Solving Temporal Constraints in Real Time and in a Dynamic Environment

Malek Mouhoub
Department of Math & Computer Science, University of Lethbridge
4401 University Drive, Lethbridge, AB, Canada, T1K 3M4
phone : (+1 403) 329 2557 fax : (+1 403) 329 2519
mouhoub@cs.uleth.ca

Abstract

In this paper we will present a study of different resolution techniques for solving Constraint Satisfaction Problems (CSP) in the case of temporal constraints. This later problem is called Temporal Constraint Satisfaction Problem (TCSP). We will mainly focus here on solving TCSPs in real time and in a dynamic environment. Indeed, addressing these two issues is very relevant for many real world applications. Solving a TCSP in real time is an optimization problem that we call MTCSP (Maximal Temporal Constraint Satisfaction Problems). The objective function to minimize is the number of temporal constraint violations. The results of the tests we have performed on randomly generated MTCSPs show that the approximation method Min-Conflict-Random-Walk (MCRW) is the algorithm of choice for solving MTCSPs. Comparison study of the different dynamic arc-consistency algorithms for solving dynamic temporal constraint problems in a preprocessing phase demonstrates that the new algorithm we propose and based on a recent arc-consistency algorithm represents a better compromise between time and space than the other dynamic arc-consistency algorithms.

Keywords : Temporal Reasoning, Constraint Satisfaction, Dynamic Arc-Consistency, Local Search.

1 Introduction

A large variety of problems from many different application areas can be seen as Constraint Satisfaction Problems (CSPs). For example, the problems of scheduling a collection of tasks, or interpreting a visual image, or laying out a silicon chip, can all be seen in this way. A CSP is a modeling tool and a problem solving mechanism which is very powerful in expressing, describing and solving many application domains.

In any constraint satisfaction problem there is a collection of variables which all have to be assigned values, subject to specified constraints. Because of the importance of these problems in so many different fields, a wide variety of techniques and programming languages from artificial intelligence, operations research and discrete mathematics are being developed to tackle problems of this kind.

One particular and interesting case of CSP is when the constraints are binary relations based on Allen's interval-based framework(Allen 1983) for representing temporal information. We talk then about a Temporal Constraint Satisfaction Problem (TCSP) [1].

Many problems in applications including planning, scheduling, molecular biology, database design and computational linguistics, can easily be represented by a TCSP. In a previous work(Mouhoub, Charpillet, & Haton 1998), we have developed a temporal model, TemPro, based on the interval algebra of Allen, to express both numeric and symbolic time information by a TCSP. In order to manage both type of time information in a efficient way, we have improved the different constraint satisfaction techniques using time properties.

An important issue when dealing with real world applications is the ability to solve over constrained TCSPs, TCSPs in real time and TCSPs in a dynamic environment. While solving TCSPs is a satisfiability problem consisting of finding a complete solution (complete assignment of values to variables such that all constraints are satisfied), solving over-constrained TCSPs and TCSPs in real time are optimization problems. The objective function to minimize is the number of temporal constraint violations. We call this later problem MTCSP (Maximal Temporal Constraint Satisfaction Problem). The goal here is to find a resolution method where the running time is proportional to the quality of the solution. In this paper we will study exact and approximation methods for solving MTCSPs.

When working in a dynamic environment, we may want to add new information or relax some constraints when, for example, there are no more solutions. In those cases we need to check if there still exist solutions to the problem every time a constraint has been added or removed. While adding constraints can easily be handled by the resolution method associated with the TemPro model, when relaxing a constraint our algorithms cannot find which value, that has

[1] Note that this name and the corresponding acronym was used in (Dechter, Meiri, & Pearl 1991). A comparison of the approach presented in this paper and our model TemPro is described in (Mouhoub, Charpillet, & Haton 1998)

been already removed, must be put back and which one must not. In this paper we will mainly focus on solving TCSPs in the pre-processing phase.

In the following section we will present through an example the different components of our model TemPro. Section 3 is dedicated to the resolution of MTCSPs using exact and approximation methods. In section 4 we will study the different dynamic arc-consistency algorithms for solving dynamic temporal constraint problems in the pre-processing phase. A modification of a recent arc-consistency algorithms to deal with dynamic constraints is also presented in this section. Experimental tests comparing the different approximation methods for solving MTCSPs are presented in section 5. We also present in this section other tests comparing the different dynamic arc-consistency algorithms and the new algorithm we propose. Finally, concluding remarks and future work are presented in section 6.

2 Knowledge Representation

Example 1 : Consider the following typical temporal reasoning problem[2] :

> *John, Mary and Wendy* **separately** *rode to the soccer game. It takes John* **30 minutes**, *Mary* **20 minutes** *and Wendy* **50 minutes** *to get to the soccer game. John either* **started or arrived** *just as Mary* **started**. *John either* **started or arrived** *just as Wendy* **started**. *John left home* **between 7:00 and 7:10**. *Mary and Wendy* **arrived together but started at different times**. *Mary arrived at work* **between 7:55 and 8:00**. *John's trip* **overlapped** *the soccer game. Mary's trip took place* **during** *the game or else the game took place* **during** *her trip. The soccer game* **starts at 7:30** *and* **lasts 105 minutes**.

The above story includes numeric and qualitative information (words in boldface). There are four main events : John, Mary and Wendy are going to the soccer game respectively and the soccer game itself. Some numeric constraints specify the duration of the different events, e.g. *20 minutes is the duration of Mary's event*. Other numeric constraints describe the temporal windows in which the different events occur. And finally, symbolic constraints state the relative positions between events e.g. *John's trip overlapped the soccer game*.

Given these kind of information, one important task is to represent and reason about such knowledge and answer queries such as : "is the above problem consistent ?", "what are the possible times at which Wendy arrived at the soccer game ?", ... etc. To reach this goal, and using an extension of the Allen algebra(Allen 1983) to handle numeric constraints, our model TemPro transforms a temporal problem involving numeric and symbolic information into a temporal constraint satisfaction problem (TCSP) including a set of events $\{EV_1, \ldots, EV_n\}$, each defined on a discrete domain standing for the set of possible occurrences (time intervals) in which the corresponding event can hold; and a set of binary constraints, each representing a qualitative disjunctive relation between a pair of events and thus restricting the values that the events can simultaneously take. A disjunctive relation involves one or more Allen primitives.

Relation	Symbol	Inverse	Meaning
X precedes Y	P	$P^{\smile}$	XXX YYY
X equals Y	E	E	XXX YYY
X meets Y	M	$M^{\smile}$	XXXYYY
X overlaps Y	O	$O^{\smile}$	XXXX YYYY
X during y	D	$D^{\smile}$	XXX YYYYYY
X starts Y	S	$S^{\smile}$	XXX YYYYY
X finishes Y	F	$F^{\smile}$	XXX YYYYY

Table 1: Allen primitives

The initial problem of figure 1 corresponds to the transformation of the temporal reasoning problem, we presented before, to a TCSP using the model TemPro. Information about the relative position between each pair of events is converted to a disjunction of Allen primitives. Indeed, Allen(Allen 1983) has proposed 13 basic relations between time intervals : starts (S), during (D), meets (M), overlaps (O), finishes (F), precedes (P), their converses and the relation equals (E) (see table 1 for the definition of the 13 Allen primitives). For example, the information "John either started or arrived just as Wendy started" is translated as follows : J $(S \vee S^{\smile} \vee M \vee E)$ W. In the case where there is no information between a pair of events, the corresponding relation is represented by the disjunction of the 13 Allen primitives (since this constraint is not considered during the resolution process, it does not appear on the graph of constraint as it is the case in figure 1 concerning the relation between Wendy and the Soccer game).
The domain of each event corresponding to the set of possible occurrences (we call it SOPO) that each event can take is generated given its earliest start time, latest end time and duration. In the case of Wendy's event, since we do not have any information about the earliest and latest time, these parameters are set respectively to 0 and the constant *horizon* (time before which all events should be performed). After a symbolic $\rightarrow$ numeric pre-process, these parameters are then set to 5 and 60 respectively.

Solving a TCSP consists of finding an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. Since we are dealing with a constraint satisfaction problem, deciding consistency is in general NP-hard[3]. In order to overcome this difficulty in practice, we have developed(Mouhoub, Charpillet, & Haton

[2]This problem is basically taken from an example presented by Ligozat, Guesgen and Anger at the tutorial : Tractability in Qualitative Spatial and Temporal Reasoning, IJCAI'01. We have added numeric constraints for the purpose of our work.

[3]Note that some CSP problems can be solved in polynomial time. For example, if the constraint graph corresponding to the

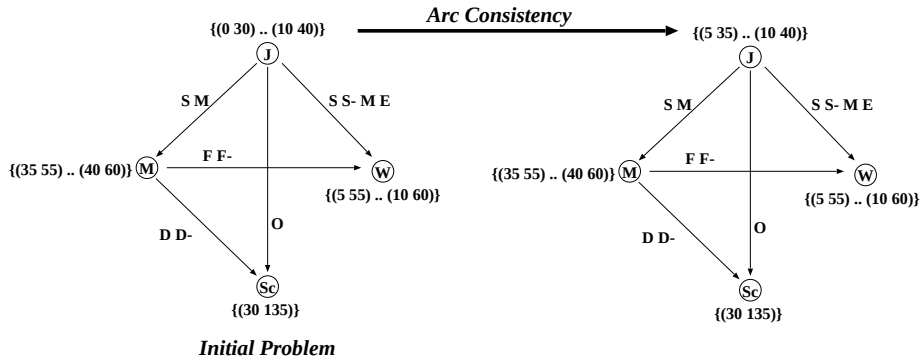

Figure 1: Applying arc-consistency to a temporal problem

1998) a resolution method performed in two stages. Local consistency algorithms are first used to reduce the size of the TCSP by removing some inconsistent values from the variable domains (in the case where arc-consistency is applied) and some inconsistent Allen primitives from the disjunctive qualitative relations (in the case where path consistency is performed). A backtrack search is then performed to look for a possible solution. When applying AC-3 to our temporal problem (see figure 1) the domain of event J is reduced.

3 Solving MTCSPs

In this section we present two different ways for solving MTCSP. The first one is an exact algorithm based on branch and bound techniques. The solution provided by this method is guaranteed to be optimal. The second way concerns approximation algorithms based on local search. Complete algorithms are the procedures of choice if the problems are small enough that all or most can be solved quickly through this approach. And as we will see in section 5, these methods can also be used to evaluate the goodness of results returned by the approximation methods. Approximation algorithms are in general used for large problems(Minton *et al.* 1992)(Selman & Kautz 1993) and, obviously, are of interest when they provide near optimal solutions with a polynomial computational complexity.

3.1 Branch and bound method

The method we present here uses partial constraint satisfaction techniques(Freuder & Wallace 1992)(Wallace 1995), capable of solving temporal constraint problems by giving a solution with a quality depending on the time allocated for computation. It consists of using a branch and bound variant of the search algorithm in order to satisfy the maximum number of constraints. We have then transformed the propagation techniques we have seen in section 2, as follows :

1. In the pre-processing phase of the resolution method, instead of performing arc-consistency algorithms to remove some values that do not belong to any solution, we use direct arc-consistency algorithms(Dechter & Pearl 1988)(Wallace 1995) to count the number of inconsistencies associated with each value (number of domains that offer no support for that value). Note that, with direct arc-consistency algorithms, checking is unidirectional so inconsistency counts are non-redundant. In order to perform direct consistency checking, a variable ordering should be first established. The ordering heuristic we use consists in sorting variables by decreasing number of constraints shared with those already instantiated. The following algorithm (see figure 2) is a modification of algorithm AC-3 to perform direct consistency checking.
2. In the search phase, a cost function corresponding to the number of violated constraints is associated with each path of the search tree. The branch and bound algorithm starts by setting an upper bound value on the cost function. Search is then performed down on each path until all variables are instantiated and in this case a new lower bound is found, or when a partial assignment of values to variables is at least as great as the actual lowest cost.

Begin
  Set the counters of variable values to 0.
  For each event e_i
    For each interval a in the domain of e_i
      For each event e_j later than e_i such that
        e_i and e_j share a qualitative relation r_{ij}
        If there is no interval b in the domain of e_j
          such that $a\, r_{ij}\, b$
          increment the counter of a
End

Figure 2: Algorithm DAC-3

3.2 Approximation methods

The different algorithms that we will consider in the following are based on a common idea known under the notion local search. In local search, an initial configuration (complete assignment of values to events) is generated randomly and the algorithm moves from the current configuration to

CSP has no loops, then the CSP can be solved in $O(nd^2)$ where n is the number of variables of the problem and d is the domain size of the different variables

a neighbor configuration (by changing one event value) until a complete solution (TCSP problems) or a good solution (MTCSP problems) has been found or the resources available are exhausted.

Min-Conflict-Random-walk method The Min-conflicts method chooses randomly any conflicting event, i.e., the event that is involved in any unsatisfied constraint, and then picks a value which minimizes the number of violated constraints (break ties randomly). If no such value exists, it picks randomly one value that does not increase the number of violated constraints (the current value of the event is picked only if all the other values increase the number of violated constraints). The problem of this method is that it is not able to leave a local minimum. In addition, if the algorithm achieves a strict local-minimum it does not perform any move at all and, consequently, it does not terminate. Thus noise strategies are introduced. Among them, the random-walk strategy that works as follows : for a given conflicting event, the random-walk strategy picks randomly a value with probability p, and apply the MC heuristic with probability 1-p. In the worst case, the time cost required in each move corresponds to the time needed to determine the value that minimizes the number of violated constraints. This time is of order $O(NMax(\frac{sup_i - inf_i - d_i}{s_i}))$.

Steepest-Descent-Random-Walk In the Steepest-Descent method, instead of selecting the event in conflict randomly, this algorithm explores the whole neighborhood of the current configuration and selects the best neighbor (neighbor with the best quality). This algorithm can be randomized by using the random-walk strategy in the same manner as for Min-Conflicts to avoid getting stuck at "local optima". The time cost required in each iteration corresponds to the time needed to find the best neighbor and is of order $O(N^2Max(\frac{sup_i - inf_i - d_i}{s_i}))$ in the worst case.

Tabu Search This method is based on the notion of tabu list used to maintain a selective history, composed of previously encountered configurations in order to prevent Tabu from being trapped in short term cycling and allows the search process to go beyond local optima. In each iteration of the algorithm, a couple $< event, v >$ that does not belong to the tabu list and corresponding to the best performance is selected and considered as an assignment of the current configuration. The previous assignment $< event, v' >$ will then replace the oldest move in the tabu list. The time cost required in each iteration is the same as for SDRW, i.e $O(N^2Max(\frac{sup_i - inf_i - d_i}{s_i}))$ in the worst case.

4 Solving Temporal Constraints in a Dynamic Environment

4.1 Dynamic Constraint Satisfaction Problem

A dynamic constraint satisfaction problem (DCSP) P is a sequence of static CSPs $P_0, \ldots, P_i, P_{i+1}, \ldots, P_n$ each resulting from a change in the preceding one imposed by the "outside world". This change can either be a restriction (adding a new constraint) or a relaxation (removing a constraint because it is no longer interesting or because the current CSP has no solution). More precisely, P_{i+1} is obtained by performing a restriction (addition of a constraint) or a relaxation (suppression of a constraint) on P_i. We consider that P_0 (initial CSP) has an empty set of constraints.

4.2 Dynamic Temporal Constraint Satisfaction Problem

Since a TCSP is a CSP where constraints are disjunctions of Allen primitives, the definition of a dynamic temporal constraint satisfaction problem (DTCSP) is slightly different from the definition of a DCSP. Indeed in the case of a DTCSP, a restriction can be obtained by removing one or more Allen primitive from a given constraint. A particular case is when the constraint is equal to the disjunction of the 13 primitives (we call it the universal relation I) which means that the constraint does not exist (there is no information about the relation between the two involved events). In this particular case, removing one or more Allen primitives from the universal relation is equivalent to adding a new constraint. Using the same way, a relaxation can be obtained by adding one or more Allen primitives to a given constraint. A particular case is when the new constraint has 13 Allen primitives which is equivalent to the suppression of the constraint. Figure 3 shows a restriction on the problem of example 1 obtained by removing some Allen primitives from the constraint between Wendy's event and the soccer game. This restriction is equivalent to the addition of the following information to our problem : *Wendy's trip took place during the game or else the game took place during her trip.*

4.3 Dynamic Arc-Consistency Algorithms

The arc-consistency algorithms we have used to solve a TCSP (Mouhoub, 1998) can easily be adapted to update the variable domains incrementally when adding a new constraint. In our example, adding the new constraint (see figure 3) will lead to an arc inconsistent TCSP which leads to an inconsistent TCSP. Let us assume now that to restore the arc-consistency we decide to relax the TCSP by adding one or more Allen primitives to a chosen constraint (one of the 10 constraints of our problem). In this case, the arc-consistency algorithms are unable to update the variable domains in an incremental way because they are not able to determine the set values that must be restored to the domains. The only way, in this case, is to reset the domains, add all the constraints (including the updated one) to the "unconstrained" TCSP (TCSP with no constraints) and then perform the arc-consistency algorithm. Dynamic arc-consistency algorithms can be used to avoid this drawback. Bessière has proposed DnAC-4(Bessière 1991) which is an adaptation of AC-4(Mohr & Henderson 1986). This algorithm stores a justification for each deleted value. These justifications are then used to determine the set of values that have been removed because of the relaxed constraint and so can process relaxations incrementally. DnAC-4 inherits the bad time and space complexity of AC-4. Indeed, comparing to AC-3 for example, AC-4 has a bad average time complexity(Wallace 1993). The worst-case space complexity of DnAC-4 is $O(ed^2 + nd)$ (e, d and n are respec-

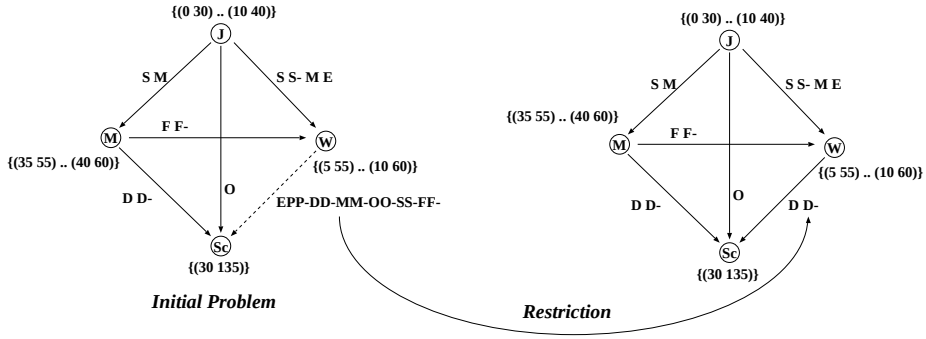

Figure 3: A restriction in a DTCSP

tively the number of constraints, the domain size of the variables and the number of variables). To work out the drawback of AC-4 while keeping an optimal worst case complexity, Bessière has proposed AC-6(Bessière 1994). Debruyne has then proposed DnAC-6 adapting the idea of AC-6 for dynamic CSPs by using justifications similar to those of DnAC-4(Debruyne 1995). While keeping an optimal worst case time complexity ($O(ed^2)$), DnAC-6 has a lower space requirements ($O(ed+nd)$) than DnAC-4. To solve the problem of space complexity, Neveu and Berlandier proposed AC|DC(Neuveu & Berlandier 1994). AC|DC is based on AC-3 and does not require data structures for storing justifications. Thus it has a very good space complexity ($O(e+nd)$) but is less efficient in time than DnAC-4. Indeed with its $O(ed^3)$ worst case time complexity, it is not the algorithm of choice for large dynamic CSPs. More recently, Zhang and Yap proposed an new version of AC-3 (called AC-3.1) achieving the optimal worst case time complexity with $O(ed^2)$ ((Zhang & Yap 2001))[4]. We have modified this algorithm in order to solve dynamic CSPs as we believe the new algorithm (that we call AC-3.1|DC) will provide better compromise between time and space than DnAC-4 and DnAC-6.

4.4 AC-3.1|DC

Before we present the algorithm AC3.1|DC, let us recall the algorithm AC-3 and the new view of AC-3 (called also AC-3.1) proposed by Zhang and Yap(Zhang & Yap 2001). Mackworth(Mackworth 1977) has presented the algorithm AC-3 for enforcing arc-consistency on a CSP. The following is the pseudo-code of AC-3 in the case of a TCSP. The worst case time complexity of AC-3 is bounded by $O(ed^3)$(Mackworth & Freuder 1985). In fact this complexity depends mainly on the way line 3 of the function *REVISE* is implemented. Indeed, if anytime the arc (i,j) is revised, b is searched from scratch then the worst case time complexity is $O(ed^3)$. Instead of a search from scratch, Zhang and Yap(Zhang & Yap 2001) proposed a new view that allows the search to resume from the point where it stopped in the previous revision of (i,j). By doing so the worst case time complexity of AC-3 is achieved in $O(ed^2)$.

Function $REVISE(i,j)$
1. $REVISE \leftarrow false$
2. **For** each interval $a \in SOPO_i$ **Do**
3. **If** $\neg compatible(a,b)$ **for each interval** $b \in SOPO_j$ **Then**
4. remove a from $SOPO_i$
5. $REVISE \leftarrow true$
6. **End-If**
7. **End-For**

Algorithm AC-3
1. Given a TemPro network $TN = (E,R)$
   (*E: set of events,*
   R: set of disjunctive relations between events)
2. $Q \leftarrow \{(i,j) \mid (i,j) \in R\}$
   (list initialized to all relations of TN)
3. **While** $Q \neq Nil$ **Do**
4. $Q \leftarrow Q - \{(i,j)\}$
5. **If** $REVISE(i,j)$ **Then**
6. $Q \leftarrow Q \sqcup \{(k,i) \mid (k,i) \in R \wedge k \neq j\}$
7. **End-If**
8. **End-While**

In the case of constraint restriction, AC-3.1|DC works in the same way as AC-3.1. The worst-case time complexity of a restriction is then $O(ed^2)$. The more interesting question is whether AC-3.1|DC's time complexity can remain the same during retractions. Indeed, if we use the same way as for AC|DC (Neuveu & Berlandier 1994), one major concern is that during the restrictions, the AC-3.1 algorithm keeps a Resume table of the last place to start checking for consistency from. Unfortunately, during retractions, this Resume table may prove useless as values in the domain of nodes are restored. Our attempt was to follow an idea observed from the DnAC6 algorithm. Instead of replacing values in the node in the order they were deleted, the algorithm should place these values to be restored at the end of the list of values for that node, thereby keeping the Resume table intact. More precisely, the constraint relaxation, of a given relation (k,m) for example, is performed in 3 steps :

1. An estimation (over-estimation) of the set of values that have been removed because of the constraint (k,m) is first determined by looking for the values removed from the

[4] Another arc consistency algorithm (called AC-2001) based on the same idea as AC-3.1 (and having an $O(ed^2)$ worst case time complexity) was proposed by Bessière and Régin(Bessière & Régin 2001). We have chosen AC-3.1 for the simplicity of its implementation

domains of k and m that have no support on (k,m),

2. the above set is then propagated to the other variables,
3. and finally a filtering procedure based on AC-3.1 is then performed to remove from the estimation set the values which are not arc-consistent with respect to the relaxed problem.

Since the time complexity of each of the above steps is $O(ed^2)$, the worst-case time complexity of a relaxation is $O(ed^2)$. Comparing to AC|DC, AC3.1|DC has a better time complexity. Indeed, the main difference between AC3.1|DC and AC|DC is the third step. This later step requires $O(ed^3)$ in the case of AC|DC (which results in a $O(ed^3)$ worst case time complexity for a restriction). In the case of AC3.1|DC, the third step can be performed in $O(ed^2)$ in the worst case because of the improvement we mentioned above. Comparing to DnAC-4 and DnAC-6, AC-3.1|DC has a better space complexity ($O(e+nd)$) while keeping an optimal worst-case time complexity ($O(ed^2)$).

5 Experimentation

The tests presented in this section are performed on consistent and inconsistent temporal problems. The consistent problems are randomly generated as follows :

Randomly pick n (n is the number of variables of the problem to generate) pairs (x,y) of integers such that $x < y$ and $x,y \in [0,\ldots,Horizon]$ ($Horizon$ is the parameter before which all events must be processed). This set of n pairs forms the initial solution where each pair corresponds to a time interval.

Metric constraints are generated as follows:

For each interval (x,y) randomly pick an interval contained within $[0..Horizon]$ and containing the interval (x,y). This newly generated interval defines the SOPO of the corresponding variable.

Qualitative constraints are obtained as follows :

Compute the basic Interval-Interval relations that can hold between each interval pair of the initial solution. Add to each relation a random number in the interval $[0,Nr]$ ($1 < Nr \leq 13$) of chosen basic Interval-Interval relations.

The generated problems can be characterized by their tightness, which can be measured, as shown in (Sabin & Freuder 1994), using the following definition :

> *The tightness of a CSP problem is the fraction of all possible pairs of values from the domain of two variables that are not allowed by the constraint.*

The tightness depends in our case on the parameters $Horizon, Nr$ and the density of the problem.

In the case of inconsistent problems, we randomly pick a number of pairs belonging to an initial solution and add other randomly generated pairs of values. This is then considered as the initial incomplete solution that we will use to generate the list of qualitative and quantitative constraints as shown above.

The experiments were performed on a SUN SPARC Ultra 5 station. All the procedures are coded in C/C++.

5.1 Comparison of the approximation methods

We use two criteria to compare the different approximation methods. The first one is the quality of the solution, i.e the minimum number of violated constraints and the second criterion is the computing effort needed by an algorithm to find its best solution. This last criterion is measured by the number of moves and the running time in seconds required by each algorithm.

The performances of the approximation algorithms we use are greatly influenced by the following parameters : the size of the tabu list, tl_size, in the case of tabu search; and the random walk probability, p, in the case of MCRW and SDRW. Preliminary tests determine the following ranges of parameter values : $10 \leq tl\_size \leq 20$ and $0.05 \leq p \leq 0.15$. Different discrete values between these ranges were further tested and the best values were identified for each class of problems.

Table 2 presents tests performed on consistent problems randomly generated. It gives a summary of the best results of MCRW, SDRW and Tabu search for the chosen instances in terms of quality of the solutions. The results correspond to the average time, number of moves and quality of solution provided by each method. To obtain these results, the algorithms were run 100 times on each instance, each run being given a maximum of 100,000 moves in the case of MCRW and 10,000 moves in the case of SDRW and tabu search. The parameter of each algorithm (the size of the tabu list tl_size and the random-walk probability p) is fixed according to the best value found during the parametric study. Note that, as noticed before in subsection 3.2, the cost in time of a move in the case of Tabu Search and SDRW is equal to N times the cost of a move in the case of the MCRW method, where N is the number of variables (events).

From the data of Table 2, we can make the following observations. For under-constrained and middle-constrained problems, the MCRW method always provides the best results. It always founds a complete solution within a reasonable amount of time which is not the case of the other two methods. It is also faster than the other two methods to find solutions of the same quality. However for over-constrained problems (see last row of table 2) SDRW and Tabu Search have better performance. We can explain this by the fact that, for under constrained problems the initial configuration is in general of good quality. A complete solution can be obtained in this case by only changing the values of some conflicting variables (case of MCRW) instead of looking for the best neighbor which is much more expensive.

Table 3 presents tests performed on inconsistent problems randomly generated. For each instance, the exact method based on branch and bound techniques(Mouhoub 2000) was first performed in order to get the optimal solution (solution with the minimum number of violated constraints). The three algorithms are then run 100 times on each instance, each run being given a maximum of 100,000 moves in the case of MCRW and 10,000 moves in the case of SDRW and

Tightness	MCRW				SDRW				Tabu Search			
of the problem	p	qual	time	# moves	p	qual	time	# moves	tl_size	qual	time	# moves
0.0002	5	0	0.12	5	15	0	2.67	80	10	0	0.17	4
0.0004	5	0	0.28	18	15	0	4.95	136	10	1	185	5000
0.001	5	0	0.46	28	5	0	8.24	193	10	0	0.6	16
0.002	5	0	0.95	68	5	0	11.22	212	10	2	294	10000
0.0037	5	0	1.74	145	10	0	126	712	10	1	270	10000
0.006	5	0	4	255	10	0	33	336	10	3	286	10000
0.03	15	0	86	3713	10	33	33802	10000	10	12	349	10000
0.044	5	0	73	1633	5	4	9595	10000	15	25	355	10000
0.045	5	0	72	1633	5	4	9614	10000	10	16	376	10000
0.058	5	0	15	433	5	74	12333	10000	15	12	364	10000
0.1	5	0	12	332	5	0	34	225	10	0	112	211
0.14	5	0	8.47	304	5	0	39	243	10	0	112	193
0.35	5	0	181	2009	15	0	66	210	20	68	714	10000
0.44	5	0	137	1291	5	220	8346	10000	10	63	646	10000
0.55	5	0	315	2505	5	0	66	210	10	0	262	190
0.67	5	372	13945	100000	5	0	130	297	10	0	422	224

Table 2: Comparative results of Tabu search, MCRW and SDRW for consistent problems

Tightness	MCRW				SDRW				Tabu Search			
of the problem	p	qual	time	# moves	p	qual	time	# moves	tl_size	qual	time	# moves
0.0002	5	**8**	0.44	32	15	8	4.5	107	10	8	0.28	6
0.001	5	**10**	0.7	53	5	10	10.26	199	10	11	242	5000
0.002	5	**2**	0.68	43	5	3	7.77	183	10	2	194	5000
0.0037	5	**14**	1237	100000	10	14	14.62	238	10	18	230	5000
0.006	5	**20**	5.83	425	10	20	33	336	10	22	377	10000
0.03	15	**21**	190	5406	10	32	3663	10000	10	85	341	10000
0.044	5	**43**	853	25	5	46	4827	10000	15	45	255	10000
0.1	5	**41**	10	318	5	106	41	233	10	91	25	230
0.14	5	**208**	10.14	279	5	208	37	215	10	230	22	197
0.35	5	**141**	259	3015	15	141	439	554	20	141	201	415
0.44	5	**531**	105	271	5	531	82	216	10	531	48	195
0.67	5	**858**	156	315	5	858	98	206	10	924	58	224

Table 3: Comparative results of Tabu search, MCRW and SDRW for non consistent problems

tabu search. Quality values in boldface correspond to the best quality (optimal solution) found by the exact method. From table 3 we can make the same observations as for table 2 i.e the MCRW method is the algorithm of choice if we have to deal with under-constrained or middle-constrained problems. The effort made by SDRW and tabu search methods to look for the best neighbor helps only in the case of over constrained problems.

5.2 Comparison of the dynamic arc-consistency algorithms

In order to compare the performance of the four dynamic arc-consistency algorithms we have seen in the previous subsection, in the case of temporal constraints, we have performed tests on randomly generated DTCSPs. The criterion used to compare the above algorithms is the computing effort needed by an algorithm to perform the arc consistency. This criterion is measured by the running time in seconds required by each algorithm. 3 classes of instances, corresponding to 3 type of tests, were generated as follows:

case 1 : actions correspond to additions of constraints. $C = N(N-1)/2$ (constraints are added until a complete graph is obtained).

case 2 : actions can be additions or retractions of constraints.
$C = N(N-1)/2$ additions $+ N(N-1)/4$ retractions (the final TCSP will have $N(N-1)/4$ constraints).

case 3 : this case is similar to case 1 but with inconsistent DTCSPs. Indeed in the previous 2 cases the generated DTCSPs are consistent. In this last case constraints are added until an arc inconsistency and thus a global inconsistency is detected (the inconsistency is detected if one variable domain becomes empty). Retractions are then performed until the arc-consistency is restored.

Figure 4a) shows the results of tests corresponding to case 1. As we can easily see, the results provided by DnAC-6 and AC-3.1|DC are better than the ones provided by AC|DC and DnAC-4 (which do not appear on the chart). Since DnAC-6 requires much more memory space than AC-3.1|DC, this latter is the algorithm of choice in the case of constraint additions. Figure 4b) and 4c) correspond to case 2 and case 3 respectively. DnAC-4 and DnAC-6 have better performance in this case than AC3.1|DC and AC|DC (the running time of AC|DC is very slow comparing to the other 3 algorithms). However, since AC3.1|DC does not require a lot of memory space, it has less limitations than DnAC-4 and DnAC-6 in terms of space requirements especially in the case of problems having large domain sizes.

6 Conclusion and Future Work

In this paper we have presented a study of the different methods for solving MTCSPs and DTCSPs. Experimental comparative study of the different approximation methods for solving MTCSPs were performed on randomly generated temporal constraint problems. Experimental tests show that the MCRW technique is the method of choice in the case of under-constrained and middle-constrained problems while

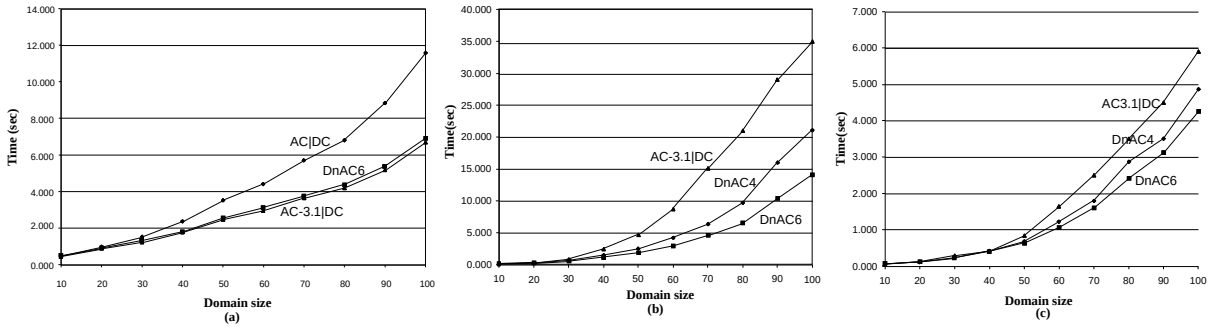

Figure 4: Experimental Tests on random DTCSPs

Tabu search and STRD have better performance for over-constrained problems. Theoretical and experimental comparison of the different dynamic arc-consistency algorithms demonstrates that the new algorithm we propose has a better compromise between time and space than the other dynamic arc-consistency algorithms.

One perspective of our work is to look for a method to maintain path consistency when dealing with dynamic temporal constraints. Indeed, as we have shown in(Mouhoub, Charpillet, & Haton 1998), path consistency is useful in the filtering phase to detect the inconsistency when solving temporal constraint problems and also in the case where the numeric information is incomplete. The other perspective is to handle the addition and relaxation of constraints during the backtrack search phase. For example, suppose that during the backtrack search a constraint is added when instantiating the current variable. In this case, the instantiation of the variables already instantiated should be reconsidered and the domains of the current and future variables should be updated.

References

Allen, J. 1983. Maintaining knowledge about temporal intervals. *CACM* 26(11):832–843.

Bessière, C., and Régin, J. C. 2001. Refining the basic constraint propagation algorithm. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 309–315.

Bessière, C. 1991. Arc-consistency in dynamic constraint satisfaction problems. In *AAAI'91*, 221–226.

Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artificial Intelligence* 65:179–190.

Debruyne, R. 1995. Les algorithmes d'arc-consistance dans les csp dynamiques. *Revue d'Intelligence Artificielle* 9:239–267.

Dechter, R., and Pearl, J. 1988. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence* 34:1–38.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58:21–70.

Mackworth, A. K., and Freuder, E. 1985. The complexity of some polynomial network-consistency algorithms for constraint satisfaction problems. *Artificial Intelligence* 25:65–74.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58:161–205.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28:225–233.

Mouhoub, M.; Charpillet, F.; and Haton, J. 1998. Experimental analysis of numeric and symbolic constraint satisfaction techniques for temporal reasoning. *Constraints: An International Journal* 2:151–164, Kluwer Academic Publishers.

Mouhoub, M. 2000. Reasoning about Numeric and Symbolic Time Information. In *the Twelfth IEEE International Conference on Tools with Artificial Intelligence(ICTAI'2000)*, 164–172. Vancouver: IEEE Computer Society.

Neuveu, B., and Berlandier, P. 1994. Arc-consistency for dynamic constraint satisfaction problems : An rms free approach. In *ECAI-94, Workshop on Constraint Satisfaction Issues Raised by Practical Applications*.

Sabin, D., and Freuder, E. C. 1994. Contradicting conventional wisdom in constraint satisfaction. In *Proc. 11th ECAI*, 125–129.

Selman, B., and Kautz, H. A. 1993. An empirical study of greedy local search for satisfiability testing. In *AAAI'93*, 46–51.

Wallace, R. J. 1993. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In *IJCAI'93*, 239–245.

Wallace, R. J. 1995. Partial constraint satisfaction. *Lecture Notes in Computer Science* 923:121–138.

Zhang, Y., and Yap, R. H. C. 2001. Making ac-3 an optimal algorithm. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 316–321.