Backtracking Algorithms for Disjunctions of Temporal Constraints*†

Kostas Stergiou **Manolis Koubarakis**

Department of Computation
UMIST
P.O. Box 88
Manchester M60 1QD, U.K.
{stergiou,manolis}@co.umist.ac.uk

Abstract

We extend the framework of simple temporal problems studied originally by Dechter, Meiri and Pearl to consider constraints of the form $x_1 - y_1 \leq r_1 \vee \ldots \vee x_n - y_n \leq r_n$, 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 $n \geq 1$. We have implemented four progressively more efficient algorithms for the consistency checking problem for this class of temporal constraints. We have partially ordered those algorithms according to the number of visited search nodes and the number of performed consistency checks. Finally, we have carried out a series of experimental results on the location of the hard region. The results show that hard problems occur at a critical value of the ratio of disjunctions to variables. This value is between 6 and 7.

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, scheduling, spatio-temporal databases, geographical information systems and medical information systems.

The class of quantitative temporal constraints has been studied originally by (Dechter et al., 1989) in the framework of *simple temporal problems* (STPs) where constraints are of the form $x - y \leq c$ 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 $l \leq x - y \leq u$ involving the *same pair* of real variables x and y (l and u are real constants). (Schwalb and Dechter, 1997) studied the performance of local consistency algorithms for processing TCSPs on hard problems in the transition region.

* This work was carried out in the context of project CHOROCHRONOS funded by the Training and Mobility of Researchers programme of ESPRIT IV.

This paper continues the work on quantitative temporal constraints, but can also be seen as a *successful* transfer of methodology (theoretical and practical!) from binary CSPs to a non-binary temporal reasoning problem. We extend the framework of STPs studied originally by (Dechter et al., 1989) to consider constraints of the form $x_1 - y_1 \leq r_1 \vee \ldots \vee x_n - y_n \leq r_n$, 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 $n \geq 1$. The reader should note that we do not restrict the variables in the disjuncts to be the same pair as (Dechter et al., 1989) do in the framework of TCSPs. The added generality is useful in many problems including temporal planning, job shop scheduling and temporal constraint databases.

We have implemented four progressively more efficient algorithms for the consistency checking problem for this class of temporal constraints (backtracking, backjumping, forward checking and forward checking with backjumping)[1]. We will only present the most efficient of these algorithms, which is forward checking with backjumping. Following the methodology of (Kondrak and van Beek, 1997), 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. We have studied the performance of the above algorithms experimentally using randomly generated sets of data. We also present a series of experimental results on the location of the hard region and investigate the transition from the region where almost all problems are soluble to the region where almost no problem is soluble.

The organization of this paper is as follows. First, we present some basic definitions and describe the problem in detail. Then, we present forward checking with backjumping and the minimum remaining values heuristic. In what follows, we briefly discuss the theoretical evaluation of the algorithms and then present the results of our empirical analysis. Finally, we conclude and discuss future work.

[1]The codes for the algorithms are available from the first author.

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 $\mathcal{R}$.

Definition 1 A *temporal constraint* is a disjunction of the form $x_1 - y_1 \leq r_1 \vee \ldots \vee x_n - y_n \leq r_n$, 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 $n \geq 1$. A temporal constraint containing only one disjunct will be called *non-disjunctive*. Temporal constraints with more than one disjuncts will be called *disjunctive*.

Example 1 The following are examples of temporal constraints:

$$x_1 - y_1 \leq 2, \quad x_1 - y_1 \leq 5 \ \vee x_2 - y_2 \leq -2$$

Definition 2 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

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

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 nonempty.

The consistency checking problem for a set of m temporal constraints in n variables can be equivalently restated as a 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 values that are part of an assignment to variables must be simultaneously satisfied. The consistency of a given set C of temporal constraints can be decided in two steps. First, we check the consistency of the subset of non-disjunctive constraints. This is done using the incremental directional path consistency algorithm IDPC of (Chleq, 1995). If the subset of non-disjunctive constraints is consistent, we consider the subset of disjunctive constraints as well. Now C is consistent if for each disjunction there exists one disjunct that can be added to the subset of non-disjunctive constraints so that the new set of constraints produced is still consistent.

The Algorithm

First we introduce some necessary terminology. Some of it is similar to the terminology of (Gerevini and Schubert, 1995). The given set of disjunctions will be denoted by D. $D(i)$ and $D(i,j)$ will represent the i-th disjunction and the j-th disjunct of the i-th 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*. 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. The *future disjunctions* are the disjunctions that have not yet been instantiated. Each disjunct can be in one of three possible states at any time. It can be *available*, *current* or *eliminated*. A disjunct is available if it is neither current nor eliminated. A disjunct is current if it is part of the currently attempted instantiation of D. A disjunct becomes eliminated when it is tried and fails. A *dead-end* is a situation when all the disjuncts of the current disjunction are rejected.

A general strategy in CSPs is to preprocess the set of constraints prior to the search. Preprocessing tries to reduce the search space that backtracking algorithms explore, and in that way improve their efficiency (Dechter and Meiri, 1994). The set of disjunctions D can be reduced to an equivalent smaller subset by applying three simple *pruning rules* prior to the execution of the search algorithm:

1. If a disjunction $D(i)$ contains a disjunct that is subsumed by a single constraint in 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 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 (Dechter et al., 1989). After the preprocessing rules have been applied, the disjunctions are ordered in ascending order of domains.

FC-BJ (Prosser, 1993) is a hybrid search algorithm that combines the forward move of FC with the backward move of BJ. In that way, the advantages of both algorithms are exploited. In the context of our problem, FC-BJ 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 using a version of the IDPC algorithm of (Chleq, 1995). IDPC adds constraints to a given directional path consistent constraint graph, propagates them in the graph, and enforces directional path consistency again. Then, all disjuncts of the future disjunctions will be checked for consistency using the same version of IDPC. If a disjunct fails it will be "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. In case there are no more disjuncts left in $D(i)$, 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. Here is an example that shows how FC-BJ works.

Example 2 Suppose that we want to determine the consistency of the following set of disjunctions:

$$\begin{array}{ll} D(1) & x_2 - x_1 \leq 5 \ \vee \ x_3 - x_4 \leq 6 \\ D(2) & x_3 - x_1 \leq 4 \ \vee \ x_3 - x_4 \leq 5 \\ D(3) & x_5 - x_4 \leq -6 \ \vee \ x_3 - x_4 \leq 4 \\ D(4) & x_1 - x_3 \leq 0 \ \vee \ x_3 - x_4 \leq 2 \\ D(5) & x_3 - x_5 \leq 2 \ \vee \ x_1 - x_3 \leq -6 \\ D(6) & x_1 - x_2 \leq -8 \ \vee \ x_4 - x_3 \leq 1 \end{array}$$

FC-BJ will first try to instantiate $D(1)$. Disjunct $D(1,1)$ is instantiated and its forward checking causes the elimination of $D(6,1)$. $D(2,1)$ 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. The forward checking of $D(5,1)$ causes the elimination of $D(6,2)$. Since there are no more available disjuncts in $D(6)$, $D(5,1)$ is rejected. This leaves $D(5)$ with no available disjuncts. 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 $D(2)$. $D(5,1)$ is eliminated because its forward checking results in the annihilation of $D(6)$. 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 $D(1)$. $D(6,2)$ is eliminated because it is in conflict with a constraint that is derived from constraints $D(5,1)$ and $D(3,1)$. Therefore, $D(3)$ is responsible for the elimination of $D(6,2)$. The past disjunctions responsible for the annihilation of $D(6)$, and thus for the elimination of $D(5,1)$, are $D(1)$ and $D(3)$. FC-BJ will backjump to the deepest disjunction that precludes either $D(5,1)$ or $D(5,2)$ (i.e., to $D(3)$). If $D(3)$ is uninstantiated, the forward checking of $D(5,1)$ will not cause the elimination of $D(6,2)$, which means that $D(5,1)$ will be consistent. Finally, $D(6,2)$ will be consistently instantiated, and the algorithm will terminate.

For each inconsistent disjunct $D(i,j)$ there can be more than one set of past disjunctions that is responsible for its rejection. FC-BJ will discover *only one* of these sets. We call this set the *culprit set* of $D(i,j)$. FC-BJ tries to select the deepest possible disjunction to backjump to when a dead-end is encountered. Therefore, if there is a dead-end at $D(i)$, FC-BJ will select the deepest disjunction among the disjunctions in the culprit sets of the disjuncts of $D(i)$. The main difficulty in adapting FC-BJ to the domain of temporal constraints was identifying the culprit set of a disjunct. When an inconsistency is encountered during the consistency checking of disjunct $D(i,j)$, the cause of the inconsistency cannot be identified immediately. This is due to the constraint propagation involved in the consistency check. In order to solve this problem, we use *dependency pointers* which connect each constraint with the constraints it is derived from. In that way we can "roll back" the changes for each of the constraints involved in the inconsistency until we reach the instantiated disjuncts of past disjunctions. These disjunctions have caused the inconsistency and will form the culprit set of $D(i,j)$.

It is well known that backtracking algorithms for CSPs benefit significantly from *dynamic variable ordering* techniques (Dechter and Meiri, 1994; Bacchus and van Run, 1995)). The most popular dynamic variable ordering heuristic is the *minimum remaining values* (MRV) heuristic (also known as the *fail-first* heuristic). 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. Due to the forward checking that FC-BJ does, 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. Ties are broken randomly.

Theoretical Evaluation

We have analyzed theoretically the behaviour of the algorithms we have developed. Our analysis is similar to Kondrak and van Beek's theoretical evaluation of the basic backtracking algorithms for binary CSPs (Kondrak and van Beek, 1997). Among other results, they were able to partially order the algorithms according to two standard peformance measures: the number of search tree nodes visited, and the number of consistency checks performed. We proved that the results of (Kondrak and van Beek, 1997) for binary CSPs are also valid for our non-binary problem. First, we proved eight theorems that specify the sufficient and necessary conditions for nodes to be visited by BT, BJ, FC and FC-BJ. The proofs are similar to the corresponding proofs for binary CSPs (Kondrak and van Beek, 1997), and can be found in (Stergiou, 1997). Based on these theorems we were able to partially order the algorithms according to the search tree nodes they visit. We proved that BJ never visits more nodes than BT, FC never visits more nodes than BJ, and FC-BJ never visits more nodes than FC. BT and BJ perform exactly one consistency check at each node. 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.

Experimental Results

In this section we present extensive results from the experimental evaluation of the search algorithms. The algorithms were tested using randomly generated sets of data. First, we made a comparison of 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. (Cheeseman et al., 1991; Prosser, 1996; Smith and Dyer, 1996; Selman et al., 1996; Gent and Walsh, 1996; Crawford and Auton, 1996) showed that for many NP-complete problems, hard problems occur arround 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 our problem has and SAT problems. In SAT problems the control parameter used is the ratio of clauses to variables which transfered to our problem corresponds to the ratio of disjunctions to variables.

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 (Selman et al., 1996). 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 $< k, n, m, L >$. As in the fixed clause length model for SAT we have kept k fixed. We have only 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 constrast with 2-SAT problems.

For given n, m, L, a random instance is produced by randomly generating m disjunctions of length 2. Each disjunction, $D(i) \equiv x_1 - y_1 \leq r_1 \vee x_2 - y_2 \leq r_2$, is constructed by randomly generating each disjunct $x_j - y_j \leq r_j$ 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.

The experiments were carried out using a PC with a Pentium processor at 100 Mhz with 16 Mb of RAM.

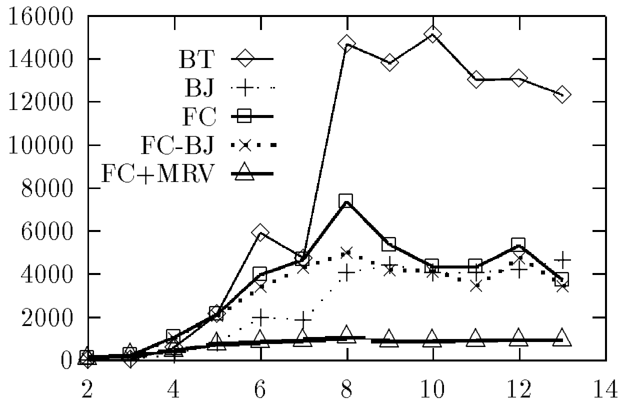

Figure 1: The median number of consistency checks as a function of the ratio of disjunctions to variables

Comparison of the Algorithms

The empirical evaluation presented here helped us to estimate quantitatively the differences between the algorithms. We evaluated the performance of BT, BJ, FC, FC-BJ, FC+MRV, and FC-BJ+MRV on randomly generated sets of problems involving 5 variables. Figure 1 shows the *median* number of consistency checks performed by each algorithm. There is no curve representing FC-BJ+MRV because the number of consistency checks for FC-BJ+MRV was only slightly less than the corresponding number for FC+MRV. Each data point gives the median number of consistency checks for 100 instances of the $< 2, 5, m, 100 >$ problem, where m is the number of disjunctions.

As expected, 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. The results with respect to the number of visited nodes and CPU time used are similar.

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 (Cheeseman et al., 1991). In both 3-SAT problems (Selman et al., 1996; Gent and Walsh, 1996; Crawford and Auton, 1996) and binary CSPs (Prosser, 1996; Smith and Dyer, 1996) *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 experimented with sets of disjunctions involving 10, 12, 15, and 20 variables. The problems created were solved using FC+MRV. We did not use FC-BJ+MRV because our experiments with 5 variables, and also additional experiments with 8, 10 and 12 variables showed that FC-BJ+MRV is only slightly better than FC+MRV[2]. All the other algorithms were unacceptably slow at solving problems with 10 or more variables. Figure 2 shows the number of consistency checks performed by FC+MRV for problems with 10, 12, 15 and 20 variables. Figure 3 shows the median and mean number of consistency checks for problems with 20 variables. 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 $r = 7$. Note, though, that the curve for 20 variables is much sharper than the one for 10 variables.

These observations suggest that the hardest problems occur in the region where the ratio of disjunctions to variables is between 6 and 7. Figure 2 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 and the CPU time used are similar to the corresponding curves in Figure 2. 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. The mean number of consistency checks is by far greater than the median (Figure 3). 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. This is due to a few exceptionally hard problems that occured at $r = 6$. Similar results regarding the differences between medians and means were observed for 10, 12, and 15 variables.

[2]Actually, in most of the generated instances FC+MRV and FC-BJ+MRV performed exactly the same number of consistency checks and visited the same number of nodes.

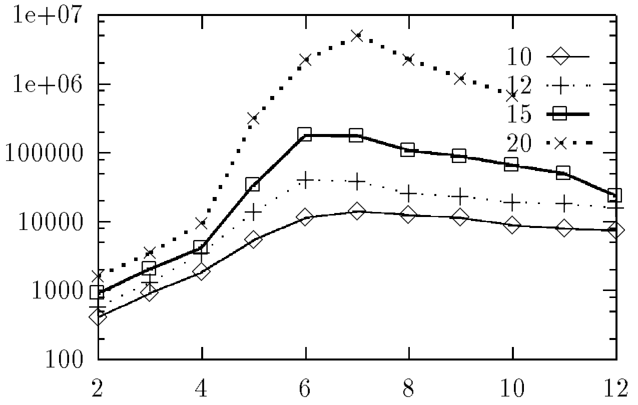

Figure 2: The median number of consistency checks for problems with 10, 12, 15 and 20 variables as a function of the ratio of disjunctions to variables

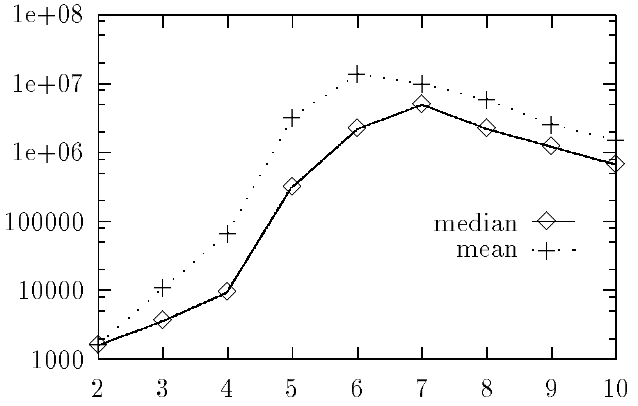

Figure 3: The median and mean number of consistency checks for 20 variables as a function of the ratio of disjunctions to variables

The Transition from Soluble to Insoluble Problems

Figure 4 shows the proportion of satisfiable problems for 5, 10, 12, 15, and 20 variables as a function of r. 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

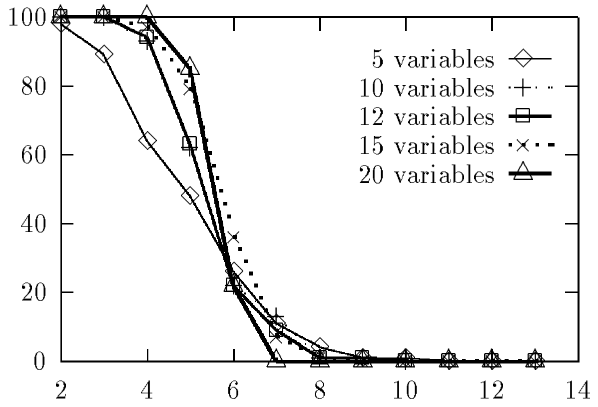

Figure 4: The percent satisfiable problems as a function of the ratio of disjunctions to variables

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 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 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 problems. In Figures 2 and 3 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. This may be due to high variance in the number of solutions (Smith and Dyer, 1996).

Future Work

We have developed backtracking algorithms for a class of disjunctive temporal constraints, and presented theoretical and experimental results concerning the behaviour of these algorithms. For future work we would like to develop tie breaking heuristics for variable ordering and discover even more efficient algorithms for determining the consistency of the studied class of temporal constraints. For this, we plan to consider *local search* algorithms similar to min-conflicts and GSAT. Local search algorithms have recently received a lot of attention in the AI literature and have produced surprising results.

Acknowledgements

We would like to thank Patrick Prosser, Ian Gent, and Toby Walsh for their help and comments.

References

Bacchus, F. and van Run, P. (1995). Dynamic variable ordering in csps. In *Proc. CP-95*, pages 258–275.

Cheeseman, P., Kanefsky, B., and Taylor, W. (1991). Where the really hard problems are. In *Proc. IJCAI-95*, volume 1, pages 331–337.

Chleq, N. (1995). Efficient algorithms for networks of quantitative temporal constraints. In *Proc. CONSTRAINTS-95*, pages 40–45.

Crawford, J. and Auton, D. (1996). Experimental Results on the Crossover Point in Random 3-SAT. *Artificial Intelligence*, 81:31–57.

Dechter, R. and Meiri, I. (1994). Experimental Evaluation of Preprocessing Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 68:211–241.

Dechter, R., Meiri, I., and Pearl, J. (1989). Temporal Constraint Networks. In *Proc. KR-89*, pages 83–93.

Gent, I. P. and Walsh, T. (1996). The Satisfiability Constraint Gap. *Artificial Intelligence*, 81:59–80.

Gerevini, A. and Schubert, L. (1995). Efficient Algorithms for Qualitative Reasoning about Time. *Artificial Intelligence*, 74:207–248.

Kondrak, G. and van Beek, P. (1997). A Theoretical Evaluation of Selected Backtracking Algorithms. *Artificial Intelligence*, 89:365–387.

Prosser, P. (1993). Hybrid Algorithms for the Costraint Satisfaction Problem. *Computational Intelligence*, 9(3):268–299.

Prosser, P. (1996). An Empirical Study of Phase Transitions in Binary Costraint Satisfaction Problems. *Artificial Intelligence*, 81:81–109.

Schwalb, E. and Dechter, R. (1997). Processing Disjunctions in Temporal Constraint Networks. *Artificial Intelligence*, 93:29–61.

Selman, B., Mitchell, D., and Levesque, H. (1996). Generating Hard Satisfiability Problems. *Artificial Intelligence*, 81:17–29.

Smith, B. and Dyer, M. (1996). Locating the Phase Transitions in Constraint Satisfaction Problems. *Artificial Intelligence*, 81:155–181.

Stergiou, K. (1997). Backtracking algorithms for checking the consistency of disjunctions of temporal constraints. Technical report, Department of Computation, UMIST.