Constraint-based Strategies for the Disjunctive Temporal Problem: Some New Results

Angelo Oddi

IP-CNR, Italian National Research Council
Viale Marx 15, I-00137 Rome, Italy
oddi@ip.rm.cnr.it

Abstract. The Disjunctive Temporal Problem (DTP) involves the satisfaction of a set of constraints represented by disjunctive formulas of the form $x_1 - y_1 \leq r_1 \vee x_2 - y_2 \leq r_2 \vee \ldots \vee x_k - y_k \leq r_k$. DTP is a quite general temporal reasoning problem which includes the well-known Temporal Constraint Satisfaction Problem (TCSP) introduced by Dechter, Meiri and Pearl. This paper describes a basic constraint satisfaction algorithm where several aspects of the current literature are integrated, in particular the so-called *incremental forward checking*. Hence, two new extended solving strategies are proposed and experimentally evaluated. The new results are both very competitive with respect to the current best results and open further research directions that concerns, in particular, the use of *arc-consistency* filtering strategies.

Keywords: constraint reasoning for planning and scheduling, temporal reasoning, constraint algorithms.

1 Introduction

In many recent Artificial Intelligence applications the need of a more expressive temporal reasoning frame is increasing more and more. For example, in continual planning applications [1] a relevant capability is the continuous management of temporal plans [2, 3]. To this purpose, the representation of temporal disjunction allows a leverage of systems' capability, for example, it avoids a too early commitment on action orderings. Other interesting applications of the temporal model proposed in this paper are discussed in the work [4], these applications range from scheduling and planning to temporal database with indefinite information.

The Temporal Constraint Satisfaction Problem (TCSP) [5] is a way to represent temporal disjuctions, it allows constraints of the form $x-y \leq r_1 \vee x-y \leq r_2 \vee \ldots \vee x-y \leq r_k$. A further generalization of the TCSP is proposed in [6, 4] where a problem has constraints of the form $x_1-y_1 \leq r_1 \vee x_2-y_2 \leq r_2 \vee \ldots \vee x_k-y_k \leq r_k$. In [7] this last problem is referred to as Disjunctive Temporal Problem (DTP) and we use this name in the paper. DTPs have been studied in several previous works: (a) in [6, 4] several constraint-based (CSP) algorithms (in the line of [8]) are defined and experimentally compared. One of them, based on *forward checking* [9], is shown to be the best; (b) in [7] DTP is modeled as a propositional satisfability (SAT [10]) problem and solved with a state-of-the-art

SAT-solver plus some additional processing. Experiments show an improvement of up to two orders of magnitude with respect to the results in [6]. (c) finally in [11] the constraint-based algorithm proposed in [6, 4] is improved exploiting the quantitative temporal information in the solution "distance graph". Using this knowledge an incremental version of the forward checking is obtained and shown to be competitive with the results proposed in [7].

Our starting point in the work [11] was the observation of the sharp difference between the results shown in [6] and [7] and the idea of using the "quantitative reasoning" that can come out from a temporal constraint network representation. We basically observe that the effects of quantitative temporal information to improve global performance of DTPs has not been explored enough in previous works. Using such knowledge we were able to define an *incremental forward checking* algorithm which has comparable performance (measured as number of forward checks) with the best SAT-based version proposed in [7].

This paper follows the same CSP-based approach and again focus on using "quantitative reasoning" that can come out from a temporal constraint network representation. In particular we propose a new heuristic strategy for *variable ordering* used in our CSP framework and an *arc-consistency* filtering algorithm. The rationale behind the new results is quite general and can be exploited in other solvers that rely on "quantitative temporal reasoning".

The paper is structured as follows. Section 2 introduces the basic concepts used in the paper. Section 3 gives a basic CSP algorithm which integrates results from both previous works, included the *incremental forward checking*. Section 4 introduces two additional solving algorithms for solving DTP instances, and an experimental evaluation of the different approaches is given in Section 5. Section 6 ends the paper with some conclusions.

2 Preliminaries

The Disjunctive Temporal Problem (DTP) involves a finite set of temporal variables $x_1, y_1, x_2, y_2 \ldots x_n, y_n$ ranging over the reals and a finite set of constraints $C = \{c_1, c_2, \ldots c_m\}$ of the form $x_1 - y_1 \leq r_1 \vee x_2 - y_2 \leq r_2 \vee \ldots \vee x_k - y_k \leq r_k$, where r_i are real numbers. A DTP is consistent if an assignment to the variables exists such that in each constraints $c_i \in C$ at least one disjunct $x_{ij} - y_{ij} \leq r_{ij}$ is satisfied. One way to check for consistency of a DTP consists of choosing one disjunct for each constraint c_i and see if the conjunction of the chosen disjuncts is consistent. It is worth observing that this is equivalent to extracting a "particular" STP (the Simple Temporal Problem defined in [5]) from the DTP and checking consistency of such a STP. If the STP is not consistent another one is selected, and so on. Both previous approaches to DTP [6, 4, 7, 11] do this basic search step.

Previous Work. All [6, 4, 7, 11] share a "two layered" algorithmic structure. An upper layer of reasoning is responsible for guiding the search that extracts the set of disjuncts, a lower layer represents the quantitative information of the temporal reasoning problem. In [6] a general CSP formulation is used at the upper level while the quantitative information is managed by using the incremental di-

rectional path consistency (IDPC) algorithm of [12]. In [7] at the upper level the DTP is encoded as a SAT problem, a SAT-solver extracts an STP to be checked, a simplified version of the Simplex algorithm is used at the lower level to check for its consistency. Stergiou and Kubarakis define different backtracking algorithms for managing the upper-level and experimentally verify that the version using forward checking is the best. Forward checking is used after each choice to test which of the possible next choices are compatible with current partial STP. In the rest of the paper their best algorithm is called SK. Armando, Castellini and Giunchiglia focus their attention on the SAT encoding, each disjunct is a propositional formula, and they use a state of the art SAT-solver enriched with a form of forward checking biased by the temporal information. Their basic version is called $TSAT$ and is shown to improve up to one order of magnitude with regard to SK. Then they add a further preprocessing step called IS that basically produces a more accurate SAT encoding because it codifies mutual exclusion conditions among propositions that exists in the temporal information, but were lost by the first standard encoding.

DTP Consistency Checking as a Meta-CSP. Before introducing our algorithm we underscore the possibility of representing the consistency checking problem as a *meta*-CSP problem, where each DTP constraint $c \in C$ represents a (meta) variable and the set of disjuncts represents variable's domain values $D_c = \{\delta_1, \delta_2, \ldots \delta_k\}$. A *meta*-CSP problem is consistent if exists at least an element S (solution) of the set $D_1 \times D_2 \times \ldots \times D_m$ such that the corresponding set of disjuncts $S = \{\delta_1, \delta_2, \ldots \delta_m\}$ $\delta_i \in D_i$ is temporally consistent.

Each value $\delta_i \in D_i$ represents an inequality of the form $x_i - y_i \leq r_i$ and a solution S can be represented as a labeled graph $G_d(V_S, E_S)$ called "distance graph" [5]. The set of nodes V_S coincides with the set of DTP variables $x_1, y_1, x_2, y_2 \ldots x_n, y_n$ and each disjunct $x_i - y_i \leq r_i$ is represented by a direct edge (y_i, x_i) from y_i to x_i labeled with r_i. A path from a node x_i to y_j on the graph is a set of contiguous edges $(x_i, y_i), (y_i, y_{i1}), (y_{i1}, y_{i2}), \ldots, (x_{il}, y_j)$ and the length of the path is the sum of the edges' labels. The set of disjuncts S corresponds to an STP. S is a solution to the *meta*-CSP problem if G_d does not contain closed path with negative length (*negative cycles*) [5]. From the graph G_d a numerical solution of the problem can be extracted as follows. Let $d_{x_i y_i}$ be the shortest path distance on G_d from the node x_i to y_i, without loss of generality we can assume a variable x_i as reference point, for example x_1, in this way the tuple $(d_{x_1 x_1}, d_{x_1 x_2}, \ldots d_{x_1 x_n})$ is a solution of the original DTP problem. In fact, the previous values represent the shortest distance from the reference node x_1 to all the other ones (in particular $d_{x_1 x_1} = 0$). For each edge $x_i - y_i \leq r_i$ in G_d as it is well known values $(d_{x_1 x_1}, d_{x_1 x_2}, \ldots d_{x_1 x_n})$ must hold the Bellman's inequalities: $d_{x_1 x_i} \leq d_{x_1 y_i} + r_i$, that is $d_{x_1 x_i} - d_{x_1 y_i} \leq r_i$. Hence $(d_{x_1 x_1}, d_{x_1 x_2}, \ldots d_{x_1 x_n})$ is a solution for the DTP.

This view of the consistency checking problem is used to define our CSP approach and in particular is useful to understand our incremental forward checking method.

```
CSP-DTP-SOLVER(dtp, S)
1.  if CheckConsistency(dtp)
2.     then if IsaSolution(dtp)
3.             then return(S)
4.             else begin
5.               c ← SelectVariable(dtp)
6.               δ ← Choose Value(dtp, c)
7.               CSP-DTP-SOLVER(dtp, S ∪ {δ})
8.             end
9.     else return(Fail)
10. end
```

Fig. 1. A CSP solver for the DTP.

3 A CSP Algorithm for DTP

In this work we mainly follow the constraint-based approach of Stergiu and Kubarakis [6, 4] for solving DTP instances. Figure 1 shows a CSP procedure which starts from an empty solution S and basically executes three steps: (a) the current partial solution is checked for consistency (Step 1) by the function *Check-Consistency*. This function filters also the search space from inconsistent states. If the partial solution is a complete solution (Step 2) the algorithm exits. If the solution is still incomplete the following two steps are executed; (b) a (meta) variable (a constraint c_i) is selected at Step 5 by a variable ordering heuristic; (c) a disjunct $x_i - y_i \leq r_i$ is chosen (Step 6) from the domain variable D_i and added to S (represented at the lower level as a G_d graph). Hence the solver is recursively called on the partial updated solution $S \cup \{\delta\}$.

The *CheckConsistency* function is the core of the CSP algorithm, it both updates the set of distances $d_{x_i y_j}$ and the domain variables D_i by forward checking. In particular it executes two main steps:

Temporal propagation. every time a new inequality $x_i - y_i \leq r_i$ is added to the G_d graph, the set of distances $d_{x_i x_j}$ is updated by a simple $O(n^2)$ algorithm.

Forward checking. After the previous step, for each not assigned meta-variable the domain D_i is checked for consistency (forward checking). Given the current solution represented by G_d, each value $x_i - y_i \leq r_i$ belonging to a not assigned variable and which induces a negative cycles on G_d is removed. In other words, each time a value $\delta_i \equiv x_i - y_i \leq r_i$ satisfies the test $r_i + d_{x_i y_i} < 0$, then δ_i is removed from the corresponding domain D_i. In the case that one domain D_i becomes empty, the function *CheckConsistency* returns *false*.

The *CheckConsistency* step contributes to avoid investigation of search states proved inconsistent and the other two steps (Steps 5 and 6 of Figure 1) are used to guide the search according to heuristic estimators.

SelectVariable. It applies the simple and effective Minimum Remaining Values (MRV) heuristic: variables with the minimum number of values are selected first. It is worth noting that the heuristic just ranks the possible choices

deciding which one to do first but all the choices should be done (it is not a non deterministic search step).

ChooseValue. This represents a non deterministic operator which starts a different computation for each domain values. Obviously in our implementation we use a *depth-first* search strategy, where there is no particular values ordering heuristic. However, in the case a constraint (variable) is always satisfied by the current partial solution S_p, that is, a constraint disjunct $x_i - y_i \leq r_i$ exists such that holds the condition $d_{y_i x_i} < r_i$, no branching is created. In fact, the current constraint is implicitly "contained" in the partial solution and it will be satisfied in all the solution created from S_p.

3.1 Integrating SAT Features

The current version of our CSP solver integrates also the so-called *semantic branching* [7]. This is a feature that in the SAT approach comes for free and that in the CSP temporal representation is to be explicitly inserted. It avoids to test again certain conditions previously proved inconsistent. The idea behind semantic branching is the following, let us suppose that the algorithm builds a partial solution $S_p = \{\delta_1, \delta_2, \ldots \delta_p\}$ and a not assigned meta-variable is selected which has a disjunct set of two elements $\{\delta^{'}, \delta^{''}\}$. Let us suppose that the disjunct $\delta^{'}$ is selected first and no feasible solution exists from the partial solution $S_p \cup \{\delta^{'}\}$. In other words, each search path from the node $S_p \cup \{\delta^{'}\}$ arrives to an infeasible state. In this case the depth-first search process removes the decision $\delta^{'}$ from the current solution and tries the other one ($\delta^{''}$). However, even if the previous computation is not able to find a solution, it demonstrates that with regard to the partial solution S_p no solution can contain the disjunct $\delta^{'}$. If we simply try $\delta^{''}$ we lose the previous information, hence, before trying $\delta^{''}$, we add the condition $\neg\delta^{'}$ (that is $x^{'} - y^{'} > r_i$) to the partial solution. It is worth nothing that in this case it is important to make explicit the semantic branching by adding the negation because the values in the domains D_i are not self-exclusive. In other cases, for example a scheduling problem, where branching is done with regard to the temporal ordering of pairs of activities A and B, semantic branching is not useful. In fact when $A\ before\ B$ is chosen the case $B\ before\ A$ is implicitly excluded.

In this section we have described our basic algorithm that integrates some of the previous analysis in a meta-CSP search framework. From now on we call this algorithm CSP and it is the base for the description of the incremental forward checking of the next section.

3.2 Incremental Forward Checking

The algorithms for solving DTP introduced at the beginning of this section is based on the *meta*-CSP schema with some additional features. In particular, it uses the enriched backtracking schema called *semantic branching*. To further improve the performance of the CSP approach we have investigated aspects connected to the quantitative temporal information. This aspect has received less attention in [6, 7, 4]. In particular in this section we introduce a method to

significantly decrease the number of forward checks by using the temporal information. Its general idea is relatively simple.

Rationale. When a new disjunct δ is added to a partial solution, the temporal propagation algorithm inside *CheckConsistency* updates only a subset of the distances $d_{x_i x_j}$ (usually a "small" subset). The forward checking test on disjuncts is performed w.r.t. the distances in the graph G_d. It is of no use to perform a forward checking test of the form $d_{x_i y_i} + r_i < 0$ on a disjunct δ_i when the distance $d_{x_i y_i}$ has not been changed w.r.t. the previous state.

This basic observation can be nicely integrated in CSP with the additional cost of a static preprocessing needed to create for each pair of nodes $\langle x_i, y_j \rangle$ the set of *affected meta values* $AMV(x_i, y_j)$.

Affected meta-values w.r.t. a pair $\langle x_i, y_j \rangle$. Given a distance $d_{x_i y_j}$ on G_d the set of *affected meta values* discriminates which subset of disjuncts are affected by an update of $d_{x_i y_j}$. The set $AMV(x_i, y_j)$ associated to the distance $d_{x_i y_j}$ (or the pair $\langle x_i, y_j \rangle$) is defined as the set of disjuncts $x - y \leq r$ whose temporal variables x and y respectively coincide with the variables y_j and x_i ($AMV(x_i, y_j) \doteq \{x - y \leq r : x = y_j, y = x_i\}$).

Given a DTP, the set of its AMVs is computed once for all with a preprocessing step with a space complexity $O(m + n^2)$ and a time complexity $O(n^3 \ln n)$ (as explained below each set AMV is represented as a sorted list according to the values r). The information stored in the AMVs can be used in a new version of CSP we call "incremental forward checking" ($CSPi$). It requires a modification of the *CheckConsistency* function. The new incremental version of the *CheckConsistency* works in two main steps:

1. The distances $d_{x_i y_j}$ are updated and the set of distances that have been changed is collected.
2. given such set, for each $d_{x_i y_j}$ the corresponding $AMV(x_i, y_j)$ is taken, and its values are forward checked. In particular, all the set $AMV(x_i, y_j)$ are represented as a list of disjuncts sorted according to the value of r and the forward checking test $d_{x_i y_j} + r < 0$ is performed from the disjunct with the smallest value of r. In this way, when a test fails on the list element δ, it will fail also on the rest of the list and the forward checking procedure can stop on $AMV(x_i, y_j)$.

In the experimental section we show that the algorithm $CSPi$ (constraint-based solver with incremental forward checking) strongly improves with respect to the basic CSP and becomes competitive with the best results available in the literature.

4 New Constraint-based Solving Strategies for DTP

In this section we propose two additional solving strategies for DTP based on the work [11]. In particular we propose: (1) a new *variable ordering* heuristic; (2) an *arc-consistency* filtering strategy.

The rational behind the first method is based on the observation that given a DTP problem, and considerd a value $\delta_i \equiv x_i - y_i \leq r_i$, during the solving process δ_i is removed by forward checking from its domain D_i when induces negative cycles in the current solution represented by the G_d graph. On the basis of the previous observation we propose the following variable ordering strategy: *select the subset of variables with minimum number of remaining values $x_i - y_i \leq r_i$ and within this subset, the variable with maximal number of negative coefficients r_i.* The values δ_i with negative coefficients r_i are crucial to the existence of a solution to a DTP. In fact, it is simple to see that a DTP instance without negative r_i values has always a solution. On the other hand, the presence of negative r_i values generate negative cycles on the graph G_d and induces inconsistent partial solutions. This strategy has the main purpose of pruning the search tree in its early stages, trying to create as many as possible negative cycles, in this way the strategy maximizes the probability of finding negative cycles at the early steps of the search tree. As we will see in the experimental section this strategy is effective in the *transistion phase* of a DTP problem where the probability of find a solution is very low.

The second solving method can be explained by giving a new version of the *CheckConsistency* algorithm used in the general algorithmic template described in Figure 1. The aim of this solving method is reducing the dimension of the search tree by the application of a more effective filtering strategy and to explore the possibility of finding tradeoffs among number of consistency checks, number of visited search nodes, and CPU time. In particular, we propose an *arc-consistency* filtering algorithm such that, among the set of filtering methods analyzed during our experimentation, is the one which gave the better performance both in CPU time and number of consistency checks. The proposed filtering algorithm works in two main steps.

1. It applies the incremental forward checking method described in Section 3.2. When at least one variable domain becames empty, *CheckConsistency* returns *false*, otherwise the following second step is executed.
2. The set of not assigned variables which are modified by the application of the first step is considered, and used to inizialize the propapagation queue Q of an *arc-consistency* filtering method. The filtering method is executed to remove further values, in the case at least one variable domain becames empty, *CheckConsistency* returns *false*, otherwise returns *true*.

Figure 2 shows the *arc-consistency* filtering algorithm. It takes as an input the set Q_{init} of modified variables and applies the 2-consistency filtering startegy by the *Revise* operator which is the core of the method. In this case the operator has the following definition: Revise(c_i, c_j) removes from the domains D_{c_i} and D_{c_j} each value $x_i - y_i \leq r_i$ which does not have *support*. That is, a value $x_i - y_i \leq r_i$ is removed from the domain D_{c_i} when there is no value $x_j - y_j \leq r_j$ in the set D_{c_j} such that $r_i + d_{x_i y_j} + r_j + d_{x_j y_i} \geq 0$ holds. When the procedure stops, it returns the set of variables with reduced domain of values.

In the experimental section we compare this strategy with the other ones, trying to find some conclusions about the relations among the number of consistency checks (we consider the test $r_i + d_{x_i y_j} + r_j + d_{x_j y_i} \geq 0$ performed inside

Arc-consistency(Q_{init})
1. $Q \leftarrow Q_{init}$
2. **while** ($Q \neq \emptyset$ **and** $\nexists D_{c_i} = \emptyset$) **do begin**
3. $c_i \leftarrow$ Pop(Q)
4. **foreach** *not assigned variable* $c_j \in C$ **do**
5. $Q \leftarrow Q \cup$ Revise(c_i, c_j)
6. **end**

Fig. 2. Arc-consistency filtering algorithm.

the *Revise* operator as equivalent to a forward checking test) the total CPU time and the number of visited search nodes.

5 Experimental Evaluation

We adopt the same evaluation procedure used in [6, 7] and use the random DTP generator defined by Stergiou. DTP instances are generated according to the parameters $\langle k, n, m, L\rangle$ (k: number of disjuncts per clause, n: number of variables, m: number of disjunction (temporal constraints); L: a positive integer such that all the constants r_i are sampled in the interval $[-L, L]$). In particular, according to [6, 7] experimental sets are generated with $k = 2$, $L = 100$ and the domain of r_i is on integers not on reals as in the general definition of DTP.

Experimental results are plotted for $n \in \{10, 12, 15, 20, 25, 30\}$, where each curve represents the number of consistency checks versus the ratio $\rho = m/n$ (in both the results of Figures 3 and 4 $\rho = m/n$ is an integer value which ranges from 2 to 14). The median number of checks over 100 random samples for different values of ρ is plotted in Figures 3(a)-3(f) where three different type of results are compared: (1) the performance of the best algorithm proposed in [6] and labeled with SK; (2) the results of the SAT-based solving methods, there are two methods: the first one labeled with $TSAT_{IS(2)}$, which corresponds to the best results claimed in the work [7], and a second one, labeled with $TSAT_{IS(3)}$, which represents some new results only published on the $TSAT$ web page (see the reference [7] for the URL); (3) the performance of our constraint-based approach, in particular the curve labeled with $CSPi$ corresponds to the best results in the paper [11], and the one labeled with $CSPineg$ represent the new results obtained with the heuristic strategy definited in Section 4. Figure 4(d) plots the percentage of problems solvable by $CSPineg$ on different n. The algorithms are implemented in Common Lisp and the reported results are obtained on a SUN UltraSparc 10 (440MHz). All the results are obtained setting a timeout of 1000 seconds of CPU time.

There are several comments on the $CSPineg$ performance: (a) all the curves have the same behavior of the previous results. It is confirmed that the harder instances are obtained for $\rho \in \{6, 7\}$ and for such values the percentage of solvable problems becomes $< 10\%$. When the number of variables n increases the hardest region narrows; (b) the median number of forward checks show that $CSPineg$ significantly improves over $CSPi$. This fact shows that the new selection variable strategy is very effective, and indirectly confirms that there could be further

space for investigating improvements of the CSP approach; (c) the $CSPineg$ compares very well with the pre-existing approaches, it outperforms the others for $n \in \{10, 12, 15, 20\}$ and it is competitive with $TSAT_{IS(3)}$ for $n \in \{25, 30\}$. However, further work will be needed to clearly outperform $TSAT_{IS(3)}$ on all n. One possible direction of research is the use of more effective filtering strategies to reduce the dimension of the search tree. However, the use of a more powerful filtering strategy has a price of an higher computational time. Hence, the real problem is find a good tradeoff among number of consistency checks, number of search nodes and CPU time.

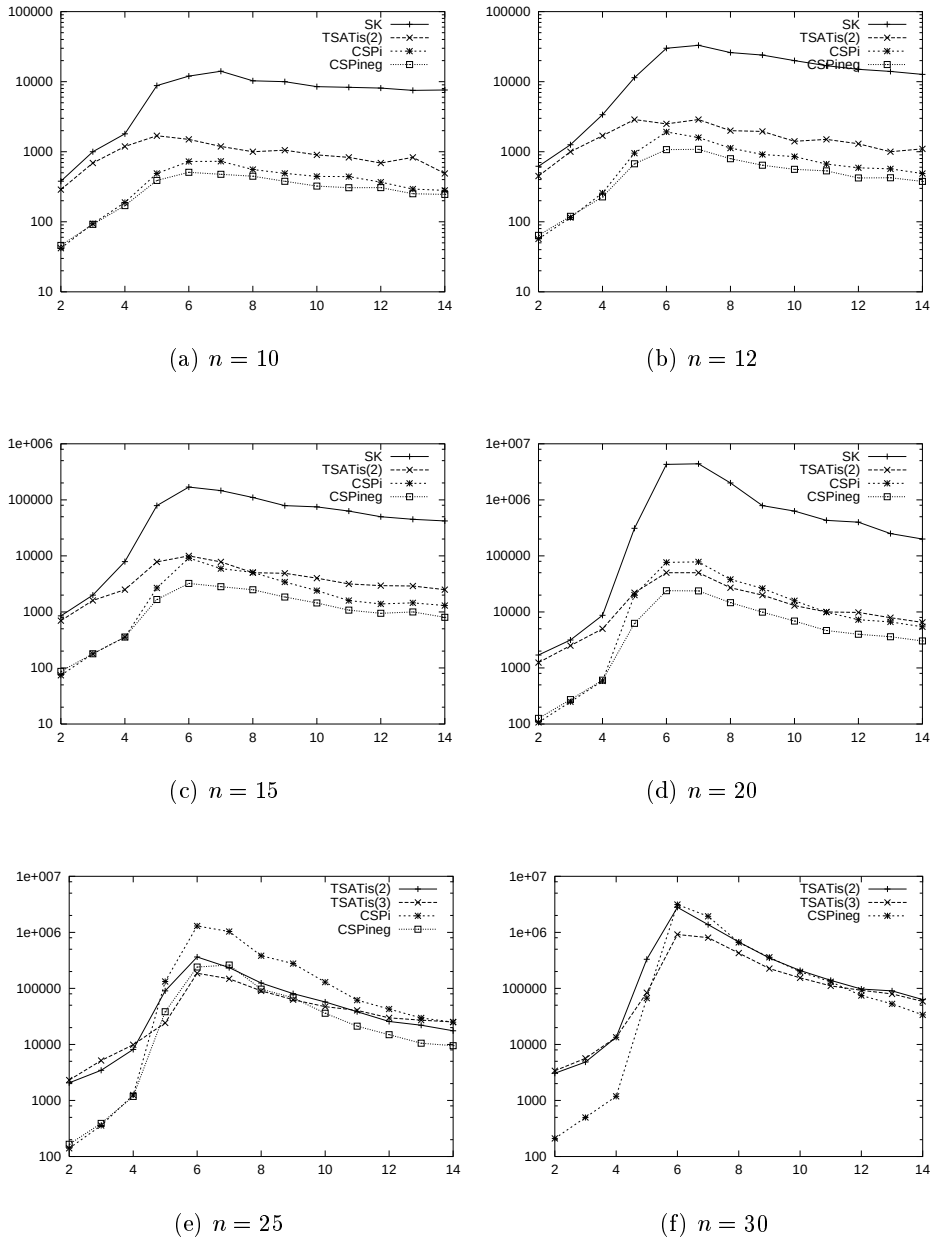
Figure 4(a) shows a comparison between the performance of our constraint-based algorithm $CSPineg$ and the other one which uses the arc-consistency filtering strategy (labeled with $CSPiac$) introduced in Section 4. With respect to number of forward checks $CSPiac$ performs about one order of magnitute worse than $CSPineg$, where in the case of the arc-consistency algorithm we consider the test $r_i + d_{x_i y_j} + r_j + d_{x_j y_i} \geq 0$ as equivalent to a forward check. On the other hand, if we consider the CPU time performance (Figure 4(c)), the ratio between the $CSPiac$ and $CSPineg$ CPU times is less than 3 in the transition phase. The analysis is completed by the results in Figure 4(d), which show that the $CSPiac$ strategy is able to reduce about 25% the number of search nodes respect to the $CSPineg$ performance.

About the results of Figure 4 we have the following observations: (a) the arc-consistency strategy performs an higher number of consistency checks respect to the forward checking strategy and many of the performed checks are unnecessary, in fact, after each solution modifications, many distances on the G_d graph remain unchanged, hence many tests of the form $r_i + d_{x_i y_j} + r_j + d_{x_j y_i} \geq 0$ are unnecessarily performed; (b) in our approch a consistency check has $O(1)$ time complexity (in the TSAT approach is at least $O(n)$) and this explain the difference in performance between number of consistency checks and CPU time shows in Figure 4.

The experimental results confirm that the CSP approch contains good ideas, in fact our results are comparable with the ones obtained by the TSAT approach which uses one of the best SAT-solver available, in addition, for lower values of the ratio $\rho = m/n$ (≤ 5) the $CSPineg$ is significantly better with respect to all the others (it is to be noted also that in many practical applications the condition $\rho \leq 5$ is likely to be verified). On the other hand, further investigation is needed to realize a competitive arc-consistency solving algorithm, in this experimentation some useful observations about tradeoffs among number of forward checks, number of search nodes explored, and CPU time are pointed out, and represent a good starting point for future research directions.

6 Conclusion

This paper has extended the constraint-based approach, initially introduced in [6] and later improved in [11], to solving the DTP temporal problem. As it is pointed out in the short discussion at the begining of the paper, DTP is going to become very relevant in many planning application. We propose two new additional solving methods for DTP. The first one is an heuristic strategy for

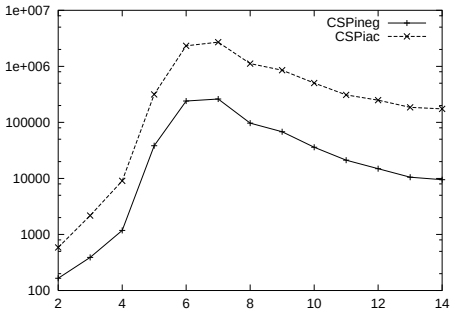


(a) $n = 10$ (b) $n = 12$

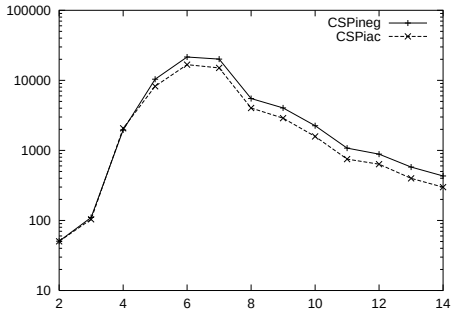
(c) $n = 15$ (d) $n = 20$

(e) $n = 25$ (f) $n = 30$

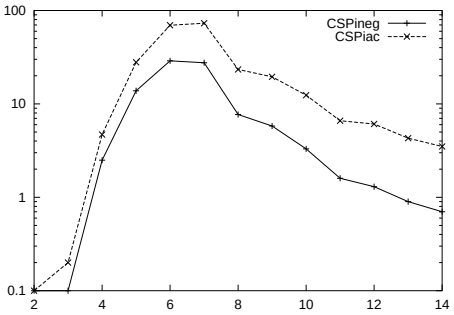
Fig. 3. Median number of forward checks for $n \in \{10, 12, 15, 20, 25, 30\}$.



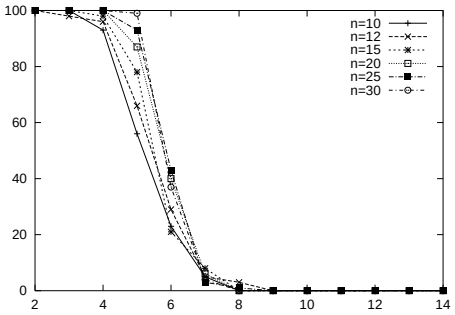
(a) Median number of forward checks for $n = 25$



(b) Average number of search nodes for $n = 25$



(c) Average CPU time in seconds for $n = 25$



(d) Percentage of solvable problems

Fig. 4. Other experimental results.

variable ordering which improves forward checking performance up to an order of magnitude respect to the results claimed in [11] and allowing a real competition with the best SAT approach. An interesting area where $CSPineg$ constantly outperforms all other approaches (when $\rho \leq 5$) emerges from an experimental evaluation. The second solving method uses a more sophisticated *arc-consistency* filtering algorithm. In this case the aim of the method is reducing the dimension of the search tree by the application of a more effective filtering strategy and to explore the possibility of finding tradeoffs among number of consistency checks, number of visited search nodes, and CPU time. The results proposed in the paper suggest that an useful research direction is the definition of an incremental version of the arc-consistency filtering algorithm.

Acknowledgments

This work is supported by ASI (Italian Space Agency) under ASI-ARS-99-96 contract and by the Italian National Research Council.

References

[1] DesJardin, M., Durfee, E., Ortiz, C., Wolverton, M.: A Survey of Research in Distributed, Continual Planning. AI Magazine **20** (1999) 13–22
[2] Pollack, M., Horty, J.: There's More to Life Than Making Plans: Plan Management in Dynamic Multiagent Environment. AI Magazine **20** (1999) 71–83
[3] Tsamardinos, I., Pollack, M.E., Horty, J.F.: Merging Plans with Quantitative Temporal Constraints, Temporally Extended Actions, and Conditional Branches. In: Proceedings of the 5th International Conference on AI Planning Systems (AIPS-2000. (2000)
[4] Stergiou, K., Koubarakis, M.: Backtracking Algorithms for Disjunctions of Temporal Constraints. Artificial Intelligence **120** (2000) 81–117
[5] Dechter, R., Meiri, I., Pearl, J.: Temporal Constraint Networks. Artificial Intelligence **49** (1991) 61–95
[6] Stergiou, K., Koubarakis, M.: Backtracking Algorithms for Disjunctions of Temporal Constraints. In: Proceedings 15th National Conference on AI (AAAI-98). (1998)
[7] Armando, A., Castellini, C., Giunchiglia, E.: SAT-based Procedures for Temporal Reasoning. In: Proceedings 5th European Conference on Planning (ECP-99). (1999) (available at `http://www.mrg.dist.unige.it/~drwho/Tsat`).
[8] Prosser, P.: Hybrid Algorithms for the Constraint Satisfaction Problem. Computational Intelligence **9** (1993) 268–299
[9] Haralick, R., Elliott, G.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems. Artificial Intelligence **14** (1980) 263–313
[10] Cook, S., Mitchell, D.: Finding Hard Instances of the Satisfiability Problem: a Survey. In: Satisfiability Problems: Theory and Applications. DIMACS Series in Discrete Mathematics and Computer Science N.35 (1998)
[11] Oddi, A., Cesta, A.: Incremental Forward Checking for the Disjunctive Temporal Problem. In Horn, W., ed.: ECAI2000. 14th European Conference on Artificial Intelligence, IOS Press (2000) 108–111
[12] Chleq, N.: Efficient Algorithms for Networks of Quantitative Temporal Constraints. In: Proceedings of the Workshop CONSTRAINTS'95 (held in conjunction with FLAIRS-95). (1995) 40–45