neg(X^- < Y^-) \wedge \neg(X^- = Y^-) \wedge \neg(X^- = Y^+) \wedge \\ \neg(X^- > Y^+) \wedge \neg(X^+ < Y^+) \wedge \neg(X^+ = Y^+)$$

Using interval formulas, we can construct comparison trees for *each* member of the subset of the 2^{13} possible disjunctions that appear in a particular problem. We term this type

```
procedure TSAT
    Randomly instantiate every event $(e_i^-, e_i^+) \in Events$
    $Cost \leftarrow$ number of unsatisfied constraints in $Events$
    $FlatMoves \leftarrow FlatWeights \leftarrow WeightIncreases \leftarrow 0$
    while($Cost > 0$)
        $StartCost \leftarrow Cost$
        $Moves \leftarrow \emptyset$
        for each $(e_i^-, e_i^+) \in Events$ do
            let $D_i$ be the domain of $(e_i^-, e_i^+)$
            for each domain pair $(d_{ij}^-, d_{ij}^+) \in D_i$ do
                $TestCost \leftarrow$ cost of $(e_i^- \leftarrow d_{ij}^-, e_i^+ \leftarrow d_{ij}^+)$
                if $TestCost < Cost$ then
                    $Cost \leftarrow TestCost$
                    $Moves \leftarrow \emptyset$
                end if
                if $TestCost = Cost$ then add $(d_{ij}^-, d_{ij}^+)$ to $Moves$
            end for
            Randomly select and instantiate $(d_{ij}^-, d_{ij}^+) \in Moves$
        end for
        if $Cost < StartCost$ then $FlatMoves \leftarrow FlatWeights \leftarrow 0$
        else if (++$FlatMoves$) $>$ MAX_FLATS then
            increment weight on all unsatisfied constraints
            increase $Cost$ by the number of unsatisfied constraints
            $FlatMoves \leftarrow 0$
            if (++$WeightIncreases$) $>$ MAX_WEIGHTS then
                decrement weight on all constraints with weight $> 1$
                decrease $Cost$ by number of decremented constraints
                $WeightIncreases \leftarrow 0$
            else if (++$FlatWeights$) $>$ MAX_FLAT_WEIGHTS then
                Randomly instantiate every event $(e_i^-, e_i^+) \in E$
                $Cost \leftarrow$ number of unsatisfied constraints in $E$
                $FlatWeights \leftarrow WeightIncreases \leftarrow 0$
            end if
        end if
    end while
end
```

Figure 1. The TSAT Local Search Algorithm for Temporal Reasoning

[1]DLM does not use a random restart strategy

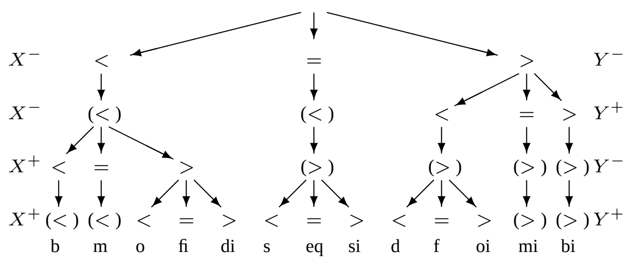


Figure 2. End-point Comparison Tree for the 13 Basic Relations

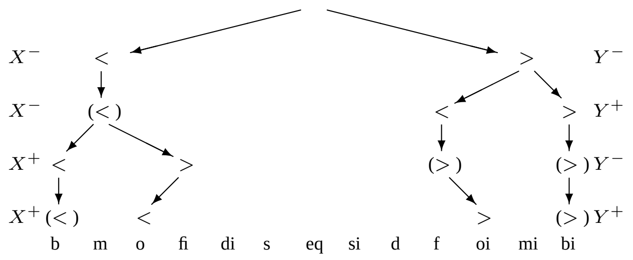


Figure 3. The Temporal Tree Constraint for $X\{b, bi, o, oi\}Y$

of constraint representation a *temporal tree constraint*. Processing these trees we can then detect *failure* with fewer comparisons, leaving the best and worst cases for success unchanged. The tree in Figure 2 represents the temporal tree constraint for all 13 possible disjunctions between X and Y and so is redundant (i.e. X and Y are unconstrained). Figure 3 shows the more useful temporal tree constraint for $X\{b, bi, o, oi\}Y$. Here we can see that an instantiation of $X^- = Y^-$ will fail at the first level and no further processing of the tree will occur.

An alternative method of constraint representation would be to express the problem as a true binary CSP, developing binary constraint extensions representing all possible combinations of end-points for a given pair of intervals. In such a model a constraint could be evaluated in a single look-up. However, we rejected this approach due to the large space overhead required.

4.3. Results

The TSAT algorithm is specifically intended to solve problems which are too large or difficult for a standard backtracking and path-consistency approach. To address this we set out to create a problem set on which backtracking has difficulty. Using Nebel's generator [6], we firstly created two large sets of random, consistent problems. The first set was made up of 40 node problems with degree = 75% and label size = 9.5. In the second set the number of nodes was increased to 80. We then ran both test sets on Nebel's backtracking-based problem solver [6] until 100 problems were found in each set that backtracking had failed to solve (the 40 node problems were timed out after 5 minutes and the 80 node after 10 minutes). We then solved each of these problems 10 times using the TSAT algorithm (all experiments were conducted on a Intel Pentium Celeron 450MHz machine with 160Mb of RAM running FreeBSD 4.2).

The graphs in Figures 4 and 5 show the proportion of problems solved against the average run-times for TSAT. Each problem is considered solved if an answer is found in at least one of the 10 runs of TSAT. The graph run-times are then calculated by dividing the total time taken for all 10 runs by the number of successful runs (in parallel with Nebel's algorithm, TSAT was timed out at 5 minutes for the 40 node problems and 10 minutes for 80 nodes). As TSAT solved all problems in less than 10 runs, both graphs show 100% success, whereas the % solved value in Table 2 reports the *overall* success rate for the 1000 runs on each problem set.

The 40 node problem results indicate that TSAT finds these instances relatively easy, with 100% of problems solved within 20 seconds and a median run-time of 4.11 seconds. This is in contrast to Nebel's algorithm which failed to solve any of these problems after 300 seconds. As would be expected, the 80 node problems proved harder for TSAT, with 18% failure at 600 seconds and a median run-time of 215 seconds. However, on average, TSAT is still able to solve *any* of the randomly generated 80-node instances in less than the 600 second time-out at which Nebel's algorithm was terminated.

While it can be argued that we have chosen just those problems on which Nebel's algorithm has difficulty, and hence that our comparison is biased, our intention is not to show that TSAT is superior to backtracking in all situations, just in those cases where backtracking has difficulty. If the lessons from SAT are carried across to the temporal reasoning domain, we would expect backtracking to be better on smaller/easier problems and always to be used when the objective is to prove inconsistency. The current TSAT algorithm represents a first pass at using local search for temporal reasoning and so has not been optimised for the temporal reasoning domain. For instance, the TSAT parameter settings were taken directly from the SAT algorithm from which it was developed. Also, given the relatively small number of average moves taken during the search (224 for 40 nodes and 1235 for 80 nodes), it is unlikely the TSAT weight reduction heuristic is having a significant effect. Finally, TSAT is performing many redundant consistency checks, which we anticipate can be eliminated us-

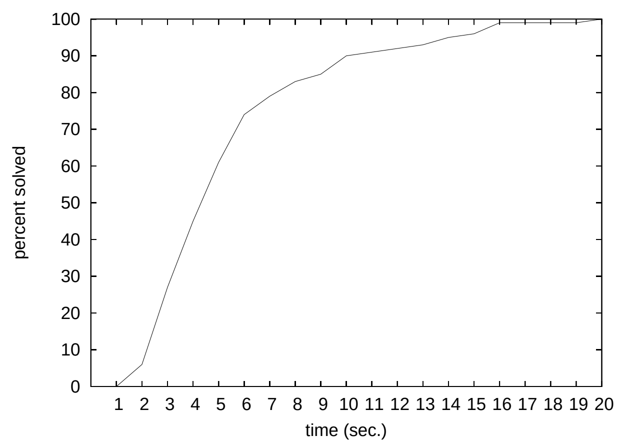


Figure 4. TSAT plot for 40 nodes

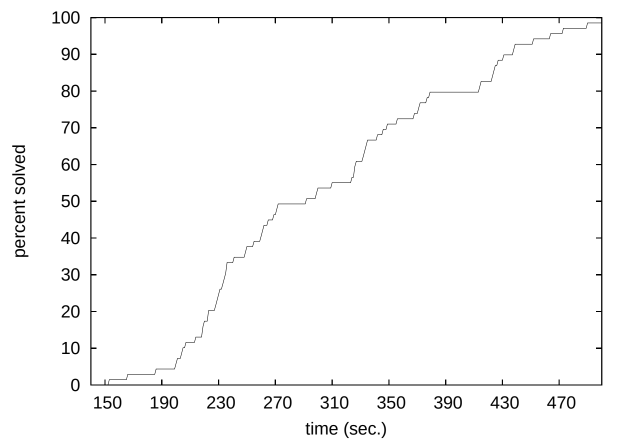


Figure 5. TSAT plot for 80 nodes

ing simple domain skipping heuristics. We are currently addressing all these areas and expect to present a more detailed empirical study of the relative merits local search for temporal reasoning in our future work.

5. Conclusion

In conclusion, the paper has demonstrated that an end-point ordering local search approach to temporal reasoning is both feasible and practical. The TSAT algorithm is a first indication that local search can outperform the traditional TCSP backtracking approach on larger, more difficult problems. Our work opens up several avenues for further research. Firstly, we have not explored alternative local search heuristics, such as tabu search or random walk. Also the TSAT algorithm can be further improved with the use of domain skipping techniques that avoid redundant tests and we have yet to perform an exhaustive search for the optimum TSAT parameter settings. Finally, local search appears especially promising for over-constrained temporal reasoning problems, where standard consistency-checking techniques become ineffective [2, 3].

		CPU Time			Number of Moves		
Problem Size	% Solved	Mean	Median	Std Dev	Mean	Median	Std Dev
40 nodes	100.0	6.38	4.11	8.43	339	224	418.14
80 nodes	82.0	244.21	215.10	116.48	1415	1235	666.81

Table 2. Average TSAT results for 1000 runs on each problem set

References

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] 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.

[3] E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1):21–70, 1992.

[4] A. Mackworth. Constraint satisfaction. Technical report, TR-85-15, University of British Columbia, Vancouver, Canada, 1985.

[5] 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.

[6] B. Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. *Constraints*, 1:175–190, 1997.

[7] B. Nebel and H. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[8] E. Schwalb and L. Vila. Temporal constraints: A survey. *Constraints*, 3:129–149, 1998.

[9] 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.

[10] Y. Shang and B. Wah. A discrete Lagrangian-based global search method for solving satisfiability problems. *J. Global Optimization*, 12:61–99, 1998.

[11] P. van Beek and D. Manchak. The design and an experimental analysis of algorithms for temporal reasoning. *Journal of AI Research*, 4:1–18, 1996.

[12] 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.