Modelling and Solving Temporal Reasoning as Propositional Satisfiability

Duc Nghia Pham [a,b,*], John Thornton [a,b] and Abdul Sattar [a,b]

[a] *SAFE Program*
NICTA Ltd., Queensland, Australia
[b] *Institute for Integrated and Intelligent Systems*
Griffith University, Queensland, Australia

{*duc-nghia.pham, john.thornton, abdul.sattar*}*@nicta.com.au*

Abstract

Representing and reasoning about time dependent information is a key research issue in many areas of computer science and artificial intelligence. One of the best known and widely used formalisms for representing interval-based qualitative temporal information is Allen's interval algebra (IA). The fundamental reasoning task in IA is to find a scenario that is consistent with the given information. This problem is in general NP-complete.

In this paper, we investigate how an interval-based representation, or IA network, can be encoded into a propositional formula of Boolean variables and/or predicates in decidable theories. Our task is to discover whether satisfying such a formula can be more efficient than finding a consistent scenario for the original problem. There are two basic approaches to modelling an IA network: one represents the relations between intervals as variables and the other represents the end-points of each interval as variables. By combining these two approaches with three different *Boolean satisfiability* (SAT) encoding schemes, we produced six encoding schemes for converting IA to SAT. In addition, we also showed how IA networks can be formulated into *satisfiability modulo theories* (SMT) formulae based on the quantifier-free integer difference logic (QF-IDL). These encodings were empirically studied using randomly generated IA problems of sizes ranging from 20 to 100 nodes. A general conclusion we draw from these experimental results is that encoding IA into SAT produces better results than existing approaches. More specifically, we show that the new point-based 1-D support SAT encoding of IA produces consistently better results than the other alternatives considered. In comparison with the six different SAT encodings, the SMT encoding came fourth after the point-based and interval-based 1-D support schemes and the point-based direct scheme. Further, we observe that the phase transition region maps directly from the IA encoding to each SAT or SMT encoding, but, surprisingly, the location of the hard region varies according to the encoding scheme. Our results also show a fixed performance ranking order over the various encoding schemes.

Key words: Temporal Reasoning, Interval Algebra, Satisfiability, Satisfiability Modulo Theories, DPLL, Search

1. Introduction and Background

Representing and reasoning about time dependent information (i.e. *temporal reasoning*) is a central research issue in many real world AI applications such as planning, plan recognition, scheduling, natural language understanding, and medical diagnosis [36]. The basic research tasks include the design and development of efficient reasoning methods to check the consistency of temporal information, to infer new information and to answer temporal queries [10].

Temporal reasoning is an important subdiscipline within the field of constraint satisfaction research and has been subdivided into two basic areas: *qualitative* and *quantitative* temporal reasoning.

1.1. *Quantitative Temporal Reasoning*

Quantitative problems have to deal with situations where there is definite metric information about time intervals, such as one event being *at least* 20 minutes long or starting at *exactly* 2 o'clock. Such scenarios can be treated as temporal constraint satisfaction problems (TCSPs) where variables represent continuous domain time points and constraints represent sets of intervals that restrict the domains of particular variables [11]. For example, we could represent the unary constraint T_i that time point x_i occurs within the intervals $([a_1, b_1], [a_2, b_2])$ as $(a_1 \leq x_i \leq b_1) \vee (a_2 \leq x_i \leq b_2)$ or the binary constraint T_{ij} that the duration between time points x_i and x_j lies within the interval $[a_1, b_1]$ as $(a_1 \leq x_j - x_i \leq b_1)$.

A *general* TCSP can be represented as a directed constraint graph with nodes representing variables and edges representing binary constraints, where each edge can be labelled with a set of intervals. If we further specify that each edge must be labelled with a single interval we arrive at a *simple* temporal problem (STP) [10].

Deciding the consistency of a general TCSP is known to be NP-complete whereas finding the consistency of an STP can be decided in polynomial time using shortest-path algorithms [11]. This has led to the development of methods that decide the consistency of a TCSP by selecting one disjunct label from each edge and testing whether the resulting problem is a consistent STP.

Subsequent work has looked at disjunct temporal problems (DTPs) that allow the inclusion of non-binary constraints. Techniques for solving these problems treat the task of selecting a consistent STP from the original DTP as a *meta-CSP* [50] that takes each constraint in the original problem as a variable in the meta-problem and performs a backtracking search with forward-checking over the space of possible STPs.

A DTP can also be represented as a satisfiability (SAT) problem, consisting of clauses containing disjunctions of literals that each represent a temporal constraint (e.g. $x_1 - x_2 \leq 2$). Such problems can be solved using generic SAT solvers that additionally test the consistency of the underlying STP problems represented by the clauses during the search [3]. Currently, some of the most promising results in the DTP area have been produced using SAT solvers that incorporate the latest advances from the general SAT solving complete search community [5].

* Corresponding author.

1.2. *Qualitative Temporal Reasoning*

However, there are notions of time used in day to day decision making that do not refer directly to quantitative metric data about time points or durations. For example, we may have a constraint that one event must start *before* another or occur *during* a third event. Here we are only concerned with the relative time ordering of events and not with exactly when each event starts and stops. For instance, consider the ordering of events in the construction of a house. Here we are typically unable to predict or control exactly when a particular task will occur but we do have hard constraints about not fitting out the interior until the roof is attached. These *qualitative* temporal relations were formalised in Allen's interval algebra (IA) [1].

In IA there are 13 atomic relations that define all the possible qualitative arrangements that can exist between two time intervals. These relations cover the basic situations that two events can be *before, during, overlapping, meeting, starting, finishing* or *equal* to each other (see Table 1). As with the formulation of a TCSP, an IA problem can be expressed as directed constraint graph. However, here each node represents an *interval* of unspecified length and the edges represent constraints labelled by sets of atomic relations. In this form, IA is an expressively rich framework and, as with the general TCSP, the reasoning problem is computationally intractable. Existing IA techniques are typically based on the backtracking approach (proposed by Ladkin and Reinefeld [29]), which uses path consistency as forward checking. Although this approach has been further improved [36,53], all variants still rely on path consistency checking at each step to prune the search space. This *native* IA approach has the advantage of being fairly compact, but is disadvantaged by the overhead of continually ensuring path-consistency. Additionally, the native IA representation of variables and constraints means that state-of-the-art local search and complete search techniques (such as unit propagation look ahead in Satz [32] or nogood recording and non-chronological backtracking in Chaff [35]) cannot be easily transferred to the IA domain.

In practice, existing native IA backtracking approaches are only able to find consistent solutions for relatively small general IA instances [51,48]. On the other hand, recent research has shown that modelling and solving hard combinatorial problems (including DTPs and planning problems) as SAT instances can produce significant performance benefits over solving problems in their original form [26,24,41,5]. These results have motivated us to undertake the current study.

In other related work, stochastic local search techniques (SLS) were applied to the IA problem, to see if performance improvements over complete search observed in other SAT and CSP domains can be translated to IA. Thornton *et al.* [48] developed the *end-point ordering* model, specifically to represent IA problems in a form suitable for processing by SLS. In this research the TSAT local search algorithm was shown to significantly outperform an existing complete search technique on a set of larger, more difficult IA problems. However, the end-point ordering model, like the native IA model, has a specialised structure that is carefully exploited by the TSAT algorithm.

1.3. *Alternative Approaches and Translations*

Another well-known approach to the qualitative temporal reasoning problem is Villain and Kautz's point algebra (PA) [54]. This formalism uses the three basic relations that exist between two time point variables, i.e. $\{<, >, =\}$. In PA (as in IA) constraints are defined as disjunctions of basic relations between two variables resulting in 2^3 possible constraint types (as opposed to

the 2^{13} in IA). These variables and constraints can again be expressed as a directed constraint graph, where the nodes represent the time point variables and the edges represent the constraints.

The greater simplicity of PA means it is both less expressive than IA and computationally tractable. In addition, PA provides a link between qualitative reasoning about time point orderings and metric reasoning about time quantities. This is shown by the fact that a PA problem is a special case of a TCSP with the metric information omitted. This allows for a simple transformation of PA into TCSP and means TCSP techniques can be easily applied [10].

However, translating from IA to TCSP is not so straightforward. This is because there are many disjunctions of IA relations that can neither be expressed in PA or as a binary TCSP. As we will discuss later in the paper, these disjunctions represent *non-binary* constraints that can involve all the four end-points of a pair of intervals. The inclusion of these non-binary constraints allows us to represent a full IA network using only the endpoints and three basic relations of PA. Such a system of constraints can already be encoded using the DTP formalism. In fact, an IA problem can be considered as a *restricted* DTP with metric information omitted, just as PA can be treated as a binary TCSP.

The previously discussed work on developing SAT-based DTP solvers overlaps with the more general area of *satisfiability modulo theories* (SMT) [4]. SMT approaches are designed to solve problems using a combination of SAT solving and theory specific decision procedures. In fact, a DTP SAT solver is a special case of an SMT solver, where the meta-CSP is handled by a standard SAT solver and the problem of deciding the consistency of the underlying STPs is handled by a theory specific decision procedure (such as the Bellman-Ford procedure) [5]. As we will show, existing general purpose SMT solvers (such as Yices [13]) can also be applied to our new IA-encoding *and* to metric DTPs, thereby providing a unified framework that can handle both the qualitative and quantitative dimensions of the temporal reasoning domain.

1.4. *An Overview of the Paper*

The aim of the current research is to investigate the best approach for solving qualitative (IA) reasoning problems, both in terms of the choice of algorithm and of problem representation. We ask the question whether the representation of IA problems using specialised models that require specialised algorithms is necessary in the general case. Given that the development of such approaches takes considerable effort, we will investigate whether any counterbalancing performance benefits actually result.

To this end, we consider a range of existing IA reasoning techniques, including backtracking over native IA problems and using local search on the end-point ordering model. We also develop a new SMT-encoding of IA and use an existing SMT solver to evaluate this encoding. Against this we contrast the approach of expressing IA as a pure propositional satisfiability problem and applying a state-of-the-art SAT-solver without the need to develop special decision procedures or representation formalisms. Despite the extensive work on solving quantitative TCSPs and DTPs using constraint satisfaction and SAT techniques, to the best of our knowledge there is no explicit and thorough work on formulating IA problems as pure SAT instances (excluding our own recent works in [38,39]). Nebel and Bürckert [37] pointed out that qualitative temporal instances can be translated to SAT instances but that such a translation causes an exponential blowup in problem size. Hence, no further investigation was provided in their work. [1]

[1] Other independent work [20] has proposed representing IA as SAT, but the authors do not specify the transformation in detail, and do not provide an adequate empirical evaluation.

In this paper, we extend our work in [39]. We provide a detailed investigation of how interval-based representations can be encoded into the conjunctive normal form (CNF) of a propositional formula. The interval-based representation, or IA network, can be modelled into two different forms: one is based on treating relations between intervals as variables; and the other is based on representing intervals in terms of end-points. By combining these two approaches with three different SAT encoding schemes, we produced six encoding schemes for converting IA to SAT.

In our empirical study, we used MiniSAT [14], a state-of-the-art complete search SAT solver, for solving SAT encoded problems. These SAT encoded temporal reasoning problems were translated from randomly generated IA problems ranging in size from 20 to 100 nodes. In addition to analysing the performance of MiniSAT on each encoding scheme, we also looked at the comparative performance of a range of other algorithms working directly on native IA, end-point ordering and our new SMT encodings.

A general conclusion we draw from these experimental results is that a SAT based approach performs better than the other alternatives. We also conclude that representing intervals using end-points produces better performance than representing intervals as variables. Further, we observe that the phase transition region maps directly from the IA encoding to each SAT or SMT encoding, but, surprisingly, the location of the hard region varies according to the encoding scheme. Our results also show a fixed performance ranking order over the various encoding schemes.

The remainder of the paper is structured as follows: firstly, we review the basic definitions of IA and then in Section 3 we introduce two models for transforming IA instances into CSP instances. Using these methods, combined with three CSP to SAT encodings, we present six IA to SAT encodings in Section 4. Section 5 describes the translation of IA to SMT, Section 6 describes the generation of our test set instances and Sections 7-10 present an empirical study to evaluate the performance of these SAT encodings relative to each other, and also to evaluate the performance of an existing complete SAT solver in comparison to the native IA backtracking, SMT and TSAT solvers. In Section 11, we further evaluate the performance of our SAT approach against the performance of the SMT approach on structured SMT problems. Finally, Section 12 presents our conclusions and discusses future research directions.

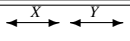
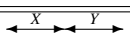
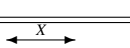
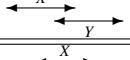
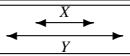
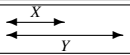
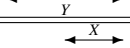
2. Interval Algebra

2.1. *Preliminaries*

Interval Algebra [1] is the most commonly used formalism to represent qualitative temporal information based on *relations* between *time intervals*. A time interval I is defined as an ordered pair of two real-valued endpoints (I^-, I^+) on the time line, where $I^- < I^+$. There are 13 *atomic* interval relations between two time intervals, $\mathcal{I} = \{eq, b, bi, m, mi, o, oi, d, di, s, si, f, fi\}$ which are jointly exhaustive and pairwise disjoint. Table 1 shows graphical representations of these atomic relations and their definitions in terms of endpoint relations. Indefinite information between two time intervals can be expressed as a subset of $\mathcal{I}$ (e.g. a disjunction of atomic relations). For example, the statement *"Event A can happen either before or after event B"* can be expressed as $A\{b, bi\}B$. Hence, there are a total of $2^{13} = 8,192$ possible IA relations between pairs of time intervals.

Qualitative temporal constraints or IA relations between two time intervals A and B are written as ARB or $R(A, B)$, where R is an IA relation. Let R_1 and R_2 be two IA relations. Then the

Table 1
The 13 IA atomic relations. Note that the endpoint relations $X^- < X^+$ and $Y^- < Y^+$ have been omitted.

Atomic relation	Symbol	Meaning	Endpoint relations
X before *Y*	b		$X^- < Y^-, X^- < Y^+$
Y after *X*	bi		$X^+ < Y^-, X^+ < Y^+$
X meets *Y*	m		$X^- < Y^-, X^- < Y^+$
Y met by *X*	mi		$X^+ = Y^-, X^+ < Y^+$
X overlaps *Y*	o		$X^- < Y^-, X^- < Y^+$
Y overlapped by *X*	oi		$X^+ > Y^-, X^+ < Y^+$
X during *Y*	d		$X^- > Y^-, X^- < Y^+$
Y includes *X*	di		$X^+ > Y^-, X^+ < Y^+$
X starts *Y*	s		$X^- = Y^-, X^- < Y^+$
Y started by *X*	si		$X^+ > Y^-, X^+ < Y^+$
X finishes *Y*	f		$X^- > Y^-, X^- < Y^+$
Y finished by *X*	fi		$X^+ > Y^-, X^+ = Y^+$
X equals *Y*	eq		$X^- = Y^-, X^- < Y^+$
			$X^+ > Y^-, X^+ = Y^+$

four operators of IA: *union* (denoted by $\cup$), *intersection* (denoted by $\cap$), *inversion* (denoted by $^{-1}$), and *composition* (denoted by $\circ$), can be defined as follows:

$$\forall\, A, B : A(R_1 \cup R_2)B \leftrightarrow (AR_1B \vee AR_2B)$$

$$\forall\, A, B : A(R_1 \cap R_2)B \leftrightarrow (AR_1B \wedge AR_2B)$$

$$\forall\, A, B : A(R_1^{-1})B \leftrightarrow BR_1A$$

$$\forall\, A, B : A(R_1 \circ R_2)B \leftrightarrow \exists\, C : (AR_1C \wedge CR_2B).$$

Hence, the *intersection* and *union* of any two temporal relations (R_1, R_2) are simply the standard set-theoretic intersection and union of the two sets of atomic relations describing R_1 and R_2, respectively. The *inversion* of a temporal relation R is the union of the inversion of each atomic relation $r_i \in R$. The *composition* of any pair of temporal relations (R_1, R_2) is the union of all results of the composition operation on each pair of atomic relations (r_{1i}, r_{2j}), where $r_{1i} \in R_1$ and $r_{2j} \in R_2$. The full composition results of these IA atomic relations are shown in Table 2.

Definition 1 *Interval Algebra is the algebra with underlying set $2^{\mathcal{I}}$, the power set or set of all subsets of $\mathcal{I}$, unary operators inversion, and binary operators intersection, union and composition [52].*

2.2. *The ISAT Problem*

An IA network is defined as a constraint satisfaction problem (CSP), where each variable represents an interval event with a domain of ordered pairs of real numbers and each binary constraint is labelled with the possible interval relations between a pair of interval events [52,36].

An *instantiation* of an IA network maps each variable to a time interval on the time line represented by an ordered pair of real values (s, e), where $s < e$. A binary constraint $R(A, B)$ is satisfied by an instantiation Θ iff $\Theta(A)$ and $\Theta(B)$ satisfy the corresponding endpoint relations of at least one atomic relation $r \in R$. A *solution* of an IA network is an instantiation that satisfies all the binary constraints.

The problem of determining whether an IA network is *satisfiable*, i.e. whether there exists a solution for that network, is called *ISAT*. ISAT is the *fundamental* reasoning task because all

Table 2
The composition table for the 12 IA atomic relations (omitting *eq*). Note that dur and con represent $\{d, s, f\}$ and $\{di, si, fi\}$, respectively.

○	b	bi	m	mi	o	oi	d	di	s	si	f	fi
b	b	*I*	b	b o m d s	b	b o m d s	b o m d s	b	b	b	b o m d s	b
bi	*I*	bi	bi oi mi d f	bi	bi oi mi d f	bi	bi oi mi d f	bi	bi oi mi d f	bi	bi	bi
m	b	bi oi mi di si	b	f fi eq	b	o d s	o d s	b	m	m	o d s	b
mi	b o m di fi	bi	s si eq	bi	oi d f	bi	oi d f	bi	oi d f	bi	mi	mi
o	b	bi oi mi di si	b	oi di si	b o m	o oi dur con eq	o d s	b o m di fi	o	o di fi	o d s	b o m
oi	b o m di fi	bi	o di fi	bi	o oi dur con eq	bi oi mi	oi d f	bi oi mi di si	oi d f	bi oi mi	oi	oi di si
d	b	bi	b	bi	b o m d s	bi oi mi d f	d	*I*	d	bi oi mi d f	d	b o m d s
di	b o m di fi	bi oi mi di si	o di fi	oi di si	o di fi	oi di si	o oi dur con eq	di	o di fi	di	oi di si	di
s	b	bi	b	mi	b o m	oi d f	d	b o m di fi	s	s si eq	d	b o m
si	b o m di fi	bi	o di fi	mi	o di fi	oi	oi d f	di	s si eq	si	oi	di
f	b	bi	m	bi	o d s	bi oi mi	d	bi oi mi di si	d	bi oi mi	f	f fi eq
fi	b	bi oi mi di si	m	oi di si	o	oi di si	o d s	di	o	di	f fi eq	fi

other interesting reasoning problems can be reduced to it in polynomial time [21] and it is one of the most important tasks in practical applications [53].

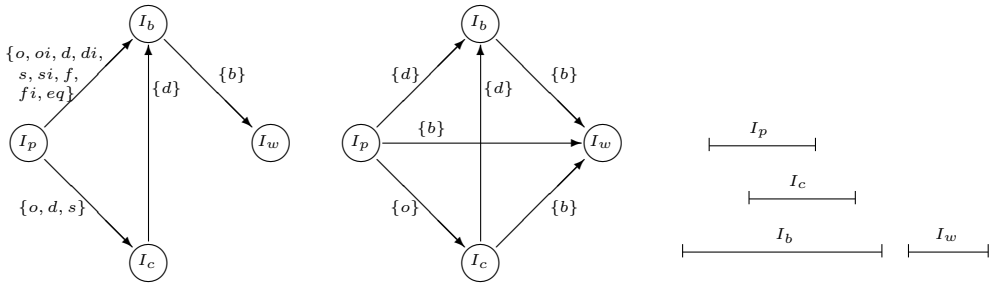
An IA network can be represented as a *constraint graph* or a *constraint network* where the nodes represent variables and the edges are labelled with binary constraints. Usually, such a constraint graph for n interval events is described by an $n \times n$ matrix M, where each entry M_{ij} is the binary constraint between the i^{th} and j^{th} intervals. An IA *scenario* is a *singleton* IA network where each edge (constraint) is labelled with *exactly one* atomic relation. A *consistent scenario* is a scenario such that there exists a corresponding CSP instantiation that satisfies all the constraints.

Figure 1(a) shows an example of an IA network expressing the situation: *"Fred was reading the paper while eating his breakfast. He put the paper down and drank the last of his coffee. After breakfast he went for a walk."*.[2] In this example, variables I_p, I_b, I_c and I_w represent the intervals that Fred is *reading the paper*, *eating breakfast*, *drinking coffee* and *walking* respectively. Figure 1(b) shows a consistent scenario of the network and Figure 1(c) shows the graphical representation of that scenario on the time line.

A CSP is *k-consistent* iff any partial solution over (*k* - 1) variables can be extended to a partial solution over *k* variables that satisfies all the relevant constraints [15]. In addition, a CSP is *strongly k-consistent* iff it is *i-consistent* for all $i \leq k$ [16]. An IA network with n interval variables is *globally consistent* iff it is strongly n-consistent. Hence, the ISAT problem is equivalent to the problem of determining whether an IA network has a globally consistent scenario.

The most useful local consistency notion in temporal reasoning is *path consistency* or *3-consistency* [33,15,16]. Allen [1] proposed a path consistency method for an IA network M that repeatedly computes the following *triangle operation* $M_{ij} \leftarrow M_{ij} \cap M_{ik} \circ M_{kj}$ for all triplets of nodes (i, j, k) until no further change occurs or until $M_{ij} = \emptyset$. These operations remove all

[2] This example was originally used in [51].



(a) IA network (b) consistent scenario (c) time line solution

Fig. 1. An example of an IA network and its consistent solution.

the atomic relations that cause an inconsistency between any triple (i, j, k) of intervals. The resulting network is a path consistent IA network. If $M_{ij} = \emptyset$, then the original IA network is path inconsistent. More sophisticated path consistency algorithms have been applied to IA networks that run in $O(n^3)$ time [52,36]. Although path consistency cannot guarantee global consistency for a *full* IA network, it was proved that enforcing path consistency is enough to ensure global consistency for the maximal tractable subclasses of IA [36].

2.3. *Current Approaches to ISAT*

In general, ISAT is an NP-complete problem [54]. Therefore, it requires *exhaustive search methods* to determine the satisfiability of a full IA network. In his seminal work, Allen [1] proposed an algorithm to systematically search through *all* possible singleton networks resulting from instantiating binary constraints with atomic relations and then to enforce path consistency on these instantiated networks. However, this sort of brute force backtracking technique is inefficient and impractical for non-trivial problems. Moreover, as the domains of the underlying IA network variables are infinite (real numbers), this makes the representation unsuitable for finite CSP and SAT solving techniques.

Ladkin and Reinefeld [29] proposed a more efficient approach to solve the ISAT problem by enforcing path consistency as *forward checking* [22] at every branching node. This allows the elimination of relations that are inconsistent with the current partial solution and hence results in a significant pruning of the search tree. In addition, backtracking algorithms can be improved by preprocessing the input network using path consistency algorithms. van Beek [51] pointed out that the branching factor of an algorithm can be significantly reduced by decomposing disjunctive relations into any set of relations for which path consistency guarantees global consistency rather than decomposing into simple atomic relations. This suggestion was later proven correct by Nebel [36]. Based on this approach, advanced algorithms have been developed that explore various variable and value ordering techniques and different decomposing sets of relations [51,30,53,31,36].

Given that the implicit constraints of the current IA formalism can only be enforced by path consistency, state-of-the-art search techniques from the CSP and SAT domains cannot be easily employed, i.e. it would require the embedding of a path consistency heuristic in the reasoning mechanism of the solver. To address this issue, in [48] we developed a new transformation method, called *end-point ordering*, that reformulates IA networks into CSPs. In this model the

variables are interval end-points and the constraints are the end-point relations as defined in Table 1. The domain of each end-point variable is defined as the integer value position or rank of that variable within the *total ordering of all end-points* [48]. To solve these problems, a specialised TSAT local search algorithm was developed that exploits the structure of the end-point domains and constraints. The main difficulty with this approach is the generation of very large variable domains (representing all possible orderings of each interval). Without the special TSAT pruning heuristics a standard general purpose solver would not prove competitive. Therefore, neither native IA or the end-point ordering models are appropriate for use with a general purpose SAT or CSP solver.

As mentioned in the introduction and elaborated in Section 5, we have also developed a translation procedure that can encode an IA network as an SMT problem, which can then be solved using a general-purpose SMT solver [44]. However, such solvers pay a price in comparison to a pure SAT solver, in having to perform consistency checks by calling separate decision procedures.

Due to the shortcomings of the techniques we have already considered, we decided to explore the development of alternative SAT representations to produce reasonable-sized problems that could then be solved using existing non-specialised SAT solvers.

3. Reformulation of IA into CSP

Recent research has shown that modelling and solving hard combinatorial problems as SAT instances can produce significant performance benefits over solving problems in their original form [26,24,41]. These results inspired us to undertake a thorough study of encoding and solving IA problems as SAT instances using existing SAT solvers.

A common approach to encode combinatorial problems into SAT is to divide the task into two steps: (i) modelling the original problem as a CSP; and (ii) mapping the new CSP into SAT. In the next two subsections, we propose two transformation methods to model IA networks as CSPs such that these CSPs can be feasibly translated into SAT. We then discuss three SAT encoding schemes to map the CSP formulations into SAT, producing in six different approaches to encode IA networks into SAT. [3]

3.1. *The Interval-Based CSP Formulation*

A straightforward method to formulate IA networks as CSPs is to represent each edge (or binary constraint) as a CSP variable. We then limit the domain values of each CSP variable to the set of permissible IA atomic relations for that edge, rather than the set of all subsets of $\mathcal{I}$ used in existing IA approaches. This allows us to reduce the domain size of each CSP variable from 2^{13} to a maximum of 13 values. Thus an instantiation of an *interval*-based CSP maps each variable (edge) to *exactly one* atomic relation in its domain. In other words, an instantiation of this new CSP model is actually a singleton network of the original IA network.

Lemma 2 *Let Θ be a singleton IA network with 3 intervals I_1, I_2, and I_3. Let r_{ij} be the label of the edge between any two I_i and I_j intervals ($i, j \in [1..3]$). Then Θ is consistent iff $r_{13} \in r_{12} \circ r_{23}$.*

[3] In practice, IA networks can be directly encoded into SAT formulae without being reformulated as CSPs. However, for the sake of clarity we first transform IA into two different CSP formulations and then to SAT.

PROOF. The proof for this lemma is trivial based on the fact that there is exactly one mapping of a singleton network onto the time line.

Theorem 3 *Let Θ be a singleton IA network with n intervals and r_{ij} be the label of the edge between I_i and I_j. Then Θ is consistent iff for any triple $(i < k < j)$ of nodes, $r_{ij} \in r_{ik} \circ r_{kj}$.*

PROOF. ($\Rightarrow$) This direction is trivial as Θ is also path consistent.

($\Leftarrow$) Given the fact that $r_{ij} \in r_{ik} \circ r_{kj}$ holds for all triplets $(i < k < j)$ of nodes, Θ is path consistent as a result of Lemma 2. In addition, Θ is singleton. Hence, Θ is globally consistent.

Based on the results of Theorem 3, an *interval*-based CSP representation of a given IA network is defined as follows:

Definition 4 *Given an IA network M with n intervals, $I_1, \ldots, I_n$; the corresponding interval-based CSP is $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where*
$\mathcal{X} = \{X_{ij} \mid i, j \in [1..n], i < j\}$ where each variable X_{ij} represents a relation between two intervals I_i and I_j;
$\mathcal{D} = \{D_{ij}\}$ where each D_{ij} is a set of domain values for X_{ij}, and $D_{ij} = M_{ij}$ the set of relations between interval I_i and I_j; and
$\mathcal{C}$ consists of the following constraints:

$$\bigwedge_{x \in D_{ik}, y \in D_{kj}} X_{ik} = x \wedge X_{kj} = y \Longrightarrow X_{ij} \in D'_{ij} \tag{1}$$

where $i < k < j$ and $D'_{ij} = D_{ij} \cap (x \circ y)$.

Theorem 5 *Let Θ be an IA network and Φ be the corresponding interval-based CSP defined by Definition 4. Then Θ is satisfiable iff Φ is satisfiable.*

PROOF. We first rewrite the constraint (1) into two clauses

$$\bigwedge_{x \in D_{ik}, y \in D_{kj}} X_{ik} = x \wedge X_{kj} = y \Longrightarrow X_{ij} \in D_{ij} \tag{2}$$

$$\bigwedge_{x \in D_{ik}, y \in D_{kj}} X_{ik} = x \wedge X_{kj} = y \Longrightarrow X_{ij} \in x \circ y \tag{3}$$

It is trivial to prove that clauses (2) and (3) are equivalent to constraint (1).

($\Rightarrow$) Let Θ' be a consistent scenario of Θ. As Θ' is a singleton network, Θ' is also an instantiation of Φ by Definition 4. Hence clause (2) is satisfied. In addition, as Θ' is globally consistent, clause (3) is also satisfied by Theorem 3. Hence Θ' satisfies all constraints of Φ. As a result, Φ is satisfiable.

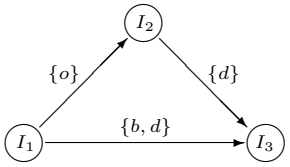
($\Leftarrow$) Let Φ' be an instantiation of Φ such that it satisfies all constraints of Φ (i.e. clauses (2) and (3) are satisfied). We construct a singleton network Θ' by labelling each edge (i, j) of Θ' with the atomic relation $\Phi'(i, j)$. As Φ' satisfies clause (2), Θ' is a singleton network of Θ. In addition, as Φ' satisfies clause (3), we have $\Theta'(i, j) \in \Theta'(i, k) \circ \Theta'(k, j)$ for all triples $(i < k < j)$ of nodes. Applying Theorem 3, Θ' is globally consistent. As a result, Θ is satisfiable.

3.1.1. *Example*

For the sake of clarity, we use the IA network in Figure 2(a) as a running example to illustrate the transformation of IA networks into CSPs and SAT encodings (which are discussed in later sections). The example represents the following scenario:

Anne usually reads her newspaper (I_1) *before or during her breakfast* (I_3)*. In addition, she always drinks a cup of coffee* (I_2) *during her breakfast. This morning, she started reading her newspaper before her coffee was served and finished reading before drinking the last of her coffee.*

The corresponding interval-based CSP of this IA network is shown in Figure 2(b), having 3 variables, which represent the temporal relations between each pair of actions. These CSP variables and their corresponding domains are described using the same order in $\mathcal{X}$ and $\mathcal{D}$. It is worth noting that as $\{o\} \circ \{d\} = \{o, d, s\}$, the constraint between I_1, I_2 and I_3 further restricts the domain of X_{13} to $\{d\}$ instead of its original $D_{13} = \{b, d\}$, i.e. Anne could not have read her newspaper before breakfast if she was still reading it while drinking her coffee *during* breakfast.



(a) IA network

$\mathcal{X} = \{ X_{12},\ X_{13},\ X_{23} \}$

$\mathcal{D} = \{ \{o\},\ \{b, d\},\ \{d\} \}$

$\mathcal{C} = \{ (X_{12} = o \wedge X_{23} = d \Longrightarrow X_{13} = d) \}$

(b) interval-based CSP

Fig. 2. An example of interval-based CSPs of IA networks.

3.2. *The Point-Based CSP Formulation*

Vilain and Kautz [54] proposed the full Point Algebra to model qualitative information between time points. PA consists of a set of 3 atomic relations $\mathcal{P} = \{<, =, >\}$ and four operators *union*, *intersection*, *inversion* and *composition*, which are defined in a similar manner to IA. The full composition results of the PA atomic relations are shown in Table 3.

Table 3
The composition table for PA atomic relations. Note that '?' represents the universal relation $\{<, =, >\}$.

$\circ$	$<$	$=$	$>$
$<$	$<$	$<$	?
$=$	$<$	$=$	$>$
$>$	?	$>$	$>$

A PA network can be represented as a constraint graph in a similar manner as an IA network, where nodes represent points and edges are labelled with PA relations between pairs of points. Hence, all the concepts of consistency discussed above for IA networks are also applicable to PA networks. Again, we use an $n \times n$ matrix P to represent a PA network with n points where P_{ij} is the relation between two points i and j.

As mentioned in Section 2, IA atomic relations can be uniquely expressed in terms of their endpoint relations. However, representing non-atomic IA relations is more complex, as not all

IA relations can be translated into conjunctions of point relations. For example, the following combination of point relations

$$(A^- \neq B^-) \wedge (A^- < B^+) \wedge (A^+ \neq B^-) \wedge (A^+ < B^+)$$

represents not only $A\{b, d\}B$ but also $A\{b, d, o\}B$. This means that PA can only cover 2% of IA [36].

In the example above, to represent $A\{b, d\}B$ we would also need to disallow that $A\{o\}B$. This is not possible in PA because ruling out the overlap relation requires a *non-binary* relation between the end-points of A and B, i.e. we would have to disallow the situation that interval A starts before interval B while also ending between the start and end of interval B, requiring a non-binary relation involving at least 3 endpoints:

$$\neg((A^- < B^-) \wedge (A^+ > B^-) \wedge (A^+ < B^+))$$

However, using the CSP formalism, we can control the instantiation of such undesired IA relations by simply introducing new constraints into the CSP model, as follows:

Let $\mu(r) = (v_{ss}, v_{se}, v_{es}, v_{ee})$ be the PA representation of an IA atomic relation r between two intervals A and B, where v_{se}, for example, is the corresponding PA relation between two endpoints A^- and B^+. We then define the *point*-based CSP model of an IA network as:

Definition 6 *Given an IA network M with n intervals and its corresponding PA network P (with $2n$ points, P_1,...,P_{2n}),* [4] *the corresponding point-based CSP of M is $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where*
$\mathcal{X} = \{X_{ij} \mid i, j \in [1..2n], i < j\}$ where each variable X_{ij} represents a relation between two points P_i and P_j of P;
$\mathcal{D} = \{D_{ij}\}$ where each D_{ij} is the set of domain values for X_{ij} and $D_{ij} = P_{ij}$, the set of point relations between P_i and P_j; and
$\mathcal{C}$ consists of the following constraints:

$$\bigwedge_{x \in D_{ik}, y \in D_{kj}} X_{ik} = x \wedge X_{kj} = y \Longrightarrow X_{ij} \in D'_{ij} \tag{4}$$

$$\bigwedge_{r \notin M_{lm}} (X_{l^-m^-}, X_{l^-m^+}, X_{l^+m^-}, X_{l^+m^+}) \neq \mu(r) \tag{5}$$

where $i < k < j$, $D'_{ij} = D_{ij} \cap (x \circ y)$, and $X_{l^*m^*}$ is the CSP variable representing the relation between one endpoint of interval l and one endpoint of interval m.

Theorem 7 *Let Ω be a singleton PA network with n points and r_{ij} be the label of the edge between two points I_i and I_j. Then Ω is consistent iff for any triple $(i < k < j)$ of nodes, $r_{ij} \in r_{ik} \circ r_{kj}$.*

Theorem 8 *Let Ω be a PA network and Ψ be the corresponding point-based CSP defined by Definition 6 without constraints (5). Then Ω is satisfiable iff Ψ is satisfiable.*

Theorems 7 and 8 are similar to the Theorems 3 and 5 respectively. We can also construct the proofs of Theorems 7 and 8 in a similar way to the proofs of Theorems 3 and 5.

[4] P may allow relations that are not allowed in M.

Theorem 9 *Let Θ be an IA network and Ψ be the corresponding point-based CSP defined by Definition 6. Then Θ is satisfiable iff Ψ is satisfiable.*

PROOF. ($\Rightarrow$) Let Θ' be a consistent scenario of Θ. As Θ' is a singleton network, its corresponding point-based CSP Ψ', defined by Definition 6, is an instantiation of Ψ. Hence, Ψ' satisfies all constraints (5). In addition, as Θ' is globally consistent, Ψ' satisfies all constraints (4) due to Theorem 7. As a result, Ψ is satisfiable.

($\Leftarrow$) Let Ψ' be an instantiation of Ψ such that all constraints (4) and (5) are satisfied. Let $\mu^{-1}(X_{l^-m^-}, X_{l^-m^+}, X_{l^+m^-}, X_{l^+m^+}) = r$ be the inversion of $\mu(r)$, such that it maps the combination of the PA atomic relations of four endpoints $(X_{l^-m^-}, X_{l^-m^+}, X_{l^+m^-}, X_{l^+m^+})$ to the IA atomic relation r between two intervals l and m. As every variable $X_{l^*m^*}$ of Ψ' is instantiated with exactly one atomic relation, $\mu^{-1}(X_{l^-m^-}, X_{l^-m^+}, X_{l^+m^-}, X_{l^+m^+})$ maps to exactly one interval relation.

We construct a singleton IA network Θ' from Ψ' by labelling each edge (l, m) with the corresponding IA atomic relation $\mu^{-1}(X_{l^-m^-}, X_{l^-m^+}, X_{l^+m^-}, X_{l^+m^+})$. As Ψ' satisfies all constraints (5), Θ' is a scenario of Θ. In addition, Θ' is globally consistent by the application of Theorem 7 as Ψ' satisfies all constraints (4). As a result, Θ is satisfiable.

3.2.1. *Example*

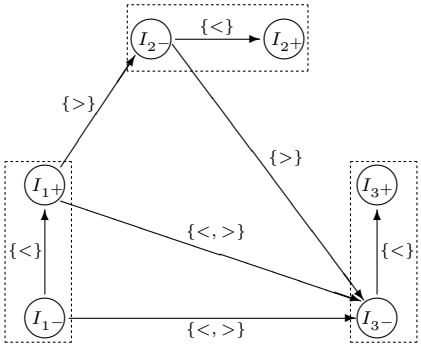
Figure 3 shows a point-based CSP corresponding to the original IA network running example from Figure 2(a), including a partial PA graph to assist in understanding the point-based CSP translation. In this graph (Figure 3(a)), each interval I_i has been replaced by its endpoints I_{i^-} (the start point) and I_{i^+} (the finish point) and all temporal relations between pairs of intervals have been replaced by corresponding relations between their endpoints. These endpoint relations are the CSP variables in the new model, which are in turn instantiated with PA atomic relations. For example, the expression $X_{1^-1^+} =$ '$<$' expresses the constraint that the edge between the endpoints I_{1^-} and I_{1^+} must be instantiated with the value '$<$', thereby expressing the underlying PA constraint $I_{1^-} < I_{1^+}$. The power we obtain from this CSP representation is that we can disallow additional unwanted interpretations that cannot be eliminated from a simple PA model. For example, in Figure 3(a) the PA graph is not a correct alternative representation of the original IA network as it allows interval I_1 to overlap (o) with interval I_3. In the CSP formalism we can disallow this overlapping relation using the third constraint in Figure 3(b):

$$\neg\,(\;X_{1^-3^-} =\text{`}<\text{'}\quad\wedge\quad X_{1^-3^+} =\text{`}<\text{'}\quad\wedge\quad X_{1^+3^-} =\text{`}>\text{'}\quad\wedge\quad X_{1^+3^+} =\text{`}<\text{'}\;)$$

It should further be noted that the order of domains in $\mathcal{D}$ is exactly preserved with respect to their corresponding variables in $\mathcal{X}$ and that all constraints of type (4) that do not further restrict the domain values of a variable have been omitted in this example.

4. Reformulation of IA into SAT

The efficiency of modern SAT solvers has triggered considerable interest in encoding problems from different domains as propositional satisfiability problems. For example, encoding binary CSPs into SAT was studied in [24,55]. In the current study we have now introduced an interval-based and a point-based non-binary IA CSP formulation. In this section, we describe three different schemes to encode these formulations into SAT, producing six new encodings.



(a) PA network

$$
\begin{aligned}
\mathcal{X} = \{\ & X_{1-1+}, && X_{2-2+}, && X_{3-3+}, && \\
& X_{1-2-}, && X_{1-2+}, && X_{1+2-}, && X_{1+2+}, \\
& X_{1-3-}, && X_{1-3+}, && X_{1+3-}, && X_{1+3+}, \\
& X_{2-3-}, && X_{2-3+}, && X_{2+3-}, && X_{2+3+} \quad \} \\
\mathcal{D} = \{\ & \{<\}, && \{<\}, && \{<\}, && \\
& \{<\}, && \{<\}, && \{>\}, && \{<\}, \\
& \{<,>\}, && \{<\}, && \{<,>\}, && \{<\}, \\
& \{>\}, && \{<\}, && \{>\}, && \{<\} \quad \} \\
\mathcal{C} = \{\ & (X_{1-1+} = \text{`}<\text{'} \ \wedge\ X_{1+3-} = \text{`}<\text{'} \ \Rightarrow\ X_{1-3-} = \text{`}<\text{'}), \\
& (X_{1+2-} = \text{`}>\text{'} \ \wedge\ X_{2-3-} = \text{`}>\text{'} \ \Rightarrow\ X_{1+3-} = \text{`}>\text{'}), \\
& \quad \neg\,(X_{1-3-} = \text{`}<\text{'} \ \wedge\ X_{1-3+} = \text{`}<\text{'} \ \wedge \\
& \quad\ \ X_{1+3-} = \text{`}>\text{'} \ \wedge\ X_{1+3+} = \text{`}<\text{'}\,) \quad \}
\end{aligned}
$$

(b) point-based CSP

Fig. 3. An example of a point-based CSP representation of an IA network.

First, we describe the one-dimensional (1-D) support scheme that naturally translates IA CSPs into CNF formulae. We then present extensions of the *direct* and *log* encoding schemes [24,55].

4.1. *The SAT* 1*-D Support Encoding*

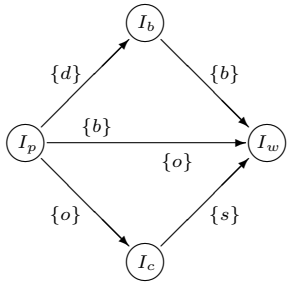
An IA network can be encoded as a SAT instance using either interval-based or point-based CSP formulations (as described in the previous section), in which each Boolean variable x_{ij}^r represents an assignment of a domain value r to a CSP variable X_{ij} such that x_{ij}^r is true iff r is assigned to X_{ij}. In a standard SAT encoding, for each CSP variable X_{ij} having a domain of values D_{ij}, two sets of at-least-one (ALO) and at-most-one (AMO) clauses are used to ensure that exactly one domain value $v \in D_{ij}$ is assigned to X_{ij} at any time:

$$ALO: \quad \bigvee_{v \in D_{ij}} x_{ij}^v \tag{6}$$

$$AMO: \quad \bigwedge_{u,v \in D_{ij}} \neg x_{ij}^u \vee \neg x_{ij}^v \tag{7}$$

It is common practice to encode a general CSP into a SAT formula without AMO clauses, thereby allowing the CSP variables to be instantiated with more than one value [55]. A CSP solution can then be extracted by taking any single SAT-assigned value for each CSP variable. However, our two CSP formulation methods strongly depend on the fact that each CSP variable can only be instantiated with exactly one value at any time. This maintains the completeness of our reformulation methods (see the proofs above). A counter-example (based on the example in Figure 1) is shown in Figure 4 where I_p is *before* I_w because I_p is *during* I_b and I_b is *before* I_w. In addition, as I_p *overlaps* I_c and I_c *starts* I_w, I_p *overlaps* I_w. As a result, I_p is either *before* or *overlaps* I_w. However, neither of the scenarios obtained from this network is consistent. Hence, the AMO clauses cannot be removed from our translation.

A natural way to encode the consistency constraints, i.e. constraints (1) and (4) above, is to add the following support (SUP) clauses:



$$X_{pb} = d \wedge X_{bw} = b \Rightarrow X_{pw} = b$$

$$X_{pc} = o \wedge X_{cw} = s \Rightarrow X_{pw} = o$$

Fig. 4. A counter-example of removing AMO clauses.

$$SUP: \bigwedge_{u \in D_{ik}, v \in D_{kj}} \neg x_{ik}^{u} \vee \neg x_{kj}^{v} \vee x_{ij}^{w_1} \vee \ldots \vee x_{ij}^{w_m} \tag{8}$$

where $D'_{ij} = D_{ij} \cap (u \circ v) = \{w_1, \ldots, w_m\}$. Note that we use the IA composition table for the interval-based reduction method and the PA composition table for the point-based reduction method.

The constraints (5) in a point-based CSP are translated into SAT formula using the following *forbidden* (FOR) clauses:

$$FOR: \bigwedge_{r \notin M_{lm}} \neg x_{l^-m^-}^{u} \vee \neg x_{l^-m^+}^{v} \vee \neg x_{l^+m^-}^{y} \vee \neg x_{l^+m^+}^{z} \tag{9}$$

where u, v, y, z are PA atomic relations and $\mu(r) = (u, v, y, z)$. For example, given the PA representation of $X_l\{o\}X_m$ is $\mu(o) = (<, <, >, <)$, the corresponding forbidden clause is $\neg x_{l^-m^-}^{<} \vee \neg x_{l^-m^+}^{<} \vee \neg x_{l^+m^-}^{>} \vee \neg x_{l^+m^+}^{<}$.

We refer to this method as the 1-D *support* encoding scheme because, it encodes the support values of the original problem. In Gent's support encoding scheme [18], the support clauses are necessary for both implication directions of the CSP constraints. However, in our scheme, only one SUP clause is needed for each triple of intervals $(i < k < j)$, and not for *all* permutation orders of this triple.

Formally, the interval and point-based SAT 1-D support encoding schemes for IA networks are described as follows:

Scheme 1 *The interval-based* 1*-D support encoding*

Input: an IA network M with n intervals

1: associate a Boolean variable x_{ij}^{r} with each assignment of relation $r \in M_{ij}$ to edge (i, j) between two intervals i and j;
2: generate ALO clauses as defined in (6);
3: generate AMO clauses as defined in (7);
4: generate SUP clauses as defined in (8);

Output: the corresponding interval-based 1D support SAT instance

4.1.1. *Example:*

The interval-based and point-based SAT 1-D support encodings of the running IA network example are respectively shown in Figures 5 and 6. Literals in the following SUP clauses are bolded to highlight the differences with other direct encoding examples in Figures 7 and 8.

Scheme 2 *The point-based* 1*-D support encoding*

Input: an IA network M with n intervals and its corresponding PA network P

1: associate a Boolean variable x_{ij}^r with each assignment of relation $r \in P_{ij}$ to edge (i, j) between two endpoints i and j;
2: generate ALO clauses as defined in (6);
3: generate AMO clauses as defined in (7);
4: generate SUP clauses as defined in (8);
5: generate FOR clauses as defined in (9);

Output: the corresponding point-based 1D support SAT instance

$$\begin{array}{ll} \text{ALO:} & (x_{12}^o) \qquad (x_{13}^b \vee x_{13}^d) \qquad (x_{23}^d) \\ \text{AMO:} & (\neg x_{13}^b \vee \neg x_{13}^d) \\ \text{SUP:} & (\neg x_{12}^o \vee \neg x_{23}^d \vee \mathbf{x_{13}^d}) \end{array}$$

Fig. 5. An interval-based 1-D support encoding of the running example.

$$\begin{array}{lllll} \text{ALO:} & (x_{1^-1^+}^<) & (x_{2^-2^+}^<) & (x_{3^-3^+}^<) & \\ & (x_{1^-2^-}^<) & (x_{1^-2^+}^<) & (x_{1^+2^-}^>) & (x_{1^+2^+}^<) \\ & (x_{1^-3^-}^< \vee x_{1^-3^-}^>) & (x_{1^-3^+}^<) & (x_{1^+3^-}^< \vee x_{1^+3^-}^>) & (x_{1^+3^+}^<) \\ & (x_{2^-3^-}^>) & (x_{2^-3^+}^<) & (x_{2^+3^-}^>) & (x_{2^+3^+}^<) \\ \text{AMO:} & (\neg x_{1^-3^-}^< \vee \neg x_{1^-3^-}^>) & & (\neg x_{1^+3^-}^< \vee \neg x_{1^+3^-}^>) & \\ \text{SUP:} & (\neg x_{1^-1^+}^< \vee \neg x_{1^+3^-}^< \vee \mathbf{x_{1^-3^-}^<}) & & (\neg x_{1^+2^-}^> \vee \neg x_{2^-3^-}^> \vee \mathbf{x_{1^+3^-}^>}) & \\ \text{FOR:} & (\neg x_{1^-3^-}^< \vee \neg x_{1^-3^+}^< \vee \neg x_{1^+3^-}^> \vee \neg x_{1^+3^+}^<) & & & \end{array}$$

Fig. 6. A point-based 1-D support encoding of the running example.

4.2. *The SAT Direct Encoding*

Another way to represent CSP constraints as SAT clauses is to encode the conflict values, i.e. the nogoods, between any pair of CSP variables [24,55]. This *direct* encoding scheme for IA networks can be derived from our 1-D support encoding scheme by replacing the SUP clauses with conflict (CON) clauses. If we represent the SUP clauses between a triple of intervals $(i < k < j)$ as a 3D array of allowable values for the CSP variable X_{ij}, given the values of X_{ik} and X_{kj}, then the corresponding CON clauses can be defined as:

$$CON: \bigwedge_{u \in D_{ik}, v \in D_{kj}, w \in D''_{ij}} \neg x_{ik}^u \vee \neg x_{kj}^v \vee \neg x_{ij}^w \tag{10}$$

where $D''_{ij} = D_{ij} - (u \circ v)$.

The *multivalued* encoding [41] is a variation of the direct encoding, where all AMO clauses are omitted. As discussed earlier, we did not consider such an encoding because in our IA transformations the AMO clauses play a necessary role.

Scheme 3 *The interval-based direct encoding*

Input: an IA network M with n intervals

1: associate a Boolean variable x_{ij}^r with each assignment of relation $r \in M_{ij}$ to edge (i, j) between two intervals i and j;
2: generate ALO clauses as defined in (6);
3: generate AMO clauses as defined in (7);
4: generate CON clauses as defined in (10);

Output: the corresponding interval-based direct SAT instance

Scheme 4 *The point-based direct encoding*

Input: an IA network M with n intervals and its corresponding PA network P

1: associate a Boolean variable x_{ij}^r with each assignment of relation $r \in P_{ij}$ to edge (i, j) between two endpoints i and j;
2: generate ALO clauses as defined in (6);
3: generate AMO clauses as defined in (7);
4: generate CON clauses as defined in (10);
5: generate FOR clauses as defined in (9);

Output: the corresponding point-based direct SAT instance

4.2.1. *Example:*

The interval-based and point-based SAT direct encodings of the running IA network example are respectively shown in Figures 7 and 8. Literals in the following CON clauses are bolded to highlight the differences with other 1-D support encoding examples in Figures 5 and 6.

$$
\begin{array}{ll}
\text{ALO:} & (x_{12}^{o}) \qquad (x_{13}^{b} \vee x_{13}^{d}) \qquad (x_{23}^{d}) \\
\text{AMO:} & (\neg x_{13}^{b} \vee \neg x_{13}^{d}) \\
\text{CON:} & (\neg x_{12}^{o} \vee \neg x_{23}^{d} \vee \neg \mathbf{x_{13}^{b}})
\end{array}
$$

Fig. 7. An interval-based direct encoding of the running example.

$$
\begin{array}{lllll}
\text{ALO:} & (x_{1^-1^+}^{<}) & (x_{2^-2^+}^{<}) & (x_{3^-3^+}^{<}) & \\
 & (x_{1^-2^-}^{<}) & (x_{1^-2^+}^{<}) & (x_{1^+2^-}^{>}) & (x_{1^+2^+}^{<}) \\
 & (x_{1^-3^-}^{<} \vee x_{1^-3^-}^{>}) & (x_{1^-3^+}^{<}) & (x_{1^+3^-}^{<} \vee x_{1^+3^-}^{>}) & (x_{1^+3^+}^{<}) \\
 & (x_{2^-3^-}^{>}) & (x_{2^-3^+}^{<}) & (x_{2^+3^-}^{>}) & (x_{2^+3^+}^{<}) \\
\text{AMO:} & (\neg x_{1^-3^-}^{<} \vee \neg x_{1^-3^-}^{>}) & & (\neg x_{1^+3^-}^{<} \vee \neg x_{1^+3^-}^{>}) & \\
\text{CON:} & (\neg x_{1^-1^+}^{<} \vee \neg x_{1^+3^-}^{<} \vee \neg \mathbf{x_{1^-3^-}^{>}}) & & (\neg x_{1^+2^-}^{>} \vee \neg x_{2^-3^-}^{>} \vee \neg \mathbf{x_{1^+3^-}^{<}}) & \\
\text{FOR:} & (\neg x_{1^-3^-}^{<} \vee \neg x_{1^-3^+}^{<} \vee \neg x_{1^+3^-}^{>} \vee \neg x_{1^+3^+}^{<}) & & &
\end{array}
$$

Fig. 8. A point-based direct encoding of the running example.

4.3. *The SAT Log Encoding*

A more compact version of the direct encoding is the *log* encoding [24,55]. Here, a Boolean variable x_i^l is true iff the corresponding CSP variable X_i is assigned a value in which the l-th bit of that value is 1. We can linearly derive log encoded IA instances from direct encoded IA instances by replacing each Boolean variable in the direct encoding with its *bitwise representation*. As a single instantiation of the underlying CSP variable is enforced by the bitwise representation, ALO and AMO clauses are omitted. However, extra bitwise prohibited (PRO) clauses are needed (if necessary) to prevent bitwise representations of undesired Boolean variables from being instantiated. For example, if the domain of variable X has three values then we will need a minimum of two bits to represent those values. As two bits can represent four values, we have to add the clause $\neg x_3^0 \vee \neg x_3^1$ to prevent the fourth value from assigning to X. Another way to remove redundant bitwise representations is to map them to valid values. However, this *binary* encoding [17] can generate exponentially more conflict clauses than the log encoding and hence is not considered further in this study.

Scheme 5 *The interval-based log encoding*

Input: an IA network M with n intervals

1: associate a *temporary* Boolean variable p_{ij}^r with each assignment of relation $r \in M_{ij}$ to edge (i, j) between two intervals i and j;
2: generate CON clauses as defined in (10);
3: replace each occurrence of a *temporary* Boolean variable p_{ij}^r with its bitwise representation bw_{ij}^r, note that each bw_{ij}^r is actually a combination of Boolean variables lv_k;
4: generate PRO clauses if necessary;

Output: the corresponding interval-based log SAT instance

Scheme 6 *The point-based log encoding*

Input: an IA network M with n intervals and its corresponding PA network P

1: associate a *temporary* Boolean variable p_{ij}^r with each assignment of relation $r \in P_{ij}$ to edge (i, j) between two endpoints i and j;
2: generate CON clauses as defined in (10);
3: generate FOR clauses as defined in (9);
4: replace each occurrence of a *temporary* Boolean variable p_{ij}^r with its bitwise representation bw_{ij}^r, note that each bw_{ij}^r is actually a combination of Boolean variables lv_k;
5: generate PRO clauses if necessary;

Output: the corresponding point-based log SAT instance

4.3.1. *Example:*

The interval-based and point-based SAT log encodings of the running IA network example are respectively shown in Figures 9 and 10.

5. Beyond SAT: The SMT Encoding

Over the last few years, there has been increasing interest in the SAT community to develop an alternative to SAT that can deal effectively with non-Boolean problems. In this paper, we

$$\text{PRO:} \quad (\neg x^1_{12}) \qquad (\neg x^1_{23})$$
$$\text{CON}_l\text{:} \quad (x^1_{12} \vee x^1_{23} \vee x^1_{13})$$

Fig. 9. An interval-based log encoding of the running example.

$$\begin{aligned}
\text{PRO:} \quad & (\neg x^1_{1^-1^+}) && (\neg x^1_{2^-2^+}) && (\neg x^1_{3^-3^+}) && \\
& (\neg x^1_{1^-2^-}) && (\neg x^1_{1^-2^+}) && (\neg x^1_{1^+2^-}) && (\neg x^1_{1^+2^+}) \\
& && (\neg x^1_{1^-3^+}) && && (\neg x^1_{1^+3^+}) \\
& (\neg x^1_{2^-3^-}) && (\neg x^1_{2^-3^+}) && (\neg x^1_{2^+3^-}) && (\neg x^1_{2^+3^+}) \\
\text{CON}_l\text{:} \quad & (x^1_{1^-1^+} \vee x^1_{1^+3^-} \vee \neg x^1_{1^-3^-}) && && (x^1_{1^+2^-} \vee x^1_{2^-3^-} \vee x^1_{1^+3^-}) && \\
\text{FOR:} \quad & (x^1_{1^-3^-} \vee x^1_{1^-3^+} \vee \neg x^1_{1^+3^-} \vee x^1_{1^+3^+}) && && && \\
\end{aligned}$$

Fig. 10. A point-based log encoding of the running example.

are particularly interested in SMT techniques, which are designed to decide the satisfiability of quantifier-free (QF) first-order logic formulae with respect to one or more background decidable theories (such as linear arithmetic, the theory of numbers, the theory of arrays or the theory of bit-vectors). Apart from successful applications in formal verification and compiler optimisation, SMT solvers are also the best for solving quantitative temporal reasoning problems such as STPs, TCSPs and DTPs.

An SMT problem is defined by a combination of a Boolean formula and predicates in other theories. Deciding the satisfiability of an SMT problem involves finding an assignment for all Boolean and theory specific variables that satisfies the Boolean formula and all theory specific predicates or proving that there is no such assignment. Consequently, developing an efficient SMT solver has involved finding a good integration between SAT and theory-specific techniques.

Currently, many SMT solvers support difference constraints, i.e. constraints in the form of $x - y \bowtie b$ where x and y are real- or integer-valued variables, b is a numeric constant and $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$. These types of constraints are called QF-RDL or QF-IDL (i.e. the quantifier-free real difference logic or the quantifier-free integer difference logic) depending on whether the values of x and y are real or integer numbers. In this paper, we use the QF-IDL logic to translate an IA network M with n intervals into a point-based SMT formula with $2n$ integer-valued variables, where each SMT variable x_i represents an endpoint P_i of the corresponding PA network P (with $2n$ points, P_1,...,P_{2n}) of M. [5]

We then add the following domain (DOM) constraints to ensure that the instantiation of SMT variables x_i and x_j satisfies the set of point relations P_{ij} between two endpoints P_i and P_j in P. Note that $u \in \{<, =, >\}$ is a PA atomic relation.

$$DOM: \quad \bigvee_{u \in P_{ij}} x_i \; u \; x_j \tag{11}$$

Here, we do not need to add constraints to ensure the path consistency between any triplets $(i < j < k)$ of endpoints of P as we did for the point-based CSP formulation in Section 3.2. As each SMT variable x_i is instantiated with integer values, an instantiation of x_i represents the position of the corresponding endpoint P_i in the time line. Consequently, a solution to this SMT

[5] P may allow relations that are not allowed in M.

formula (i.e. all DOM constraints are satisfied) represents a consistent mapping of all endpoints of P onto the time line, i.e. a consistent scenario of P.

Finally, we add the following forbidden (FOR) constraints to rule out all IA relations r that are not in the set of IA relations M_{lm} between two intervals I_l and I_m but may be enabled during the translation of the original IA network M into its corresponding PA network P:

$$FOR: \quad \bigwedge_{r \notin M_{lm}} \neg(x_{l^-}\ u\ x_{m^-}) \vee \neg(x_{l^-}\ v\ x_{m^+}) \vee \neg(x_{l^+}\ y\ x_{m^-}) \vee \neg(x_{l^+}\ z\ x_{m^+}) \quad (12)$$

where u, v, y, z are PA atomic relations and $\mu(r) = (u, v, y, z)$.

Formally, the point-based SMT QF-IDL encoding scheme (or in short, the point-based SMT encoding) for IA networks is described as follows:

Scheme 7 *The point-based SMT QF-IDL encoding*

Input: an IA network M with n intervals and its corresponding PA network P

1: associate an integer-valued variable x_i with each endpoint P_i in P.
2: generate DOM constraints as defined in (11);
3: generate FOR constraints as defined in (12);

Output: the corresponding point-based SMT QF-IDL formula

Theorem 10 *Let Θ be an IA network and Π be the corresponding point-based SMT formula formulated by Algorithm 7. Then Θ is satisfiable iff Π is satisfiable.*

PROOF. The proof of this Theorem is trivial and can be constructed in a similar manner as the proof of Theorem 9.

5.1. *Example:*

The point-based SMT encoding of the running IA network example is shown in Figure 11 using the Yices' input language. [6] The 'run' commands are to tell Yices to check for the satisfiability of the input formula and to return a satisfiable model if one exists.

6. Test Instances and Algorithm Selection

6.1. *Test Instances*

As there is a lack of large sized realistic IA benchmarks, the majority of empirical studies in this field have relied on randomly generated IA problems [29,30,53,36,31]. This reliance has the advantage of allowing researchers to "investigate how algorithmic performance depends on problem characteristics" and be able to "predict how an algorithm will perform on a given problem class" [23].

Among the different current models for randomly generating IA instances [53,36,31], the models proposed by Nebel [36] provide more control of the characteristics of generated instances. We therefore based our empirical study on random problems generated using Nebel's $A(n, d, s)$ model [36]. The resulting instances are temporal constraint graphs with n nodes (i.e. intervals)

[6] See http://yices.csl.sri.com/language.shtml

TYPE: (define-type $dVal$ (subrange 1 6))

VAR: (define x_{1^-}::$dVal$) (define x_{2^-}::$dVal$) (define x_{3^-}::$dVal$)

(define x_{1^+}::$dVal$) (define x_{2^+}::$dVal$) (define x_{3^+}::$dVal$)

DOM: (assert ($< x_{1^-}\ x_{1^+}$)) (assert ($< x_{2^-}\ x_{2^+}$)) (assert ($< x_{3^-}\ x_{3^+}$))

(assert ($< x_{1^-}\ x_{2^-}$)) (assert ($< x_{1^-}\ x_{2^+}$))

(assert ($> x_{1^+}\ x_{2^-}$)) (assert ($< x_{1^+}\ x_{2^+}$))

(assert (or ($< x_{1^-}\ x_{3^-}$) ($> x_{1^-}\ x_{3^-}$))) (assert ($< x_{1^-} x_{3^+}$))

(assert (or ($< x_{1^+}\ x_{3^-}$) ($> x_{1^+}\ x_{3^-}$))) (assert ($< x_{1^+}\ x_{3^+}$))

(assert ($> x_{2^-}\ x_{3^-}$)) (assert ($< x_{2^-}\ x_{3^+}$))

(assert ($> x_{2^+}\ x_{3^-}$)) (assert ($< x_{2^+}\ x_{3^+}$))

FOR: (assert (or (not ($< x_{1^-}\ x_{3^-}$)) (not ($< x_{1^-}\ x_{3^+}$))

(not ($> x_{1^+}\ x_{3^-}$)) (not ($< x_{1^+}\ x_{3^+}$))))

RUN: (set-evidence! true)

(check)

Fig. 11. A point-based SMT encoding (in Yices' input language) of the running example.

and an average degree of d constrained edges (i.e. interval relations). Constrained edges are then uniformly labelled with an average size of s IA atomic relations, where $1 \leq s \leq 12$. Unconstrained edges are labelled with all 13 IA atomic relations.

We chose the $A(n, d, s)$ model as it allows the constraints to be uniformly chosen from the entire set of IA relations and not limited to only a small sets of very hard constraints like the $H(n, d)$ model, or to generating only satisfiable instances like the $S(n, d, s)$ model [36]. In his study, Nebel [36] pointed out that the average degree d is a critical parameter that can define the phase transition of IA instances independently from the number of nodes. He showed that when s is fixed to 6.5, the phase transition happens around $d = 9.5$.

To investigate the performance effects of our six SAT encoding schemes and one SMT encoding, we use Nebel's generator [7] to generate IA instances where the number of nodes was varied from 20 to 100. We then pre-processed these IA instances using the path consistency algorithm proposed by van Beek [53] before encoding them into SAT or SMT formulae. We found that this preprocessing significantly reduced the problem size of SAT-encoded and SMT-encoded IA instances both in terms of the number of variables and the number of clauses. Section 7.1 discusses in detail the effects of this pre-processing approach on our SAT encodings.

6.2. *Algorithm Selection*

SAT solving is one of the most active research areas in computer science and artificial intelligence. It has been shown that SLS techniques (such as the WalkSAT family [46]) and recent

[7] Available for download at ftp://ftp.informatik.uni-freiburg.de/documents/papers/ki/allen-csp-solving.programs.tar.gz

clause weighting techniques (such as SAPS [25] and PAWS [49]) are significantly better than systematic search on random SAT problems. However, although SLS solvers are the best choice for many realistic problems [28,6,27], systematic solvers were still the leading performers in the recent SAT competitions [8] on the crafted and industrial categories problems, where test cases are highly structured.

As local search solvers (e.g. PAWS [49] and its many-valued variant MV-PAWS [40]) performed poorly in comparison with the systematic solver zChaff [35] (version 2004.11.25) on all six different SAT encodings of IA instances in our preliminary study [38], we decided to use only systematic search in this study. [9] Another reason for choosing systematic solvers is that they work on both satisfiable and unsatisfiable instances and hence allow us to evaluate the performance of SAT solvers on the phase transition of IA instances.

In this study, we selected MiniSAT [14] version 2 (build 070721) [10] as our benchmarking SAT solver as it is one of the best systematic solvers for both random and structured problems. MiniSAT uses an efficient mechanism for learning clauses and won the championship in the 2005 SAT competition. As our SMT solver, we used Yices [13] version 1.0.9 [11] as it was shown to be the best overall SMT solver in the 2006 and 2007 SMT competitions. [12]

To compare with existing IA approaches, we chose Nebel's solver [36] as it is the best known systematic solver for native IA representation. [13] In particular, we used two variants of Nebel's backtracking algorithm, NBT$_{\mathcal{I}}$ and NBT$_{\mathcal{H}}$. NBT$_{\mathcal{I}}$ instantiates each edge with an atomic relation in $\mathcal{I}$, whereas NBT$_{\mathcal{H}}$ assigns a relation in the set $\mathcal{H}$ of ORD-Horn relations to each edge. The other heuristics used in Nebel's backtracking algorithm were set to default.

Finally, we included TSAT [48], the specialised SLS solver that uses an end point ordering representation of IA instances, described in Section 2.3. By exploiting special heuristics suited to this representation (such as domain and constraint skipping) TSAT was shown to perform better than both NBT$_{\mathcal{I}}$ and NBT$_{\mathcal{H}}$ on hard IA instances [48].

7. The Effects of Encoding on the Problem Size

In 1995, Nebel and Bürckert [37] pointed out that qualitative temporal instances can be translated to SAT instances but such a translation causes an exponential blowup in problem size. In order to study the effect of our SAT translation on the size of the generated problems, we first generated an extensive benchmark test set of $A(n,\, d, 6.5)$ IA instances by varying the average degree d from 1 to 20 (in steps of 0.5 from 8 to 11 and in steps of 1 otherwise) and n from 20 to 50 (in steps of 5). We generated 500 instances for each n/d data point to obtain a set of $23 \times 7 \times 500 = 80,500$ test instances. We then used our SAT and SMT encoding schemes to translate these instances to SAT and SMT formulae respectively. Figure 12 graphs the median number of variables, clauses and literals of the corresponding SAT and SMT encodings and the median time for translation. We use PS, PD, and PL to denote the point-based 1-D support, di-

[8] Results for SAT competitions are available at http://www.satcompetition.org/

[9] Although PAWS with its weight decay parameter fixed to 10 lost to R+AdaptNovelty$^+$ in the 2005 SAT competition, a tuned version of PAWS strongly dominates other local search solvers [2]. In our preliminary study, both variants of PAWS were run with their optimal tuned parameter settings.

[10] Source code is available at http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html

[11] Binary version is available at http://yices.csl.sri.com/

[12] Results for SMT competitions are available at http://www.smtcomp.org

[13] Source code is available at ftp://ftp.informatik.uni-freiburg.de/documents/papers/ki/allen-csp-solving.programs.tar.gz

rect and log SAT instances; IS, ID and IL to denote the interval-based 1-D support, direct and log SAT instances; and SMT to denote the point-based SMT instances; respectively.

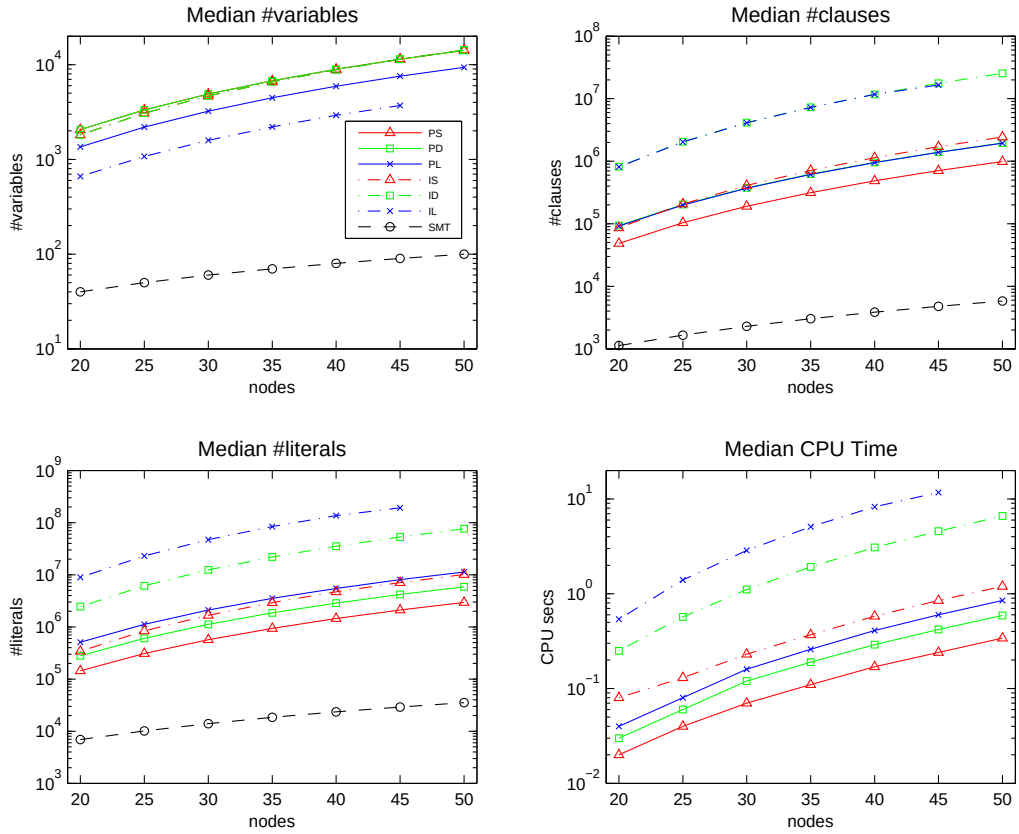


Fig. 12. The median number of variables, clauses and literals, and the median time taken to translate $A(n, d, 6.5)$ IA instances into SAT- and SMT-encoded instances without path consistency preprocessing ($11,500$ instances, where $d = 1 - 20$, per each data point). The CPU time of the point-based SMT encoding is omitted as the translation was almost instant.

The results show that the point-based SMT encoding dominates the other encoding schemes in producing both the smallest problem sizes and the fastest translation times. As graphed in Figure 12, the SMT encoding produces instances that are at least 40 times smaller than its closest SAT rival. In addition, the translation of IA instances into SMT formulae happened almost instantly (hence the SMT translation times are omitted from Figure 12). This result is not surprising as the translation of an IA network into an SMT formula is a relatively straightforward carry over from the corresponding PA network.

Putting aside the results of the SMT encoding, the graphs in Figure 12 clearly show that point-based SAT encoding produces instances with significantly smaller problem sizes (in terms of the number of clauses) and in a shorter time than interval-based SAT encoding. Among the three SAT encoding schemes, 1-D support was the best, followed by the direct encoding scheme. Although the interval-based and point-based log schemes generated SAT instances with the smallest number of variables, they took the longest time to translate and produced instances with more

lengthy clauses. As shown in Figure 12, the median CPU time used for the point-based 1-D support scheme was from 2 to 20 times faster than other SAT encoding schemes. Similar results were also observed when comparing the median number of clauses and literals of the point based 1-D support SAT instances with the other SAT encoded instances.

7.1. *The Impact of Path Consistency Preprocessing*

Since the path consistency algorithm is an efficient approximation approach to solve IA problems [51,29,53], most successful IA backtracking algorithms use it before and during the search to remove as many inconsistent relations between intervals (or endpoints) as possible [30,53,36,31]. Hence, it would appear reasonable to try and minimise the problem size of the SAT and SMT encoded instances by applying path consistency algorithms to native IA instances before translation. In addition, as path consistency algorithms are polynomial, we would expect a combination of path consistency and SAT (or SMT) to be more efficient than trying to solve unpreprocessed SAT (or SMT) encoded instances.

Here, we used the vBM path consistency algorithm [53], [14] taken from Nebel's backtracking solvers (NBT$_{\mathcal{I}}$ and NBT$_{\mathcal{H}}$), to evaluate the impacts of preprocessing native IA instances before SAT or SMT translations. We ran the vBM algorithm on all benchmark instances generated in the previous experiment. Figure 13 graphs the median number of variables, clauses and literals, and the median time taken to translate these IA instances, with and without the path consistency preprocessing, into point-based SAT instances.

The results clearly show the advantage of preprocessing native IA instances using path consistency algorithms before SAT translation. With path consistency preprocessing, the higher the probability of inconsistency in the IA instances (i.e. the higher average degree d), the more improvement in minimising the problem size of SAT encoded instances was observed. Similar benefits were also observed when combining path consistency with other SAT and SMT encoding schemes.

8. The Phase Transition of SAT/SMT Encodings

An important advantage of using randomly generated benchmark test sets is that the average difficulty of these instances can be controlled. Cheeseman *et al.* [8] pointed out that the satisfiability distribution of problem sets can be separated into two regions: one where most of the problems have many solutions and are relatively easy to solve; and one where most of the problems are unsatisfiable and this is also relatively easy to prove. They conjectured that the phase transition, the border between the two regions, depends strongly on the critical value of at least one order parameter. They also found that instances around the phase transition are harder to solve than other instances. Later, several empirical studies reconfirmed this conjecture [42,47].

Ladkin and Reinefeld [30] were the first to study the phase transition of IA instances. They observed this transition to be in the range of $6 \leq c \times n \leq 15$ for $c \geq 0.5$, where c is the completion coefficient [29] and n is the number of intervals. However, this phase transition depends on the problem size n, and hence does not allow "the generation of arbitrarily hard instances" [36]. In his study, Nebel [36] pointed out that the average degree d is a critical parameter that can define the phase transition of IA instances independently of the number of nodes. He showed that with

[14] The source code of this path consistency algorithm was included in Nebel's solvers.

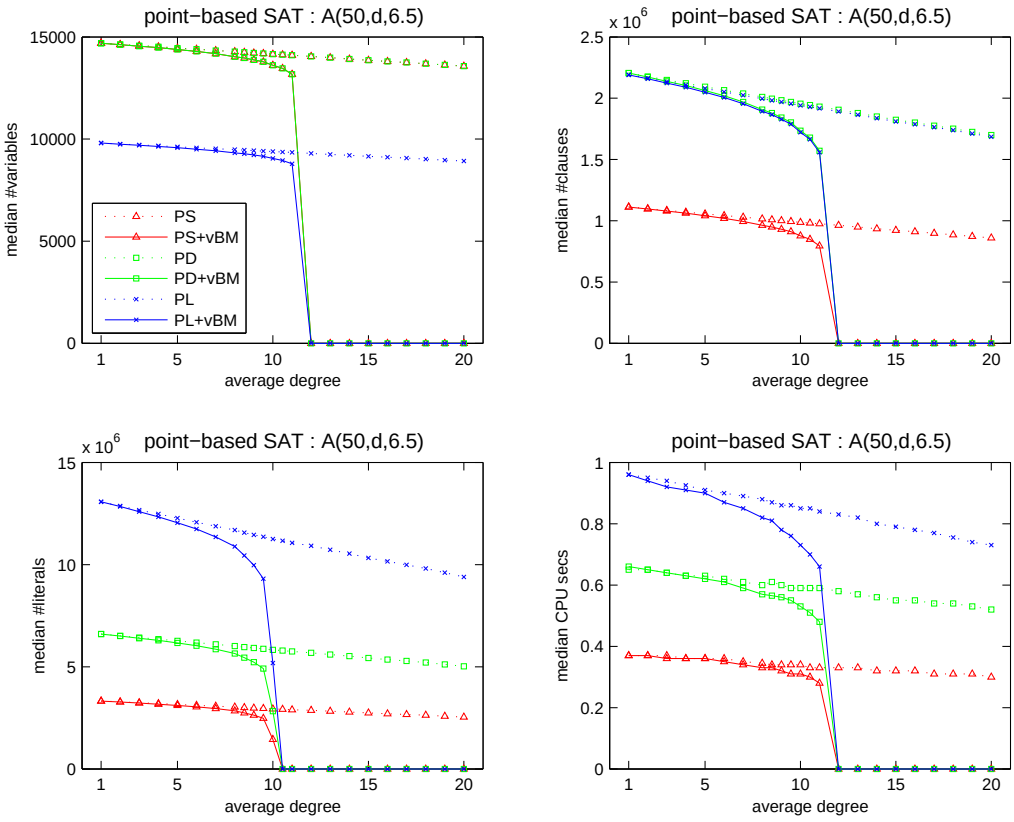


Fig. 13. The median number of variables, clauses and literals, and the median time taken to translate $A(50, d, 6.5)$ IA instances into the point-based SAT instances with and without path consistency preprocessing (500 instances, where $d = 1 - 20$, per each data point).

s fixed to 6.5, the phase transition happens around $d = 9.5$ for the $A(n, d, s)$ model. He also observed that the runtime of his backtracking algorithm peaked at $d = 9.5$ for $s = 6.5$.

As our SAT and SMT translation methods were theoretically proved sound and complete, we expected that the following properties would also be true for our SAT-encoded and SMT-encoded IA instances:

i) The phase transition of SAT-encoded and SMT-encoded instances happens at the same critical value of the average degree parameter d as for the original IA instances; and
ii) The performance of SAT solvers on SAT-encoded instances is proportionally similar to the performance of temporal backtracking algorithms on the original IA instances; and
iii) The performance of SMT solvers on SMT-encoded instances is proportionally similar to the performance of temporal backtracking algorithms on the original IA instances.

In order to verify these properties, we conducted a similar experiment to that reported in Nebel's study [36]. We ran NBT$_{\mathcal{I}}$ and NBT$_{\mathcal{H}}$ on IA instances generated in the previous section and MiniSAT and Yices respectively on the corresponding SAT-encoded and SMT-encoded instances. All solvers were timed out after one hour.

As expected, the probability of satisfiability for our SAT-encoded and SMT-encoded instances was the same as the probability of satisfiability for the original IA instances, regardless of the

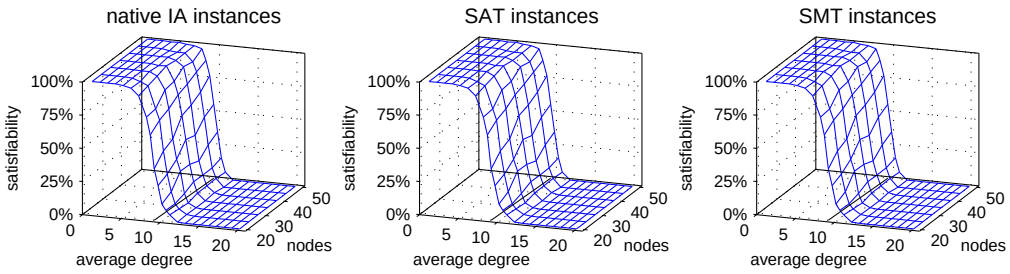


Fig. 14. The probability of satisfiability for $A(n, d, 6.5)$ (500 instances per data point).

translation method. This is illustrated in Figure 14 which shows that the phase transition happens around $d = 9.5$ for $s = 6.5$ regardless of instance size or representation. These results are consistent with the earlier work of Nebel [36].

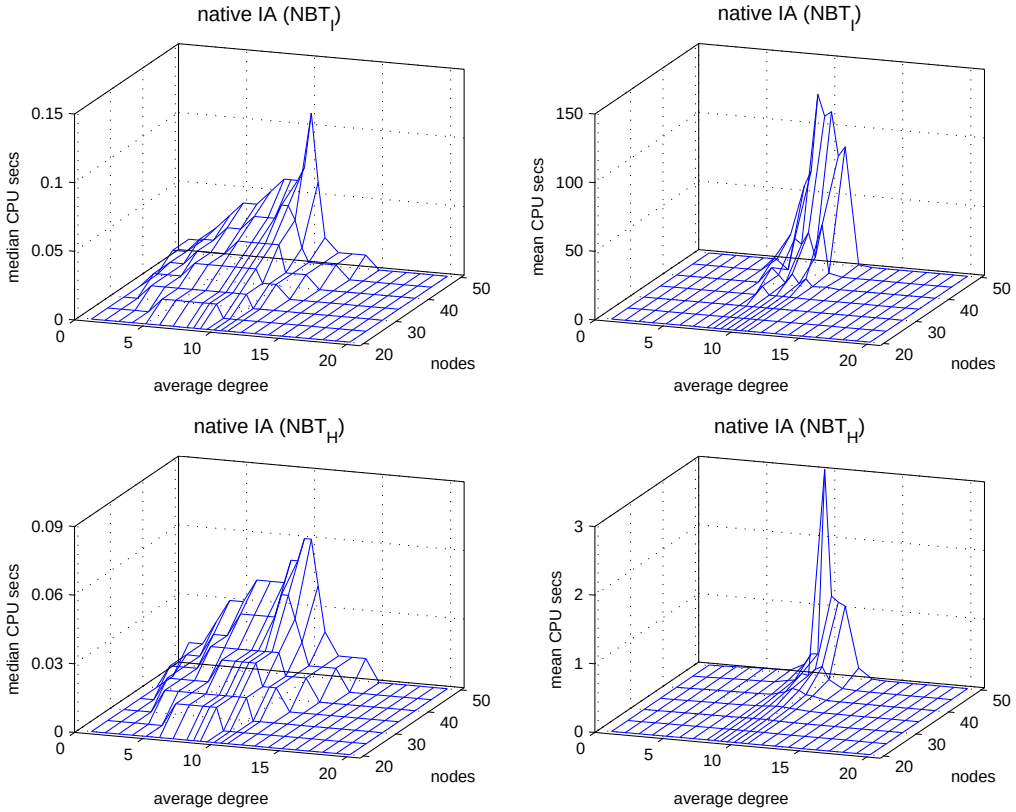


Fig. 15. The performance of NBT$_{\mathcal{I}}$ and NBT$_{\mathcal{H}}$ on native IA instances.

However, the performance of MiniSAT on our six different SAT encodings was significantly different from the performance of NBT$_{\mathcal{I}}$ and NBT$_{\mathcal{H}}$ on the native IA representations. As graphed in Figure 15, the runtime of NBT$_{\mathcal{I}}$ and NBT$_{\mathcal{H}}$ both peaked around the phase transition point, i.e. $d = 9.5$. In contrast, the runtime peaks of MiniSAT on the SAT instances were shifted away from

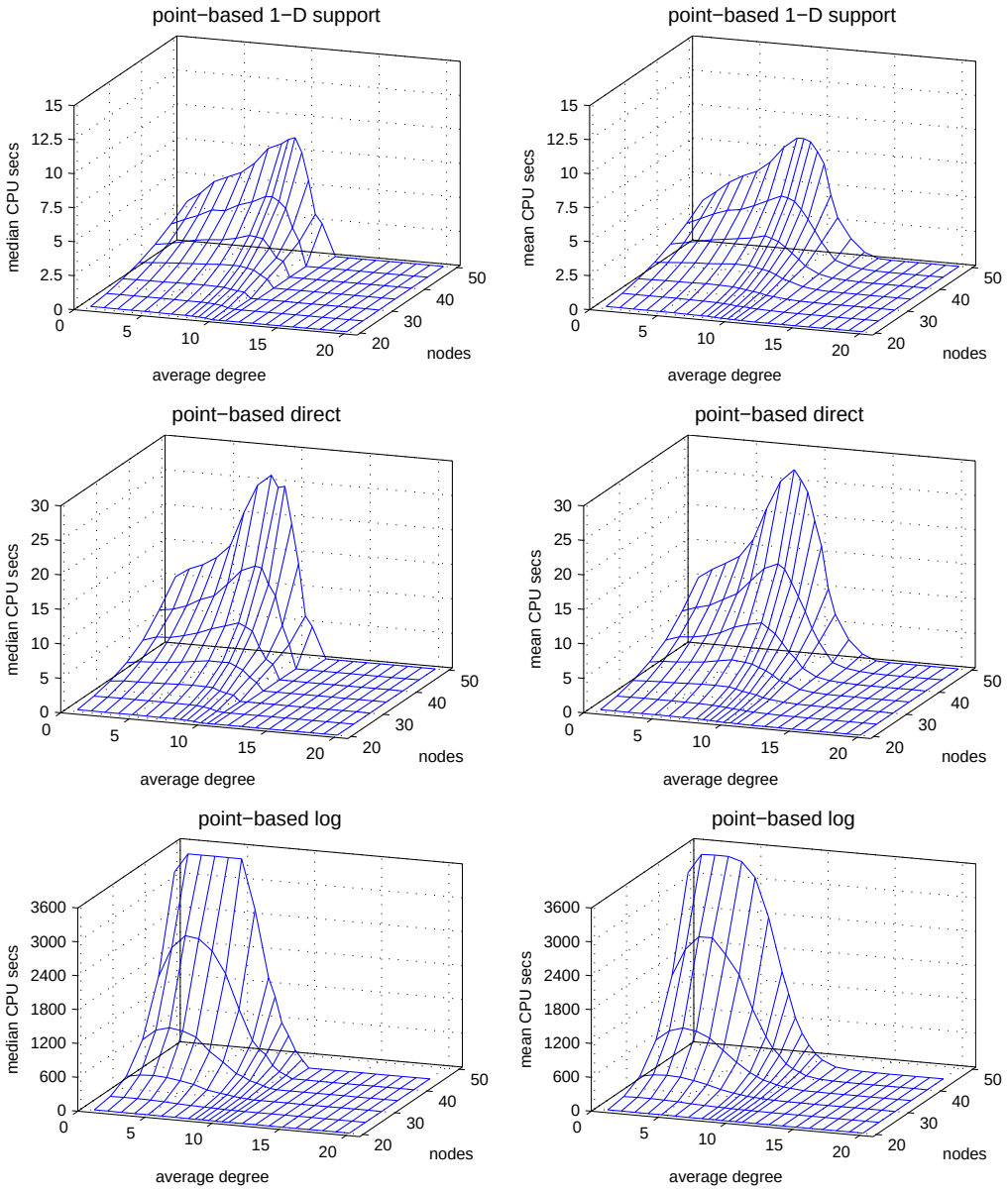


Fig. 16. The performance of MiniSAT on point-based 1-D support, direct and log SAT-encoded instances.

the phase transition. Figure 16 shows that the median and mean CPU time of MiniSAT on the point-based 1-D support, direct and log SAT-encoded instances peaked around a d value of 8, 8 and 3, respectively. In addition, the CPU time of MiniSAT on instances surrounding these peaks was relatively similar, regardless of which SAT encoding scheme was used.

This result is further supported when we take into account the performance of MiniSAT on the interval-based SAT instances. Figure 17 graphs the median and mean CPU time of MiniSAT

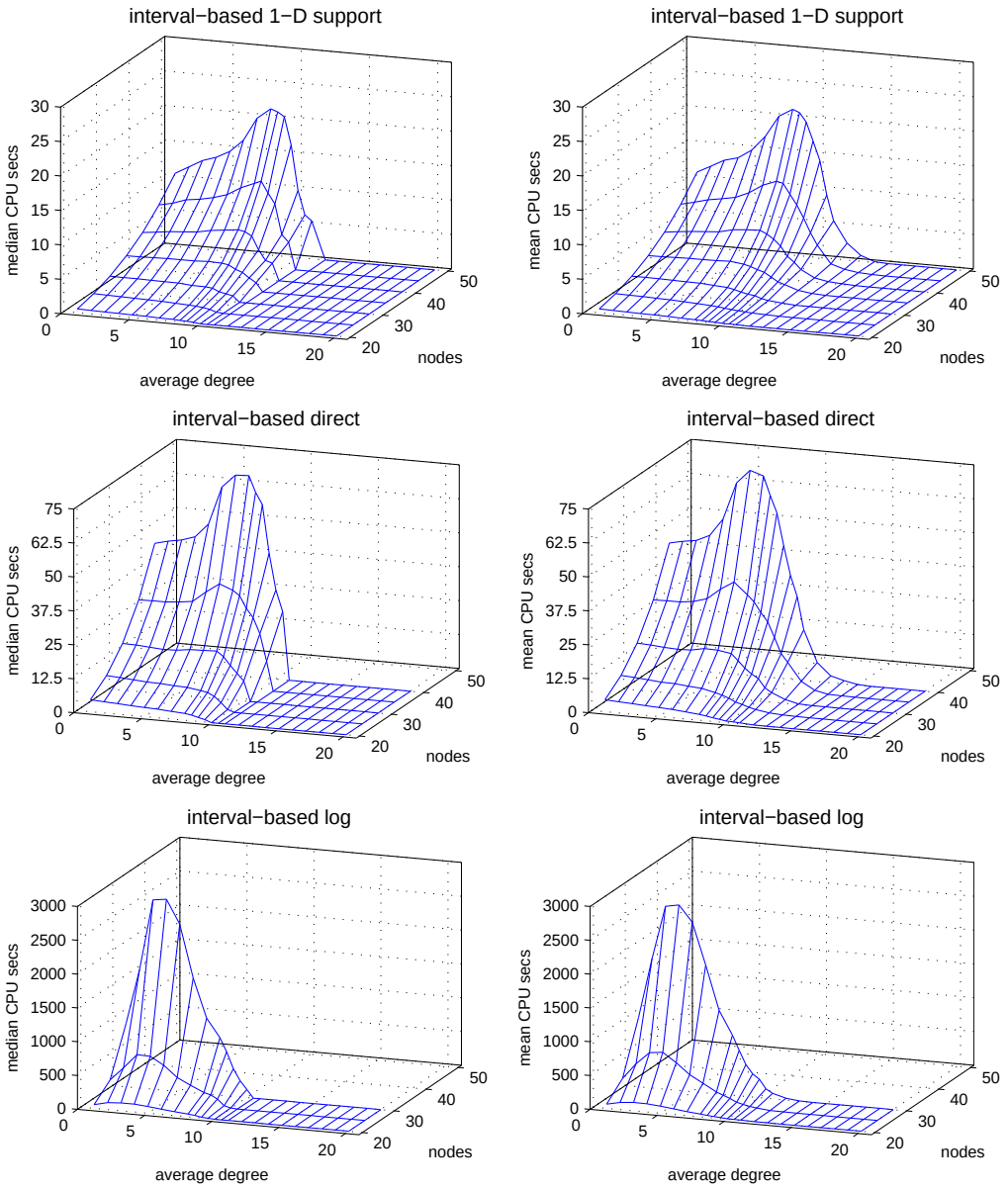


Fig. 17. The performance of MiniSAT on interval-based 1-D support, direct and log SAT-encoded instances.

on the corresponding interval-based 1-D support, direct and log SAT-encoded instances, respectively. Here we can see that the runtime peaks of MiniSAT are shifted away from the phase transition in the same as they were on the point-based SAT instances, regardless of which SAT encoding scheme is used. However, the CPU time of MiniSAT on the interval-based direct and log instances peaked at slightly different points in comparison to their corresponding point-based instances, i.e. with d equal to 7 (compared to 8) and 4 (compared to 3), respectively. Only the

runtime of MiniSAT on the interval-based 1-D support instances remained the same as the point-based instances (both peaking at 8).

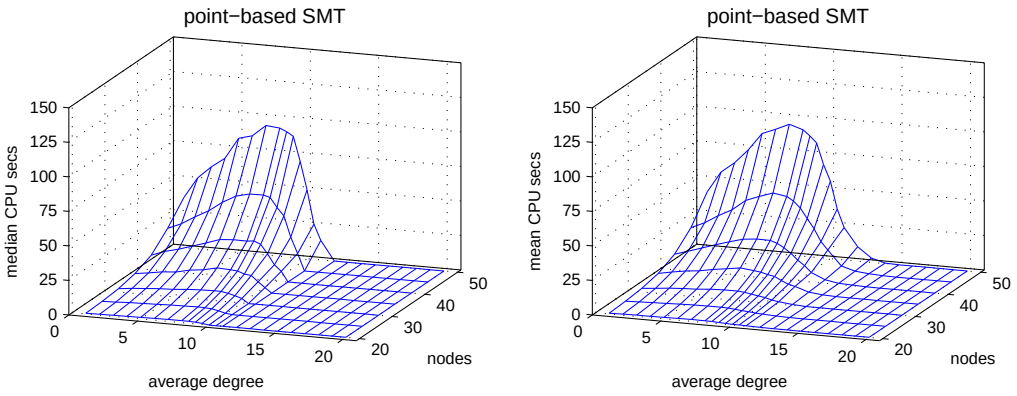


Fig. 18. The performance of Yices on point-based SMT-encoded instances.

A similar story develops when we look at the performance of Yices on the SMT-encoded instances. Figure 18 graphs the median and mean CPU time of Yices on the point-based SMT instances. Here we can see that the runtime peaks of Yices (at $d = 7$) are shifted further away from the phase transition than the runtime peaks achieved by MiniSAT (at $d = 8$) on both the point-based 1-D support and direct SAT instances.

These results are quite surprising and contrast with the results of previous studies on the phase transition of random problems [8,36,45,47]. Intuitively, the further left we move from the phase transition, the more solutions an instance has and as a consequence, the easier this instance should be to solve. However, this conjecture is not true for our SAT and SMT encoding schemes. Our empirical results clearly show that the hard region, where instances take significantly more time to solve, does not always happen around the phase transition. In contrast, the representation or encoding of the problem instance plays an important role in determining where hard region will occur.

9. An Empirical Comparison between SAT and SMT Encodings

From the graphs in Figures 16-18, we can conclude that:

i) A point-based formulation method produces better results than an interval-based formulation method, regardless of how IA instances are generated (in terms of the number of nodes n, the average degree d or the average label size s) or the SAT encoding employed.

ii) The 1-D support encoding scheme produces the best results, followed by the direct and log encoding schemes, regardless of how IA instances being generated (in terms of the number of nodes n, the average degree d or the average label size s) or the formulation method employed.

iii) SAT encoding schemes produce better results than the SMT encoding scheme. In particular, the performance of Yices on SMT instances is worse than the performance of MiniSAT on the top three SAT encodings (i.e. point-based and interval-based 1-D support and point-based direct encodings) but is better than the performance of MiniSAT on the other three SAT encodings.

iv) Among the six SAT encoding schemes considered, the point-based 1-D support encoding is the most suitable scheme to translate IA instances into SAT formulae.

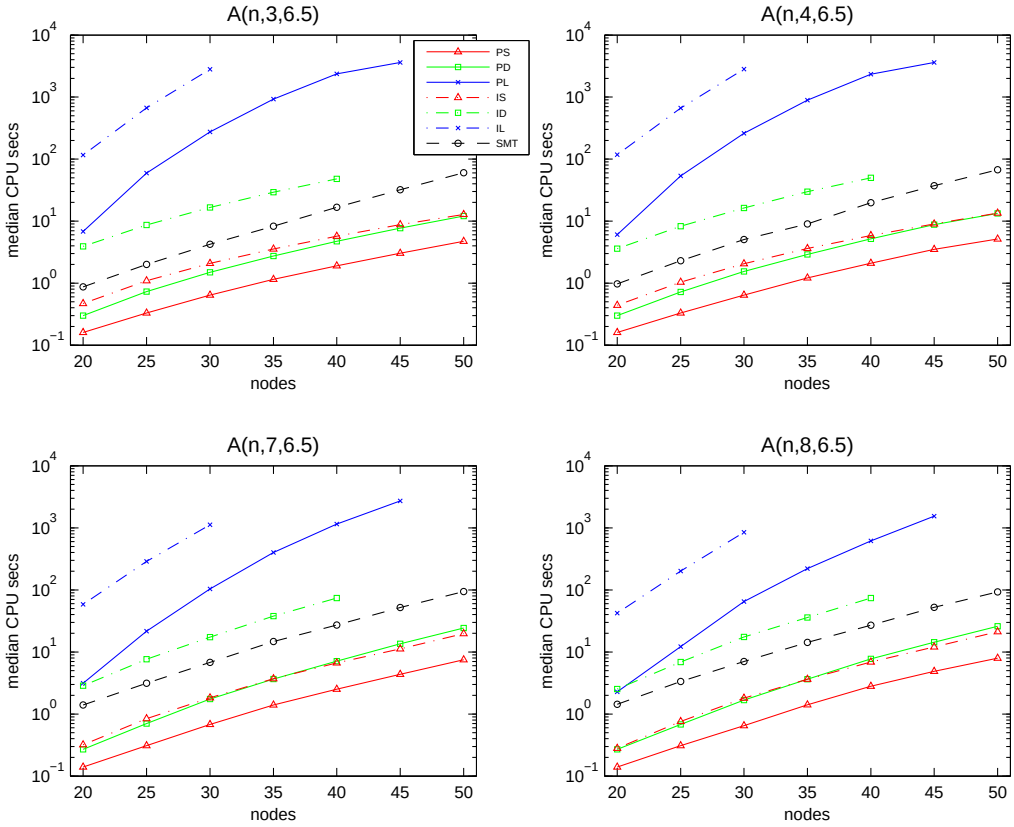


Fig. 19. The performance of MiniSAT and Yices on SAT-encoded and SMT-encoded IA instances respectively. The 'SMT' key represents Yices. All other keys represent MiniSAT on the various encodings, where 'P' = point-based, 'I' = interval-based, 'S' = support, 'D' = direct and 'L' = log.

Figure 19 graphs the median CPU time of MiniSAT and Yices respectively on the SAT and SMT encodings of IA instances at different problem sizes. These graphs give us a closer look at how the SAT and SMT encodings affect performance in the hardest regions (as discussed in the previous sections).

From this we can see that the performance of MiniSAT on a point-based encoding scales better than its performance on an interval-based encoding, i.e. the bigger the problem instance is, the bigger the gap between the point versus interval performance lines plotted in these graphs. Among the various SAT encodings, the performance of MiniSAT on the point-based 1-D support instances scaled steadiest and was nearly linear, having an average runtime of around 8.36 seconds while the average runtime of MiniSAT on the second best encoding (the interval-based 1-D support scheme), was around 21.16 seconds at A$(50, 8, 6.5)$. The performance of MiniSAT on these six different SAT encodings is ordered from best to worst as follows: point-based 1-D support (PS) $<$ interval-based 1-D support (IS) $<$ point-based direct (ID) $<$ interval-based direct (ID) $<$ point-based log (PL) $<$ interval-based log (IL). If we take into account the results

of point-based SMT instances, this order is clearly divided into two exact halves with Yices' performance on SMT instances being worse than MiniSAT's performance on point-based direct instances but better than MiniSAT's performance on interval-based direct instances. In summary, these results clearly demonstrate the benefit of using the 1-D support SAT encoding over the direct and log SAT encodings and the SMT encoding.

It is worth noting that SMT solvers for QF-IDL or QF-RDL usually translate an SMT formula (either eagerly or lazily) into a SAT formula and pass it to a DPLL SAT solver [13]. Consequently, the question arises as to whether the inferior performance of Yices on the SMT-encoded IA instances (in comparison to MiniSAT on the SAT-encoded instances) is because Yices' underlying SAT engine performs significantly worse than MiniSAT. In order to test this, we re-ran Yices on the point-based 1-D support SAT-encoded A(50,d,6.5) instances with Yices simply calling its SAT engine to solve the CNF input formula.[15] Yices' results on the SAT-encoded instances are graphed in Figure 20 against its performance on the corresponding SMT-encoded instances and against MiniSAT on the same SAT-encoded instances. These graphs clearly show that the Yices internal SAT solver has almost identical performance to MiniSAT on the 1-D support SAT-encoded instances and that the large difference in performance between Yices on the SMT-encoded instances and MiniSAT on the SAT-encoded instances is due to the use of SMT-encoding and the subsequent translation of SMT into SAT.

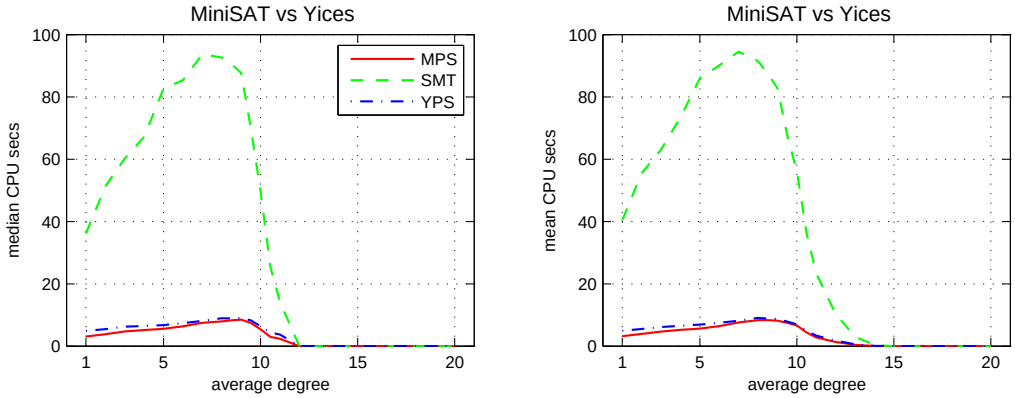


Fig. 20. The performance of MiniSAT and Yices on SAT-encoded or SMT-encoded IA A(50,d,6.5) instances. The 'SMT' key represents Yices on SMT-encoded instances. Other keys represent MiniSAT and Yices on the SAT-encoded instances, where 'M' = MiniSAT, 'Y' = Yices, and 'PS' = point-based 1-D support.

Although the search trees (i.e. the number of variables) for 1-D support and direct SAT-encoded instances are the same size, the graphs in Figure 12 show that the clause-to-variable ratios of the 1-D support SAT-encoded instances are significantly smaller than for the direct SAT-encoded instances. This difference partly explains the superior performance gained from using the 1-D support encoding over the direct encoding on satisfiable IA instances. However, MiniSAT also gained a significant performance boost when solving unsatisfiable 1-D support SAT-encoded instances in comparison with its performance when solving unsatisfiable direct SAT-encoded instances, despite previous studies [8,34,9,19] showing that the smaller the clause-

[15] The developers of Yices confirmed via email discussion that Yices will exclusively employ its SAT engine to solve a pure CNF formula.

to-variable ratio of a CNF formula, the harder it is to prove the unsatisfiability of that formula. This motivated us to find an explanation for these observations.

Theoretically, it has been shown that a DPLL solver with unit propagation on a support SAT-encoded instance maintains full arc consistency during the search [18], while on direct SAT-encoded instances it only maintains a weaker form of arc consistency (equivalent to a forward checking CSP solver) [55]. However, these results are limited to SAT instances translated from *binary* CSPs whereas our two IA-to-CSP formulations produce *ternary* CSPs. According to Bessiere *et al.* [7], unit propagation on our 1-D support and direct SAT-encoded IA instances would enforce relational 2-arc-consistency and relational 3-arc-consistency respectively. Although their experiments confirmed that support encoding is better than direct encoding for ternary CSPs, to the best of our knowledge there is no theoretical comparison between these two encodings on ternary CSPs.

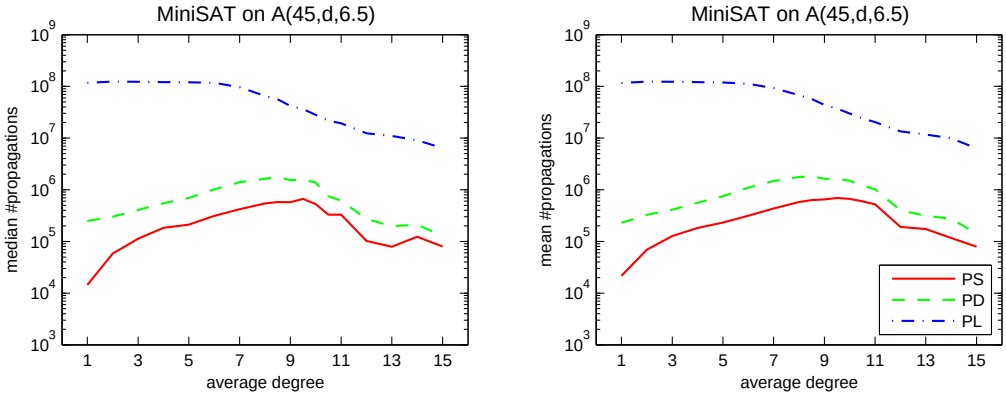


Fig. 21. The median and mean number of unit propagations performed by MiniSAT on point-based SAT-encoded IA A(45,d,6.5) instances. The keys represent MiniSAT on the various encodings, where 'P' = point-based, 'S' = support, 'D' = direct, and 'L' = log.

From an examination of MiniSAT's use of unit propagation and its branching heuristic we found two explanations for its superior performance on the support-encoded instances. Firstly, the direct scheme replaces each support clause $(\neg x^u_{ik} \vee \neg x^v_{kj} \vee x^{w_1}_{ij} \vee \ldots \vee x^{w_m}_{ij})$ by a series of n_c conflict clauses $(\neg x^u_{ik} \vee \neg x^v_{kj} \vee \neg x^{w'_l}_{ij})$ (n_c varies from 1 to 12 conflict clauses), meaning unit propagation on direct SAT-encoded instances will do more work than on support SAT-encoded instances. As unit propagation accounts for more than 90% of the run time of a DPLL solver [35,12], such solvers should benefit significantly from support encoding. This is confirmed by the plots of the median and mean number of unit propagations made by MiniSAT on the support and direct SAT-encoded instances shown in Figure 21.

Secondly, MiniSAT selects the variable with the highest number of occurrences in recent conflict clauses to split the search tree. Consequently, variables x^u_{ik} and x^v_{kj} in a series of conflict clauses $(\neg x^u_{ik} \vee \neg x^v_{kj} \vee \neg x^{w'_l}_{ij})$ have a higher chance of selection at each branching node than other variables in the same series. In contrast, all variables in the corresponding support clauses are treated equally. Furthermore, MiniSAT always branches on the negative value of a variable first. Consequently, MiniSAT on direct SAT-encoded instances becomes less effective and efficient in enforcing path consistency constraints (e.g. constraints 1 or 4) than for support SAT-encoded instances. In other words, MiniSAT gains less information and guidance from direct encoding than

from support encoding. This is illustrated in Figure 22 which shows MiniSAT made more decisions and produced more conflict clauses when solving direct SAT-encoded instances than for support SAT-encoded instances. (Note that the higher the number of conflict clauses generated, the higher the number of times MiniSAT needs to backtrack.)

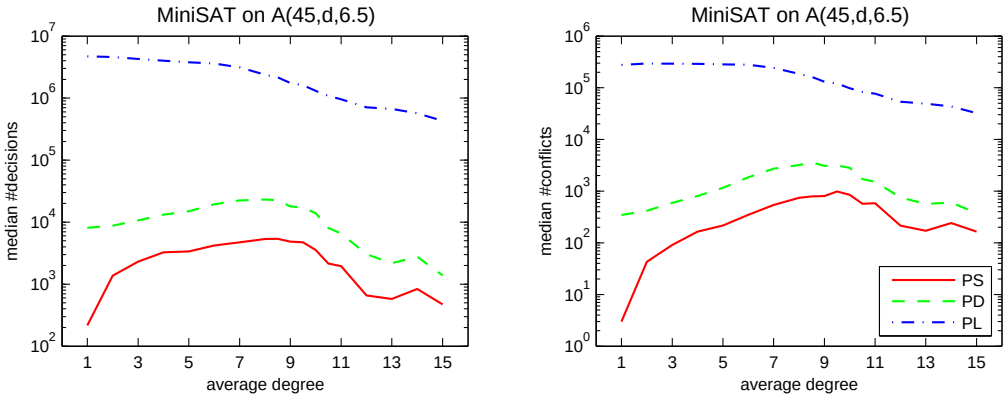


Fig. 22. The median number of decisions and conflict clauses produced by MiniSAT on point-based SAT-encoded IA A(45,d,6.5) instances. The keys represent MiniSAT on the various encodings, where 'P' = point-based, 'S' = support, 'D' = direct, and 'L' = log.

We then altered MiniSAT so that it always branches on the positive value of the branching variable first, meaning it will more strictly enforce path consistency constraints on direct SAT-encoded IA instances. The results in Figure 23 show the superiority of the branch-positive-first strategy, supporting our conjecture that the original MiniSAT with its branch-negative-first policy is not the most suitable for direct SAT-encoded IA instances.

Finally, although the log SAT-encoded encoding scheme produces instances that are $O(n \times (|s| - log|s|))$ times smaller than the other two schemes in terms of the number of variables [24], its bitwise representation of variables also loosens the relationship between variables in their constraints. Consequently, a SAT solver has to spend more time correcting the values of the bitwise variables of the log encoding than it would on the direct encoding. This is confirmed by the large number of decisions MiniSAT had to make when solving log SAT-encoded instances shown in Figure 22.

10. An Empirical Evaluation of SAT/SMT versus Existing Approaches

As the results in previous sections provided encouraging evidence of the practical feasibility of our SAT/SMT approaches, we decided to perform a thorough evaluation of the best performing versions of each approach (MiniSAT on point-based 1-D support SAT instances and Yices on point-based SMT instances) in comparison to the two most well-known existing approaches (NBT$_{\mathcal{H}}$ on native IA instances and TSAT on the endpoint ordering model).

We generated another benchmark test set of $A(n, d, 6.5)$ IA instances by varying the average degree $d \in \{1, 3, 5, 7, 8, 8.5, 9, 9.5, 10, 10.5, 11, 12, 14, 16, 18, 20\}$ across nine values of n varied from 60 to 100 (in steps of 5). We generated 100 instances for each n/d data point to obtain a set of $16 \times 9 \times 100 = 14,400$ test instances. This test set allows us a closer look at the performance of different approaches around the phase transition while still providing a general

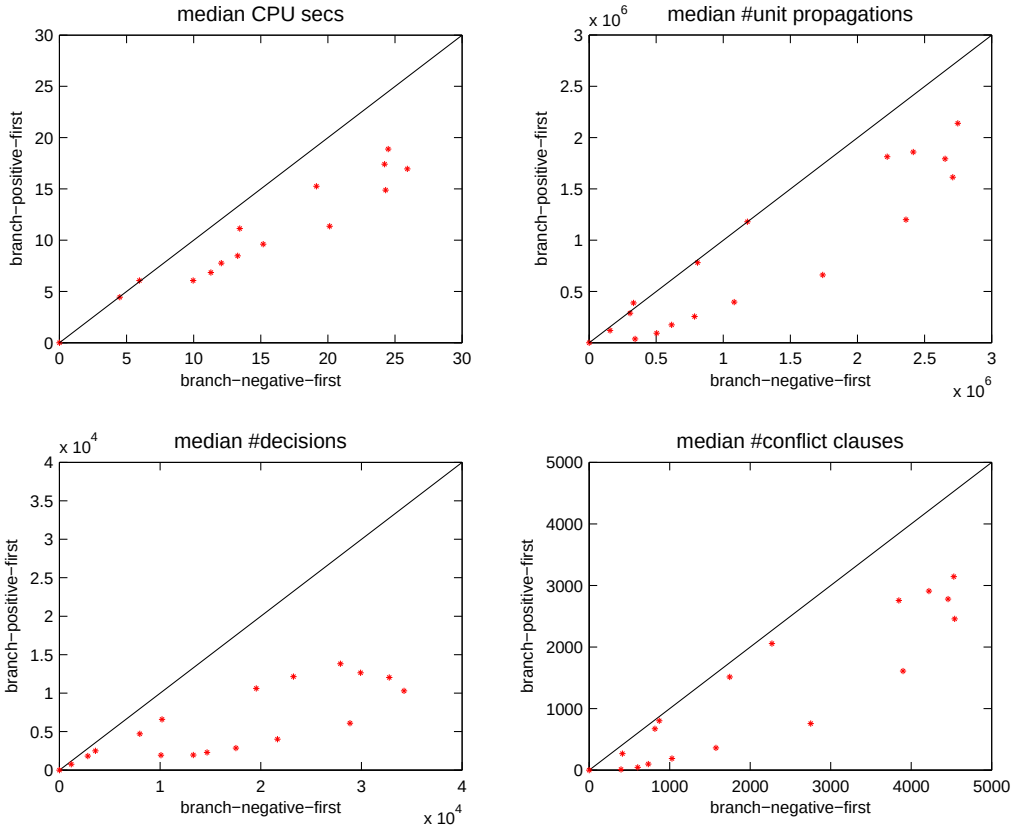


Fig. 23. The head-to-head comparison on the effects of the branch-negative-first and branch-positive-first policies on the performance of MiniSAT on point-based direct SAT-encoded IA A(50,d,6.5) instances (500 instances per each d data point, where $d = 1 - 20$).

view across the entire distribution. We then ran NBT$_{\mathcal{H}}$ on these instances and MiniSAT and Yices on the corresponding point-based 1-D support SAT instances and point-based SMT instances respectively. All solvers were timed out after one hour for $n < 80$ and four hours for $n \geq 80$.

Figure 24 plots the median and mean CPU time of MiniSAT, Yices and NBT$_{\mathcal{H}}$ on these benchmark instances. Here we can see that MiniSAT strongly dominates Yices and NBT$_{\mathcal{H}}$ as the problem size grows. In particular, the CPU time of MiniSAT at $n = 100$ is about 30 times better than for Yices (both in terms of median and mean time) and about 10 times better than for NBT$_{\mathcal{H}}$ (in terms of mean time). In addition, when the test instances became bigger (e.g. $n \geq 80$), the median and mean time curves of Yices and NBT$_{\mathcal{H}}$ were exponentially increasing while the MiniSAT curves remained nearly polynomial. A closer look at the results produced further evidence to support this conclusion: with a one hour time limit, MiniSAT was unable to solve only 1 of the entire benchmark set of $14,400$ instances (this instance took MiniSAT an extra 207.92 seconds to solve), while 323 and $1,774$ instances remained unsolved for NBT$_{\mathcal{H}}$ and Yices respectively. When the time limit was raised to four hours, NBT$_{\mathcal{H}}$ and Yices were still unable to solve 103 and 812 instances respectively. This shows that the performance of MiniSAT scaled significantly better than NBT$_{\mathcal{H}}$ and Yices on these extremely hard instances. The detailed probabilities of

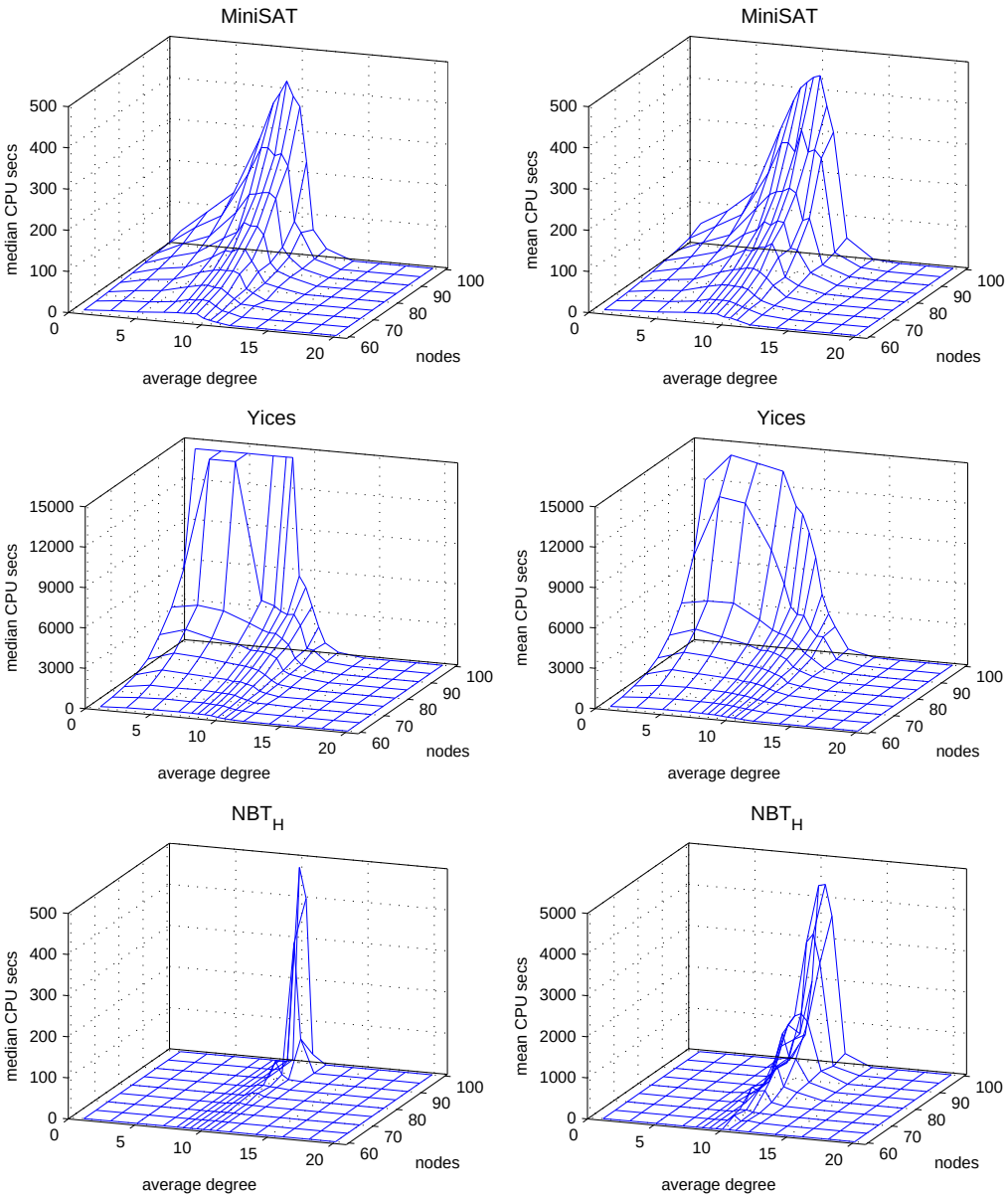


Fig. 24. The CPU time of MiniSAT, Yices and NBT$_{\mathcal{H}}$ on the point-based 1-D support SAT instances, the point-based SMT instances and the native IA instances respectively.

failure for Yices and NBT$_{\mathcal{H}}$ are graphed in Figure 25.

It should also be noted that although the average CPU time of MiniSAT was significantly better than for NBT$_{\mathcal{H}}$ (as graphed in Figure 24), the median CPU time of MiniSAT was only slightly better. This observation led us to conjecture that MiniSAT performs better than NBT$_{\mathcal{H}}$ on hard instances. A more thorough analysis of the results revealed that all the instances that NBT$_{\mathcal{H}}$

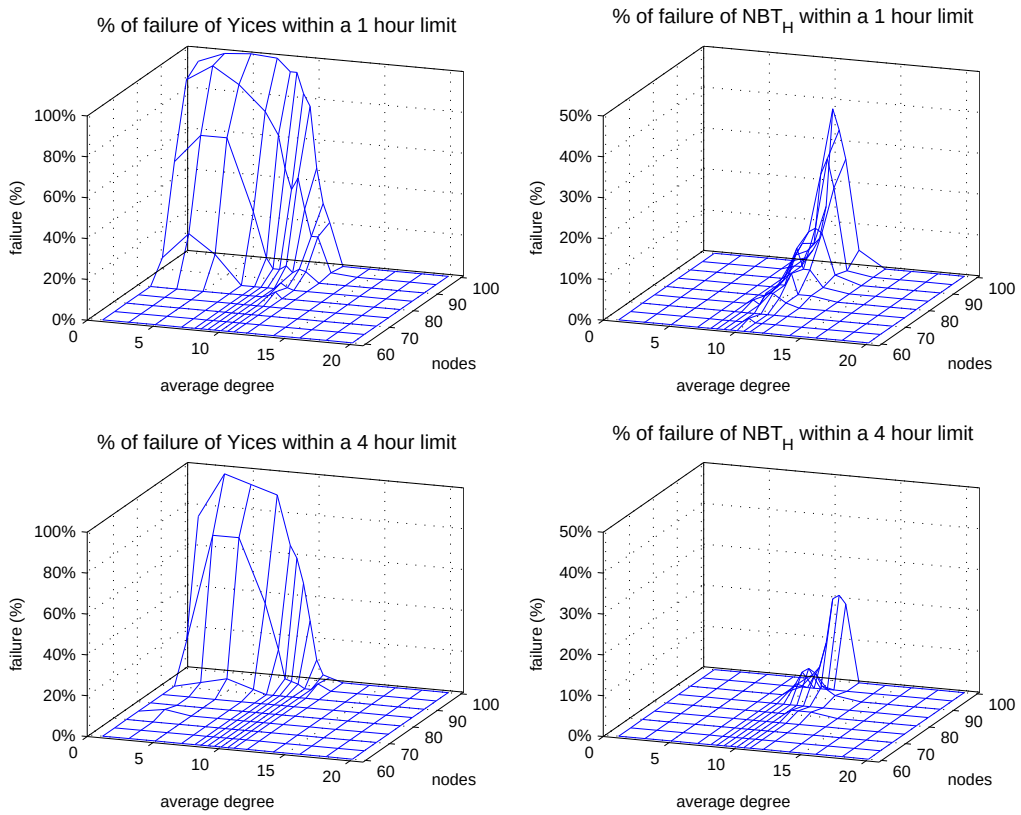


Fig. 25. The probability of failure of Yices and NBT$_{\mathcal{H}}$ on the point-based SMT instances and the native IA instances respectively.

failed to solve (either within a one hour limit or a four hour limit) fall in the area surrounding the phase transition (see Figure 25). This conjecture is further supported when we look at the runtime distribution of MiniSAT and NBT$_{\mathcal{H}}$ on all instances within the range of $d = 8$ to 11 in Figure 26. These graphs also confirm that the performance of MiniSAT scales much better than the other methods as the number of time intervals in the test instances increases.

The performance of TSAT on satisfiable instances was first shown to be better than NBT$_{\mathcal{H}}$ in [48] and this result was initially confirmed in our preliminary study [38]. In a further investigation, we ran TSAT on all $3,985$ hard satisfiable instances (i.e. instances within the range of $d = 8$ to 11) generated in this study in order to obtain a better view of the relative performance of our approaches, NBT$_{\mathcal{H}}$ and TSAT. As TSAT is a local search algorithm, we ran TSAT 100 times for each satisfiable instance with a time-out of one hour for $n < 80$ and four hours for $n \geq 80$.

We then plotted the performance of TSAT against the performance of MiniSAT, Yices and NBT$_{\mathcal{H}}$ extracted from the previous experiment in Figure 27. These results further emphasise the superior performance of our SAT approach against existing approaches, but showed a reversal of performance between TSAT and NBT$_{\mathcal{H}}$, with NBT$_{\mathcal{H}}$ now appearing convincingly better than TSAT. This reversal is perhaps explained by the relatively small number of instances used in our preliminary study.

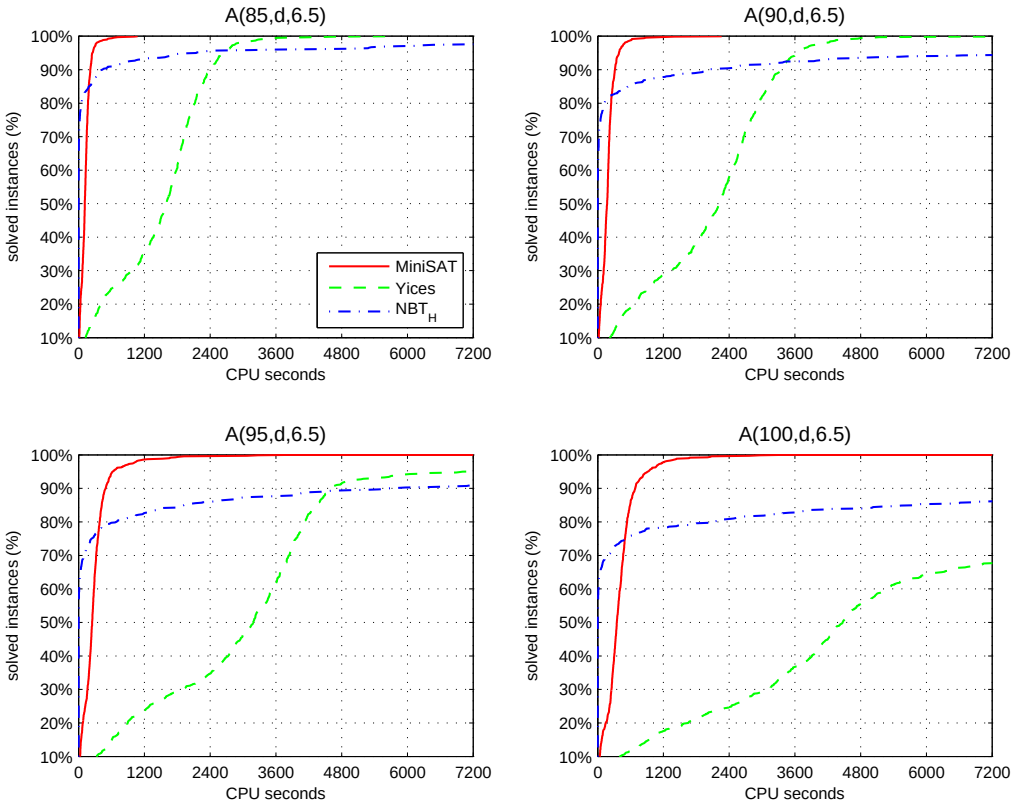


Fig. 26. The runtime distribution of MiniSAT, Yices and NBT$_{\mathcal{H}}$ on $A(n, d, 6.5)$ instances, where $d = 8 - 11$.

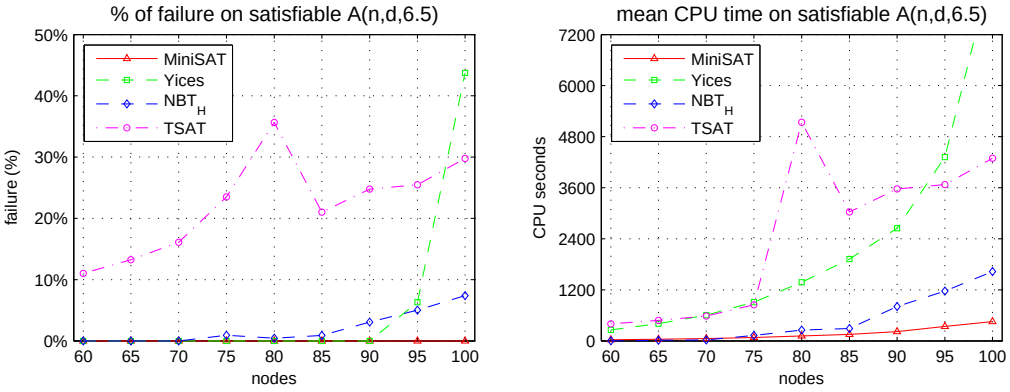


Fig. 27. The results of MiniSAT, Yices, NBT$_{\mathcal{H}}$ and TSAT on satisfiable $A(n, d, 6.5)$ instances, where $d = 8 - 11$.

11. An Empirical Evaluation of SAT versus SMT on Structured SMT Problems

So far, we have shown that our SAT approach significantly outperforms other approaches for solving qualitative temporal problems. As discussed in Section 6.1, we have followed the

common trend of IA-related research [29,30,53,36,31] and limited our benchmarks to randomly generated IA problems. However, although randomly generated problems have useful features (such as controllable size and hardness), they lack the structure that can be found in non-random instances. For this reason, we decided to extend our benchmark set with the more realistic *diamonds* instances. [16] These problems were generated by Ofer Strichman [17] and later translated into SMT-LIB format by Albert Oliveras. Each *diamonds* instance is specified by (n, s), where n is the number of diamonds and s is the number of edges of each diamond divided by 2. A *diamonds* instance is encoded into a SMT formula as a conjunction of disjunct clauses, where each clause contains elements in the following form: $(\ \geq\ (\ -\ a\ b\ )\ 1\ )$ or $(not\ (\ \geq\ (\ -\ a\ b\ )\ 1\ ))$. Given that the variables a and b are integer numbers, the translation of these instances into our qualitative temporal SAT encodings is a straightforward task, as each element is equal to $(\ a > b\ )$ or $(\ a \leq b\ )$, respectively.

The left graph in Figure 28 compares the performance (in seconds) of MiniSAT and Yices on *diamonds* instances where n varies from 10 to 18 and s varies from 2 to 10. MiniSAT was run on the point-based 1-D support encodings of these instances whilst Yices was run on the original SMT encodings. The results shows that our SAT approach dominates on all the $(n, 2)$ instances and the three biggest $(n, 3)$ instances but is outperformed by Yices on all other instances (which make up the majority of the test set). In addition, the SAT encodings of the *diamonds* $(n, 10)$ instances become too big for MiniSAT to handle when $n > 12$.

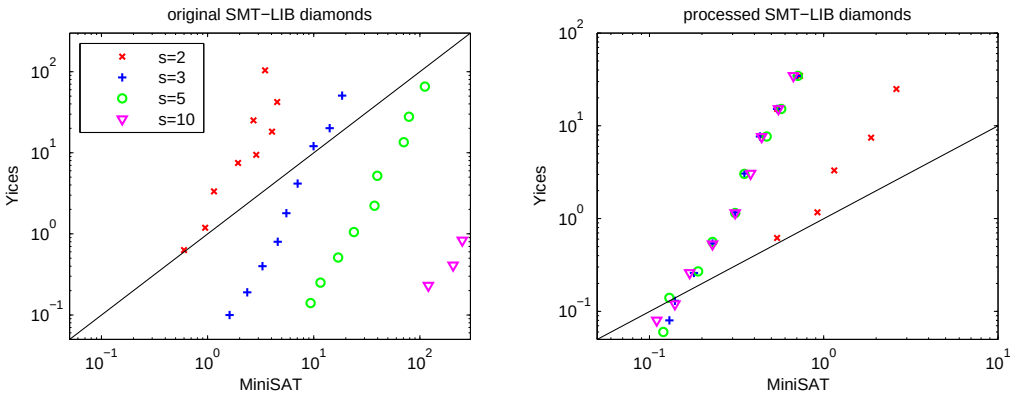


Fig. 28. A comparison of the performance of MiniSAT and Yices on the original and processed SMT-LIB diamonds problems.

However, a closer inspection of the *diamonds* problem set provided us with a solution to this size blow-up problem: we found that most instances contain several long chains of inequalities of the form: $(\ a \leq c_1 \leq c_2 \leq \ldots \leq c_k \leq b\ )$ where $c_1, c_2 \ldots c_k$ do not appear in any other inequalities. Our conjecture is that the original intention of these c_i variables was to enforce a metric distance constraint between a and b. However, such constraints appear to be very 'weak' as the solver can freely set the distance between a and b to any number ≥ 0. In addition, the constraint becomes redundant from the qualitative point of view as once $a < b$ we can allow an unlimited number of c_i between a and b. It is the inclusion of c_i variables in the *diamonds*

[16] The majority of QF-IDL benchmarks in the SMT-LIB repository (http://www.sat-lib.org/) require reasoning with metric information and hence are unsuitable for our study.
[17] http://iew3.technion.ac.il/Home/Users/ofers.phtml

instances that makes the size of the corresponding SAT encoding grow exponentially (due to the requirement of extra clauses to maintain consistency). Once these redundant c_i variables are removed, the size of SAT-encoded *diamonds* instances is greatly reduced, leading to significant boost in the performance of MiniSAT. This is shown in the right graph of Figure 28, where MiniSAT now clearly dominates Yices on the majority of instances where $n > 11$.

12. Conclusions and Future Work

In conclusion, we have proposed six different methods to formulate IA networks into SAT formulae and one point-based method to formulate IA networks into SMT instances. We also provided the theoretical proofs of completeness of these transformation techniques. Although our empirical results confirmed that the phase transition of IA networks mapped directly into these SAT and SMT encodings, they also showed that the hard regions of these problems were surprisingly shifted away from the phase transition areas after transformation into SAT or SMT. Evaluating the effects of these encodings, we found that the point-based 1-D support SAT scheme is the best among the seven different encoding schemes examined. Our results also revealed that MiniSAT combined with our point-based 1-D support SAT scheme could solve IA instances significantly faster than existing IA solvers working on the equivalent native IA networks.

In future work we anticipate that the performance of our SAT-based approach can be further improved by exploiting the special structure of IA problems in a manner analogous to the work on TSAT [48]. In addition, we plan to extend our SAT-based approach to cover INDU networks [43], which enable qualitative reasoning with the duration between pairs of intervals. The possibility also opens up of integrating our approach to temporal reasoning into other well known real world problems such as planning. Given the success of SAT solvers in many real world domains, our work promises to expand the reach of temporal reasoning approaches for IA to encompass larger and more practical problems.

Acknowledgments

We thankfully acknowledge the financial support from NICTA and the Queensland government. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[2] Anbulagan, Duc Nghia Pham, John Slaney, and Abdul Sattar. Old resolution meets modern SLS. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 354–359, 2005.

[3] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. SAT-based procedures for temporal reasoning. In *Proceedings of the Fifth European Conference on Planning*, pages 97–108, 2000.

[4] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, Massimo Idini, and Marco Maratea. TSAT++: an open platform for satisfiability modulo theories. In *Proceedings of the Second Workshop on Pragamatics of Decision Procedures in Automated Reasoning (PDPAR-04)*, 2004.

[5] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, and Marco Maratea. A SAT-based decision procedure for the boolean combination of difference constraints. In *Proceedings of the Seventh International Conference on*

Theory and Applications of Satisfiability Testing SAT 2004, Selected Revised Papers, volume 3542 of *Lecture Notes in Computer Science*, pages 16–29, 2004.

[6] Ramoón Béjar and Felip Manyà. Solving the round robin problem using propositional logic. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pages 262–266, 2000.

[7] Christian Bessière, Emmanuel Hebrard, and Toby Walsh. Local consistencies in SAT. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing SAT 2003, Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 299–314, 2003.

[8] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the Twelveth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

[9] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1–2):31–57, 1996.

[10] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, 2003.

[11] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):61–95, 1991.

[12] Heidi E. Dixon, Matthew L. Ginsberg, and Andrew J. Parkes. Genralizing Boolean satisfiability I: Background and survey of existing work. *Journal of Artificial Intelligence Research*, 21:193–243, 2004.

[13] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proceedings of the Eighteenth International Conference on Computer Aided Verification*, pages 81–94, 2006.

[14] Niklas Eén and Niklas Sörensson. An extensible SAT solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT-03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, 2003.

[15] Eugene C. Freuder. Synthesizing constraint expressions. *Communication of ACM*, 21(11):958–966, 1978.

[16] Eugene C. Freuder. A sufficient condition for backtrack-free search. *Journal of ACM*, 29(1):24–32, 1982.

[17] Alan M. Frisch and Timothy J. Peugniez. Solving non-Boolean satisfiability problems with stochastic local search. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 282–290, 2001.

[18] Ian P. Gent. Arc consistency in SAT. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI-02)*, pages 121–125, 2002.

[19] Ian P. Gent and Toby Walsh. The SAT phase transition. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 105–109, 1994.

[20] K. Ghiathi and G. Ghassem-Sani. Using satisfiability in temporal planning. *WSEAS Transactions on Computers*, 3(4):963–969, 2004.

[21] Martin C. Golumbic and Ron Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *Journal of ACM*, pages 1108–1133, 1993.

[22] Robert M. Haralick and Gordon L. Elliott:. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[23] J. N. Hooker. Needed: an empirical science of algorithms. *Operations Research*, 42(2):201–212, 1994.

[24] Holger H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 296–302, 1999.

[25] Frank Hutter, Dave A.D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-02)*, pages 233–248, 2002.

[26] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, pages 374–384, 1996.

[27] Henry Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla Gomes, Bart Selman, and Mark Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 351–358, 2001.

[28] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, 1996.

[29] Peter Ladkin and Alexander Reinefeld. Effective solution of qualitative interval constraint problems. *Artificial Intelligence*, 57(1):105–124, 1992.

[30] Peter Ladkin and Alexander Reinefeld. A symbolic approach to interval constraint problems. In *Artificial Intelligence and Symbolic Mathematical Computation*, pages 65–84, 1993.

[31] Peter Ladkin and Alexander Reinefeld. Fast algebraic methods for interval constraint problems. *Annals of Mathematics and Artificial Intelligence*, 19:383–411, 1997.

[32] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP-97)*, pages 341–355, 1997.
[33] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
[34] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, 1992.
[35] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC-01)*, pages 530–535, 2001.
[36] Bernhard Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. *Constraints*, 1(3):175–190, 1997.
[37] Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's Interval Algebra. *Journal of ACM*, 42(1):43–66, 1995.
[38] Duc Nghia Pham, John Thornton, and Abdul Sattar. Modelling and solving temporal reasoning as satisfiability. In *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 117–131, 2005.
[39] Duc Nghia Pham, John Thornton, and Abdul Sattar. Towards an efficient SAT encoding for temporal reasoning. In *Proceedings of the Twelfth International Conference on the Principles and Practice of Constraint Programming (CP-06)*, pages 421–436, 2006.
[40] Duc Nghia Pham, John Thornton, Abdul Sattar, and Adelraouf Ishtaiwi. SAT-based versus CSP-based constraint weighting for satisfiability. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 455–460, 2005.
[41] Steven Prestwich. Local search on SAT-encoded colouring problems. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT-03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 105–119, 2003.
[42] Patrick Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):81–109, 1996.
[43] Arun K. Pujari, G. Vijaya Kumari, and Abdul Sattar. INDU: an interval and duration network. In *Proceedings of the Twelfth Australian Joint Conference on Artificial Intelligence*, pages 291–303, 1999.
[44] Silvio Ranise and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2006.
[45] Jochen Renz and Bernhard Nebel. Efficient methods for qualitative spatial reasoning. *Journal of Artificial Intelligence Research*, 15:289–318, 2001.
[46] Bart Selman, Henry Kautz, and David McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 50–54, 1997.
[47] Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.
[48] John Thornton, Matthew Beaumont, Abdul Sattar, and Michael Maher. A local search approach to modelling and solving Interval Algebra problems. *Journal of Logic and Computation*, 14(1):93–112, 2004.
[49] John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-04)*, pages 191–196, 2004.
[50] Ioannis Tsamardinos and Martha Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151(1):43–89, 2003.
[51] Peter van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.
[52] Peter van Beek and Robin Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.
[53] Peter van Beek and Dennis W. Manchak. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, 4:1–18, 1996.
[54] Marc Vilain and Henry Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 377–382, 1986.
[55] Toby Walsh. SAT v CSP. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP-00)*, pages 441–456, 2000.