Constraint Propagation versus Local Search for Incremental Temporal Constraint Problems

Malek Mouhoub
Department of Computer Science, University of Regina
Regina SK, Canada
mouhoubm@cs.uregina.ca

Abstract

Our aim in this paper is to maintain the global consistency of a constraint satisfaction problem involving temporal constraints anytime a new constraint is added. This problem is of practical relevance since it is often required to check whether a solution to a CSP continues to be a solution when a new constraint is added and if not, whether a new solution satisfying the old and new constraints can be found.

The two methods that we will present here are respectively a complete search technique based on constraint propagation and an approximation method based on stochastic local search. The goal of both methods is to check whether the existence of a solution is maintained anytime a new constraint is added. The approximation method does not guarantee the completeness of the solution provided, but is of interest for those problems where it is impossible or impractical to find a complete solution. This is the case of real time applications where a solution should be returned within a given deadline and over constrained problems where a complete solution does not exist.

Keywords: Temporal Reasoning, Constraint Satisfaction, Planning and Scheduling.

1 INTRODUCTION

In any constraint satisfaction problem (CSP) there is a collection of variables which all have to be assigned values from their discrete domains, 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. An important issue when dealing with a constraint satisfaction problem in the real world is the ability of maintaining the consistency of the problem in a dynamic environment i.e anytime there is a constraint restriction or relaxation. Indeed, in the case of constraint restriction, this change may affect the solution already obtained with the old constraints. In the past decade several algorithms based on constraint propagation have been proposed to enforce a particular case of local consistency, called arc consistency (or 2-consistency), in a dynamic environment. Our goal in this paper is to maintain the global consistency, in a dynamic environment, of a constraint satisfaction problem involving qualitative and quantitative temporal constraints. This is of practical relevance for many real world applications such as reactive scheduling and planning. In scheduling problems, for example, a solution corresponding to an ordering of tasks to be processed can no longer be consistent if a given machine becomes unavailable. We have then to look for another solution (ordering of tasks) satisfying the old constraints and taking into account the new information.

In a previous work[1], we have developed a temporal model, TemPro, based on the interval algebra, to express numeric and symbolic time information in terms of qualitative and quantitative temporal constraints. More precisely, TemPro translates an application involving temporal information into a binary Constraint Satisfaction Problem[1] where constraints are temporal relations. We call it Temporal Constraint Satisfaction Problem (TCSP)[2]. Managing temporal information consists then of maintaining the consistency of the related TCSP using constraint satisfaction techniques. Local consistency is enforced by applying the arc consistency for numeric constraints and the path consistency for symbolic relations. Global consistency is then obtained by using a backtrack search algorithm to look for a possible solution. Note that for some TCSPs local consistency implies global consistency[3].

In order to check for the global consistency of a TCSP in a dynamic environment, we have adapted the above local consistency techniques and backtrack search in order to handle the addition of constraints in an efficient way. This method has the advantage to be complete, however for large size problems it suffers from the exponential time complexity of the backtrack search algorithm. This motivates us to develop another incremental technique based on stochastic local search. Indeed the underlying local search paradigm is well suited for recovering solutions after local changes (addition of constraints) of the problem occur. Also, the stochastic local search method is well suited for real time and over-constrained problems. Indeed, in the case where it is impossible or impractical to find a complete

[1] A binary CSP involves a list of variables defined on finite domains of values and a list of binary relations between variables.

[2] Note that this name and the corresponding acronym was used in [2]. A comparison of the approach proposed in this later paper and our model TemPro is described in [1].

solution, these techniques have the ability to provide a partial one with a quality proportional to the allocated time. The quality corresponds here to the number of solved constraints.

The rest of the paper is organized as follows : in the next section, we will present the notion of TCSPs in general and in the case of dynamic environment. The two methods for maintaining the global consistency of TCSPs in a dynamic environment are then presented respectively in sections 3 and 4. Section 5 is dedicated to the experimental evaluation on randomly generated TCSPs of the two methods we propose. Concluding remarks and possible perspectives of our work are then presented in section 6.

2 TCSPs AND DYNAMIC TCSPs

2.1 Temporal Constraint Satisfaction Problems (TCSPs)

We define a TCSP as :

- a list of temporal variables (events) defined on domains of possible values (numeric intervals) that each event can take,
- and a list of of binary temporal relations between variables defined as disjunctions of Allen primitives (see table 1 for the definition of the 13 Allen primitives).

Table 1: **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

Let us consider the following example[3].

Example 1

> *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**.

[3]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.

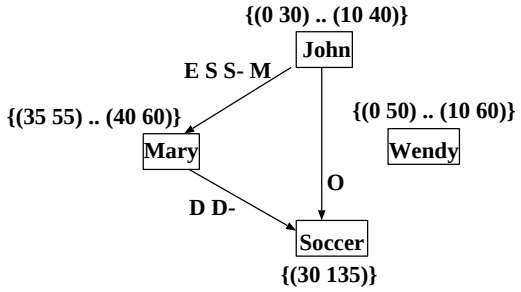

Figure 1: **A Temporal Constraint Satisfaction Problem.**

> *John left home* **between 7:00 and 7:10**. *Mary arrived at work* **between 7:55 and 8:00**. *Wendy left home* **between 7:00 and 7:10**. *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**.

Using our model TemPro [1], the above example is transformed to the TCSP represented by the graph in figure 1.

2.2 Dynamic Temporal Constraint Satisfaction Problems (DTCSPs)

A dynamic temporal constraint satisfaction problem (DTCSP) is a sequence of static TCSPs : $TCSP_0$, ..., $TCSP_i$, $TCSP_{i+1}$, ..., $TCSP_n$ each resulting from a change in the preceding one imposed by the "outside world". This change corresponds to a constraint restriction or relaxation. In this paper we will focus only on constraint restriction. More precisely, in the case of constraint restriction, $TCSP_{i+1}$ is obtained by performing a restriction on $TCSP_i$. We consider that $TCSP_0$ (initial TCSP) has an empty set of constraints. A restriction can be obtained by removing one or more Allen primitives from a given constraint. A particular case is when the initial 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.

3 DYNAMIC MAINTENANCE OF GLOBAL CONSISTENCY USING CONSTRAINT PROPAGATION

Given that we start from a consistent TCSP, the goal of the resolution method we present here consists of maintaining the global consistency (existence of a solution) anytime a new constraint is added (constraint restriction). Anytime a new constraint is added (disjunction of some Allen primitives), the method works as follows :

1. Compute the intersection of the new constraint with the corresponding constraint in the consistent graph.

   If the result of the intersection is not an empty relation **then**

   (a) Replace the current constraint of the graph by the result of the intersection.

   (b) **If** the new constraint is inconsistent with the current solution **then**

   i. Perform the numeric $\rightarrow$ symbolic conversion for the updated constraint. If the symbolic relation becomes empty then the new constraint cannot be added. The numeric $\rightarrow$ symbolic conversion works as follows : from the numeric information, we can extract the corresponding symbolic relation. An intersection of this relation with the given qualitative information will reduce the size of the latter which simplifies the size of the original problem. Although the exact algorithm that converts numeric to symbolic time information requires $O(e(Max(\frac{sup_i - inf_i - d_i}{s_i}))^2)$ in time where e is the number of qualitative constraints, we have defined a method that extracts most of the primitives within a relation between each pair of events in constant time reducing the complexity to $O(e)$. The method consists of using the information concerning the lower bound, upper bound and duration of the event temporal window instead of its occurrences. For example :

   - if $inf_i > sup_j$ then $e_i \ P^{\smile} \ e_j$,
   - if $d_i > d_j$ then E, S, F, D cannot belong to the relation between e_i and e_j,
   - ...etc.

   e_i and e_j are two events and inf_i, sup_j, d_i and d_j are respectively the earliest start time of e_i, latest end time of e_j, duration of e_i and duration of e_j.

   ii. Perform dynamic path consistency (*DPC*) in order to propagate the update of the constraint to the rest of the graph. If the resulting graph is not path consistent then the new constraint cannot be added.

   iii. Perform dynamic arc consistency (*DAC*) starting with the updated constraints. If the new graph is not arc consistent then the new constraint cannot be added.

   iv. Perform the backtrack search algorithm in order to look for a new solution to the problem. The backtrack search will start here from the point (resume point) it stopped in the previous search when it succeeded to find a complete assignment satisfying all the constraints. This way the part of the search space already explored in the previous searches will be avoided. The search will explore the rest of the search space. If a solution is found then the point where the backtrack search stopped is saved as new resume point and the new solution is returned. Otherwise the graph is inconsistent (when adding the new constraint). The new constraint cannot be added.

   Else the new constraint cannot be added otherwise it will violate the consistency of the graph.

DPC it the path consistency algorithm PC-2[5] we have adapted to handle constraint additions in an incremental way. *DAC* is the new arc consistency algorithm AC-3[6, 7] we have adapted for temporal constraints in a dynamic environment. A detailed description of *DAC* can be found in [8].

Example 2 :

The top right graph of figure 2 is the consistent TCSP obtained after performing our resolution method to the constraints of example 1 . Note that the constraint $(M^{\smile} \vee O^{\smile} \vee P^{\smile})$ between the soccer game and Wendy is an implicit constraint deduced after the numeric $\rightarrow$ symbolic conversion and the path consistency phases (same for the constraint between John and Wendy, and the constraint between Mary and Wendy). The numeric solution obtained is : $\{John : (5,35), Mary : (35,55), Soccer : (30,135), Wendy : (0,50)\}$. Let us assume now that we have the following constraint restrictions :

1. John either **started or arrived** just as Wendy **started**.
2. Mary and Wendy **arrived together but started at different times**.
3. Wendy **arrived** just as the soccer game **started**.

Figure 2 shows the application of our resolution method when adding each of the above first two constraints.

The first operation corresponds to the addition of the relation $S \vee S^{\smile} \vee M \vee E$ between John and Wendy. The intersection of this relation with the current constraint between the two events will lead to the relation S. After applying the dynamic arc, path consistency and backtrack search the new solution obtained is : $\{John : (5,35), Mary : (35,55), Soccer : (30,135), Wendy : (5,55)\}$.

The second operation corresponds to the addition of the relation $F \vee F^{\smile}$ between Mary and Wendy. The intersection of this relation with the current constraint between the two events will lead to the relation F. The relation F does not conflict with the global solution obtained so far. Thus the consistency of the graph is maintained.

The third operation corresponds to the addition of the relation P between Wendy and soccer game. The intersection of this relation with the current constraint between the two events will lead to an empty relation. Thus this third constraint cannot be added.

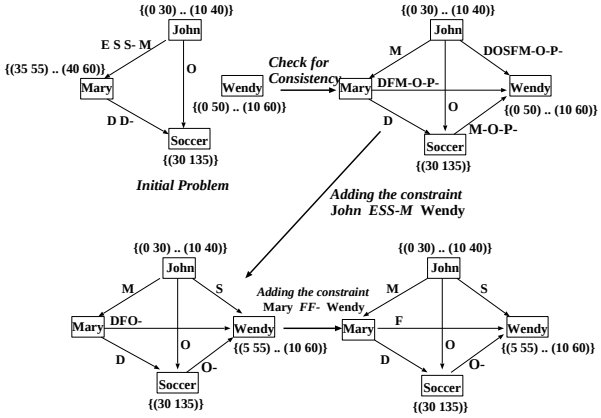

Figure 2: **Maintaining the global consistency of a TCSP**

4 DYNAMIC MAINTENANCE OF GLOBAL CONSISTENCY USING STOCHASTIC LOCAL SEARCH

In this section we present the way to solve dynamic TCSPs using stochastic local search methods. We will use the following terms:

State : one possible assignment of all events i.e set of couples (ev_i, occ_j), where ev_i is an event and occ_j is a possible interval belonging to the domain of ev_i; the number of states is equal to the product of domains sizes.

State or solution quality : the number of constraint violations of the state or the solution.

Neighbor : the state which is obtained from the current state by changing one event value.

Local-minimum : the state that is not a solution and the evaluation values of all of its neighbors are larger than or equal to the evaluation value of this state.

Strict local-minimum : the state that is not a solution and the quality of all of its neighbors are larger than the evaluation value of this state.

The three algorithms that we will consider in the following are based on a common idea known under the notion of local search. In local search, an initial configuration (assignment of events) is generated randomly and the algorithm moves from the current configuration to a neighborhood configurations until a complete solution or a good one has been found. When a new constraint is added, the algorithm restarts the search from the point corresponding to the last solution obtained, and iterates until a new solution respecting the old constraints and the new one is found. Note that this iterative algorithm can be interrupted at any time (case or real time problems) and provides a partial solution with a quality proportional to the allocated time. Also, in the case where a complete solution does not exist, the algorithm returns a partial one with the maximum of solved constraints.

4.1 Min-Conflict-Random-Walk method (MCRW)

After an initial configuration is randomly generated, 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 (numeric interval) 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 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 should be 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 Min Conflict 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 $O(N\ Max_{1\leq i\leq N}(\frac{sup_i - inf_i - d_i}{s_i}))$ where N is the number of variables and sup_i, inf_i, s_i and d_i are respectively the latest end time, earliest start time, duration and step of a given event evt_i. Max is the function that returns the maximum of a list of numbers.

4.2 Steepest-Descent-Random-Walk (SDRW)

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 $O(N^2 Max_{1\leq i\leq N}(\frac{sup_i - inf_i - d_i}{s_i}))$ in the worst case.

4.3 Tabu Search (TS)

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 $< event, intv >$ 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. $< event, intv >$ 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 $O(N^2 Max_{1 \leq i \leq N}(\frac{sup_i - inf_i - d_i}{s_i}))$ in the worst case.

5 EXPERIMENTATION

In order to evaluate the performance of the two methods we propose, we have performed experimental tests on randomly generated DTCSPs. The criteria used to evaluate the two different methods is the running time needed to maintain the global consistency of the DTCSP. The experiments are performed on a SUN SPARC Ultra 5 station. All the procedures are coded in C/C++.

5.1 Comparison Criteria

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

5.2 Generated Instances

Each generated problem is characterized by two parameters : N the number of events and $Horizon$ the parameter before which all events must be processed. In the following we will describe the generation of consistent and inconsistent problems.

Consistent problems of size N are those having at least one complete numeric solution (set of N numeric intervals satisfying all the constraints of the problem). Thus, to generate a consistent problem we first randomly generate a numeric solution and then add other numeric and symbolic information to it. More precisely the generation is performed using the following steps.

1. Generation of the numeric solution Randomly pick N 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.

2. Generation of the numeric constraints 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.

3. Generation of the symbolic constraints Compute the basic Allen primitives that can hold between each interval pair of the initial solution. Add to each relation a random number in the interval $[0,Nr]$ $(1 \leq Nr \leq 13)$ of chosen Allen primitives.

Example 3 :

Let us assume we want to generate a consistent problem with $N = 3$ and $Horizon = 10$.

1. First a numeric solution is generated : S = {(1 4), (2 8), (5 7)}.

2. Numeric constraints (SOPOs) are then randomly generated from the numeric solution.

Numeric Interval		Corresponding SOPO
(1 4)	→	[2 9]
(2 8)	→	[2 10]
(5 7)	→	[3 8]

3. The Allen primitives are then computed from the pairs of intervals of the numeric solution :

(1 4) *and* (2 8)	→	Overlaps (O)
(1 4) *and* (5 7)	→	Overlaps (O)
(2 8) *and* (5 7)	→	During inverse ($D^{\smile}$)

and finally the symbolic constraints are generated from the above Allen primitives.

O	→	POM
O	→	$DD^{\smile}EO$
$D^{\smile}$	→	$FSDD^{\smile}PE$

Each inconsistent problem of size N (N is the number of variables) is generated using the following steps.

1. Generation of numeric constraints Randomly pick N pairs of ordered values (x,y) such that $x,y \in [0,\ldots,Horizon]$. x and y are respectively considered the earliest start time and the latest end time of a given event. For each pair of value (x,y), randomly pick a number $d \in [1\ldots y-x]$. d is considered the duration of the event.

2. Generation of symbolic constraints Randomly generate C constraints between the N events where $C \in [1\ldots\frac{N(N-1)}{2}]$ ($C = \frac{N(N-1)}{2}$ in the case of a complete graph of constraints). Each constraint C is a disjunction of a random number Nb ($Nb \in [1\ldots13]$) of relations chosen randomly from the set of the 13 Allen primitives.

3. Consistency check of the generated problem Perform a backtrack search method on the generated problem. If a solution is found **goto 1** otherwise the problem is inconsistent.

Table 2: **Comparative tests on randomly generated DTC-SPs.**

N	C	Constraint Propagation	MCRW
20	95	0.10	0.11
40	390	0.35	0.22
60	885	1.02	0.79
80	1580	2.58	1.24
100	2475	6.10	1.89
200	9950	28	2.23

The generated problems are characterized by their tightness, which can be measured, as shown in [9], by 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 $Horizon$ (time before which all tasks should be processed), Nr (the maximal number of Allen primitives per symbolic constraint) and the density of the problem ($\frac{2C}{N(N-1)}$ where C is the number of constraints of the problem).

5.3 Results

After a random consistent TCSP is generated, each of the two resolution techniques will process the list of temporal relations in an incremental way. Preliminary experiments comparing the stochastic local search methods we have seen in section 4, namely MCRW, SDRW and Tabu have demonstrated the efficiency of the MCRW over the other two methods to deal with dynamic TCSPs. This is justified by the fact that, as we mentioned in section 4, the cost in time of a move, from one state to another, 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).

Table 2 presents the results of tests (time in seconds) performed on randomly generated TCSPs defined by the number of variables N and the number of constraints C. As we can easily see, the approximation method is faster than the exact one. This is justified by the fact that the systematic method is based on a backtracking algorithm with an exponential time cost while the two approximation methods are based polynomial time iterative algorithms. However, as we said in introduction, the approximation method does not guarantee in general the completeness of the solution provided at each time.

6 CONCLUSION AND FUTURE WORK

In this paper we have presented two different ways for maintaining in a dynamic environment, the global consistency of a temporal constraint satisfaction problem. The methods are of interest for any application where qualitative and numeric temporal information should be managed in an evolutive environment. This can be the case of real world applications such as reactive scheduling and planning where any new information corresponding to a constraint restriction should be handled in an efficient way. Although the approximation method does not guarantee the completeness of the solution returned, it is of interest when an answer needs to be returned within a given deadline (case of real time problems) and also when a complete solution does not exist (case of over constrained problems). In both cases, a solution maximizing the number of constraints to be satisfied is returned.

One perspective of our work is to handle the relaxation of constraints during the resolution process. For example, suppose that during the search a given constraint is removed. Would it be worthwhile to find those values removed previously because of this constraint and to put them back in the search space or would it be more costly than just continuing on with search without considering these values. Another question to consider in the case of the approximation methods is whether, when processing a list of new constraints in an incremental way, is it better to start search with the best solution found for the previous problem or to start with a random initial assignment.

References

[1] Mouhoub, M., Charpillet, F., Haton, J.: Experimental Analysis of Numeric and Symbolic Constraint Satisfaction Techniques for Temporal Reasoning. Constraints: An International Journal **2** (1998) 151–164, Kluwer Academic Publishers

[2] Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991) 61–95

[3] Meiri, I.: Combining qualitative and quantitative constraints in temporal reasoning. Artificial Intelligence **87** (1996) 343–385

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

[5] van Beek, P., Manchak, D.W.: The design and experimental analysis of algorithms for temporal reasoning. Journal of Artificial Intelligence Research **4** (1996) 1–18

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

[7] Bessière, C., Régin, J.C.: Refining the basic constraint propagation algorithm. In: Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, WA (2001) 309–315

[8] Mouhoub, M., Yip, J.: Dynamic CSPs for Interval-based Temporal Reasoning. In: Fifteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-2002), Cairns, Australia (2002) To appear.

[9] Sabin, D., Freuder, E.C.: Contradicting conventional wisdom in constraint satisfaction. In: Proc. 11th ECAI, Amsterdam, Holland (1994) 125–129