ELSEVIER

Artificial Intelligence 120 (2000) 81–117

Artificial Intelligence

www.elsevier.com/locate/artint

Backtracking algorithms for disjunctions of temporal constraints ☆

Kostas Stergiou [a,*], Manolis Koubarakis [b,1]

[a] *APES Research Group, Department of Computer Science, University of Strathclyde, Glasgow, G1 1HX, Scotland, UK*

[b] *Department of Electronic and Computer Engineering, Technical University of Crete, University Campus—Kounipidiana, 73100 Chania, Crete, Greece*

Received 22 February 1999; received in revised form 1 February 2000

Abstract

We extend the framework of simple temporal problems studied originally by Dechter, Meiri and Pearl to consider constraints of the form $x_1 - y_1 \leqslant r_1 \ \vee \cdots \vee \ x_n - y_n \leqslant r_n$, where $x_1, \ldots, x_n, y_1, \ldots, y_n$ are variables ranging over the real numbers, $r_1, \ldots, r_n$ are real constants, and $n \geqslant 1$. This is a wide class of temporal constraints that can be used to model a variety of problems in temporal reasoning, scheduling, planning, and temporal constraint databases. We have implemented several progressively more efficient algorithms for the consistency checking problem for this class of temporal constraints: backtracking, backjumping, three variations of forward checking, and forward checking with backjumping. We have partially ordered the above algorithms according to the number of visited search nodes and the number of performed consistency checks. Although our problem is non-binary, our results agree with the results of Kondrak and van Beek who consider only binary constraints. We have also studied the performance of the above algorithms experimentally using randomly generated sets of data and job shop scheduling problems. The experiments with random instances allowed us to locate the hard region for this class of problems. The results show that hard problems occur at a critical value of the ratio of disjunctions to variables.

Keywords: Temporal reasoning; Constraint satisfaction problems; Search; Scheduling; Spatio-temporal databases

☆ This work was supported in part by the TMR project Chorochronos funded by ESPRIT IV. Most of the results presented in this paper were obtained while the authors where with the Department of Computation, UMIST, Manchester, UK. A preliminary version of this paper was presented at AAAI-98.

* Corresponding author. Email: ks@cs.strath.ac.uk.

[1] Email: manolis@ced.tuc.gr.

0004-3702/00/$ – see front matter
PII: S0004-3702(00)00019-9

1. Introduction

Reasoning with temporal constraints has been a hot research topic for the last fifteen years. The importance of this problem has been demonstrated in many areas of artificial intelligence and databases, e.g., planning [2], scheduling [11], spatio-temporal databases [13,44], geographical information systems [24] and medical information systems [69].

The work carried out in this field can be categorized in terms of the classes of temporal constraints studied. The class of qualitative temporal constraints in the *Interval Algebra* and *Point Algebra* has been studied primarily in [1,49,50,70–72]. Nebel and Bürckert [56] have introduced the ORD-Horn subclass of Interval Algebra and showed that it is a maximal tractable subclass. Drakengren and Jonsson [21,22] have also studied other tractable subclasses of the Interval Algebra. Finally, Gerevini and Schubert [31–33] have studied several algorithms for qualitative temporal constraints and have implemented them efficiently in their TimeGraph system.

The class of quantitative temporal constraints has been studied originally by Dechter et al. [18] in the framework of *simple temporal problems* (STPs) where constraints are of the form $x - y \leqslant c$ where x and y are real variables and c is a real constant, and *temporal constraint satisfaction problems* (TCSPs) where constraints are disjunctions of formulas $l \leqslant x - y \leqslant u$ involving the *same pair* of real variables x and y (l and u are real constants). Subsequently several researchers [12,30,40,46,52] continued the study of the class of simple temporal problems, and some tractable extensions of it. Koubarakis [45] and Jonsson and Bäckström [38] have studied *Horn linear constraints*, a significant tractable extension of simple temporal problems that allows disjunctions involving an arbitrary number of linear disequations (e.g., $3x + 5y + 7z \neq 8$) but *at most one* linear inequality. Naturally, the applications of this latter class go beyond temporal reasoning. Schwalb and Dechter [62] studied the performance of local consistency algorithms for processing TCSPs on hard problems in the transition region. Finally, Staab [65] proposed a very expressive temporal reasoning framework which can handle sets of disjunctions of conjunctions of binary constraints between time points.

This paper continues the work on quantitative temporal constraints and makes the following contributions:

(1) We extend the framework of simple temporal problems studied originally by Dechter et al. [18] to consider constraints of the form $x_1 - y_1 \leqslant r_1 \ \vee \cdots \vee \ x_n - y_n \leqslant r_n$, where $x_1, \ldots, x_n, y_1, \ldots, y_n$ are variables ranging over the real numbers, $r_1, \ldots, r_n$ are real constants, and $n \geqslant 1$.
The reader should note that we do not restrict the variables in the disjuncts to be the same pair as Dechter et al. [18] do in the framework of temporal constraint satisfaction problems. The added generality is useful in many problems including temporal planning [73], scheduling [11] and indefinite temporal constraint databases [43,44,47]. We demonstrate this with the examples and the experiments of Section 7.

(2) We have implemented several progressively more efficient algorithms for the consistency checking problem for this class of temporal constraints (backtracking, backjumping, three variations of forward checking and forward checking with backjumping). We have also implemented the minimum remaining values heuristic in conjunction with the forward checking algorithms.

(3) Following the methodology of Kondrak and van Beek [42], we have proved the correctness of all of the above algorithms, and partially ordered them according to the number of visited search nodes and the number of performed consistency checks. Although our problem is *not* binary, our results agree with the results of [42] for binary constraint satisfaction problems.
(4) We have studied the performance of the above algorithms experimentally using randomly generated sets of data and small job shop scheduling problems. Our results show that the best of our algorithms is one of the versions of forward checking with backjumping (an algorithm we call FC1-BJ). The same version of forward checking without backjumping is only slightly less efficient for the class of problems we considered in our experimental evaluation.
(5) We present a series of experimental results on the location of the region where hard problems occur. The results show that hard problems occur at a critical value of the ratio of disjunctions to variables. We also investigate the transition from the region where almost all problems are consistent to the region where almost no problem is consistent. Our empirical results show that the hard region does not occur around the 50% satisfiability point as is the case with other NP-complete problems like SAT problems [14,29,63] and binary CSPs [60,64]. It occurs at a point where almost all problems are unsatisfiable.

The organization of this paper is as follows. In Section 2 we present some basic definitions and describe the problem in detail. Section 3 discusses consistency checking for non-disjunctive constraints. In Section 4 we introduce necessary terminology and describe how the problem is solved using our most efficient algorithm, forward checking with backjumping. In Section 5 we evaluate the performance of the algorithms theoretically. Section 6 presents the results of our empirical analysis. In Section 7 we discuss possible applications of our framework and give some results on job shop scheduling problems. Finally, in Section 8 we conclude and discuss future work.

2. Preliminaries

We consider time to be linear, dense and unbounded. *Points* will be our only time entities. Points are identified with the real numbers. The set of real numbers will be denoted by $\mathbb{R}$.

Definition 1. A *temporal constraint* is a disjunction of the form $x_1 - y_1 \leqslant r_1 \ \vee \cdots \vee x_n - y_n \leqslant r_n$, where $x_1, \ldots, x_n, y_1, \ldots, y_n$ are variables ranging over the real numbers, $r_1, \ldots, r_n$ are real constants, and $n \geqslant 1$. The special variable $x_0 = 0$ is also allowed so that disjuncts of the form $x \leqslant r$ and $-x \leqslant r$ are included in the definition. A temporal constraint with only one disjunct will be called *non-disjunctive*. Temporal constraints with more than one disjuncts will be called *disjunctive*.

Example 2. The following are examples of temporal constraints:

$$x_1 - y_1 \leqslant 2, \quad x_1 - y_1 \leqslant 5 \ \vee x_2 - y_2 \leqslant -2 \vee x_3 - y_3 \leqslant 4.$$

Although our formalism does not allow time intervals, these can be modelled as pairs of points as the following example demonstrates.

Example 3. Let I, J be intervals, I^-, J^- their beginning points and I^+, J^+ their ending points. The following constraints express the fact that intervals I and J have duration between 5 and 10 minutes and they cannot overlap.

$$I^+ - I^- \leqslant 10, \quad I^- - I^+ \leqslant -5, \quad J^+ - J^- \leqslant 10, \quad J^- - J^+ \leqslant -5,$$
$$I^+ - J^- \leqslant 0 \vee J^+ - I^- \leqslant 0.$$

2.1. Relationship to other temporal reasoning formalisms

The constraint class that we will study is more expressive than the classes of STP constraints and TCSP constraints of [18]. The last constraint of Example 3 cannot be captured by the TCSP model but can be modelled naturally in our framework. Here is another example.

Example 4. Let I and J be intervals corresponding to operations O_I and O_J. O_I and O_J will be executed on a machine that can handle only one operation at a time and has a set up time of 2 minutes. Let I^-, J^- be the beginning points of I and J and I^+, J^+ their ending points.

The following is a constraint on the scheduling of operations O_I and O_J:

$$I^+ - J^- \leqslant -2 \vee J^+ - I^- \leqslant -2.$$

Such a disjunctive constraint with disjuncts having different pairs of variables cannot be expressed in the TCSP framework.

However, it is probably unfair to simply say that our framework is more expressive than TCSP because TCSP constraints are modelled in our framework rather awkwardly as the following example demonstrates.

Example 5. Consider the TCSP constraint $x_i - x_j \in \{[a, b], [c, d]\}$ or equivalently:

$$a \leqslant x_i - x_j \leqslant b \vee c \leqslant x_i - x_j \leqslant d.$$

This constraint can only be expressed in our framework using a set of four disjunctions:

$$x_i - x_j \leqslant b \vee x_i - x_j \leqslant d,$$
$$x_i - x_j \leqslant b \vee x_j - x_i \leqslant -c,$$
$$x_j - x_i \leqslant -a \vee x_i - x_j \leqslant d,$$
$$x_j - x_i \leqslant -a \ \vee \ x_j - x_i \leqslant -c.$$

Our framework can be easily extended in order to handle TCSP constraints effectively. We only need to extend Definition 1 to allow disjunctions of the following form:

$$a_1 \leqslant x_1 - y_1 \leqslant b_1 \vee \cdots \vee \ a_n \leqslant x_n - y_n \leqslant b_n.$$

In this case one might also want to consider allowing disjunctions of the form $\phi_1 \vee \cdots \vee \phi_n$ where $\phi_i, \ i = 1, \ldots, n$, is a conjunction of constraints $x_i - x_j \leqslant c$. This will allow one to reach the expressiveness of the formalism proposed in [65].

Both of the alternatives sketched above can easily be accommodated by the work presented in this paper. The algorithms discussed in forthcoming sections need only simple modifications in order to work with the new classes of constraints. [2]

2.2. Consistency checking

In the rest of this paper we will study algorithms for deciding whether a set of temporal constraints is consistent or not. Let us first define consistency.

Definition 6. Let C be a set of temporal constraints in variables $x_1, \ldots, x_n$. The *solution set* of C, denoted by $Sol(C)$, is

$$\{(\tau_1, \ldots, \tau_n)\colon\ (\tau_1, \ldots, \tau_n) \in \mathbb{R}^n \text{ and for every } c \in C,\ (\tau_1, \ldots, \tau_n) \text{ satisfies } c\}.$$

Each member of $Sol(C)$ is called a *solution* of C. A set of temporal constraints is called *consistent* if and only if its solution set is non-empty.

The consistency checking problem for a set of m temporal constraints in n variables can be equivalently restated as an m-ary *meta*-CSP, where disjunctions can be viewed as variables, and the disjuncts of each disjunction as the possible values of the corresponding variable. The m-ary constraint between the variables is that all disjuncts that are part of an assignment to variables must be simultaneously satisfied. We now give a formal definition of the problem as an m-ary meta-CSP.

Definition 7. A *consistency checking* problem for a set of temporal constraints consists of the following:

- A set of m variables $\{D_1, \ldots, D_m\}$ representing disjunctions.
- A set of domains $\{dom_1, \ldots, dom_m\}$ for variables $D_1, \ldots, D_m$. The elements of the domains are inequalities of the form $x_i - x_j \leqslant r$, where x_i, x_j are variables ranging over the real numbers, and r is a real constant.
- An m-ary constraint which states the following. For all k, where $1 \leqslant k \leqslant m$, if

  D_1 is assigned value d_1

  $\vdots$

  D_k is assigned value d_k

  then inequalities $d_1, \ldots, d_k$ are consistent.

Since the problem is expressed as a constraint satisfaction problem, we can solve it by search. We check the consistency of a given set C of temporal constraints in two steps. First, we group together all the non-disjunctive constraints and check their consistency [19]. If the set of non-disjunctive constraints is inconsistent then the original set of constraints is also inconsistent and we stop. Otherwise, we consider the subset of disjunctive constraints as well. Now C is consistent if for each disjunction there exists one

[2] This point has already been made in [65].

disjunct that can be added to the subset of non-disjunctive constraints so that the new set of constraints produced is still consistent.

The second step of consistency checking is performed using a backtracking based search algorithm. The basic backtracking algorithm is *simple* or *chronological backtracking* (usually denoted by BT) [36]. In this problem, BT successively selects a disjunct from each disjunction and adds it to the set of non-disjunctive constraints, as long as this set remains consistent. In case none of the constraints of the current disjunction can be consistently added to the set of non-disjunctive constraints, the algorithm backtracks to the previous disjunction, removes the selected constraint from the set of non-disjunctive constraints, and chooses another one. The algorithm terminates successfully if all the disjunctions have been considered. If there is no disjunction left to backtrack to, the algorithm fails. There are many ways to make BT more efficient and we will discuss some of them in the rest of this paper.

We will now give more details of the two steps briefly sketched in the above paragraphs.

3. Consistency checking for non-disjunctive constraints

Several algorithms that check the consistency of a set of non-disjunctive temporal constraints have been proposed in the literature. All these algorithms make use of a graph representation of the constraints.

Definition 8. A *constraint graph* is a directed graph whose nodes represent variables and whose arcs represent binary constraints between these variables. If C is set of non-disjunctive temporal constraints in n variables $x_1, \ldots, x_n$ then the constraint graph of C is $G = (V, E)$ where $V = \{x_1, \ldots, x_n\}$ and $E = \{x_j \xrightarrow{r} x_i \colon x_i - x_j \leqslant r \text{ is a member of } C\}$. We will use the notation G_{ij} to refer to the label (weight) of edge $i \to j$ in a constraint graph G.

Dechter, Meiri and Pearl in [19] introduced the algorithm DPC (i.e., *d*irectional *p*ath *c*onsistency) for checking the consistency of a set of non-disjunctive temporal constraints. Chleq presented an incremental version of DPC, called IDPC [12]. DPC and IDPC work by propagating constraints in the given constraint graph so that the resulting graph is directional path consistent.

Definition 9 (*Dechter et al.* [19]). Let C be a set of non-disjunctive temporal constraints and G its constraint graph completed in the following way. For each pair of nodes (x_i, x_j), if there is no edge $x_i \to x_j$ then a new edge is added labelled with ∞ (this edge represents the tautologically true constraint $x_j - x_i \leqslant \infty$). Let $\prec$ be an ordering of the variables of C (or nodes of G), and let $x_i \prec x_j$ iff $i < j$. The constraint graph G is *directional path consistent* with respect to $\prec$, if for every triple i, j and k such that $i, j \prec k$ then $G_{ij} \leqslant G_{ik} + G_{kj}$.

Dechter, Meiri and Pearl prove that DPC always detects an inconsistency in a constraint graph, if one exists [19]. In fact Dechter et al. [19] and Chleq [12] present their results

for *constraint networks* but this is not a problem since constraint graphs are an equivalent representation. In our case, consistency checking is performed using an algorithm, called GRAPH_IDPC, which is the equivalent of Chleq's IDPC algorithm for distance graphs. In GRAPH_IDPC, the nodes are ordered according to a predetermined ordering $\prec$. We assume that the ordering $\prec$ is the ordering in which the variables are numbered.

The input of GRAPH_IDPC is a directional path consistent graph G with n nodes, and a constraint $x_m - x_l \leqslant r$, with $x_l \prec x_m$, to be added to G. GRAPH_IDPC will add the new constraint to G and enforce directional path consistency again, exploiting the fact that G is already directional path consistent. GRAPH_IDPC will start by examining node x_m and move towards node x_1. All the nodes that are after x_m in the ordering $\prec$ are ignored, since the constraints they are involved in cannot be possibly affected by the new constraint.

The worst-case complexity of GRAPH_IDPC is $O(nW^2)$ where n is the number of nodes, and W is the width of the ordering $\prec$ in the graph that results after GRAPH_IDPC terminates [19]. The concept of width of an ordering is defined below.

Definition 10 (*Freuder* [26], *Dechter and Pearl* [20]). Let G be a constraint graph and $\prec$ an ordering of its nodes. The *parent set* of a node x relative to $\prec$ is the set of nodes y such that there is an edge between x and y (direction does not matter) and y comes before x in the ordering $\prec$. The *width of a node* relative to ordering $\prec$ is the cardinality of its parent set. The *width of an ordering* $\prec$ is the maximum width of nodes along this ordering.

Instead of using GRAPH_IDPC to add constraints to an already directional path consistent graph, we could use DPC, since the worst-case complexity of GRAPH_IDPC is the same as DPC. However, the time spent by GRAPH_IDPC to establish directional path consistency incrementally is significantly less than the time spent by DPC, as shown experimentally in [12].

4. Search algorithms

Having explained how consistency checking for non-disjunctive constraints is performed, we will now discuss consistency checking for disjunctive constraints.

4.1. Terminology

First, we introduce some necessary terminology. Our terminology is similar to the terminology of [31,32] where algorithms for disjunctive qualitative temporal constraints are proposed. The given set of disjunctions will be denoted by D. $D(i)$ and $D(i, j)$ will represent the ith disjunction and the jth disjunct of the ith disjunction, respectively. The set of non-disjunctive constraints will be represented by a labelled directed graph and will be denoted by G. We call the set of selected disjuncts an *instantiation* of D in G. When a constraint is consistently added to G, we say that it has been *instantiated*, and when a constraint is retracted from G that it has been *uninstantiated*. Trying to instantiate a disjunct means adding it to G and checking that the produced set of constraints is consistent.

The *order of instantiation* is the ordering in which the disjunctions are examined. We will assume that this order is *static* (i.e., it is predetermined and does not change during the execution of the algorithm). In Section 4.5 we will discuss *dynamic* ordering of the disjunctions. The *current disjunction* is the disjunction chosen for instantiation at each step of the algorithm. The *past disjunctions* are the disjunctions that have been already instantiated. If $D(i)$ is the current disjunction then all the disjunctions that precede $D(i)$ in the order of instantiation are the past disjunctions. We say that these disjunctions are *before* $D(i)$ and $D(i)$ is *after* them. The *future disjunctions* are the disjunctions that have not yet been instantiated. They are all the disjunctions that are after the current disjunction in the order of instantiation. Each disjunct can be in one of three possible states at any time. It can be *available*, *current* or *eliminated*. We say that a disjunct is available if it is neither current nor eliminated. Initially all the disjuncts of all the disjunctions are available. A disjunct is current if it is part of the currently attempted instantiation of D. All the instantiated disjuncts of the past disjunctions are current disjuncts. A disjunct becomes eliminated when it is tried against the graph and fails to be consistently instantiated. All the disjuncts of the past disjunctions that have failed consistency checking are eliminated disjuncts. A *dead-end* is a situation when all the disjuncts of the current disjunction are rejected.

4.2. Preprocessing

A general strategy in CSPs is to preprocess the set of constraints prior to search. The aim of preprocessing is to transform the given CSP into an equivalent CSP that is easier for a backtracking algorithm to solve. To be precise, preprocessing tries to reduce the search space that backtracking algorithms explore, and in that way improve their efficiency [17].

The set of disjunctions D can be reduced to an equivalent smaller subset by exploiting the information provided by the constraint graph G. Three simple *pruning rules* originally used in [32] in the context of qualitative temporal constraints are applied to each disjunction in D prior to the execution of the search algorithms. In this way the search space can sometimes be significantly reduced. The process of applying the pruning rules to D is called *preprocessing*. The pruning rules are the following:

(1) If a disjunction $D(i)$ contains a disjunct that is subsumed by a single constraint in the constraint graph G then disjunction $D(i)$ can be eliminated from D.
(2) If a disjunction $D(i)$ is subsumed by another disjunction $D(j)$ then $D(i)$ can be eliminated from D.
(3) If a disjunct $D(i, j)$ is inconsistent relative to G then it can be eliminated from disjunction $D(i)$.

The first two rules, in a way, correspond to checking for *constraint entailment* in constraint programming languages. In case a disjunct $D(i, j)$ is subsumed by a single constraint in G, $D(i)$ can always be consistently instantiated simply by selecting $D(i, j)$. Therefore, D can be reduced to a subset $D' = D \setminus D(i)$. A disjunction $D(i)$ is subsumed by another disjunction $D(j)$ if every disjunct of $D(j)$ subsumes a disjunct of $D(i)$. Pruning rule (3) checks the consistency of the disjuncts using GRAPH_IDPC. If a disjunct $D(i, j)$ is found to be inconsistent, it can be discarded, resulting in a shorter disjunction $D(i)$. In

terms of constraint programming, this rule can be thought of as enforcing arc consistency for every disjunction with two or more disjuncts with respect to the disjunctions with one disjunct which form the initial constraint graph.

The worst-case complexity of preprocessing is $O(|N||D|^2|d|^2nW^2)$ where $|N|$ is the number of non-disjunctive constraints, $|D|$ is the number of disjunctions, $|d|$ is the maximum number of disjuncts in a disjunction, n is the number of variables, and W is the width of graph G. After the preprocessing rules have been applied, the disjunctions are ordered in ascending order of domain sizes. This can reduce the number of backtracks required significantly [67].

4.3. Forward checking and forward checking with backjumping

We have implemented several progressively more efficient algorithms for the consistency checking problem for this class of temporal constraints: backtracking (BT), backjumping (BJ), three variations of forward checking (which we call FC, FC1 and FC2) and forward checking with backjumping (FC-BJ). A detailed presentation of some of these algorithms can be found in [66]. For reasons of brevity we only present the three versions of forward checking and FC-BJ in this paper.

In the context of our problem, FC will attempt to instantiate a disjunct from each disjunction, starting with the first. First, the current disjunct $D(i, j)$ will be added to G and propagated using the GRAPH_IDPC algorithm discussed in Section 3. Then, all disjuncts of the future disjunctions will be checked for consistency. That is, each disjunct will be added to G and propagated, using again GRAPH_IDPC. If an inconsistency is detected then the disjunct fails and will be temporarily removed from the disjunction it belongs to. If during the *filtering* of the domains, one of the future disjunctions is annihilated then the disjuncts that were removed due to $D(i, j)$ will be restored, the attempted instantiation will be rejected, and the next disjunct of the current disjunction $D(i)$ will be tried.

We also study two variations of FC. The first one, FC1, is similar to FC, with the only difference being that the forward checking process is switched off when the current disjunction has only one available disjunct. We call such disjunctions *unary*. Forward checking is again turned on after all the unary disjunctions have been instantiated. This algorithm is inspired from [3] and may save many consistency checks.

The second variation of forward checking, FC2, works in the following way. First, it forward checks the current instantiation in the same way that FC does. Then, it adds all the disjuncts in unary future variables to the constraint graph and checks for consistency. If they are inconsistent, the algorithm continues with the next available disjunct of the current variable. If no inconsistency is detected, the unary disjunctions are removed from the graph and the algorithm continues by picking another disjunction. In Sections 5 and 6, we show theoretically and experimentally how these algorithms compare with FC.

FC-BJ is a hybrid search algorithm that combines the forward move of FC with the backward move of BJ [59]. In that way, the advantages of both algorithms are exploited. In case there are no more disjuncts left in the current disjunction $D(i)$, FC-BJ will backjump to one of the past disjunctions that are responsible for the dead-end, uninstantiate its instantiated disjunct and try the next available one. If there are no disjuncts available FC-BJ

will backtrack chronologically. Naturally, backjumping can be added to any of the versions of FC that we described. Here is an example that shows how the standard version of FC augmented with BJ works. Algorithms FC1-BJ and FC2-BJ work in a similar way.

Example 11. Suppose that we want to determine the consistency of the following set of disjunctions:

$$D(1)\ x_2 - x_1 \leqslant 5 \vee x_3 - x_4 \leqslant 6,$$
$$D(2)\ x_3 - x_1 \leqslant 4 \vee x_3 - x_4 \leqslant 5,$$
$$D(3)\ x_5 - x_4 \leqslant -6 \vee x_3 - x_4 \leqslant 4,$$
$$D(4)\ x_1 - x_3 \leqslant 0 \vee x_3 - x_4 \leqslant 2,$$
$$D(5)\ x_3 - x_5 \leqslant 2 \vee x_1 - x_3 \leqslant -6,$$
$$D(6)\ x_1 - x_2 \leqslant -8 \vee x_4 - x_3 \leqslant 1.$$

FC-BJ will begin by trying to instantiate the first disjunction. Disjunct $D(1,1)$ is instantiated and is forward checked against all the future disjuncts. The forward checking causes the elimination of $D(6,1)$. In the same way, $D(2,1)$ is instantiated and its forward checking causes the elimination of $D(5,2)$. $D(3,1)$ and $D(4,1)$ are instantiated without affecting any future disjunctions. Now, FC-BJ moves on to $D(5)$. The forward checking of $D(5,1)$ causes the elimination of $D(6,2)$. This happens because from constraints $D(5,1)$ and $D(3,1)$ we get the new constraint $x_3 - x_4 \leqslant -4$, which is obviously in conflict with $D(6,2)$. Since there are no more available disjuncts in $D(6)$, $D(5,1)$ is rejected. This leaves $D(5)$ with no available disjuncts, since $D(5,2)$ has been eliminated due to the forward checking of $D(2,1)$. Therefore, FC-BJ has reached a dead-end.

FC-BJ will backjump to the deepest past disjunction that precludes a disjunct of $D(5)$. This disjunction can be discovered by reasoning as follows. Disjunct $D(5,2)$ is eliminated because of the forward checking of $D(2,1)$. Therefore, we can say that the *culprit* for the elimination of $D(5,2)$ is disjunction $D(2)$. Disjunct $D(5,1)$ is eliminated because its forward checking results in the annihilation of $D(6)$. If $D(6)$ were not annihilated, $D(5,1)$ would be available. Therefore, the disjunctions responsible for the elimination of $D(5,1)$ are the past disjunctions whose instantiations, together with $D(5,1)$, cause the annihilation of $D(6)$. Disjunct $D(6,1)$ is eliminated because of the forward checking of $D(1,1)$. Thus, the culprit for the elimination of $D(6,1)$ is disjunction $D(1)$. Disjunct $D(6,2)$ is eliminated because it is in conflict with a constraint that is derived from constraints $D(5,1)$ and $D(3,1)$. Therefore, disjunction $D(3)$ is responsible for the elimination of $D(6,2)$. As we can see, the past disjunctions responsible for the annihilation of $D(6)$, and thus for the elimination of $D(5,1)$, are $D(1)$ and $D(3)$. The deepest of all the past disjunctions that cause the elimination of the disjuncts of $D(5)$ is therefore $D(3)$. This means that FC-BJ will backjump to $D(3)$. If FC-BJ jumps back to $D(3)$ and uninstantiates it then the forward checking of $D(5,1)$ will not cause the elimination of $D(6,2)$, which means that $D(5,1)$ will be consistently instantiated. Finally, $D(6,2)$ will be consistently instantiated, and the algorithm will terminate.

4.4. Discovering the causes of inconsistencies

Assuming that a dead-end is encountered at $D(i)$ then for each disjunct of $D(i)$ there can be more than one set of past disjunctions that is responsible for its rejection. This is demonstrated in the following example.

Example 12. Let us assume that the set of instantiated disjuncts at some point in time is:

$$x_3 - x_1 \leqslant 1,$$
$$x_3 - x_2 \leqslant 2,$$
$$x_2 - x_4 \leqslant 0,$$
$$x_1 - x_4 \leqslant 0$$

and that the current disjunction contains the disjunct $x_4 - x_3 \leqslant -3$. When this disjunct will be added to the set of instantiated disjuncts there will be two sources of inconsistency:

(1) the constraints $x_3 - x_2 \leqslant 2$, $x_2 - x_4 \leqslant 0$ and $x_4 - x_3 \leqslant -3$,
(2) the constraints $x_3 - x_1 \leqslant 1$, $x_1 - x_4 \leqslant 0$ and $x_4 - x_3 \leqslant -3$.

In the above example there are two sets of constraints, and therefore two sets of disjunctions that are responsible for the rejection of the current disjunct. For each disjunct $D(i, j)$, these sets of past disjunctions are the *conflict sets* of $D(i, j)$. In our implementation of backjumping, the algorithm will discover *only one* of the conflict sets of $D(i, j)$. This set will be the set that causes the first inconsistency that is encountered when disjunct $D(i, j)$ is added and propagated in the constraint graph. For instance, in Example 12 the constraint $x_4 - x_3 \leqslant -3$ is checked by adding it to the graph formed by the other constraints and propagating it using directional path consistency. The propagation algorithm GRAPH_IDPC will start with variable x_4, it will check the triplet x_4, x_3, x_2 and will discover the first inconsistency. The conflict set returned to the algorithm will be the disjunction(s) that has (have) created the constraints $x_3 - x_2 \leqslant 2$ and $x_2 - x_4 \leqslant 0$.

The conflict set discovered by the algorithm will be called the *culprit set* of $D(i, j)$. The aim of BJ is to select the deepest possible disjunction to backjump to. Therefore, BJ will select the deepest disjunction among the disjunctions in the culprit sets of the disjuncts of $D(i)$. In order to identify the culprit set of a disjunct $D(i, j)$, we have to know which constraints have produced the current labels on the edges involved in the inconsistency. If we have that information then we can "roll back" the changes, for each of the edges involved in the inconsistency, until we reach the instantiated disjuncts of past disjunctions. These disjunctions will form the culprit set of $D(i, j)$. The rolling back of the changes requires that each edge should be connected with the edges it is derived from. This can be done by using *dependency pointers*. If the current label on edge $x_k \rightarrow x_l$ is a result of the addition of the labels on edges $x_k \rightarrow x_m$ and $x_m \rightarrow x_l$ then edge $x_k \rightarrow x_l$ should be associated through dependency pointers with edges $x_k \rightarrow x_m$ and $x_m \rightarrow x_l$. Instead of using two pointers to point to edges $x_k \rightarrow x_m$ and $x_m \rightarrow x_l$, we have chosen to use only one pointer that points to node x_m. Obviously, if we can identify node x_m then we automatically know that the edges responsible are $x_k \rightarrow x_m$ and $x_m \rightarrow x_l$. Using these dependency pointers we can trace the changes in every edge back to an instantiated disjunct. This

is done simply by following the dependency pointers backwards. The search for the inconsistency cause stops when we reach a point where there are no more dependency pointers to be followed. That is, when all the paths of pointers lead us to edges which are not the result of propagated disjuncts, but have been directly added to the graph.

4.5. Heuristics

So far, we have assumed a static ordering of the disjunctions for all the algorithms presented. It is well known, though, that backtracking algorithms for binary CSPs benefit significantly from *dynamic variable ordering* (DVO) techniques [4,17,27]. In this section, we investigate the applicability of dynamic ordering techniques in our n-ary temporal constraint satisfaction problem.

Several heuristics have been developed for the static or dynamic ordering of variables mainly for binary CSPs [28,67]. The most popular dynamic variable ordering heuristic is based on the *fail-first (FF) principle*, proposed by Haralick and Elliot [37]. The FF principle is the following: "To succeed, try first where you are most likely to fail". Applied as a variable ordering heuristic, this suggests that at each step we choose to instantiate the variable that has the fewest values compatible with the previous instantiations. Following [4], we will call this heuristic the *minimum remaining values* (MRV) heuristic. The best way to implement the MRV heuristic is in conjunction with a forward checking algorithm. In the context of the problem we are studying, the MRV heuristic suggests that at each step we select to instantiate the disjunction that has the fewest available disjuncts. Augmenting any version of forward checking and forward checking with backjumping with the MRV heuristic is straightforward. Due to the forward checking that these algorithms do, we can find the future disjunction with the fewest available disjuncts simply by counting the disjuncts in each future disjunction. The one with the fewest available disjuncts is selected as the next disjunction to be instantiated.

When the MRV heuristic is used with FC, the algorithm will always instantiate the disjunction with the smallest domain first, which means that unary disjunctions are instantiated before all the others. When a unary disjunction is selected, forward checking may create new unary disjunctions, which in turn will be picked before the others. A non-unary disjunction will be picked only if there are no unary ones left. FC1 with MRV works in a slightly different way. At each step of the search all the unary disjunctions are instantiated, one after the other, without forward checking them, which means that no new unary disjunctions are created during this process. When all the unary disjunctions have been instantiated, a disjunct from a non-unary one will be selected and its forward checking may create new unary disjunctions. FC2 with MRV is similar to FC. Unary disjunctions are again instantiated before the others and are forward checked. The difference with FC is that at each step of the search the future unary disjunctions are checked for consistency to prune the search space earlier.

As we explain in Section 6, when many interesting real-world problems are modelled in our framework, the disjunctions have small domain sizes (usually 2 disjuncts per disjunction). For this reason our experiments were carried out on problems with disjunctions of maximum domain size 3. Due to the small domain size, the MRV heuristic leaves a large number of disjunctions having the same minimum domain size. So far we

break "ties" by randomly selecting one of these disjunctions. But it would be very helpful to devise and use informed heuristics for tie breaking. We have experimented with a tie breaking heuristic based on information from the disjunction set, as well as with some ad-hoc heuristics without success. The tie breaking heuristic we investigated selects a disjunct based on the number of times that the temporal variables in the disjunct appear in other constraints. In that way, we hope to select a disjunct that is more heavily constrained than the others. Apart from tie breaking during variable ordering, this heuristic is also doing value ordering. However, experimental results did not show any benefits from using this heuristic instead of random selection as Fig. 3 in Section 6 shows. We also experimented with the anti-heuristic that selects the constraint whose variables appear in the least number of constraints but with disappointing results. This suggests that the idea behind selecting the most heavily constrained disjunct may be worth persuing. We should note that, because of the nature of the problem we are addressing, we are unable to use heuristics for binary CSPs like the *Brelaz* heuristic [9] and the heuristics of [28] or static heuristics that depend on topological information from the constraint graph. For instance, the Brelaz heuristic chooses the variable with the maximum future degree among the variables that have minimum domain size. In this problem, though, all variables are involved in the same constraint and therefore all have the same degree.

4.6. Consistency levels of forward checking

Before moving to the theoretical and experimental analysis of our algorithms, it is instructive to compare our algorithms with the well-known local consistency enforcing algorithms [16,25,26,51,55].

We start by giving a few definitions. Let C be a set of constraints in variables $x_1, \ldots, x_n$. For any i such that $1 \leqslant i \leqslant n$, $C(x_1, \ldots, x_i)$ will denote the set of constraints in C involving only variables $x_1, \ldots, x_i$.

The following definition is from [16].

Definition 13. Let C be a set of constraints in variables $x_1, \ldots, x_n$ and $1 \leqslant i \leqslant n$. C is called *i-consistent* iff for every $i-1$ distinct variables $x_1, \ldots, x_{i-1}$, every valuation $u = \{x_1 \leftarrow x_1^0, \ldots, x_{i-1} \leftarrow x_{i-1}^0\}$ such that u is a solution of $C(x_1, \ldots, x_{i-1})$ and every variable x_i different from $x_1, \ldots, x_{i-1}$, there exists a real number x_i^0 such that u can be extended to a valuation $u' = u \cup \{x_i \leftarrow x_i^0\}$ which is a solution of $C(x_1, \ldots, x_{i-1}, x_i)$. C is called *strong i-consistent* if it is j-consistent for every j, $1 \leqslant j \leqslant i$. C is called *globally consistent* iff it is i-consistent for every i, $1 \leqslant i \leqslant n$.

At each node of the search tree, each one of our algorithms is adding constraints to a constraint graph and is making this graph directionally path consistent. Due to the special form of this graph, directional path consistency is equivalent to global consistency (this has been shown in [18]). However this equivalence refers to the constraint graph built incrementally at each node of the search tree, and has nothing to do with local consistency for the original set of temporal constraints (our algorithms do not attempt to enforce any level of consistency of that sort).

Let us now consider what our algorithms are doing, in terms of the above local consistency notions, to the m-ary meta-CSP equivalent to the original problem (m is the number of disjunctions of the original problem, see Section 2.2). Algorithm FC (and its variants) is a generalization of forward checking as defined for binary constraints [37]. At each node of the search tree, FC considers the projections of the single m-ary constraint over the past variables, the current variable, and one future variable. Each such projection defines a "virtual" constraint that involve the past, the current and one future variable. Whenever FC removes disjuncts from the future disjunctions, it is essentially enforcing a form of $(k + 2)$-consistency where k is the number of past variables. FC1 does the same when the current variable is not a unary one. If the current variable is unary then FC1 behaves like chronological backtracking. FC2 enforces a stronger level of consistency than FC. It first goes through the same process as FC and then it considers the constraint projection comprising of the past variables, the current variable, and all the future variables with unary domains. This projection creates a "virtual" constraint that is checked for consistency. Therefore, if k is the number of past variables and u is the number of future variables with unary domains, FC2 enforces a form of $(k + u + 2)$-consistency.

Recently, Bessière et al. [8] have discussed possible extensions of forward checking to non-binary constraints. We should note that our algorithms FC, FC1 and FC2 are not directly comparable to nFC1–nFC5, the generalizations of forward checking studied in [8], because we only have a single constraint in our problem. nFC1, which also works on projections, makes each projection of the current variable and one future variable arc-consistent but, to do so, it only considers values that are present in valid tuples of the m-ary constraint. Since we only have a single global m-ary constraint, the valid tuples of this constraint are the solutions to the problem, so, strictly speaking, to use nFC1 we would have to find all the solutions to the problem first. Similar comments can be made for algorithms nFC2–nFC5.

5. Theoretical results

In this section we analyse theoretically the behaviour of the backtracking algorithms we have developed. Our analysis is similar to Kondrak and van Beek theoretical evaluation of the basic backtracking algorithms for binary CSPs [42]. Kondrak and van Beek have partially ordered these algorithms according to two standard performance measures: the number of search tree nodes visited and the number of consistency checks performed.

Although our problem is *non-binary*, we show in this section that the results of [42] are also valid in our case. Additionally, we study theoretically the algorithms FC1 and FC2 and include them in the partial hierarchy. To order the algorithms, some of the analysis is based on theorems that state the necessary and sufficient conditions for nodes to be visited by the search algorithms. The proofs are very similar to the corresponding proofs for binary CSPs [42] so we do not need to repeat all of them here (some can be found in [66]).

The following two theorems summarize the sufficient and necessary conditions for a node to be visited by BT, BJ, FC, and FC-BJ. We assume that there is a static ordering of the variables and we are looking for all solutions. All the following results can be extended to the case where we are looking for the first solution [42]. Furthermore, the results are valid

when the MRV heuristic is used for dynamic variable ordering, as Kondrak and van Beek in [42] explain.

Theorem 14.
(a) *If the parent of a node is consistent then BT visits the node.*
(b) *If the parent of a node is consistent with every variable then BJ visits the node.*
(c) *If a node is consistent and the parent of the node is consistent with every variable then FC visits the node.*
(d) *If a node is consistent and the parent of the node is consistent with every set of variables then FC-BJ visits the node.*

Proof. The proofs are the same as in [42]. We should note that the proof for FC-BJ is not included in [42] but it is very similar to the proof for FC-CBJ. □

Theorem 15.
(a) *If a node is visited by BT then its parent is consistent.*
(b) *If a node is visited by BJ then its parent is consistent.*
(c) *If a node is visited by FC then it is consistent and its parent is consistent with all variables.*
(d) *If a node is visited by FC-BJ then it is consistent and its parent is consistent with all variables.*

Proof. The proofs are very similar to those in [42]. □

Based on the above theorems we were able to partially order the algorithms according to the search tree nodes they visit. We say that an algorithm *A dominates* an algorithm *B* iff any search node visited by *A* is also visited by *B* and *A strictly dominates B* iff *A* dominates *B* and there is at least one problem where *A* visits less nodes than *B*.

Corollary 16.
(a) *BJ strictly dominates BT.*
(b) *FC strictly dominates BJ.*
(c) *FC-BJ strictly dominates FC.*

Proof. The proofs are straightforward and identical to the proofs in [41]. □

Using the theorems we can also prove the correctness (i.e., the soundness and completeness) of the algorithms in a straightforward way. We can also include the variations of forward checking in the hierarchy. The following theorems prove that FC strictly dominates FC1 and it is strictly dominated by FC2.

Theorem 17. *FC strictly dominates FC1.*

Proof. To prove that FC dominates FC1, consider a node in the search tree visited by algorithm FC1. If the node corresponds to a disjunction with more than one disjuncts left in

its domain then FC1 and FC will both forward check the current assignment and remove exactly the same disjuncts from future disjunctions. If the node corresponds to a unary disjunction then FC1 will visit the next node without forward checking, and therefore, without removing any future disjuncts. FC will first forward check the current assignment and then visit the next node. Therefore, if FC1 removes a disjunct from the domain of a disjunction then FC will also remove it. In other words, if FC visits a node then FC1 will also visit it.

To prove strictness, consider a problem consisting of the following disjunctive constraints.

$$D(1)\ x_1 - x_2 \leqslant 0,$$
$$D(2)\ x_2 - x_4 \leqslant 0 \vee x_2 - x_1 \leqslant -3,$$
$$D(3)\ x_4 - x_3 \leqslant 0 \vee x_2 - x_1 \leqslant -2,$$
$$D(4)\ x_3 - x_2 \leqslant -1 \vee x_2 - x_1 \leqslant -1,$$
$$D(5)\ x_4 - x_2 \leqslant -1 \vee x_4 - x_1 \leqslant -1.$$

FC1 will instantiate $D(1, 1)$ and forward check it. This will leave the future disjunctions $D(2)$, $D(3)$ and $D(4)$ with single-element domains. FC1 will then instantiate the three unary disjunctions and discover that there is no solution when $D(5)$ is checked. Therefore, FC1 will discover the insolubility after visiting 6 nodes. FC will instantiate $D(1, 1)$ and forward check it in the same way as FC1. Then, it will instantiate $D(2, 1)$ and forward check it. This will remove both disjuncts from $D(5)$, and thus FC will discover that the problem is insoluble after visiting only 2 nodes. □

Theorem 18. *FC2 strictly dominates FC.*

Proof. The first step of FC2 is forward checking. Therefore, FC can never do more pruning than FC2, which means that FC2 can never visit more nodes than FC.

To prove strictness, consider a problem consisting of the following disjunctive constraints.

$$D(1)\ x_1 - x_2 \leqslant 0,$$
$$D(2)\ x_2 - x_4 \leqslant 0 \vee x_2 - x_1 \leqslant -3,$$
$$D(3)\ x_4 - x_3 \leqslant 0 \vee x_2 - x_1 \leqslant -2,$$
$$D(4)\ x_3 - x_2 \leqslant -1 \vee x_2 - x_1 \leqslant -1,$$
$$D(5)\ x_5 - x_6 \leqslant 0 \vee x_6 - x_7 \leqslant 0.$$

FC2 will instantiate $D(1, 1)$ and forward check it. This will leave the future disjunctions $D(2)$, $D(3)$ and $D(4)$ with single-element domains. FC2 will then check the unary disjunctions and find that they are inconsistent. Therefore, FC2 will discover the insolubility in 1 node. FC will make the instantiations $D(1, 1)$, $D(2, 1)$ and $D(3, 1)$, and thus visit 3 nodes, before discovering that the problem is insoluble.

It trivially follows that the relationship between FC and FC-BJ carries through to FC1 and FC2. That is, FC1-BJ strictly dominates FC1 and FC2-BJ strictly dominates FC2. □

The ordering of the algorithms according to the number of visited nodes helps us to partially order them according to the number of performed consistency checks as well.

BT and BJ perform exactly one consistency check at each node (i.e., they check the consistency of the current disjunct). This means that BJ never performs more consistency checks than BT. The fact that FC and FC-BJ perform the same consistency checks at each node means that FC-BJ never performs more consistency checks than FC. A relationship between the backward checking algorithms and the forward checking algorithms with respect to the number of performed consistency checks cannot be established. There are problems where BT performs more consistency checks than FC and FC-BJ, as well as problems where the opposite occurs. FC, FC1 and FC2 are also incomparable with respect to consistency checks.

The theoretical results presented above can be very useful for somebody who wants to select an algorithm for an application that uses temporal constraints. For example, we know that FC-BJ will always visit less search tree nodes and performs less consistency checks than FC, so it makes sense to choose FC-BJ (over FC) in any application domain. But will FC-BJ offer substantially better performance over FC? In what cases? To find the answers to such questions, one needs to move *from theory to practice* and study the behaviour of the two algorithms experimentally. This is what we do in the next section (for all the algorithms we have designed and implemented).

Let us also stress here that theoretical results can be misleading if they are not used carefully and accompanied by experimental analysis. For example, the experiments with random data of Section 6 and the scheduling problems of Section 7.1 show that, in practice, FC1 clearly beats FC with respect to consistency checks performed. Therefore, among the two, FC1 is the algorithm of choice. The theoretical results of this section alone could never tell us that! The only thing they tell us is that FC1 is strictly dominated by FC with respect to visited nodes, and is incomparable to it with respect to consistency checks performed.

6. Experimental results

In this section we present results from the empirical evaluation of the search algorithms. The algorithms were tested using randomly generated sets of data. First, we compared the algorithms with respect to the number of search tree nodes visited, the number of consistency checks performed, and the CPU time used. Then, we tried to investigate the phase transition in the problem we study. It was shown in [10,14,29,60,63,64] that for many NP-complete problems, hard problems occur around a critical value of a control parameter. The control parameter in our problem is the ratio r of disjunctions to variables. We have chosen this parameter because of the similarities between the problem we study and SAT problems. The disjunctions in our problem correspond to *clauses* in SAT and the disjuncts correspond to *atomic propositions*. Each disjunct can be either consistent or inconsistent, which is similar to an atomic proposition being true or false. In SAT problems the control parameter used is the ratio of clauses to variables which transferred to this problem corresponds to the ratio of disjunctions to variables.

6.1. The random generation model

The random problem generation model used is in some ways similar to the *fixed clause length model* for SAT, as described in [63]. For each set of problems there are

four parameters: the number of constrained variables n, the number of disjuncts per disjunction k, the number of disjunctions m, and the maximum integer value L. Therefore, each problem is described by the 4-tuple $\langle k, n, m, L\rangle$. As in the fixed clause length model for SAT we have kept k fixed. We mainly examined problems with $k = 2$, since this is the most interesting class in planning and scheduling applications. Many such problems can be formulated as sets of disjunctions with $k = 2$. Problems with two disjuncts per disjunction are NP-complete, in contrast with 2-SAT problems. We also present results from problems where all disjunctions have domain size 3.

For disjunction with domain size 2, we do the following. For given n, m, L, a random instance is produced by randomly generating m disjunctions of length 2. Each disjunction, $D(i) \equiv x_1 - y_1 \leqslant r_1 \vee x_2 - y_2 \leqslant r_2$, is constructed by randomly generating each disjunct $x_j - y_j \leqslant r_j$ in the following way:

(1) Two of the n variables are randomly selected with probability $1/n$. It is made sure that the two variables are different.
(2) r_j is a randomly selected integer in the interval $[0, L]$. r_j is negated with probability 0.5.
(3) If the pair of variables in $D(i, 1)$ is the same as in $D(i, 2)$ then it is made sure that r_1 is not equal to r_2 so that the disjuncts are different.

Disjunctions of size 3 were generated in a similar way.

6.2. Comparison of the algorithms

The empirical evaluation presented here helped us to estimate quantitatively the differences between the algorithms. First, we evaluated the performance of BT, BJ, FC, FC-BJ, FC+MRV, and FC-BJ+MRV on small randomly generated sets of problems involving 5 variables. Fig. 1 shows the *median* number of consistency checks performed by each algorithm. There is no curve representing FC-BJ+MRV because the number of visited nodes and consistency checks for FC-BJ+MRV are only slightly less than the corresponding numbers for FC+MRV. Along the horizontal axis is the ratio r of

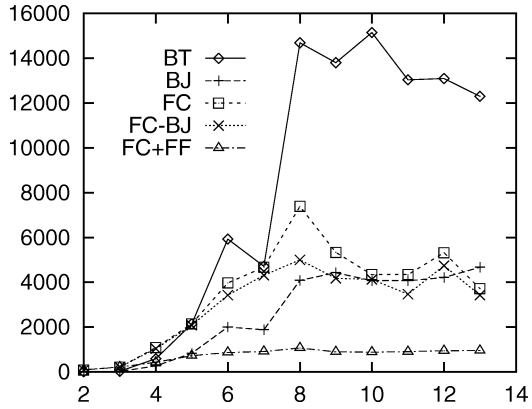

Fig. 1. The median number of consistency checks as a function of the ratio of disjunctions to variables.

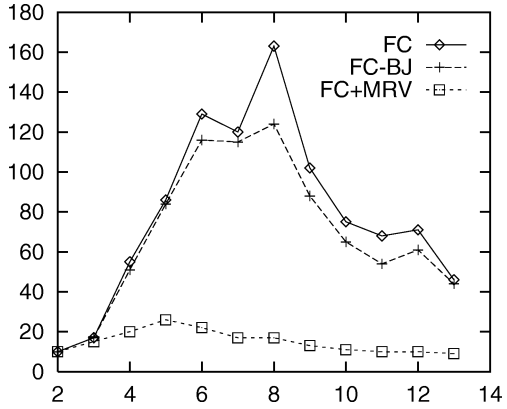

Fig. 2. The median number of visited nodes as a function of the ratio of disjunctions to variables.

disjunctions to variables. The ratio was incremented in steps of 1, starting from $r = 2$. Each data point gives the median number of consistency checks for 100 random instances of the $\langle 2, 5, m, 100\rangle$ problem, where m is the number of disjunctions.

From Fig. 1 it is obvious that FC+MRV and FC-BJ+MRV are by far the best algorithms and BT is by far the worst. It seems that BJ performs less consistency checks than FC and FC-BJ. This is caused by the forward checking that FC and FC-BJ do. For small values of r most of the problems are solved after very few, if any, backtracks. Therefore, for small problems there are no real gains by the forward checking of FC and FC-BJ. Experiments with 10 or more variables showed that this is not true for larger problems. We should note that for values of r greater than 7, the mean consistency checks performed by BJ are marginally more than the ones performed by the forward checking algorithms. This is caused by a few hard instances in which BJ performs poorly.

Fig. 2 shows the median number of nodes visited by FC, FC-BJ, and FC+MRV with random tie breaking. BT and BJ are omitted because for these algorithms the number of visited nodes is by far greater than the corresponding number for the forward checking algorithms. As we explained in Section 5, the number of visited nodes for BT and BJ is always the same as the number of consistency checks performed. Fig. 2 displays the effectiveness of dynamic variable ordering. For instance, for $r = 8$ the median number of nodes visited by FC+MRV is 17, while for FC and FC-BJ they are 163 and 124, respectively. The corresponding numbers for BJ and BT are 4088 and 14694.

We then compared the performance of the tie breaking heuristic discussed in Section 4.5 (denoted by FC+MRV+TB) to the performance of FC+MRV with random tie breaking. Fig. 3 shows the percentage of instances where the tie breaking heuristic performed better than random tie breaking. We also show the performance of the anti-heuristic (denoted by FC+MRV+antiTB). As we can see, FC+MRV with the tie breaking heuristic is in general inferior to FC+MRV with random tie breaking. Even in the cases where more than 50% of the instances benefited from FC+MRV+TB, the gain was not substantial. The anti-heuristic is clearly bad which gives an indication that the idea behind FC+MRV+TB may have potential.

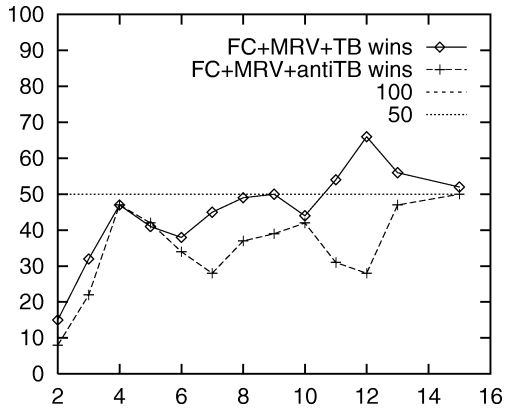

Fig. 3. The percentage of instances where FC+MRV+TB and FC+MRV+antiTB performed less consistency checks than FC+MRV. The instances have 10 temporal variables. Each disjunction has 2 disjuncts. On the x-axis is the ratio of disjunctions to temporal variables.

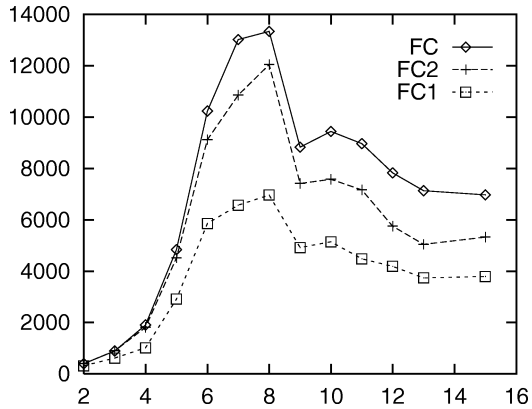

Fig. 4. Median consistency checks performed by FC, FC1 and FC2. The instances have 10 temporal variables. Each disjunction has 2 disjuncts. On the x-axis is the ratio of disjunctions to temporal variables.

Then, we compared the three versions of forward checking, FC, FC1, and FC2. The MRV heuristic was used for variable ordering. In general, FC2 visits less nodes but it performs more consistency checks at each node, while FC1 visits more nodes, but performs less consistency checks because for some nodes no forward checking is performed. Figs. 4 and 5 show the performance of FC, FC1, and FC2 on randomly generated instances with 10 and 20 temporal variables. FC2 performs less consistency checks than FC but the gain is not very significant initially when the ratio of disjunctions to variables is small. As the ratio of disjunctions to variables grows, FC2 outperforms FC in an increasing percentage of instances. This is caused by the increasing number of unary disjunctions that help FC2 to identify dead-ends earlier. FC1 is the best algorithm as it performs considerably less consistency checks than the other two algorithms. FC1 visits more nodes than the

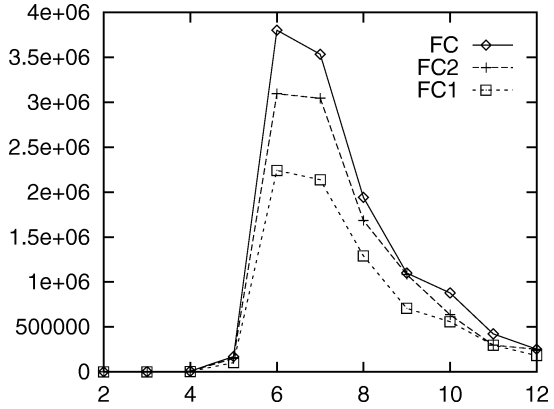

Fig. 5. Median consistency checks performed by FC, FC1 and FC2. The instances have 20 temporal variables. Each disjunction has 2 disjuncts. On the x-axis is the ratio of disjunctions to temporal variables.

other algorithms but the consistency checks that are saved when unary disjunctions are instantiated more than compensate for that. The differences between algorithms FC-BJ, FC1-BJ, and FC2-BJ were similar to those of Figs. 4 and 5. Our *best algorithm* overall was FC1-BJ+MRV, but it was only slightly better than FC1+MRV.

We have also measured the CPU times used by the algorithms we studied. As expected, the CPU times are proportional to the number of consistency checks.

6.3. The hard problems

For many NP-complete problems there is a problem parameter such that the hardest problems tend to be those for which the parameter is in a particular range [10]. In both 3-SAT problems [14,29,63] and binary CSPs [60,64] *under-constrained* problems appear to be easy to solve because they generally have many solutions. *Over-constrained* problems also appear to be easy because such problems generally have no solutions, and a sophisticated algorithm is able to quickly identify dead-ends and abandon most or all the branches in the search tree. The hardest problems generally occur in the region where there is a *phase transition* from easy problems with many solutions to easy problems with no solutions. These problems are very important for the accurate evaluation of the performance of algorithms. The region in which hard problems occur is called the *hard region*.

In order to locate the hard region, we first experimented with problems where all disjunctions have two disjuncts. We considered sets of disjunctions involving 10, 12, 15, and 20 variables. The problems created were solved using FC+MRV. The other versions of forward checking as well as forward checking with backjumping demonstrate a very similar behaviour to FC+MRV with respect to the location of the hard region. All the other algorithms were unacceptably slow at solving problems with 10 or more variables. Then, we experimented with disjunctions of domain size 3 to find out how the number of disjuncts affects the location of the hard region.

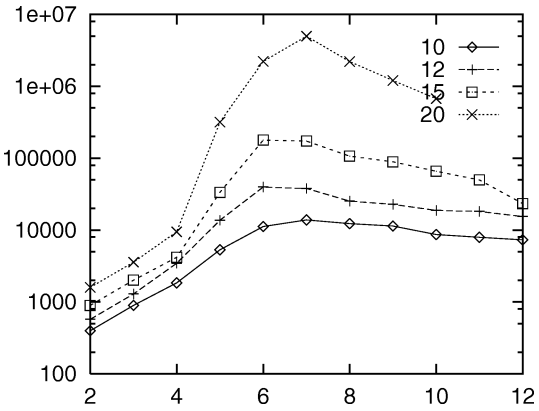

Fig. 6. The median number of consistency checks for problems with 10, 12, 15 and 20 variables as a function of the ratio of disjunctions to variables.

Fig. 6 shows the number of consistency checks performed by FC+MRV for problems with 10, 12, 15 and 20 variables.

In Fig. 6 we can notice the following:

- The curve representing the consistency checks for 10 variables starts to rise sharply at $r = 5$, reaches its peak at $r = 7$, and then slowly falls away.
- The curves representing the consistency checks for 12 and 15 variables follow the same pattern, with the difference that their peak is reached at $r = 6$.
- The curve representing the median consistency checks for 20 variables is similar to the corresponding curve for 10 variables in the sense that it reaches its peak at $r = 7$. Notice, though, that the curve for 20 variables is much sharper than the one for 10 variables.

The above observations suggest that the hardest problems occur in the region where the ratio of disjunctions to variables is between 6 and 7. Fig. 6 also shows that the hard region becomes narrower as the number of variables is increased. We should note that the curves representing the median number of visited nodes are similar to the corresponding curves in Fig. 6. The difference is that the number of consistency checks declines slower than the number of visited nodes for $r > 7$. As we shall see later, most of the problems with r more than 7 have no solutions. In this case, FC+MRV is able to discover dead-ends soon and thereby avoid visiting redundant nodes. This results in the decline of the median number of visited nodes. The median number of consistency checks declines relatively slower because as the number of disjunctions rises, FC+MRV has to do more forward checking early in the search, and therefore perform more consistency checks.

Fig. 7 shows the median and mean number of consistency checks for problems with 20 variables. The location of the peak of the mean curve is at $r = 6$, which means that it does not coincide with the peak of the median curve. As a matter of fact, the mean number of consistency checks at $r = 6$ is as much as seven times more than the median number. This is due to a few exceptionally hard problems that occurred at $r = 6$. We should mention that the same results regarding the differences between medians and means were observed for 10, 12, and 15 variables. Generally, as the number of variables increases, the distance

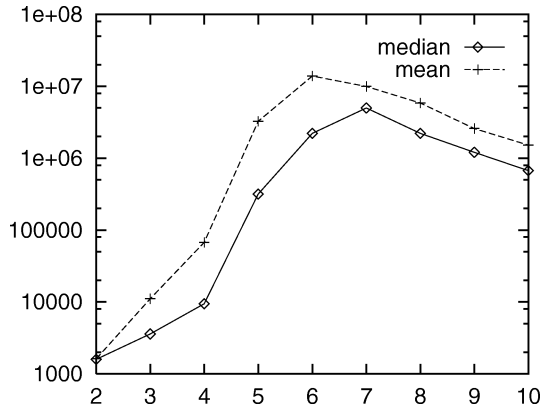

Fig. 7. The median and mean number of consistency checks for problems with 20 variables as a function of the ratio of disjunctions to variables.

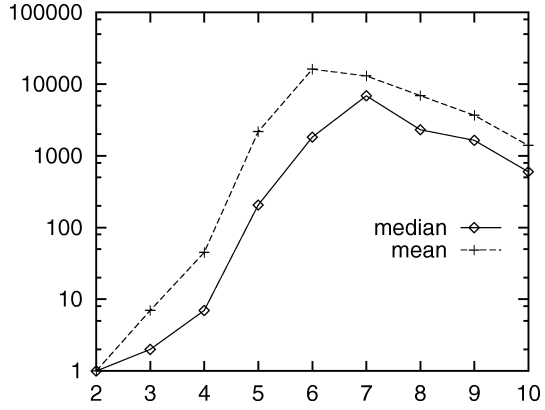

Fig. 8. The median and mean CPU time in seconds as a function of the ratio of disjunctions to variables.

between the median and mean curves increases too. This is caused by the few very hard instances that occur, which as the search space increases, tend to affect the average number of consistency checks and visited nodes more and more. In Fig. 8 we give the mean and median CPU time (in seconds) to illustrate the hardness of random problems with as few as 20 variables.

Figs. 9 and 10 shows the median number of consistency checks and visited nodes for instances with 5 and 10 variables, each having 3 disjuncts per disjunction. The peak in the number of checks and nodes has shifted to the right compared to problems with two disjuncts. Such a shift has also been observed in SAT problems when the number of literals per clause increases [54], and in binary CSPs when the domain size of the variables increases [60]. An interesting observation is that problems remain relatively hard, in terms of consistency checks, even for very high ratios of disjunctions to variables. The same is also true for problems with two disjuncts. There is, however, a clear transition in the

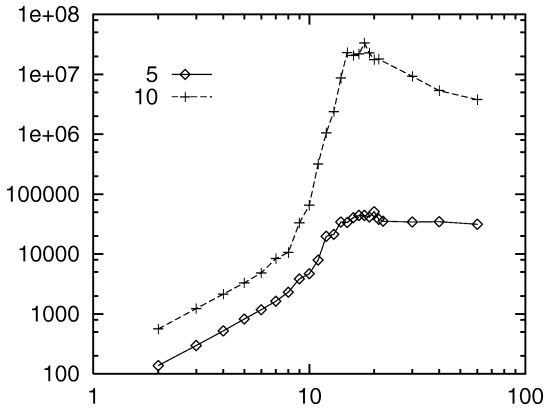

Fig. 9. The median number of consistency checks for problems with 5 and 10 variables and 3 disjuncts per disjunction as a function of the ratio of disjunctions to variables.

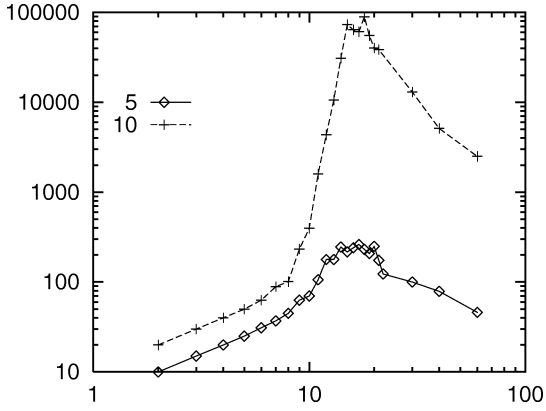

Fig. 10. The median number of visited nodes for problems with 5 and 10 variables and 3 disjuncts per disjunction as a function of the ratio of disjunctions to variables.

number of visited nodes as Fig. 10 shows. The difference in the rate that the number of nodes declines when r rises, compared to consistency checks, is made clear when we look at the numbers for $r = 18$ and $r = 60$. At $r = 18$ we encountered the hardest problems with both visited nodes and consistency checks reaching their peak, while at $r = 60$ they have their lowest values. However, the median number of nodes at $r = 60$ is 36 times lower than the corresponding number at $r = 18$, while the median number of checks is only 9 times lower.

The slow decline in the number of consistency checks can be explained by the extra effort in forward checking as the number of disjunctions increases. While the number of visited nodes starts falling after a point, the number of consistency checks that are performed at each node stays high because of the greater number of future disjunctions that have to be checked. As a result, the total number of consistency checks remains high too. A possible explanation is that forward checking is not very effective in identifying

dead-ends soon. It takes a large number of forward checks before dead-ends are identified making insoluble problems hard. This is not very surprising because of the way that forward checking is performed. As we explained in Section 4, after a disjunct is chosen for the current disjunction, it is forward checked against the disjuncts of the future disjunctions, one by one. This ensures that all the future disjuncts are consistent with the current partial solution. But since we are dealing with a non-binary CSP, this is only the lowest level of consistency that can be guaranteed. For example, we could enforce consistency among all the triples of disjuncts involving the current disjunct and two future ones. This would be very expensive, however. The scheme we are using for forward checking is cheap but does not achieve the same pruning as more expensive higher level schemes would. It remains an open question whether the use of more sophisticated variable ordering heuristics can reduce the number of checks in the insoluble region even further.

6.3.1. The phase transition from soluble to insoluble problems

Fig. 11 shows the proportion of satisfiable problems for 5, 10, 12, 15, and 20 variables as a function of r for problems with 2 disjuncts. Each data point represents the number of consistent instances out of 100 instances. As we can see, at small ratios ($r < 4$) almost all problems are satisfiable and at high ratios ($r > 7$) almost all problems are unsatisfiable. There is a range of r values over which the proportion of satisfiable problems abruptly changes from almost 100% to almost 0%. This transition region from soluble to insoluble problems becomes narrower as the number of variables increases. For 10, 12 and 15 variables the transition region appears to occur at values of r between 4 and 7. For 20 variables the proportion of satisfiable problems at $r = 5$ is 85% and at $r = 6$ it is 22%. This suggests that the phase transition from soluble to insoluble problems occurs in this area.

As we saw previously, the hard region for problems with 2 disjuncts occurs at values of r between 6 and 7. For 10 variables the hardest problems occur at $r = 7$. At this point the proportion of satisfiable problems is 13%. For 12 variables the hardest problems occur at $r = 6$ where the proportion of satisfiable problems is 22%. For 15 variables the

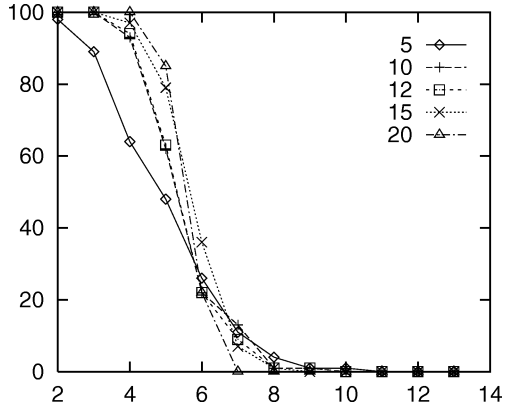

Fig. 11. Percent of satisfiable instances for problems with 5, 10, 12, 15, and 20 variables and 2 disjuncts per disjunction.

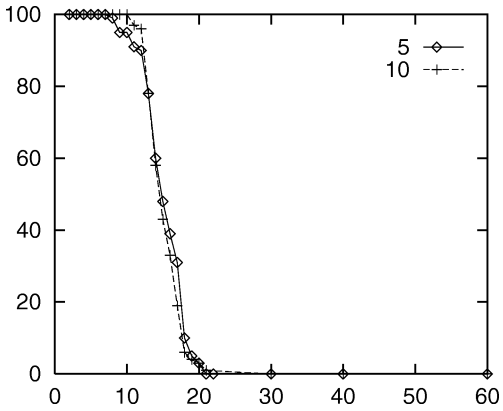

Fig. 12. Percent of satisfiable instances for problems with 5 and 10 variables and 3 disjuncts per disjunction.

proportion of satisfiable problems at $r = 6$ is 36% and at $r = 7$ it is 7%. For 20 variables the hardest problems occur at $r = 7$ where all the generated problems are unsatisfiable. It seems therefore, that the hard region does not coincide with the transition from soluble to insoluble problems. Unlike SAT problems and binary CSPs, problems near the *50%-satisfiability* point are easy. The hardest problems occur in a region where there are very few, if any, soluble instances. In Fig. 6 we can see that for values of r greater than 7 problems become easy again. This means that the problems follow the expected pattern: easy with many solutions, hard, easy with no solutions. The difference with SAT problems and binary CSPs is that the hard problems occur in the area where there are very few soluble problems. Fig. 12 shows the phase transition for problems with 3 disjuncts per disjunction. There is a shift to the right compared to problems with 2 disjuncts, but again the hard instances are encountered in a region where almost all problems are insoluble. This may be due to high variance in the number of solutions [64].

Phase transition behaviour in quantitative temporal constraint satisfaction problems has also been studied by Schwalb and Dechter. In [62], they report results on phase transition phenomena in random TCSPs, as defined in [18]. Schwalb and Dechter observed a phase transition in the number of consistent instances when varying the number of variables in the constraint graph, and also when varying the tightness of the constraints. In their experiments they used randomly generated TCSPs involving 10–20 variables with each constraint having three possible labelings. The structure of the general problem we consider is different to that of a TCSP and thus there can be no meaningful comparison of our results with the results of [62].

7. Applications

Disjunctive temporal constraints like the ones we studied here can be found in many scheduling problems [11], temporal planning problems [15,23,34,48,58,73–75], and temporal databases with indefinite information [43,44,47]. In this section we discuss relevant work and possible applications of the algorithms presented in earliest sections.

7.1. Job shop scheduling

A scheduling problem involves the performance of a set of tasks over a specified interval of time using a collection of available resources. Several CSP models and heuristics have been investigated as a means of solving scheduling problems [5,11,53,57,61]. Much of this work has focused on variations of the classic job shop scheduling problem. A *job shop scheduling problem* involves synchronization of the production of n jobs in a facility with m machines (or resources), where:

(1) each job j requires execution of a sequence of operations within a time interval specified by its *ready time* r_j and *deadline* d_j,
(2) each operation $O_{j,i}$ of job j requires exclusive use of a designated machine M_i for a specified amount of processing time p_i.

The objective is to determine a schedule for production that satisfies all temporal and resource capacity constraints. The basic job shop scheduling problem as defined above can be extended in various ways. For example, by allowing minimum and maximum bounds in operation processing times instead of fixed durations, or by adding separation constraints between consecutive operations. In job shop scheduling problems, the fact that a resource M is required by two operations A and B can be expressed by a disjunctive temporal constraint of the form

$$A \textit{ before } B \vee A \textit{ after } B.$$

This constraint states that operation A must be completed before operation B starts or vice versa.

There are different ways to formulate a job shop scheduling problem as a CSP. A job shop scheduling problem can be formulated as the problem of finding a consistent set of start times for each operation of each job. A lot of research has been carried out on constraint propagation methods, search strategies, and variable/value ordering heuristics for this formulation of the problem, see [53,57,61] among others.

Alternatively, a job shop scheduling problem can be formulated as the problem of trying to establish ordering relations between pairs of operations requiring the same resource. Following this idea, Cheng and Smith introduced the *Precedence Constraint Posting* (PCP) framework [11]. In the PCP model a decision variable $Ordering_{ij}$ is defined for each pair of operations (O_i, O_j) that compete for the same machine. This variable represents the disjunction

$$O_i \textit{ before } O_j \vee O_i \textit{ after } O_j$$

and can take two possible values: O_i *before* O_j or O_i *after* O_j. The job shop scheduling problem can now be solved by running a backtracking based search method on the *meta-CSP* whose variables are the new decision variables representing disjunctions and the values are the disjuncts of the disjunctions. In this case, the search proceeds by incrementally adding new precedence relations into an underlying temporal constraint graph and propagating each new constraint to verify consistency. If an inconsistency is encountered, backtracking occurs. The formulation of a job scheduling problem as a meta-

CSP allows it to be modelled in our framework too. This is demonstrated by the following example.

Example 19. Let us assume that we have the following 2 job, 2 machine problem. Job 1 consists of operations O_1 and O_2. Job 2 consists of operations O_3 and O_4. s_i and e_i represent the starting and ending points of operation O_i. r_i and d_i represent the ready time and deadline of job i, respectively. X_0 is an auxiliary variable representing "time 0". We have the following constraints:

- r_1 is minute 3, r_2 is minute 4, d_1 is minute 18 and d_2 is minute 20.
- O_1 and O_3 require machine M_1 while O_2 and O_4 require machine M_2.
- The *variable* durations of the operations are as follows: O_1 is 5 to 7 minutes, O_2 is 6 to 10 minutes, O_3 is 8 to 10 minutes and O_4 is 5 to 8 minutes.

The above natural language statements can be translated into temporal constraints involving endpoints of operations, ready times and deadlines. For ready times and deadlines we get the constraints

$$X_0 - s_1 \leqslant -3, \quad X_0 - s_3 \leqslant -4, \quad e_2 - X_0 \leqslant 18, \quad e_4 - X_0 \leqslant 20.$$

For the operations of Job 1, we get the constraints

$$e_1 - s_1 \in [5, 7], \quad e_2 - s_2 \in [6, 10], \quad e_1 - s_2 \leqslant 0.$$

In the same way, for Job 2 we get

$$e_3 - s_3 \in [8, 10], \quad e_4 - s_4 \in [5, 8], \quad e_3 - s_4 \leqslant 0.$$

The capacity constraint for machines together with the requirements of the operations result in the following constraints:

$$e_1 \leqslant s_3 \vee e_3 \leqslant s_1, \quad e_2 \leqslant s_4 \vee e_4 \leqslant s_2.$$

Finally, we end up with the following set of temporal constraints:

$$\begin{aligned}
&X_0 - s_1 \leqslant -3, \quad X_0 - s_3 \leqslant -4, \quad e_2 - X_0 \leqslant 18, \quad e_4 - X_0 \leqslant 20,\\
&e_1 - s_1 \leqslant 7, \quad s_1 - e_1 \leqslant -5, \quad e_2 - s_2 \leqslant 10, \quad s_2 - e_2 \leqslant -6,\\
&e_3 - s_3 \leqslant 10, \quad s_3 - e_3 \leqslant -8, \quad e_4 - s_4 \leqslant 8, \quad s_4 - e_4 \leqslant -5,\\
&e_1 - s_2 \leqslant 0, \quad e_3 - s_4 \leqslant 0,\\
&e_1 - s_3 \leqslant 0 \vee e_3 - s_1 \leqslant 0, \quad e_2 - s_4 \leqslant 0 \vee e_4 - s_2 \leqslant 0.
\end{aligned}$$

A schedule for the operations exists if and only if the above set of constraints is consistent.

Such a formulation of a job shop scheduling problem with n jobs and m machines results in a temporal CSP with $2nm + 2$ temporal variables, $n(3m + 1) + 2$ non-disjunctive constraints, and $\frac{1}{2}n(n - 1)m$ disjunctive constraints. Table 1 gives some experimental results on small job shop scheduling problems formulated in our framework and solved using our best algorithm, FC1+MRV-BJ. The first instance, the 6×6 one, is a benchmark taken from the ORLib, and the rest of the instances were generated by randomly rearranging and adding jobs and/or machines to the 6×6 instance. For each combination of jobs and machines, 20 instances were generated and the median was taken. We are trying

Table 1
Number of consistency checks performed and nodes visited to find the optimal solution to small job scheduling problems and prove optimality. In the "F Optimal" columns we give the checks and nodes required to find a solution when the deadline is equal to the optimal. In the "P Optimality" columns we give the checks and nodes required to show that no solution exists when the deadline is equal to optimal-1

Instance	Variables	Non-disjunctive constraints	Disjunctive constraints	F Optimal		P Optimality	
				Checks	Nodes	Checks	Nodes
6×6	74	116	90	8924	3932	10358	3184
6×7	86	134	105	1528	746	948	632
7×6	86	135	126	36575	30785	273289	30023
7×7	100	156	147	38300	20081	257651	17182
8×6	98	154	168	3535964	741580	5934959	483505
8×8	130	202	224	1231321	632821	1803430	590818

to minimize the makespan (i.e., the distance between the ready time and the deadline). To do that, we solve a series of decision problems until we reach the optimal makespan.

The instances in Table 1 were relatively easy (solved in a few seconds) except the 8×6 and 8×8 instances which took hours. We were not able to solve larger problems in reasonable time. For example, benchmark 10×10 instances that are relatively easy for state of the art scheduling methods could not be solved in reasonable time. This is not surprising as we are using a generic search algorithm and variable ordering heuristic that do not take account domain specific information. The generic algorithm we use performs a large amount of redundant forward checking as we now explain.

Example 20. Let us assume that we have a 10 job, 10 machine problem. Such a problem involves 202 temporal variables, 312 non-disjunctive constraints, and 450 disjunctive constraints. The generic forward checking algorithms we described in this paper will try to solve the problem in the following way: First, the set of non-disjunctive constraints will be checked for consistency and, assuming it is consistent, the search over the disjunctions will start. The algorithms will pick a disjunction and forward check it against all the future disjunctions. Let us assume that the first disjunction in the ordering is $e_{1,1} - s_{2,4} \leqslant 0 \vee e_{2,4} - s_{1,1} \leqslant 0$, which means that the first operation of the first job and the fourth operation of the second share a resource. The first disjunct of this disjunction, which puts $Operation_{1,1}$ before $Operation_{2,4}$, will be forward checked against all the remaining 449 disjunctions. Most of these checks will be redundant as the decision to schedule $Operation_{1,1}$ before $Operation_{2,4}$ does not directly prohibit decisions on the ordering of operations on other resources. Such redundant consistency checks will be performed throughout the search. Moreover, the heuristic which picks the disjunctions does not use any information domain specific from the constraint graph of the job shop problem.

Cheng and Smith in [11] solve the job shop scheduling problem in a similar way. In their approach, first, all the non-disjunctive constraints are grouped together and the minimal network of the constraint graph is calculated using full path consistency. Then, one of

the disjunctive constraints is selected and backtracking search starts. Each time a new constraint (i.e., a disjunct) is added to the network, it is propagated using a path consistency algorithm and thus the minimal network is obtained again. If an inconsistency is detected, the algorithm selects the other disjunct. In case of a dead-end, backtracking takes place. Information about the shortest paths is also exploited to provide efficient dynamic variable and value ordering heuristics. We believe that a combination of the look-ahead scheme and the heuristics of [11] or [7] with the forward checking and backjumping algorithms we have discussed will result in a more efficient search algorithm. The forward checking process would have to be modified so that redundant consistency checks are avoided. This could done by exploiting heuristic information to focus forward checking on particular disjunctions instead of blindly forward checking against all the future disjunctions.

The usefulness of our framework for disjunctive scheduling is not restricted to problems such as the classic job shop where disjunctions are of the form

$$e_i - s_j \leqslant 0 \vee e_j - s_i \leqslant 0.$$

More general constraints about operations that share the same resource can be easily expressed, as well as temporal constraints between operations in different jobs that do not share the same resource. For example consider the following disjunctive constraint: "Operation O_i can start on Machine 1 at least 10 minutes after operation O_j has finished, or operation O_j can start on Machine 1 at least 8 minutes after operation O_i has finished". The constraints imposed between the ending and starting times of different operations that share the same machine may be due to several reasons. For example, overhead for the set up of the machine, or a brief lapse of the system, or a third operation being scheduled between O_i and O_j. Beck et al. [6] and Cheng and Smith [11] point out that complex temporal constraints, like the above, are present in real-world industrial applications, but have been overlooked or have not been addressed properly by most current scheduling research.

Let us close this section by stressing that the aim of this section was *not* to present efficient algorithms for scheduling problems. The algorithms described in this paper solve a disjunctive temporal reasoning problem which can be used to model a wide variety of problems, including job shop scheduling, in a straightforward way. In the future, we plan to study in more detail the extensions sketched above, and investigate whether they are actually competitive with state of the art scheduling algorithms.

7.2. Planning with temporal constraints

There has been a lot of work on temporal planning. The most sophisticated temporal planners currently available are SIPE [73–75], O-Plan [15], IxTeT [34,48], Zeno [58] and *parc*Plan [23]. At the core of all of these planners there is a temporal reasoner which handles conjunctions of temporal constraints of the form $x - y \leqslant c$. But these are not the only useful temporal constraints in temporal planning problems. For example, when there are two actions A_1 and A_2 competing for the same resource R_1 then a *disjunctive* constraint of the form

$$A_1 \textit{ before } A_2 \vee A_1 \textit{ after } A_2$$

needs to be enforced.[3] In current temporal planners this is typically handled by creating two different branches in the search tree and considering each disjunct individually. It would be interesting to develop a temporal planner with a temporal reasoner capable of handling disjunctions of temporal constraints at its core. Incremental versions of the algorithms presented in this paper could then be used for deciding consistency of a given set of disjunctive temporal constraints. In addition the ideas of [39] (where only non-temporal planning is considered) could also be applied so that good performance is achieved by the planner.

Because the ideas presented in this section are very preliminary, we did not have the opportunity to evaluate them in practice.

7.3. Indefinite temporal constraint databases

Let us now demonstrate with an example why the problem solved in this paper is useful for query evaluation in the model of *indefinite temporal constraint databases* proposed in [44,47]. The following example is from [47]. Very similar examples appear in the AI literature (see, for example [69], for methods of evaluating queries over interval constraint networks).

Example 21. Let us consider a planning database used by a medical laboratory for keeping track of patient appointments for the year 1996. The set of rationals $\mathbb{Q}$ will be our time line. The year 1996 is assumed to start at time 0 and every interval $[i, i+1)$ represents a day (for $i \in \mathbb{Z}$ and $i \geqslant 0$). Time intervals will be represented by their endpoints. They will always be assumed to be of the form $[B, E)$ where B and E are the endpoints.

The following indefinite temporal constraint database illustrates our discussion (Table 2).

CONSTRAINT_STORE :

$$\omega_1 \geqslant 0, \quad \omega_2 \geqslant 0, \quad \omega_3 \geqslant 0, \quad \omega_4 \geqslant 0, \quad \omega_5 \geqslant 0, \quad \omega_6 \geqslant 0,$$
$$\omega_2 = \omega_1 + 1, \quad \omega_4 = \omega_3 + 1, \quad \omega_6 = \omega_5 + 2$$
$$\omega_2 \leqslant 91, \quad \omega_3 \geqslant 91, \quad \omega_4 \leqslant 182,$$
$$\omega_3 - \omega_2 \geqslant 60, \quad \omega_5 - \omega_4 \geqslant 20, \quad \omega_6 \leqslant 213.$$

Table 2

APPOINTMENT

PATIENT	TREATMENT	BEGIN	END	CON
Smith	Chemotherapy	ω_1	ω_2	*true*
Smith	Chemotherapy	ω_3	ω_4	*true*
Smith	Radiation	ω_5	ω_6	*true*

[3] More complicated disjunctive constraints are also useful. For example, "A_1 should take place at least two minutes before A_2 or at least two minutes after A_2".

This database consists of the single relation APPOINTMENT. In the model of indefinite temporal constraint databases relations are as in the standard relational model [68], but they also allow *Skolem constants* as attribute values. Skolem constants are denoted by ω_1, ω_2 etc. and represent values that are not known precisely. There is a *constraint store* that represents all the information known about the Skolem constants in the database.

The reader can now see that the above database represents the following information:

(1) There are three scheduled appointments for patient Smith. This is represented by three tuples in relation APPOINTMENT.
(2) Chemotherapy appointments must be scheduled for a single day. Radiation appointments must be scheduled for two consecutive days. This information is represented by constraints $\omega_2 = \omega_1 + 1,\ \omega_4 = \omega_3 + 1$, and $\omega_6 = \omega_5 + 2$.
(3) The first chemotherapy appointment for Smith should take place in the first three months of 1996 (i.e., days 0–91). This information is represented by the constraints $\omega_1 \geqslant 0$ and $\omega_2 \leqslant 91$.
(4) The second chemotherapy appointment for Smith should take place in the second three months of 1996 (i.e., days 92–182). This information is represented by constraints $\omega_3 \geqslant 91$ and $\omega_4 \leqslant 182$.
(5) The first chemotherapy appointment for Smith must precede the second by at least two months (60 days). This information is represented by constraint $\omega_3 - \omega_2 \geqslant 60$.
(6) The radiation appointment for Smith should follow the second chemotherapy appointment by at least 20 days. Also, it should take place before the end of July (i.e., day 213). This information is represented by constraints $\omega_5 - \omega_4 \geqslant 20$ and $\omega_6 \leqslant 213$.

Let us now consider the following query to the above database: "Is it certain that patient Smith is scheduled for chemotherapy during the first three months of 1996?" In the query language of [47] this is called a *closed necessity query* because it contains no output variables and can be expressed using the modal operator for necessity as follows:

$$\Box(\exists t_1, t_2)(\mathit{APPOINTMENT}(\mathit{Smith}, \mathit{Chemotherapy}, t_1, t_2) \wedge t_1 \geqslant 1 \wedge t_1 < t_2 \wedge t_2 \leqslant 91).$$

If $CS(\omega_1, \ldots, \omega_6)$ is the conjunction of constraints in the constraint store of Example 21 then evaluating the above query amounts to *eliminating quantifiers* from the following first order formula which mentions only temporal constraints: [4]

$$(\forall \omega_1 \cdots \omega_6)\big(CS(\omega_1, \ldots, \omega_6) \supset (\exists t_1, t_2)\ (t_1 = \omega_1 \wedge t_2 = \omega_2 \wedge t_1 \geqslant 1 \wedge t_1 < t_2 \wedge t_2 \leqslant 91) \vee (t_1 = \omega_3 \wedge t_2 = \omega_4 \wedge t_1 \geqslant 1 \wedge t_1 < t_2 \wedge t_2 \leqslant 91)\big).$$

Let us first eliminate quantifiers $(\exists t_1, t_2)$ from the above formula. The result will be

$$(\forall \omega_1 \cdots \omega_6)\big(CS(\omega_1, \ldots, \omega_6) \supset (\omega_1 \geqslant 1 \wedge \omega_1 < \omega_2 \wedge \omega_2 \leqslant 91) \vee (\omega_3 \geqslant 1 \wedge \omega_3 < \omega_4 \wedge \omega_4 \leqslant 91)\big)$$

[4] The interested reader can see [47] for details on the equivalence between query evaluation and quantifier elimination for the model of indefinite constraint databases.

or equivalently

$$\neg(\exists\omega_1 \cdots \omega_6)\big(CS(\omega_1, \ldots, \omega_6) \wedge (\omega_1 < 1 \vee \omega_1 \geqslant \omega_2 \vee \omega_2 > 91) \wedge (\omega_3 < 1 \vee \omega_3 \geqslant \omega_4 \vee \omega_4 > 91)\big).$$

Now instead of eliminating quantifiers $(\exists\omega_1 \cdots \omega_6)$ we can use the algorithms of this paper to decide whether the inner conjunction of disjunctions is consistent. If it is consistent then the equivalent formula with no quantifiers is *true*, otherwise it is *false*. The reader can verify that in this case we have an inconsistency, so the answer to the query is *true* (i.e., YES).

The above example is simple and its only purpose is to illustrate our application. In general a closed necessity query over an indefinite temporal constraint database is of the form $\Box\phi$ where ϕ is an arbitrary first order logic formula with two kinds of predicates: relation names (e.g., *APPOINTMENT*) and temporal predicates (e.g., $<$). If we evaluate such a query using the methods of [47], we will always end up deciding the consistency of a set of disjunctive temporal constraints at the last step of query evaluation.

The usefulness of our algorithms for indefinite temporal constraint databases should now be clear. Unfortunately, there is no implementation of the indefinite temporal constraint database model yet, thus we did not have any experimental data that we could use for analysing the performance of our algorithms in this application domain.

8. Conclusion and future work

We have developed several backtracking algorithms for a class of disjunctive temporal constraints. This class of temporal constraints can be used to easily model a wide range of problems from scheduling, planning, and temporal constraint databases. We presented theoretical and experimental results concerning the behaviour of the algorithms we developed. Through our theoretical analysis we were able to partially order the algorithms using Kondrak and van Beek's [42] methodology, originally introduced for binary CSPs. We have shown that this methodology can be successfully transferred to a non-binary CSP such as the one we have tackled. In our experimental analysis we compared the algorithms quantitatively and identified the phase transition in this problem using randomly generated instances. We also discussed three possible applications of our algorithms and gave experimental results on small job shop scheduling problems. We believe that our best algorithm, FC1-BJ, is a possible candidate for solving a wide range of disjunctive scheduling and planning problems, given the right domain specific heuristics. Our algorithms can be easily adapted to other temporal reasoning frameworks with disjunctive temporal constraints [65].

For future work we would like to consider the following:

- The implementation of more sophisticated backjumping mechanisms such as CBJ [59] or dynamic backtracking [35]. Such mechanisms may produce more efficient algorithms, although, as our experiments show, backjumping offers very little, when forward checking and dynamic variable ordering is used.
- We intend to further investigate the phase transition behaviour for this class of problems. This is interesting since our results show that the region of hard problems

does not coincide with the transition in solubility, as is the norm in almost all NP-complete problems. The exact reasons behind that are worth investigating using larger data sets and various combinations of parameters.

- We would like to develop new versions of our algorithms that take into account the tractability results of [38,45]. In the new versions we could start with a collection of *Horn temporal constraints* that can be solved in PTIME. A temporal constraint is called Horn if it is a disjunction of an arbitrary number of disequations of the form $x_i - y_i \neq c_i$ and at most one inequality of the form $x_k - y_k \leqslant c_k$ where x_i, y_i, x_k, y_k are real variables and c_i, c_k are real constants. Then, the disjuncts of a disjunction of the form

$$\phi \wedge x_1 - y_1 \leqslant c_1 \wedge \cdots \wedge x_n - y_n \leqslant c_n,$$

  where ϕ is a Horn temporal constraint, can be explored in only $n + 1$ steps by a backtracking algorithm.
- Finally, we would like to further investigate the applicability of our algorithms in disjunctive scheduling and planning by implementing and evaluating the ideas sketched in Section 7. An obvious starting point is job shop scheduling where we have already tested the generic algorithms presented in this paper. As explained in Section 7, there are many ways to extend our work by using domain specific heuristics and cutting down the redundant amount of work done by the generic algorithms.

Acknowledgements

We would like to thank all members of the APES Research Group and in particular Ian Gent, Patrick Prosser, Toby Walsh and Peter van Beek. Also, thanks to Alessandro Armando, Claudio Castellini and Enrico Giunchiglia. Finally, we are grateful to the referees of this paper, whose comments and suggestions have led to substantial improvements.

References

[1] J.F. Allen, Towards a general model of action and time, Artificial Intelligence 23 (2) (1984) 123–154.
[2] J.F. Allen, H. Kautz, R. Pelavin (Eds.), Reasoning about Plans, Morgan Kaufmann, San Mateo, CA, 1991.
[3] A. Armando, C. Castellini, E. Giunchiglia, Sat-based procedures for temporal reasoning, in: Proc. European Conference on Planning (ECP-99), 1999.
[4] F. Bacchus, P. van Run, Dynamic variable ordering in CSPs, in: Proc. 1st International Conference on Principles and Practice of Constraint Programming (CP-95), Cassis, France, 1995, pp. 258–275.
[5] P. Baptiste, C. Le Pape, A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling, in: Proc. IJCAI-95, Montreal, Quebec, Vol. 1, 1995, pp. 600–606.
[6] C. Beck, A. Davenport, M. Fox, Five pitfalls of empirical scheduling research, in: Proc. 3rd International Conference on Principles and Practice of Constraint Programming (CP-97), Linz, Austria, 1997, pp. 390–404.
[7] S. Belhadji, A. Isli, Temporal constraint satisfaction techniques in job shop scheduling problem solving, CONSTRAINTS 3 (1998) 203–211.
[8] C. Bessière, P. Meseguer, E. Freuder, J. Larrosa, On forward checking for non-binary constraint satisfaction, in: Proc. 5th International Conference on Principles and Practice of Constraint Programming (CP-99), Alexandria, VA, 1999, pp. 88–102.

[9] D. Brelaz, New methods to color the vertices of a graph, J. ACM 22 (4) (1979) 251–256.
[10] P. Cheeseman, B. Kanefsky, W. Taylor, Where the really hard problems are, in: Proc. IJCAI-91, Sydney, Australia, Vol 1, 1991, pp. 331–337.
[11] C.C. Cheng, S.F. Smith, Generating feasible schedules under complex metric constraints, in: Proc. AAAI-94, Seattle, WA, 1994.
[12] N. Chleq, Efficient algorithms for networks of quantitative temporal constraints, in: Proc. CONSTRAINTS-95, Melbourne Beach, FL, 1995, pp. 40–45.
[13] J. Chomicki, P.Z. Revesz, Constraint-based interoperability of spatiotemporal databases, in: Proc. SSD-97, Lecture Notes in Computer Science, Vol. 1262, Springer, Berlin, 1997, pp. 142–161.
[14] J. Crawford, D. Auton, Experimental results on the crossover point in random 3-SAT, Artificial Intelligence 81 (1996) 31–57.
[15] K. Currie, A. Tate, O-plan: The open planning architecture, Artificial Intelligence 52 (1) (1991) 49–86.
[16] R. Dechter, From local to global consistency, Artificial Intelligence 55 (1992) 87–107.
[17] R. Dechter, I. Meiri, Experimental evaluation of preprocessing algorithms for constraint satisfaction problems, Artificial Intelligence 68 (1994) 211–241.
[18] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, in: R. Brachman, H. Levesque, R. Reiter (Eds.), Proc. 1st International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario, 1989, pp. 83–93.
[19] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence (Special Volume on Knowledge Representation) 49 (1–3) (1991) 61–95.
[20] R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, Artificial Intelligence 34 (1) (1988) 1–38.
[21] T. Drakengren, P. Jonsson, Towards a complete classification of tractability in Allen's algebra, in: Proc. IJCAI-97, Nagoya, Japan, Vol. 2, 1997, pp. 1466–1471.
[22] T. Drakengren, P. Jonsson, Twenty-one large tractable subclasses of Allen's algebra, Artificial Intelligence 93 (1997) 297–319.
[23] A. El-Kholy, B. Richards, Temporal and resource reasoning: The *parc*Plan approach, in: Proc. 12th European Conference on Artificial Intelligence (ECAI-96), Budapest, Hungary, 1996, pp. 614–618.
[24] A.U. Frank, I. Campari, U. Formentini (Eds.), Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, Lecture Notes in Computer Science, Vol. 639, Springer, Berlin, 1992.
[25] E. Freuder, Synthesizing constraint expressions, Comm. ACM 21 (11) (1978) 958–966.
[26] E. Freuder, A sufficient condition for backtrack-free search, J. ACM 29 (1982) 24–32.
[27] D. Frost, R. Dechter, In search of the best constraint satisfaction search, in: Proc. AAAI-94, Seattle, WA, 1994, pp. 301–306.
[28] I. Gent, E. MacIntyre, P. Prosser, B. Smith, T. Walsh, An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem, in: Proc. 2nd International Conference on Principles and Practice of Constraint Programming (CP-96), Cambridge, MA, 1996, pp. 179–193.
[29] I.P. Gent, T. Walsh, The satisfiability constraint gap, Artificial Intelligence 81 (1996) 59–80.
[30] A. Gerevini, M. Cristani, On finding a solution in temporal constraint satisfaction problems, in: Proc. IJCAI-97, Nagoya, Japan, 1997.
[31] A. Gerevini, L. Schubert, Efficient temporal reasoning through timegraphs, in: Proc. IJCAI-93, Chambéry, France, 1993, pp. 648–654.
[32] A. Gerevini, L. Schubert, Efficient algorithms for qualitative reasoning about time, Artificial Intelligence 74 (1995) 207–248.
[33] A. Gerevini, L. Schubert, S. Schaeffer, Temporal reasoning in timegraph I-II, SIGART Bulletin 4 (3) (1993) 21–25.
[34] M. Ghallab, H. Laruelle, Representation and control in IxTeT, a temporal planner, in: Proc. 2nd International Conference on AI Planning Systems, Chicago, IL, 1994, pp. 61–67.
[35] M. Ginsberg, Dynamic backtracking, J. Artificial Intelligence Res. 1 (1993) 25–46.
[36] S. Golomb, L. Baumert, Backtrack programming, J. ACM 12 (1965) 516–524.
[37] R. Haralick, G. Elliot, Increasing tree efficiency for constraint satisfaction problems, Artificial Intelligence 14 (1980) 263–314.
[38] P. Jonsson, C. Bäckström, A linear programming approach to temporal reasoning, in: Proc. AAAI-96, Portland, OR, 1996.

[39] S. Kambhampati, X. Yang, On the role of disjunctive representations and constraint propagation in refinement planning, in: Proc. 5th International Conference on the Principles of Knowledge Representation and Reasoning (KR-96), Cambridge, MA, 1996.
[40] H. Kautz, P. Ladkin, Integrating metric and qualitative temporal reasoning, in: Proc. AAAI-91, Anaheim, CA, 1991, pp. 241–246.
[41] G. Kondrak, A theoretical evaluation of selected backtracking algorithms, Technical Report TR94-10, University of Alberta, 1994.
[42] G. Kondrak, P. van Beek, A theoretical evaluation of selected backtracking algorithms, Artificial Intelligence 89 (1997) 365–387.
[43] M. Koubarakis, Complexity results for first-order theories of temporal constraints, in: Proc. 4th International Conference on the Principles of Knowledge Representation and Reasoning (KR-94), Bonn, Germany, Morgan Kaufmann, San Francisco, CA, 1994, pp. 379–390.
[44] M. Koubarakis, Database models for infinite and indefinite temporal information, Information Systems 19 (2) (1994) 141–173.
[45] M. Koubarakis, Tractable disjunctions of linear constraints, in: Proc. 2nd International Conference on Principles and Practice of Constraint Programming (CP-96), Cambridge, MA, 1996, pp. 297–307.
[46] M. Koubarakis, From local to global consistency in temporal constraint networks, Theoret. Comput. Sci. 173 (1997) 89–112; Invited submission to the special issue dedicated to the 1st International Conference on Principles and Practice of Constraint Programming (CP-95), U. Montanari, F. Rossi (Eds.).
[47] M. Koubarakis, The complexity of query evaluation in indefinite temporal constraint databases, Theoret. Comput. Sci. 171 (1997) 25–60; Special Issue on Uncertainty in Databases and Deductive Systems, L.V.S. Lakshmanan (Ed.).
[48] P. Laborie, M. Ghallab, Planning with sharable resource constraints, in: Proc. IJCAI-95, Montreal, Quebec, 1995, pp. 1643–1649.
[49] P. Ladkin, R. Maddux, On binary constraint problems, J. ACM 41 (3) (1994) 435–469.
[50] P. Ladkin, A. Reinefeld, Effective solution of qualitative interval constraint problems, Artificial Intelligence 57 (1992) 105–124.
[51] A.K. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1977) 99–118.
[52] I. Meiri, Combining quantitative and qualitative constraints in temporal reasoning, Artificial Intelligence 87 (1996) 343–384.
[53] S. Minton, M.D. Johnston, A.B. Philips, P. Laird, Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence 58 (1992) 161–205.
[54] D. Mitchell, H. Levesque, Some pitfalls for experimenters with random SAT, Artificial Intelligence 81 (1996) 111–125.
[55] U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, Information Sciences 7 (1974) 95–132.
[56] B. Nebel, H.-J. Bürckert, Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra, J. ACM 42 (1) (1995) 43–66.
[57] W.P.M. Nuijten, Time and resource-constrained scheduling: A constraint satisfaction approach, Ph.D. Thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, 1994.
[58] J.S. Penberthy, D. Weld, Temporal planning with continuous change, in: Proc. AAAI-94, Seattle, WA, 1994, pp. 1010–1015.
[59] P. Prosser, Hybrid algorithms for the constraint satisfaction problem, Comput. Intelligence 9 (3) (1993) 268–299.
[60] P. Prosser, An empirical study of phase transitions in binary constraint satisfaction problems, Artificial Intelligence 81 (1996) 81–109.
[61] N. Sadeh, K. Sycara, Y. Xiong, Backtracking techniques for the job shop scheduling constraint satisfaction problem, Artificial Intelligence 76 (1995) 455–480.
[62] E. Schwalb, R. Dechter, Processing disjunctions in temporal constraint networks, Artificial Intelligence 93 (1997) 29–61.
[63] B. Selman, D. Mitchell, H. Levesque, Generating hard satisfiability problems, Artificial Intelligence 81 (1996) 17–29.
[64] B. Smith, M. Dyer, Locating the phase transitions in constraint satisfaction problems, Artificial Intelligence 81 (1996) 155–181.

[65] S. Staab, On non-binary temporal relations, in: Proc. ECAI-98, Brighton, UK, 1998, pp. 567–571.
[66] K. Stergiou, Backtracking algorithms for checking the consistency of disjunctions of temporal constraints, Master's Thesis, Department. of Computation, UMIST, Manchester, UK, 1997.
[67] E. Tsang, Foundations of Constraint Satisfaction, Academic Press, New York, 1993.
[68] J. Ullman, Principles of Data Base and Knowledge Base Systems, Vol. 1, Computer Science Press, Rockville, MD, 1988.
[69] P. van Beek, Temporal query processing with indefinite information, Artificial Intelligence in Medicine 3 (1991) 325–339.
[70] P. van Beek, Reasoning about qualitative temporal information, Artificial Intelligence 58 (1992) 297–326.
[71] P. van Beek, D. Manchak, The design and experimental analysis of algorithms for temporal reasoning, J. Artificial Intelligence Res. 4 (1996) 1–18.
[72] M. Vilain, H. Kautz, P. van Beek, Constraint propagation algorithms for temporal reasoning: A revised report, in: D.S. Weld, J. de Kleer (Eds.), Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann, San Mateo, CA, 1989, pp. 373–381.
[73] D.E. Wilkins, Practical Planning: Extending the Classical AI Planning Paradigm, Morgan Kaufmann, San Mateo, CA, 1988.
[74] D.E. Wilkins, Can AI planners solve practical problems, Comput. Intelligence 6 (4) (1990) 232–246.
[75] D.E. Wilkins, K. Myers, A common knowledge representation for plan generation and reactive execution, Technical Report 532R, SRI International, Menlo Park, CA, 1994; Available from http://www.ai.sri.com/people/wilkins/papers.html. To appear in Journal of Logic and Computation.