GRIFFITH UNIVERSITY

A Local Search Approach to Modelling and Solving Interval Algebra Problems

Author

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

Published

2004

Journal Title

Journal of Logic and Computation

Rights statement

© 2004 Oxford University Press. Please refer to the link for the definitive publisher-authenticated version.This is a pre-copy-editing, author-produced PDF of an article accepted for publication in Journal of Logic and Computation following peer review. The definitive publisher-authenticated version J Logic Computation 2004 14: 93-112 is available online at: http://logcom.oxfordjournals.org/cgi/reprint/14/1/93

Downloaded from

http://hdl.handle.net/10072/5165

Link to published version

http://logcom.oxfordjournals.org/cgi/content/abstract/14/1/93

Griffith Research Online

https://research-repository.griffith.edu.au

A Local Search Approach to Modelling and Solving Interval Algebra Problems *

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}@griffith.edu.au

Michael Maher
Department of Computer Science,
Loyola University,
Chicago, IL 60626, USA
mjm@cs.luc.edu

Abstract

Local search techniques have attracted considerable interest in the artificial intelligence community since the development of GSAT and the min-conflicts heuristic for solving propositional satisfiability (SAT) problems and binary constraint satisfaction problems (CSPs) respectively. Newer techniques, such as the discrete Langrangian method (DLM), have significantly

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

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) 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 existing consistency-enforcing algorithms on those problems that the existing techniques find most difficult.

1 Introduction

Representing and reasoning with temporal information is a basic requirement for many AI applications, such as scheduling, planning and natural language processing [8]. 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 [6]. However, in a TCSP, constraints are modelled as *intensional* disjunctions of temporal relations [10], 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 [10].

In this paper we look at applying local search to solving TCSPs. Local search techniques such as GSAT [11] and the min-conflicts heuristic [7] 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 [12]). In Section 5 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]. 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* aspects 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 time intervals X and Y. 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 $\{B_1..B_n\} \subseteq$ **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 [8] maps each interval variable to an interval. It *satisfies* a basic relation $X\{B\}Y$ iff the real-valued end-points of the intervals assigned to X and Y satisfy the corresponding end-point constraints (see Table 1). 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 [8]. This problem

is known to be NP-complete [16].

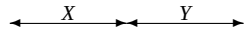
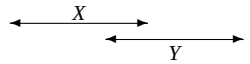
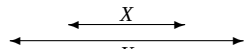
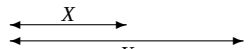
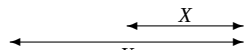
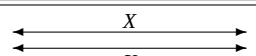
Basic Relation	Symbol	Diagram of Meaning	End-point Relations
X before *Y*	b		$(X^- < Y^-) \wedge (X^- < Y^+) \wedge$
Y after *X*	bi		$(X^+ < Y^-) \wedge (X^+ < Y^+)$
X meets *Y*	m		$(X^- < Y^-) \wedge (X^- < Y^+) \wedge$
Y met by *X*	mi		$(X^+ = Y^-) \wedge (X^+ < Y^+)$
X overlaps *Y*	o		$(X^- < Y^-)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

Table 1: The set **B** of 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 TCSP approach outlined in the introduction [8, 15], using a combination of specialised path-consistency and backtracking algorithms. Applying path-consistency to an IA network, involves testing the consistency of each basic relation B_i in each interval formula $X\{B_1..B_n\}Y$, such that for each path between X and Y passing through a third variable Z, an instantiation $X\{B_j\}Z$ and $Z\{B_k\}Y$ exists that is consistent with

$X\{B_i\}Y$. If no such instantiation exists then $X\{B_i\}Y$ is inconsistent and is deleted from $X\{B_1..B_n\}Y$. For example, if we are testing $X\{b\}Y$ and the path ($X\{bi, eq\}Z$, $Z\{bi, eq\}Y$), then $X\{b\}Y$ is inconsistent as X cannot be simultaneously before Y and equal to or after Z, when Z is also equal to or after Y.

As path-consistency does not guarantee global consistency, except when each arc is labelled with a singleton [10], a further search is needed to find a consistent scenario. Typically TCSPs are solved using a combination of backtracking and forward-checking, which proceeds by instantiating an interval formula with a single basic relation and then checking that the remaining formulae are path-consistent. If so, the algorithm instantiates another interval formula with a single relation, and so on, until an inconsistency is found, or all formulae are instantiated with a single basic relation (thus arriving at a globally consistent scenario). In the event of an inconsistency, the algorithm tries another instantiation in the current formula, and, if no further instantiations remain, it *backtracks* to an earlier formula and tries another instantiation. As backtracking is a complete search, it will either find a consistent scenario, or it will backtrack to the point where no further instantiations are available, and report that no consistent scenario exists.

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 [9]. These sub-classes are subsets of the 2^{13} possible interval formulas allowed in the full IA. Interval algebra 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 [15].

3.2 Local Search and TCSPs

Unfortunately, little of this work is of direct relevance in applying local search to IA. A local search algorithm 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 [7]. Local search techniques have been particularly successful in solving propositional satisfiability (SAT) problems, resulting in the development of the discrete Lagrangian method (DLM) [12] on which our later work is based. In a SAT problem, a greedy local search will attempt to minimise the number of unsatisfied clauses, where each clause contains a set of disjunct and optionally negated true/false variables. For example, a clause $(x \vee \neg y \vee z)$ would be satisfied by $x \leftarrow$ true or $y \leftarrow$ false or $z \leftarrow$ true. In such a problem, the cost of a move is measured by the number of false clauses that will result from changing the instantiation (flipping) a particular variable. At any point in the search, the algorithm will flip the variable that causes the least number of clauses to remain false, continuing until no further improvements are possible. At this point, the algorithm has either found a consistent solution, or it is "stuck" in a local minimum or trap. Most work in local search has concentrated on trap avoiding or trap escaping strategies, using combinations of random instantiations, avoiding previous instantiations and changing costs by penalising frequently violated constraints [13].

In applying local search to IA, it is natural to take a variable instantiation to be the selection of a basic relation from each interval formula. The task is then to determine 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 the number 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 [10], we can conclude that a local search of this sort will not achieve any benefits over existing approaches.

3.3 End-Point Ordering

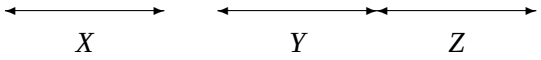
An IA problem Θ can easily be translated to propositional logic (using a variation of the method used by Nebel and Bürckert [9], Section 3). Hence, an obvious alternative for representing the ISAT problem is to translate Θ into a propositional satisfiability (SAT) formula. This would enable the application of existing SAT local search techniques without modification. However, as Nebel has already pointed out [8], expressing the implicit dependencies between time interval end-points in that translation 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 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 end-point, 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).

The advantage of using values to represent an 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).

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.

Given an IA problem Θ involving m interval variables, an *ordering* or O-interpretation maps each interval variable to an interval with integer end-points in the range $1..2m$. We say Θ is O-satisfiable if there is an O-interpretation that satisfies Θ. *OSAT* is the problem of deciding whether an IA problem Θ is satisfiable by an O-interpretation.

Proposition 3.1 *Let Θ be an IA problem. Then Θ is I-satisfiable iff Θ is O-satisfiable.*

> **Proof:** *($\Leftarrow$) This direction is trivial, since every O-interpretation is also an I-interpretation.*
>
> *($\Rightarrow$) Let ψ be an I-interpretation that satisfies Θ. ψ maps every interval variable X to an interval (a, b). Consider the set of values used by ψ as end-points. Since these values are real numbers, they can be ordered and ranked; let r be the ranking function. Notice that*

$x < y$ iff $r(x) < r(y)$, and $x = y$ iff $r(x) = r(y)$. Thus, r preserves the truth value of end-point constraints.

Let ψ' be the I-interpretation that maps each interval variable X to the interval $(r(a), r(b))$. Obviously ψ' is an O-interpretation, since there are $2m$ end-points and thus the range of r is $1..2m$. Furthermore, ψ' satisfies a basic interval formula $X\{B\}Y$ iff ψ satisfies $X\{B\}Y$, because satisfaction is determined entirely by satisfaction of the end-point constraints of B, and r preserves the truth value of these constraints. Thus, ψ' satisfies Θ iff ψ satisfies Θ.

Thus, OSAT is equivalent to ISAT.

Furthermore, we can formulate OSAT as a CSP: Let (Ψ, Θ) be an IA problem. The set of variables is $\{X^-, X^+ \mid X \in \Psi\}$; each variable has domain $\{1, 2, \ldots, 2|\Psi|\}$. For each interval formula $X\{B_1, B_2, \ldots, B_n\}Y$, there is a constraint on X^-, X^+, Y^-, Y^+ equivalent to $\bigvee_{i=1}^{n} \phi_{B_i}(X^-, X^+, Y^-, Y^+)$, where ϕ_{B_i} is the end-point relation for B_i.

This reformulation allows us to apply conventional CSP techniques to solve ISAT. In particular, we can now apply local search methods.

4 Solving OSAT using Local Search

4.1 Constraint Weighting Local Search

In order to apply a local search to IA, we need to decide how a local move is taken and how the solution cost is calculated. 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 or SAT problem, 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 finish-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 [12]. 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 *temporal* SAT or TSAT algorithm (see Figure 1) applies the basic DLM heuristics to the temporal reasoning domain, and is controlled by two parameters: MAX_FLATS (set to 4) which specifies how many consecutive non-improving (flat) moves can be taken before constraint weights are increased and MAX_WEIGHTS (set to 10) which specifies how many constraint weight increases can occur before the weights are reduced. The TIME_LIMIT parameter is further used stop the algorithm running indefinitely on problems for which it can find no answer. In the case of running TSAT on known over-constrained problems, we would also add code to record the best solution found *so far* in the search.

As TSAT is a special purpose algorithm that exclusively processes temporal constraints in the form of end-point relations (see Table 1), we were also able to

```
procedure TSAT(Events, Constraints)
  Randomly instantiate every event (e_i^-, e_i^+) ∈ Events
  Cost ← number of unsatisfied constraints ∈ Constraints
  FlatMoves ← WeightIncreases ← 0
  while Cost > 0 and execution time < TIME_LIMIT 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_j^-, d_j^+) ∈ Moves
    end for
    if Cost < StartCost then FlatMoves ← 0
    else if (++FlatMoves) > MAX_FLATS then
      increment weight on all unsatisfied constraints
      increase Cost by the number of unsatisfied constraints
      FlatMoves ← 0
      if (++WeightIncreases) > MAX_WEIGHTS then
        decrement weight on all constraints with weight > 1
        decrease Cost by number of decremented constraints
        WeightIncreases ← 0
      end if
    end if
  end while
end
```

Figure 1: The TSAT local search procedure for interval algebra

develop various optimisations that exploit the special structure of these constraints. These optimisations are implemented in the TSAT functions $FindBestMoves$ (Figure 4) and $FindCost$ (Figure 5), and are explained in the following Sections on temporal tree constraints, domain skipping and constraint skipping.

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 [8]. 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[1] 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 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

[1]Where the length of a path is defined as the number of comparisons needed to evaluate the constraint.

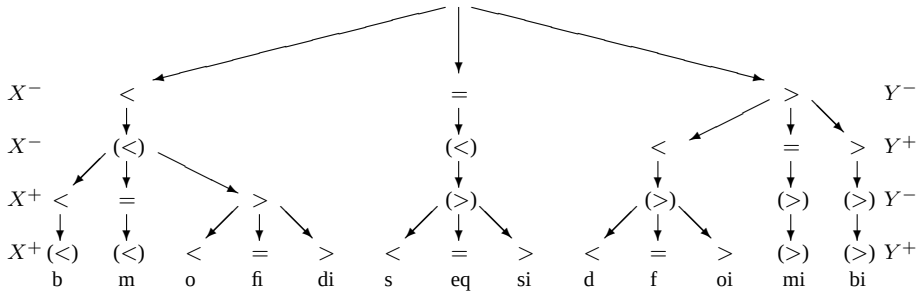


Figure 2: The end-point comparison tree for the 13 basic relations

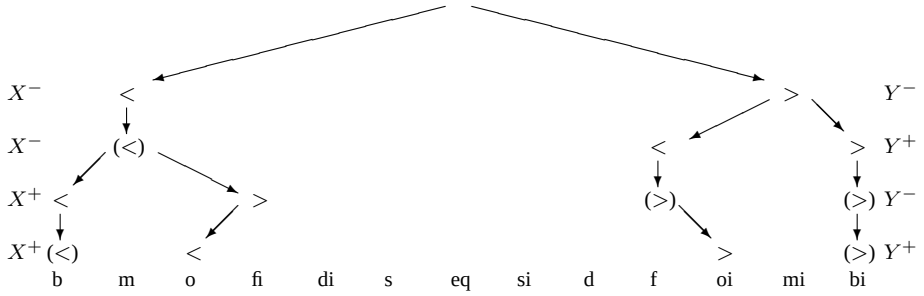


Figure 3: The temporal tree constraint for $X\{b, bi, o, oi\}Y$

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 Domain Skipping

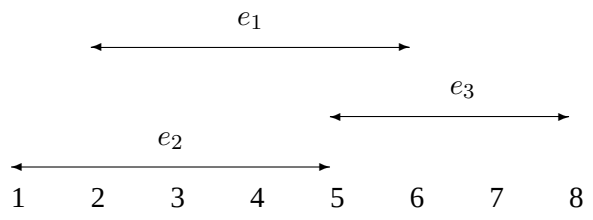
As discussed in Section 4.1, the TSAT algorithm considers all possible combinations of event start and finish-point pairs (e_i^-, e_i^+) before selecting a particular instantiation. To speed up this process, we developed a domain skipping technique that avoids redundant domain value evaluations.

From Section 3.3, we found the upper bound on the number of integers needed for a complete end-point ordering of an IA problem is $2m$, where m = the total

```
function FindBestMoves(Constraints, Cost, e_i^-, e_i^+)
  Moves ← ∅, OuterCost ← 0
  OuterConstraints ← all c_{i,j} ∈ 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^+) ← FindCost(e_i^+, d_min^+, InnerConstraints, InnerCost)
        if TestCost < Cost then
          Cost ← TestCost
          Moves ← ∅
        end if
        if TestCost = Cost then add domain values ((d_min^- ... d_max^-), (d_min^+ ... d_max^+)) to Moves
        else if InnerCost > Cost then d_max^+ ← max domain value of e_i^+
        d_min^+ ← d_max^+ + 1
      end for
    end if
    d_min^- ← d_max^- + 1
  end while
  return (Moves, Cost)
end
```

Figure 4: The $FindBestMoves$ TSAT move selection function

number of events. However, as we know $e_i^- < e_i^+$, it follows that the upper bound for the domain size of a particular end-point is $2m - 1$, (i.e. $domain(e_i^-) = \{1 \ldots 2m - 1\}$ and $domain(e_i^+) = \{2 \ldots 2m\}$) and the corresponding upper bound for the domain size of an event (e_i^-, e_i^+) is $2m^2 - 3m + 1$. In practice, the overhead in evaluating this domain can be considerably reduced by recognising *ranges* of domain values for which the same constraint evaluations apply. For example, consider the three events e_1, e_2 and e_3 shown in the following diagram:



```
function FindCost(e_i, d_min, ConstraintList, FixedCost)
  TestCost ← 0, d_max ← domain size of e_i + 1
  for each c_{i,j} ∈ ConstraintList do
    let e_j be the second event in c_{i,j}
    if c_{i,j} is consistent with domain value d_min for e_i then
      mark c_{i,j} as consistent
      if d_min > e_j^+ then remove c_{i,j} from ConstraintList
    else
      mark c_{i,j} as inconsistent
      if d_min > e_j^+ then
        remove c_{i,j} from ConstraintList
        FixedCost ← FixedCost + weighted cost of c_{i,j}
      else TestCost ← TestCost + weighted cost of c_{i,j}
      end if
    end if
    d_skip ← largest domain value skip allowed for e_j from d_min by c_{i,j}
    if d_skip < d_max then d_max ← d_skip
  end for
  return (TestCost + FixedCost, FixedCost, d_max)
end
```

Figure 5: The $FindCost$ TSAT cost evaluation function

Here we are considering the move for event e_1 given the end-point instantiations $(e_2^-, e_2^+) \leftarrow (1, 5)$ and $(e_3^-, e_3^+) \leftarrow (5, 8)$. From an examination of the problem, it trivially follows that no difference in cost can be caused by moving from $e_1^- \leftarrow 2$ to $e_1^- \leftarrow 3$ or $e_1^- \leftarrow 4$, whereas moving to $e_1^- \leftarrow 5$ can effect a change as it alters the end-point relationship from $e_1^- < e_2^+$ to $e_1^- = e_2^+$. Therefore, we can compress the domain of e_1^- by *skipping* values 3 and 4 and immediately trying $e_1^- \leftarrow 5$. Similarly moving from $e_1^+ \leftarrow 6$ to $e_1^+ \leftarrow 7$ leaves all end-point relations unaffected so a further evaluation of $e_1^+ \leftarrow 7$ is redundant.

The domain skipping technique is implemented in the final two lines of the **for** loop of the TSAT function $FindCost$ (Figure 5). In this function, the d_{min} parameter holds the domain value of event e_i currently being tested in $FindBestMoves$ (Figure 4). The $FindCost$ line:

$d_{skip} \leftarrow$ largest domain value skip allowed for e_j from d_{min} by $c_{i,j}$

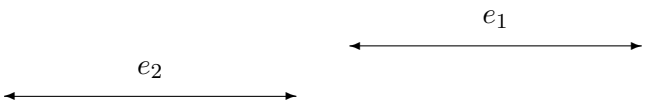
then checks, for each constraint $c_{i,j}$, to find the next consecutive domain value of e_i greater than d_{min} that alters the current evaluation of $c_{i,j}$ and stores the value *preceding* this in d_{skip}. The range $d_{min} \ldots d_{skip}$ therefore holds values for which the current evaluation of $c_{i,j}$ remains unchanged. The next line:

if $d_{skip} < d_{max}$ **then** $d_{max} \leftarrow d_{skip}$

calculates the smallest value of d_{skip} for all $c_{i,j}$ and stores this in d_{max}. The $d_{min} \ldots d_{max}$ range therefore holds the values for which *all* current $c_{i,j}$ evaluations remain unchanged, and which therefore can be safely skipped. Then, the value of d_{max} is returned to $FindBestMoves$ (Figure 4) and used to instantiate the next domain evaluation at $d_{max} + 1$ (thereby skipping the redundant domain values between d_{min} and $d_{max} + 1$).

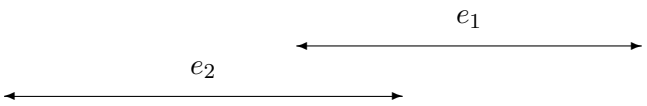
4.4 Constraint Skipping

A central feature of the $FindBestMoves$ function in Figure 4 is that the evaluation of domain values for an event e_i is separated into an outer loop that tests domain values for the start-point of the event (e_i^-) and an inner loop that tests the finish-point (e_i^+). This structure allows us to exploit the property that once the current test domain value d_{min}^- for e_i^- in the outer loop exceeds the finish-point e_j^+ of any event e_j that shares a constraint $c_{i,j}$ with e_i, then it is no longer possible for the violation cost of $c_{i,j}$ to alter during the subsequent evaluation of e_i. Hence, such past constraints can be *skipped* in the future evaluation of domain values for e_i. In addition, the *sum* of fixed costs incurred by these skipped constraints becomes a lower bound on the best cost that can be achieved by any future move for e_i. From this it follows that if the fixed cost value exceeds the current solution cost, we can abandon further evaluation of e_i as it cannot improve on the current cost. As an example, consider the following diagram depicting two events e_1 and e_2:



Here the start-point e_1^- of e_1 is greater than the end-point e_2^+ of e_2 and due to the order in which the domain values are tested (Figure 4) no further domain value change for e_1 can alter the situation. Additionally, if we assume a violated constraint exists between e_1 and e_2 (e.g. $e_1\{b\}e_2$), then no further domain value change for e_1 can alter this violation.

Again, due to the division of processing between the start and finish-points of events, the constraint skipping technique can be extended to skip constraints involved in finish-point evaluations that could still be affected by future untested start-point evaluations for the same event. This is because the start-point in the outer loop (Figure 4) is held constant while all valid finish-points are evaluated in the inner loop. For example, consider the following diagram:



In this case e_1^+ has passed e_2^- and, as far as e_1^+ is concerned, the constraint $c_{1,2}$ between e_1 and e_2 will no longer evaluate differently. Using the outer loop in $FindBestMoves$ (Figure 4), we can exploit this situation by *partially* evaluating $c_{1,2}$, i.e. by only considering point relations involving e_i^- in the outer loop and then completing the evaluation for each domain value of e_i^+ in the inner loop. Hence, we can skip the overhead of re-evaluating any e_i^- relations of $c_{i,j}$ when processing e_i^+. Further, if $c_{i,j}$ is found to be inconsistent in the outer loop, it must remain inconsistent in the inner loop and can therefore be skipped entirely.

The efficiency of partial constraint evaluation is further enhanced using the temporal tree constraint representation described in Section 4.2, as the first two levels of the tree only concern the point e_i^- and the last two levels only concern e_i^+. Therefore we can easily divide the tree to reflect the processing between the inner and outer loops of Figure 4.

As an indication of the performance improvements achieved by our domain and constraint skipping techniques, we compared the results for the current version of TSAT, running on the random problem set described in Section 5.2, with our initial TSAT algorithm reported in [14]. We found an average speed up of $\times 4.5$ on the 40 node problems and $\times 8$ on the 80 node problems[2]. We therefore concluded that the overhead costs of the skipping techniques are more than compensated for by the reduction in constraint and domain evaluations.

5 Empirical Study

5.1 Problem Generation

Due to the absence of a collection of large IA benchmark problems, earlier work in the area [8, 15] conducted empirical evaluations of backtracking using randomly generated IA problem instances. To remain in line with this work, we selected Nebel's random instance generator [8] to build a set of satisfiable problems on which to compare the performance of local search with the existing backtracking approaches. Of the three models available in Nebel's generator, we chose the $S(n, d, s)$ model, as it guarantees the generation of satisfiable instances[3]. Here n

[2]Here our measure of performance speed up is the ratio of average moves executed per unit time for each algorithm.

[3]We required satisfiable instances as a local search will not terminate on an unsatisfiable instance due to the incomplete nature of the algorithm.

determines the number of nodes (i.e. intervals) in the problem and d determines the degree or the proportion of all possible arcs (i.e. interval formulas) that are constrained. For example, if degree $= 25\%$ then, on average, a randomly selected 25% of the possible $n - 1$ arcs for each node will be constrained, where a constrained arc is one that is labelled with *less than* the full 13 IA basic relations. Then s sets the average label size for each arc, such that if $s = a$, then an average of a basic relations will be randomly selected for each *constrained* arc, where $domain(a) = \{1 \ldots 12\}$. In a local search the unconstrained arcs remain unlabelled whereas for backtracking + path-consistency unconstrained arcs are labelled with all 13 basic relations. This full labelling is needed because path-consistency may later infer a reduced set of relations on an unconstrained arc, thereby making it constrained.

To ensure a problem is satisfiable the $S(n, d, s)$ model generates a random problem P_1 and a random I-interpretation for that problem. The I-interpretation is then used to generate a second problem P_2 with a unique labelling for each arc, such that the I-interpretation is satisfied. Problems P_1 and P_2 are then combined so that each label in P_2 is added to the corresponding arc in P_1 unless that label already exists in P_1. In this way P_1 becomes satisfiable but loses the property of being a pure random problem.

5.2 Initial Tests

We initially set out to investigate the feasibility of the local search approach by running TSAT on problems which a standard backtracking and path-consistency approach has difficulty. Using the $S(n, d, s)$ model, we started generating two sets of random, consistent problems $S(40, 75, 9.5)$ and $S(80, 75, 9.5)$. We then ran each problem on Nebel's backtracking-based problem solver (with all heuristics

		CPU Time			Number of Moves		
nodes	% solved	mean	median	std dev	mean	median	std dev
40	100.0	0.206	0.188	0.096	48.64	46	12.70
80	100.0	4.809	4.343	2.021	215.90	200	66.38

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

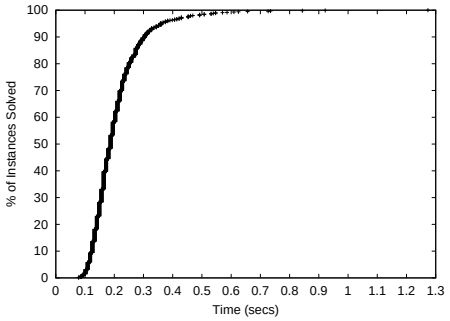


Figure 6: TSAT plot for 40 nodes

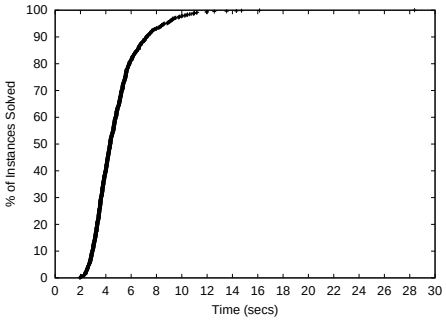


Figure 7: TSAT plot for 80 nodes

turned off) [8] until 100 problems were found in each set that backtracking had failed to solve (the 40 node problems were timed out after 300 seconds and the 80 node after 600 seconds). We then solved each of these problems 10 times with the TSAT algorithm described in Figures 1, 4 and 5 (all experiments were conducted on a Intel Pentium Celeron 450MHz machine with 160Mb of RAM running FreeBSD 4.2).

The TSAT feasibility results are shown in Table 2 and in the graphs of Figures 6 and 7 which plot the proportion of problems solved against the CPU time for each run. The 40 node problem results indicate that TSAT finds these instances relatively easy, with all runs being solved within 1.27 seconds and a median CPU time of 0.188 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 larger 80 node

problems proved harder for TSAT, but all runs were still solved within 28.38 seconds with a median run-time of 4.34 seconds. These results show that TSAT is up to two orders of magnitude faster than standard backtracking + path-consistency, at least on this set of problems which backtracking already finds difficult.

5.3 Comparison with Backtacking

As the initial TSAT results were promising, we decided to obtain a clearer picture of the relative performance of TSAT and Nebel's backtracking algorithm by developing a more extensive test set of 80 node problems. In these we varied the label size from 1 to 12 (in steps of 0.5) across four degrees of constraint connectivity (25%, 50%, 75%, and 100%). By randomly generating 100 problems at each degree/label size combination we obtained a set of $23 \times 4 \times 100 = 9200$ problems. We then ran two versions of Nebel's algorithm on each problem, one (standard backtracking) with all special heuristics turned off, and one (heuristic backtracking) using a combination of the heuristics that proved successful in Nebel's earlier empirical study [8][4]. As before each run was timed out at 600 seconds and TSAT was allowed 10 runs on each problem. The results for these runs are shown in Tables 3, 4, 5, and 6 where the backtracking results average the 100 problems generated at each degree/label size category and the TSAT results average 10 runs on each of the 100 problems in each category.

5.4 Results

The comparison between backtracking and local search on the 80 node problem set does not yield a simple yes/no answer as to which algorithm is better. How-

[4]Specifically, we used the *ORD-Horn* subclass for the *Split* set, with the *static* constraint evaluation heuristic, the weighted *queue* scheme and the *global* heuristic criteron to evaluate constrainedness.

	Standard Backtrack			Heuristic Backtrack			TSAT Local Search		
label size	% fail	mean time	median time	% fail	mean time	median time	% fail	mean time	median time
1	0	0.13	0.13	0	0.08	0.08	1	0.51	0.39
1.5	0	0.15	0.15	0	0.08	0.09	1	0.52	0.40
2	0	0.17	0.17	0	0.09	0.09	0	0.48	0.41
2.5	0	0.20	0.20	0	0.10	0.10	0	0.51	0.45
3	0	0.23	0.23	0	0.11	0.12	0	0.52	0.46
3.5	1	0.27	0.27	1	0.13	0.13	0	0.58	0.51
4	0	0.31	0.31	0	0.15	0.15	0	0.61	0.55
4.5	4	0.37	0.37	0	0.50	0.18	0	0.58	0.52
5	8	0.44	0.43	0	2.74	0.21	0	0.60	0.53
5.5	19	2.47	0.46	3	2.66	0.23	0	0.57	0.51
6	29	14.27	0.48	4	11.39	0.30	0	0.51	0.45
6.5	26	33.00	0.48	2	12.16	0.50	0	0.47	0.42
7	26	9.44	0.49	3	12.86	0.48	0	0.41	0.38
7.5	11	0.62	0.48	3	24.01	0.93	0	0.37	0.34
8	5	6.80	0.49	2	16.21	0.47	0	0.32	0.30
8.5	3	1.12	0.51	2	14.44	0.30	0	0.29	0.28
9	1	0.52	0.52	2	1.80	0.30	0	0.24	0.23
9.5	0	0.54	0.54	1	1.19	0.31	0	0.21	0.20
10	0	0.56	0.56	1	0.32	0.32	0	0.17	0.17
10.5	0	0.57	0.57	0	0.32	0.32	0	0.14	0.13
11	0	0.57	0.57	0	0.30	0.30	0	0.11	0.10
11.5	0	0.53	0.53	0	0.24	0.23	0	0.07	0.07
12	0	0.46	0.46	0	0.13	0.13	0	0.04	0.04

Table 3: Results for 80 node problems at 25% degree size

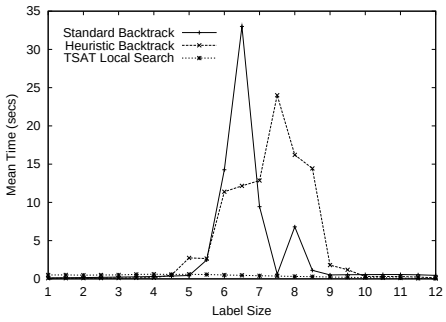


Figure 8: Time plots at 25% degree size

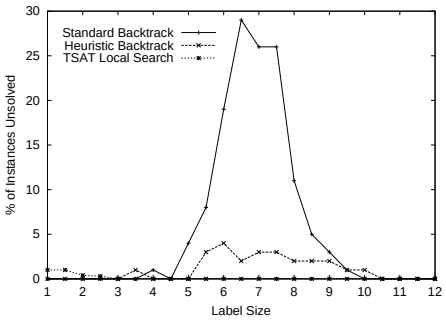


Figure 9: Fail plots at 25% degree size

	Standard Backtrack			Heuristic Backtrack			TSAT Local Search		
label size	% fail	mean time	median time	% fail	mean time	median time	% fail	mean time	median time
1	0	0.10	0.09	0	0.07	0.07	1	6.35	3.79
1.5	0	0.10	0.10	0	0.08	0.08	0	5.28	3.17
2	0	0.11	0.11	0	0.08	0.08	1	4.55	2.81
2.5	0	0.12	0.13	0	0.08	0.08	1	4.04	2.53
3	0	0.14	0.13	0	0.08	0.09	0	3.39	2.25
3.5	0	0.15	0.16	0	0.09	0.09	1	3.38	2.25
4	0	0.17	0.17	0	0.09	0.09	0	3.15	2.20
4.5	0	0.20	0.20	0	0.10	0.10	0	2.83	2.14
5	0	0.23	0.23	0	0.24	0.11	0	2.86	2.11
5.5	0	0.28	0.27	0	0.13	0.13	0	2.63	2.18
6	0	0.35	0.34	0	0.15	0.15	0	2.61	2.26
6.5	2	0.59	0.44	1	1.23	0.19	0	2.90	2.51
7	17	21.56	0.62	3	11.00	0.49	0	2.64	2.34
7.5	71	52.14	11.27	15	35.67	4.41	0	2.59	2.29
8	90	49.27	39.55	27	69.78	16.18	0	2.49	2.23
8.5	88	57.37	1.14	49	108.12	36.32	0	2.16	1.95
9	85	29.31	0.98	52	182.30	168.09	0	1.92	1.75
9.5	75	37.77	0.69	57	121.16	52.64	0	1.53	1.42
10	63	8.40	0.62	49	104.19	33.91	0	1.29	1.23
10.5	22	1.34	0.61	19	37.72	2.33	0	0.97	0.94
11	0	0.63	0.63	3	12.50	0.38	0	0.69	0.66
11.5	0	0.67	0.66	2	0.38	0.38	0	0.44	0.43
12	0	0.66	0.66	0	0.78	0.32	0	0.21	0.20

Table 4: Results for 80 node problems at 50% degree size

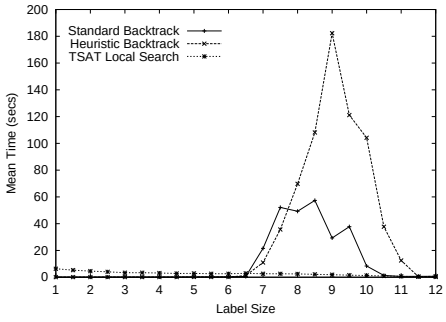


Figure 10: Time plots at 50% degree size

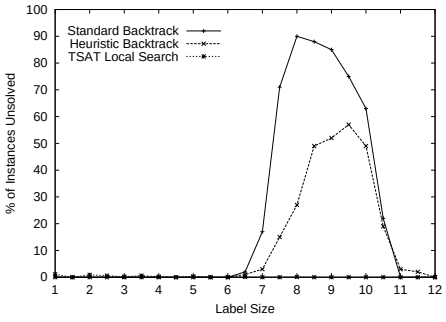


Figure 11: Fail plots at 50% degree size

	Standard Backtrack			Heuristic Backtrack			TSAT Local Search		
label size	% fail	mean time	median time	% fail	mean time	median time	% fail	mean time	median time
1	0	0.09	0.09	0	0.08	0.08	5	24.79	15.57
1.5	0	0.09	0.09	0	0.08	0.08	2	22.67	13.86
2	0	0.09	0.09	0	0.08	0.08	1	22.34	14.52
2.5	0	0.10	0.10	0	0.08	0.08	1	17.73	11.02
3	0	0.11	0.11	0	0.08	0.09	1	16.44	10.03
3.5	0	0.11	0.12	0	0.09	0.09	0	17.38	10.24
4	0	0.13	0.13	0	0.09	0.09	1	14.74	8.90
4.5	0	0.14	0.14	0	0.09	0.09	0	13.23	8.40
5	0	0.16	0.16	0	0.09	0.09	0	11.64	6.56
5.5	0	0.18	0.18	0	0.10	0.10	0	9.91	6.44
6	0	0.21	0.21	0	0.11	0.11	0	8.67	5.33
6.5	0	0.26	0.26	0	0.12	0.12	0	7.92	5.24
7	1	0.33	0.33	0	0.19	0.14	0	6.81	5.10
7.5	3	0.45	0.43	0	0.39	0.19	0	6.04	5.09
8	11	6.0	0.65	0	17.86	0.60	0	6.23	5.23
8.5	69	111.84	57.13	12	46.62	10.53	0	5.91	5.13
9	98	1.14	1.49	61	184.94	122.04	0	5.37	4.77
9.5	97	67.32	23.49	80	265.25	236.04	0	4.80	4.30
10	99	23.91	23.91	86	149.08	162.91	0	3.77	3.46
10.5	86	44.09	0.79	87	225.61	242.43	0	3.08	2.85
11	56	10.09	0.67	61	99.92	61.14	0	2.11	2.01
11.5	10	2.60	0.67	7	36.80	3.13	0	1.30	1.27
12	0	0.71	0.71	3	1.21	0.39	0	0.61	0.59

Table 5: Results for 80 node problems at 75% degree size

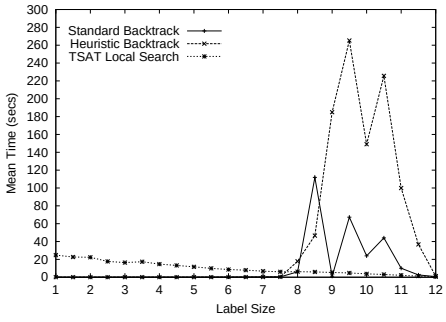


Figure 12: Time plots at 75% degree size

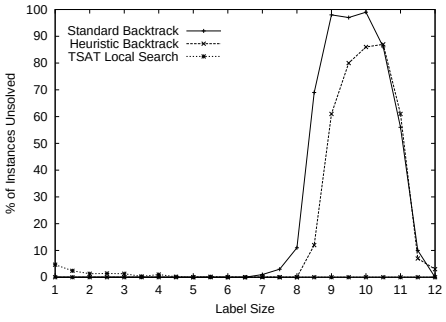


Figure 13: Fail plots at 75% degree size

	Standard Backtrack			Heuristic Backtrack			TSAT Local Search		
label size	% fail	mean time	median time	% fail	mean time	median time	% fail	mean time	median time
1	0	0.08	0.08	0	0.08	0.08	14	47.99	35.38
1.5	0	0.08	0.09	0	0.08	0.08	8	45.17	33.63
2	0	0.09	0.09	0	0.08	0.08	6	41.00	30.10
2.5	0	0.09	0.09	0	0.08	0.09	6	44.07	31.11
3	0	0.09	0.09	0	0.08	0.09	5	42.20	30.81
3.5	0	0.10	0.09	0	0.09	0.09	6	40.05	29.37
4	0	0.10	0.10	0	0.09	0.09	2	36.80	25.26
4.5	0	0.11	0.12	0	0.09	0.09	3	36.04	24.27
5	0	0.12	0.13	0	0.09	0.09	2	32.06	20.94
5.5	0	0.14	0.14	0	0.09	0.09	2	29.33	18.45
6	0	0.16	0.16	0	0.10	0.10	1	25.27	16.15
6.5	0	0.19	0.19	0	0.10	0.10	1	21.36	11.82
7	0	0.23	0.23	0	0.12	0.12	1	17.61	10.77
7.5	0	0.30	0.30	0	0.13	0.13	0	13.08	8.62
8	0	0.40	0.40	0	0.22	0.17	0	11.17	8.69
8.5	7	1.26	0.55	0	0.80	0.38	0	11.08	8.73
9	26	87.97	22.47	5	48.67	8.18	0	9.92	8.60
9.5	95	140.42	140.71	55	114.33	71.76	0	9.10	8.29
10	100	600.00	600.00	94	303.18	279.67	0	8.39	7.47
10.5	99	1.41	1.41	96	219.24	249.64	0	6.46	5.94
11	93	1.56	0.71	88	144.59	86.09	0	4.55	4.28
11.5	62	5.91	0.70	74	121.47	46.21	0	2.84	2.75
12	0	0.71	0.70	8	12.42	0.41	0	1.30	1.26

Table 6: Results for node problems at 100% degree size

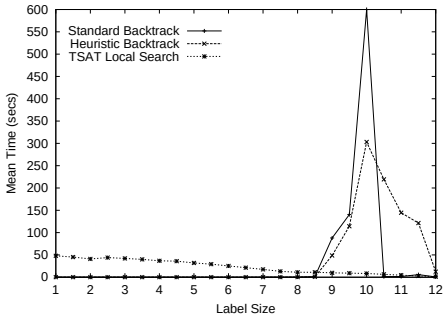


Figure 14: Time plots at 100% degree size

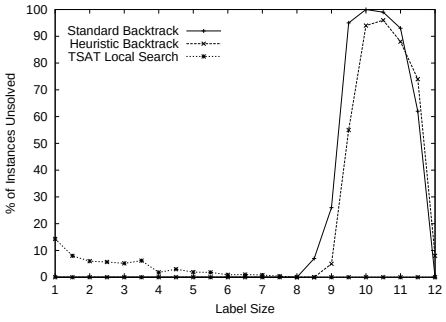


Figure 15: Fail plots at 100% degree size

ever, various patterns do emerge from the data, which are detailed in the following points:

Firstly, for all degree values d, both the heuristic and standard backtracking techniques proved better than TSAT for smaller label sizes. Further, as d increases, the number of label size values for which backtracking is better also increases, from a range of $\{1 \ldots 5\}$ for $d = 25$ (Table 3) to $\{1 \ldots 8.5\}$ for $d = 100$ (Table 6).

Secondly, as d increases, the label size at which the backtracking techniques have their worst performance also increases. For example, the worst performance of heuristic backtracking moves from label size $s = 7.5$ at $d = 25$ to $s = 10.5$ at $d = 100$. Also, as d increases, the execution times and failure rates for backtracking on the larger label sizes grow significantly, whereas there is little change for the smaller label size problems. For example, the worst performance of heuristic backtracking for $d = 25$ is at $s = 7.5$ with a 3% failure rate and a mean time of 24.01 seconds, whereas for $d = 100$ the worst failure rate has grown to 96% with a mean time of 249.64 seconds (at $s = 10.5$). Conversely, the best performance for heuristic backtracking actually decreases slightly as d increases from a 0% failure rate and mean time of 0.13 seconds for $d = 25$, $s = 1$ to a 0% failure rate and mean time of 0.08 seconds for $d = 100$, $s = 1$.

Thirdly, TSAT execution times and failure rates show a more consistent relationship to degree and label size than the backtracking techniques, with times and failure rates generally *increasing* as d increases and *decreasing* as s increases. This is in marked contrast to backtracking, and results in the situation that the problems backtracking finds easier (smaller label size), TSAT finds harder and the problems TSAT finds easier (larger label size) backtracking finds harder. Additionally, the problems TSAT finds easier, it finds much easier than backtracking does, whereas the problems backtracking finds easier are only somewhat more difficult for TSAT. This is illustrated by the fact that TSAT can reach any of our 9200 problems in a

worst case average time of 35 seconds, whereas heuristic backtracking was unable to solve 1118 or 12% of the problem set within 600 seconds.

Finally, the results were also able to confirm that the backtracking heuristics selected for our experiments were able to improve on the performance of standard backtracking, except for the largest label sizes ($s = 11.5$ and $s = 12$).

5.5 Analysis

In analysing the results, our first observation is that a portfolio algorithm using TSAT and heuristic backtracking appears to be the best solution on the range of problems we are considering. Such portfolio algorithms [5] have already been considered in the CSP literature, and operate by running two or more algorithms simultaneously on the same problem, and terminating all algorithms when one finds a solution (or the whole system is timed out). This approach would be especially useful on problems for which the question of consistency is undecided, as a local search will never terminate on an inconsistent problem[5]. A portfolio algorithm is also justified by the significant difference in run-times exhibited by the two algorithms, as the potential benefit of running a TSAT in the hard region for backtracking would then outweigh the redundant cost of using TSAT on the easier problems. A refined portfolio technique may further consider eliminating TSAT for smaller label sizes.

However, our results also raise questions about the hardness of the underlying problem distribution and the reasons for the wide divergence of performance. From a CSP perspective, we would expect problems to be hardest when the label size is small, as this is where the number of expected solutions would be at a minimum

[5] In the case that no algorithm can successfully terminate on a problem, we would rely on a time out value to terminate all searches, as per Figure 1).

(i.e. a problem with exactly one label on each arc has either zero or one possible consistent O-instantiations). Hence TSAT performs poorly on problems with smaller label sizes because there are fewer expected solutions and the reduction in label size does not reduce the size of the search space (i.e. label size does not affect the number of constraints or the size of any domain).

Conversely, as backtracking + path-consistency techniques explicitly work by testing the consistency of individual relations, a problem with fewer relations is actually a smaller problem, and the size of the search tree is reduced. Consequently we would expect problem difficulty to grow as label size increases due to the effect of increasing the size of the search tree. However, it can be seen from the results that an opposite effect predominates once the label size exceeds a certain critical point for each degree size, and problem instances start becoming easier. From this we conjecture that the existence of more possible solutions at higher label sizes finally counteracts effects from the increasing size of the search tree.

Finally, the question of a phase-transition in the underlying problem distribution remains unanswered. Many NP-complete problem domains exhibit phase-transition behaviour, with an order parameter that is known to have a critical value where hard to solve problems exist [3]. Examples are the 4.3 clause to variable ratio in the 3-SAT problem domain, and the known relationship between the number of variables, the domain size and the density and tightness of constraints in the binary CSP domain. Nebel conjectured a similar phase-transition in randomly generated temporal reasoning problems, existing between the average degree d and the average label size s [8]. His empirical tests indicated, for a fixed $s = 6.5$, that most hard problems (for backtracking) exist around $d = 9.5$. Our local search results on the transformed OSAT problems show no such transition, with problems generally becoming more difficult as label size becomes smaller. However, this may partly or fully be explained by the fact we did not use a true random problem

generation model (i.e. we generated only *satisfiable* problems as explained in Section 5.1). Hence, our method of eliminating over-constrained problems may also have eliminated phase-transition behaviour. In our future work we intend to revisit this question by running a local search on a larger range of fully random OSAT problems.

6 Conclusion

In conclusion, the paper has demonstrated that an end-point ordering local search approach to solving interval algebra problems is both feasible and practical. The TSAT algorithm is a first indication that local search can outperform existing backtracking + path-consistency approaches on a range of difficult problems. Our main conclusion is that a portfolio algorithm, simultaneously applying local search and backtracking to the same underlying problem, is the best approach for the kind of random problems we have been considering.

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, and a fuller evaluation of these techniques seems called for. Also, we have only considered a fairly narrow range of random problems for which consistency is guaranteed. A larger empirical study looking at both pure random problems and problems exhibiting real-world structure would allow us to draw more general conclusions about the applicability of our end-point ordering approach. Finally, local search appears especially promising for over-constrained temporal reasoning problems, where standard consistency-checking techniques become ineffective [2, 4], but local search can be applied relatively easily.

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] B. Cheesman, P. Kanefski and W. Taylor. Where the *really* hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

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

[5] I. Gent, H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

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

[7] 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.

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

[9] 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.

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

[11] 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.

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

[13] J. Thornton. *Constraint Weighting Local Search for Constraint Satisfaction*. PhD thesis, School of Information Technology, Griffith University Gold Coast, Australia, January 2000.

[14] 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.

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

[16] 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.