Gaining Efficiency and Flexibility in the Simple Temporal Problem

Amedeo Cesta

IP-CNR
National Research Council of Italy
Viale Marx 15, I-00137 Rome, Italy
amedeo@pscs2.irmkant.rm.cnr.it

Angelo Oddi

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Rome, Italy
oddi@assi.dis.uniroma1.it

Abstract

The paper deals with the problem of managing quantitative temporal networks without disjunctive constraints. The problem is known as Simple Temporal Problem. Dynamic management algorithms are considered to be coupled with incremental constraint posting approaches for planning and scheduling. A basic algorithm for incremental propagation of a new time constraint is presented that is a modification of the Bellman-Ford algorithm for Single Source Shortest Path Problem. For this algorithm a sufficient condition for inconsistency is given based on cycle detection in the shortest paths graph. Moreover, the problem of constraint retraction from a consistent situation is considered and properties for repropagating the network locally are exploited. Some experiments are also presented that show the usefulness of the properties.

1 Introduction

Knowledge-based architectures for planning and scheduling based on constraint propagation, e.g. [5, 3, 8, 2], perform incremental constraint posting and retraction on a current partial solution. A complete plan is created by efficiently searching in partial plans space, and, in other cases, it is adapted to new situations by partially removing parts of the solution. A module for temporal constraint management that supports plan space search and current solution maintenance should be extremely efficient because is called into play at any modification (monotonic or not) of the current plan. Such an efficiency is usually guaranteed by restricting the expressive power of the temporal representation. Usually the so called Simple Temporal Problem (STP) [7] is used that allows the representation of binary quantitative constraints without disjunction. In spite of the restriction of expressivity, also for STP it results useful to consider how the efficiency of manipulation primitives may be improved. In our research, we have been investigating possible algorithms for managing temporal information that: (a) allow dynamic changes of the constraint set for both incremental constraint posting and retraction; (b) exploit the localization of effects of any change in a subnetwork of the whole constraint graph; (c) do not compute the minimal network as done in [7] but just check for consistency. A previous paper [1], in the same line of [6], has concerned the specialization of *arc-consistency* algorithm to the STP. The choice of arc-consistency to propagate temporal constraints was motivated by the good trade-off wrt space and time complexity. In the same paper some properties were given that were shown experimentally to improve the performance of the algorithm in the average case. The present paper contains a further step in the direction of gaining efficiency in the solution of the STP. After presenting the essentials of STP (Section 2), it presents dynamic algorithms based on the well known Bellman-Ford algorithm for computing Single Source Shortest Paths (Section 3). It also introduces (Section 4) the concept of *dependency* that computes a particular spanning tree on the constraint graphs that allows the definitions of a sufficient condition for inconsistency detection (Section 5) and an algorithm for local constraint retraction (Section 6). Some experiments (Section 7) show the usefulness of the properties.

2 The Temporal Problem

A Simple Temporal Problem is defined in [7] and involves a set of temporal variables $\{X_1, \ldots, X_n\}$, having continuous domains $[lb_i, ub_i]$ and a set of constraints $\{a_{ij} \leq X_j - X_i \leq b_{ij}\}$, where $a_{ij} \geq 0$, $b_{ij} \geq 0$ and $a_{ij} \leq b_{ij}$. A special variable X_0 is added to represent the origin of the time (the beginning of the considered temporal horizon) and its domain is fixed to $[0, 0]$. A solution of the STP is a tuple $(x_i \ldots x_n)$ such that $x_i \in [lb_i, ub_i]$ and every constraint $a_{ij} \leq X_j - X_i \leq b_{ij}$ is satisfied. An STP is *inconsis-*

tent if no solution exists. In order to find the set of possible values $[lb_i, ub_i]$ for every variable X_i, a direct constraint graph $G_d(V_d, E_d)$ is associated to the STP, where the set of nodes V_d represents the set of variables $\{X_1, \ldots, X_n\}$ and the set of edges E_d represents the set of constraints $\{a_{ij} \leq X_j - X_i \leq b_{ij}\}$.
Given a constraint $a_{ij} \leq X_j - X_i \leq b_{ij}$, we can rewrite it as a pair of inequalities:

- $X_j - X_i \leq b_{ij}$ • $X_i - X_j \leq -a_{ij}$

For every linear inequality $X_j - X_i \leq w_{ij}$ (with w_{ij} equal to b_{ij} or $-a_{ij}$) we have an edge (i, j) in $G_d(V_d, E_d)$ labeled with the weight w_{ij}. Each path in G_d from the node i to the node j, $i = i_0, i_1 \ldots i_m = j$ induces between the variables X_j and $X_i$the constraint $X_j - X_i \leq l_{ij}$, where l_{ij} is the sum of weights along the path, that is $l_{ij} = w_{01} + w_{12} + \ldots + w_{(m-1)m}$. Considering the set of all paths between the nodes i and j, these paths induce a constraint $X_j - X_i \leq d_{ij}$, where d_{ij} is the length of a shortest path between the nodes i and j. Finally a *cycle* on the graph G_d is closed path $i = i_0, i_1 \ldots i_m = i$ and a *negative cycle* is a cycle with associated a negative length $(l_{ii} < 0)$.
In [7] some useful properties of an STP are given and reported in the following theorems.

Theorem 1 [7] *A Simple Temporal Problem is consistent iff G_d does not have negative cycles.*

Defining d_{0i} as the length of a shortest path on the graph G_d from the origin 0 and the node i and d_{i0} as the length of a shortest path from the node i to the origin 0 we can also have the other following theorem.

Theorem 2 [7] *Given a consistent Simple Temporal Problem, the set $[lb_i, ub_i]$ of feasible values for the variable X_i is the interval $[-d_{i0}, d_{0i}]$.*

Theorem 2 shows that the Simple Temporal Problem is a Shortest Paths Problem and precisely we have to calculate two sets of shortest paths length: (a) the set of shortest paths from the node 0 (that represent the variable X_0) to the nodes $1 \ldots n$; (b) and the set of shortest paths from the nodes $1 \ldots n$ to node 0.

3 An Algorithm for the STP

To solve the basic STP we use the Bellman-Ford algorithm for the Single Source Shortest Paths Problem [4] giving an incremental version of the algorithm named *Propagation*, which accepts as an input the graph G_d and a new constraint C_{ij} (where $C_{ij} = a_{ij} \leq X_j - X_i \leq b_{ij}$) and produces in output a new set of feasible values $[-d_{i0}, d_{0i}]$ for every variable X_i or a value *fail* in the case the new constraint induces a inconsistent situation.
To understand the algorithm, shown in Figure 1, some simple definitions are useful: given a node i of the graph G_d we define $EdgesOut(i)$ as the set of edges which leave from the node i and $EdgesIn(i)$ as the set of edges which arrive to the node i. T and F are the boolean constants $True$ and $False$.
The algorithm has two differences wrt the standard implementation on Bellman-Ford with a queue. First, it calculates at the same time two sets of shortest distances. Second, the algorithm has an internal test which detects negative cycles on the graph G_d which contain the reference node X_0. In addition, every node $u \in V_d$ has two boolean marks: $LB(u)$ and $UB(u)$. This marks are useful in order to distinguish the two types of propagation in the graph G_d, that is, respectively $UB(u) = T$ and $LB(u) = T$ when a node is modified by the propagation process for the distance d_{0i} and the distances d_{i0}. The *Propagation* calculates the set of distances $\{d_{0i}\}$ between Steps 6 and 14 and the set of distances $\{d_{i0}\}$ between Steps 16 and 24. This last section of the algorithm, in order to calculates the set of distances d_{i0}, (that is, the length of the shortest paths on the graph G_d between the nodes $1 \ldots n$ and the node 0) considers the set of direct edges in G_d as oriented in the opposite direction. In this way when a shortest path between the nodes 0 and i is found, it is actually a shortest path in the opposite direction. Finally, the tests at Steps 10 and 20 check for *negative cycles* in the graph G_d when they contain the node 0. The algorithm calculates also two shortest path trees. In fact Steps 11 and 21 respectively update the *predecessor function pu*, which represents the shortest path tree of the distances $\{d_{0i}\}$ and the *predecessor function pl*, which represents the shortest path tree of the distances $\{d_{i0}\}$.

The complexity of the algorithm, as well known, is $O(EN)$. Where N and E are respectively the number of nodes and the number of edges in G_d.

negative cycles

4 Focusing on Dependency

The temporal meaning of shortest path trees on the G_d graph is simple. Every bound $\{d_{0i}\}$ (or $\{d_{i0}\}$) is induced by the set of temporal constraints in the shortest paths between the origin 0 and the node i (or between the node i the origin 0). The following definitions are useful:

Definition 1 *Let G_d a consistent distance graph. The tree DT_{ub} of the shortest paths from the origin 0 to the nodes $1 \ldots n$ is called Upper Bounds' Dependency Tree.*

Definition 2 *Let G_d a consistent distance graph. The tree DT_{lb} of the shortest paths from to the nodes*

```
Propagation ($G_d$, $C_{ij}$)
1.  begin
2.      $Q \leftarrow \{i, j\}$
2a.     $LB(i) ::= T$; $UB(i) ::= T$
2b.     $LB(j) ::= T$; $UB(j) ::= T$
3.      While $Q \neq \emptyset$ do begin
4.          $u \leftarrow Pop(Q)$
5.          if $UB(u)$ then
6.              Foreach $(u, v) \in EdgesOut(u)$ do
7.                  if $d_{0u} + w_{uv} < d_{0v}$
8.                      then begin
9.                          $d_{0v} ::= d_{0u} + w_{uv}$
10.                         if $d_{0v} + d_{v0} < 0$ then exit(fail)
11.                         $pu(v) ::= u$
12.                         $UB(v) ::= T$
13.                         if $v \notin Q$ then $Q \leftarrow Q \cup \{v\}$
14.                     end
15.         if $LB(u)$ then
16.             Foreach $(u, v) \in EdgesIn(u)$ do
17.                 if $d_{u0} + w_{vu} < d_{v0}$
18.                     then begin
19.                         $d_{v0} ::= d_{u0} + w_{vu}$
20.                         if $d_{0v} + d_{v0} < 0$ then exit(fail)
21.                         $pl(v) ::= u$
22.                         $LB(v) ::= T$
23.                         if $v \notin Q$ then $Q \leftarrow Q \cup \{v\}$
24.                     end
25.         $LB(u) ::= F$
26.         $UB(u) ::= F$
27.     end
28. end
```

Figure 1: *Propagation* algorithm

$1 \ldots n$ to origin 0 is called Lower Bounds' Dependency Tree.

If a given graph G_d is consistent then the trees DT_{ub} and DT_{lb} are always defined. In fact, without negative cycles, the distances $\{d_{0i}\}$ and $\{d_{i0}\}$ are always defined. In general, the trees DT_{ub} and DT_{lb} may not be single. In fact, the graph G_d may contain several paths with the same length.

A relevant situation is verified when the graph G_d contains at least a negative cycle. In this case, the following Theorem holds.

Theorem 3 *Give a distance graph G_d. If during the update process of the* Propagation *algorithm the predecessor function pu (pl) represents a graph containing at least a cycle then the graph G_d is inconsistent.*

Proof. We give the proof for the distances $\{d_{0i}\}$, but an analogous proof can be given for the distances $\{d_{i0}\}$. Suppose by hypothesis that during the update process of the algorithm, a dependency path exists between the nodes i and j named $p_1 : i = i_0, i_1 \ldots i_r = j$, that is, a path such that $pu(i_k) = i_{k-1}$, with $k = 1 \ldots r$. If we sum the weights along this path, we have the following relation:

$$d_{0j} - d_{0i} = w_{01} + w_{12} + \ldots + w_{(r-1)r}. \qquad (1)$$

If successively the *Propagation* algorithm builds a dependency path $p_2 : j = j_0, j_1 \ldots j_s = j$, we can write the following relation:

$$d_{0i}^{new} - d_{0j} = w_{01} + w_{12} + \ldots + w_{(s-1)s}. \qquad (2)$$

Where d_{0i}^{new} is the new value of the distances d_{0i} updated along the path p_2. If we sum the relations 1 and 2 we obtain the length of the cycle l_{ii}:

$$l_{ii} = d_{0i}^{new} - d_{0i}. \qquad (3)$$

Observing that the link of two paths p_1 and p_2 is a cycle and $d_{0i}^{new} < d_{0i}$, then the length l_{ii} is negative and this proves the inconsistency of the graph G_d. □

5 Cycle Detection

In order to use the property expressed by Theorem 3 few changes are introduced in the *Propagation* algorithm. Each edge (i, j) in the graph G_d have three new boolean marks: $NEW((i, j))$, $LBP((i, j))$ and $UBP((i, j))$. The mark NEW is useful in order to distinguish the new edges introduced in G_d, by the new temporal constraint C_{ij}. In fact, if in the graph there is at least a negative cycle, then it must contain at least one of the new edges introduced. Instead, the two marks $LBP((i, j))$ and $UBP((i, j))$ are used to check when a bound changes two times as explained in the next Theorem 4:

Theorem 4 *Let G_d a consistent distance graph and $C_{ij} = a_{ij} \leq X_j - X_i \leq b_{ij}$ the new constraint added. If during the propagation process the distance d_{0j} (d_{i0}) changes two times, then the constraint C_{ij} is inconsistent with the other constraints represented in G_d.*

Proof. We give the proof for the distances $\{d_{0j}\}$, but an analogous proof can be given for the distances $\{d_{j0}\}$. If the constraint represented by the edge (i, j) changes the distance d_{0j} a first time, this means every new shortest paths built by the *Propagation* algorithm will contain the node j. If the distances is changed a second time, then the algorithm has built a closed dependency path and for the Theorem 3 the graph G_d is inconsistent. □

Figure 2 shows the modified version of the algorithm to check for cycle detection. It is interesting to notice the complexity of the algorithm with cycles detection is the same of the *Propagation* algorithm. In fact, the only difference with the previous algorithm is the check of the boolean marks $NEW((i,j))$ $LBP((i,j))$ and $UBP((i,j))$.

```
Propagation-cd ($G_d$, $C_{ij}$)
1. - 9.  as in the Propagation algorithm
10a.     if $d_{0v} + d_{v0} < 0$
10b.         then exit(fail)
10c.         else if $NEW((u,v))$
10d.             then if $UBP((u,v))$
10e.                 then exit(fail)
10f.                 else $UBP((u,v)) ::= T$
11. - 19.  as in the Propagation algorithm
20a.     if $d_{0v} + d_{v0} < 0$
20b.         then exit(fail)
20c.         else if $NEW((u,v))$
20d.             then if $LBP((u,v))$
20e.                 then exit(fail)
20f.                 else $LBP((u,v)) ::= T$
24. - 28.  as in the Propagation algorithm
```

Figure 2: Differences introduced by cycle detection

the average time

6 Retraction of Temporal Constraints from a Consistent Context

This paragraph deals with the problem of removing temporal constraints from a consistent graph G_d (a graph without negative cycles). A basic way to do this consists of: physically removing the constraint from the graph G_d; setting every distance $\{d_{0i}\}$ and $\{d_{i0}\}$ to the value $+\infty$; finally, running the *Propagation* algorithm on the whole graph.
As a matter of fact, this method is not very efficient. In fact, when retracting a constraint from the time map a lot of distances are likely not to be affected by the removal. The dependency information may be used to focalize the part of the network actually affected by the removal and to run the *Propagation* algorithm on that part of the graph.
To state same properties some definitions are useful. Given an upper bounds' dependency tree $DT_{ub}(V_{DT_{ub}}, E_{DT_{ub}})$, each sub-tree $ST_{ub}[i](V_{ST_{ub}}, E_{ST_{ub}})$ of root $i \in V_{DT_{ub}}$ is called an *Upper Bounds' Dependency Sub-tree.* Given a lower bounds' dependency tree $DT_{lb}(V_{DT_{lb}}, E_{DT_{lb}})$ every sub-tree $ST_{lb}[i](V_{ST_{lb}}, E_{ST_{lb}})$ of root $i \in V_{DT_{lb}}$ is called a *Lower Bounds' Dependency Sub-tree.* Given a a distance graph $G_d(V_{G_d}, E_{G_d})$ and a node $i \in V_{G_d}$, $IN(i)$ is the set of start nodes of the edges which enter in the node i (in the edge (j,i), j is the start node and i is the end node). The next Proposition explains the real effects of a removal constraints from a graph G_d and it is a starting point to write a new algorithm to remove temporal constraints from G_d.

Proposition 1 *Let G_d be a consistent graph and $DT_{ub}(V_{DT_{ub}}, E_{DT_{ub}})$ its upper bounds' dependency tree ($DT_{lb}(V_{DT_{lb}}, E_{DT_{lb}})$ its lower bounds' dependency tree). The retraction of an edge $(i,j) \in E_{DT_{ub}}$ ($(i,j) \in E_{DT_{lb}}$) modifies at most the distances of the nodes $k \in V_{ST_{ub}[j]}$ ($k \in V_{ST_{lb}[j]}$). No distances are modified when $(i,j) \notin E_{DT_{ub}}$ ($(i,j) \notin E_{DT_{lb}}$).*

Proof. We give the proof for the distances $\{d_{0i}\}$, but an analogous proof can be given for the distances $\{d_{i0}\}$. The removal of an edge $(i,j) \in E_{DT_{ub}}$ can't modify a node's distance $\{d_{0k}\}$ in the case $k \notin V_{ST_{ub}}[k]$. In fact the removal of (i,j) does not change the shortest path between the origin 0 and the node k. If $(i,j) \notin E_{DT_{ub}}$ then no distance is changed because no shortest path is changed. □

The basic idea to write an efficient removal algorithm is run the *Propagation* algorithm on the only part of the G_d graph affected by the removal of the constraint. The next Theorem formalize this concept and explains how to initialize the *Propagation* algorithm.

Theorem 5 *Let G_d be a consistent distance graph. To remove the effects of the constraint represented by the edge $(i,j) \in E_{DT_{ub}}$ ($(i,j) \in E_{DT_{lb}}$) the queue Q of the Propagation algorithm and the set of distances $\{d_{0i}\}$ ($\{d_{i0}\}$) in the graph G_d need of the following initialize operations.*

1. $Q \leftarrow \bigcup_{k \in V_{ST_{ub}}[j]} IN(k) \quad (Q \leftarrow \bigcup_{k \in V_{ST_{lb}}[j]} IN(k))$
2. $d_{0u} ::= +\infty, u \in V_{ST_{ub}}[j] \quad (d_{u0} ::= +\infty, u \in V_{ST_{lb}}[j])$

Proof. We give the proof for the distances $\{d_{0i}\}$, but an analogous proof can be given for the distances $\{d_{i0}\}$. By Proposition 1, for every node $k \in V_{ST_{ub}}[j]$, the distance $\{d_{0k}\}$ can change after the removal. The *Propagation* algorithm have to rebuild the new shortest paths for every node $k \in V_{ST_{ub}}[j]$. In order to update these distances to the new values, it is necessary to initialize them to the maximum possible value $+\infty$. In fact, it is not known what the new values will be and the *Propagation* algorithm can only reduce the bounds. In addition, we have to put in the queue Q all the nodes of the constraints (i, j) which enter in the set of updated nodes. That is, the nodes in the set $\bigcup_{k \in V_{ST_{ub}}[i]} IN(k)$. In fact, these are the only nodes of the graph from which can start the new shortest paths of the nodes $k \in V_{ST_{ub}}[j]$. □

The *Remove* algorithm is shown in Figure 3. It accepts as an input a graph G_d and a constraint C_{ij} which have to be removed from G_d and return the graph G_d updated. At the step 13 is used the *RePropagation* algorithm that is similar to the *Propagation* algorithm but accepts as an input a list of nodes Q instead of an edge C_{ij}. The parameter Q is used as an initialization for the internal queue. Moreover *RePropagation* does not check for the consistency of a modification because the removal of one or more constraints, relax the STP holding the consistency property.

```
Remove ($G_d$, $C_{ij}$)
1.  begin
2.      $V_m \leftarrow \emptyset$
3.      $Q \leftarrow \emptyset$
4.      if $(i, j) \in E_{DT_{ub}}$
5.          then $V_m \leftarrow V_m \cup V_{ST_{ub}}[j]$
6.          else if $(j, i) \in E_{DT_{ub}}$
7.                  then $V_m \leftarrow V_m \cup V_{ST_{ub}}[i]$
8.      if $(i, j) \in E_{DT_{lb}}$
9.          then $V_m \leftarrow V_m \cup V_{ST_{lb}}[i]$
7.          else if $(j, i) \in E_{DT_{lb}}$
8.                  then $V_m \leftarrow V_m \cup V_{ST_{lb}}[j]$
9.      Foreach $u \in V_m$ do begin
10.         $Q \leftarrow Q \cup IN(u)$
11.     end
12.     $E_{G_d} \leftarrow E_{G_d} - \{(i, j), (j, i)\}$
13.     RePropagation($G_d$, $Q$)
14. end
```

Figure 3: *Remove* algorithm

7 Performance Evaluation

In order to get some realistic evaluations of the algorithms, we have used a scheduling system described in [3] and the time network generated by the scheduler. This scheduler solves instances of the Deadline Job Shop Scheduling Problem (DJSSP) by incremental precedence constraint posting between the activities until any conflict in the use of resources is resolved.

In the DJSSP, each activity in a job can request only one resource and a resource is requested only once in a job. The sequence of resources requested by the activities in a job is random. Every job has a fixed release date and a due date. More details on the random problem generator are described in [3].

All the evaluations are given as number of time points explored by the algorithms. This choice is motivated from the fact that such number is both proportional to the time of computation and machine independent.

We have built two different types of time networks from the resolution of two different DJSSPs: the $8 \times 8 \times 8$ (named $P8$) and the $10 \times 10 \times 10$ (named $P10$), where the first number indicate the number of jobs, the second one the number of activities in a job and the third one the number of resources. The data are obtained running ten instances of each type of problem.

Table 1 shows the number of time points N, the maximum number of distance constraints E_{max} and maximum connectivity C_{max} for each problem. The *connectivity* is defined as the ratio between the number of distance constraints E and the number of time points N. The value N is two times the number of activities plus two (the *origin point* and *horizon point*). The value E_{max} represents the maximum number of distance constraints which can be contained in a time network associated to the solution of the instance of the DJSSP. E_{max} is obtained by the sum of the maximum values of the number of precedence constraints for each resource and the number of constraints before the scheduling algorithm starts to find a solution. Table 2 and Table 3 present the performance of the *Propagation* algorithm when a modification is either consistent or inconsistent respectively. This values are shown as a function of the average connectivity *Av-conn*, that is, every row of the table represents the average value obtained in the interval *Av-conn* ± 0.25. In order to get several values of the connectivity we have built a solution of an instance of a DJSSP and progressively reduced the number of edges and selected a time constraint C_{ij} in random way. In order to get the results showed in Table 2, we have modified the distance constraint selected $C_{ij} = a_{ij} \leq X_j - X_i \leq b_{ij}$, in the constraint

Table 1: Number of time points and maximum connectivity for the experimental time networks

Problem	N	E_{max}	$C_{max} = E_{max}/N$
$P8$	130	333	2.56
$P10$	202	661	3.27

$C_{ij} = a_{ij} + (d_{ij} - a_{ij})U[0.05, 01] \leq X_j - X_i \leq b_{ij}$. Where $U[x, y]$ represents a random value r with uniform distribution such that $x \leq r \leq y$ and d_{ij} is minimal temporal distance between the nodes i and j on the G_d graph. In this case, it is possible to make a comparison between the number of nodes scanned by the *Propagation* algorithm (*Loc-prop* values) and the number of nodes scanned by an algorithm which works from scratch (*Scratch* values). In order to get the results showed in Table 3, we have induced an inconsistent situation by modifying the constraint C_{ij} in the constraint $d_{ij}(1 + U[0.05, 01]) \leq X_j - X_i \leq b_{ij}$ In this other case, it is possible make a comparison between the number of nodes visited by the *Propagation* algorithm which uses the property expresses by Theorem 4 (*Cycle-det* values) and the number without the previous property (*No-cycle-det* values).

Table 2: Incremental vs scratch propagation

Problem	Av-conn	Loc-prop	Scratch
$P8$	1.25	38.76	652.67
	1.75	52.33	1111.47
	2.25	34.08	1641.19
	2.75	39.51	2048.28
$P10$	1.25	51.42	1108.38
	1.75	67.20	1928.54
	2.25	64.34	2876.22
	2.75	57.00	3817.79
	3.25	63.92	4388.71

Table 3: Propagation with and without cycle detection

Problem	Av-conn	Cycle-det	No-cycle-det
$P8$	1.25	3.02	114.42
	1.75	2.92	77.30
	2.25	2.47	43.87
	2.75	1.92	9.75
$P10$	1.25	3.21	199.45
	1.75	2.78	133.81
	2.25	2.68	86.85
	2.75	2.55	27.15
	3.25	2.63	14.58

Finally, Table 4 presents the performance of the *Remove* algorithm. These results are obtained in the same way as the previous ones. First we have built a solution; then we have reduced progressively the number of time constraints by using the *Remove* algorithm. In this case, is possible to make a comparison between the average number of nodes scanned by the *Remove* algorithm (*Loc-rem* values) and the number of nodes scanned in the same case by a scratch algorithm (*Scratch-rem* values). The scratch algorithm eliminates first the constraint from the time map; then puts all the bounds of the time points to the value $+\infty$; finally updates all the network.

Table 4: Incremental vs scratch remove

Problem	Av-conn	Loc-rem	Scratch-rem
$P8$	1.25	2.37	652.67
	1.75	35.34	1111.47
	2.25	56.02	1641.19
	2.75	96.35	2048.28
$P10$	1.25	2.69	1108.38
	1.75	33.12	1928.54
	2.25	55.06	2876.22
	2.75	70.58	3817.79
	3.25	156.97	4388.71

Acknowledgments

This research is partially supported by: ASI - Italian Space Agency, CNR Special Project on Planning, CNR Committee 04 on Biology and Medicine.

References

[1] Cervoni, R., Cesta, A., Oddi, A., Managing Dynamic Temporal Constraint Networks, *Proceedings of the Second International Conference on AI Planning Systems (AIPS94)*, AAAI Press, 1994.

[2] Cesta, A., Oddi, A., DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains, *Proceedings of the 3rd European Workshop on Planning (EWSP95)*, IOS Press, 1996.

[3] Cheng, C. Smith, S.,F., Generating Feasible Schedules under Complex Metric Constraints, *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, AAAI Press, 1994.

[4] Cormen, T.H., Leierson, C.E., Rivest, R.L., *Introduction to Algorithms*, MIT Press, 1990.

[5] Currie, K., Tate, A., O-Plan: the open planning architecture, *Artificial Intelligence*, 52, 1991, 49-86.

[6] Davis, E., Constraint Propagation with Interval Labels, *Artificial Intelligence*, 32, 1987, 281-331.

[7] Dechter, R., Meiri, I., Pearl, J., Temporal constraint networks. *Artificial Intelligence*, 49, 1991, 61-95.

[8] Ghallab, M, Laruelle, H., Representation on Control in IxTeT, a Temporal Planner, *Proceedings of the Second International Conference on AI Planning Systems (AIPS94)*, AAAI Press, 1994.