Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Maintenance of solution in a dynamic constraint satisfaction problem

A. Bellicha

LIRMM, 161 rue ADA, 34392 Montpellier Cedex 5, France

ABSTRACT

Constraint Satisfaction Problems (*CSP*) have been shown to be a useful way of formulating problems such as design, scene labelling and temporal reasoning. As many problems using constraints need a dynamic environment, the static framework of CSPs extend into DCSP (*Dynamic Constraint Satisfaction Problems*). Up to now, most papers about DCSPs have dealt with the problem of the existence of a solution and the filtering techniques. The problem of the maintenance of a solution, after the DCSP has evolved, has mainly been approached through re-execution or delay to the computation of the solution. This paper first presents the CSP framework and its dynamic evolution DCSP, and then assigns bounds to the study of the problem of the maintenance of solution: given an instance of a binary DCSP, a solution to it and a new constraint which disables that solution, we achieve the computation (if possible) of a new solution as "close" as possible to the previous one - with several criteria of closeness. The paper presents an efficient algorithm restricted to acyclic binary DCSPs and new constraints which do not modify the constraint-graph, and then its extension to the cyclic case with any new constraint.

INTRODUCTION

Constraint networks (or Constraint Satisfaction Problems (*CSP*)) have been shown to be a useful way of formulating problems such as design, scene labelling, diagnosis and temporal reasoning. A CSP is defined as being the problem of finding one or all consistent labelling for a fixed set of variables satisfying all given constraints between those variables.

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

There are various resolution techniques: filtering techniques, heuristics, structural methods...

Nevertheless the CSP framework is static (the sets of variables and constraints are fixed and constraints cannot evolve), whereas a great number of problems involve a dynamic environment.

For instance, the data of a design process may not be completely defined at the beginning of the process, and may need to change during the resolution; interactivity between the resolver and the user is a necessity and a satistying solution may be built step by step [9].

In scheduling problems, a first solution can be destroyed by an unexpected event, and we need to compute a new solution which does not differ much from the former one; for instance, Gates [2] is a constraint satisfaction expert system that assigns gates to arriving and departing flights at New York's JFK International Airport; schedules are created monthly, but may be interrupted by unanticipated events such as mechanical failure, bad weather, late arrival and so forth. In such situations, existing gate assignments must be rearranged quickly, and with few modifications.

For those dynamic problems, two approaches may be distinguished: a predicting one where the selected solutions must be resistant to disturbances, and where information must be maintained in order to help further computations, and a reactive one with no preparation of the future computations.

This paper focuses only on the second approach. It uses the Dynamic Constraint Satisfaction Problem framework *(DCSP)* which has been defined by R. & A. Dechter [8] and Janssen et al [6]: thus a DCSP is a sequence of static CSPs, where a component (or instance) of this sequence differs from the previous one only with one constraint (added or removed).

A great number of papers about DCSPs deal with the adaptation of filtering techniques to the dynamic framework in order to maintain partial consistency (Bessière [1]) or global consistency (Dechter & Dechter [8]).

Van Hentenryck [5] gives an efficient way to maintain a solution to the problem once the current solution has been disabled by a new constraint. But as we have seen, in many applications this new solution must be similar to the previous one.

This paper presents an adaption of the work of R. & A. Dechter[8] to the computation of a new solution, if there exists one, as "close" as possible to the former one. It is organized as follows: section 2 presents the *CSP* and *DCSP* frameworks, and some related notions. Section 3 deals with the notion of closeness and presents the method in the case of a binary acyclic DCSP. Section 4 presents an extension of this method to all binary DCSPs.

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

DEFINITIONS

Definition 1

A Constraint Satisfaction Problem $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R})$ *involves:*
- a finite set of variables $\mathcal{X} = \{X, Y, Z, \ldots\}$.
- a set of domains $\mathcal{D} = \{D_X, D_Y, D_Z, \ldots\}$*; each variable* X *takes value in the domain* D_X*; we suppose each* D_X *is finite .*
- a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$*; each* C_i *is a subset of* $\mathcal{X}$ *.*
- a set of relations $\mathcal{R} = \{R_1, \ldots, R_m\}$*; if* $C_i = \{X_1, \ldots, X_{k_i}\}$, R_i *is a proper subset of the Cartesian product* $D_{X_1} \times \ldots \times D_{X_{k_i}}$ *.*

The CSP is binary if all the constraints are binary, i.e. involve two variables.

If $C_i = \{Y, Z\}$, then the associated relation R_i may also be denoted as R_{YZ} .

The constraint-graph associated with a binary CSP is the graph in which vertices represent variables, and edges connect these pairs of variables for which constraints are given. Edges are labelled with corresponding relations.

n denotes the set of variables size.
m denotes the set of constraints size.
d denotes the size of the largest domain.

Definition 2

Let $\mathcal{X}'$ *be a subset of* $\mathcal{X}$ *and* $L(\mathcal{X}')$ *an assignment of* $\mathcal{X}'$.
$L(\mathcal{X}')$ *is said to be consistent if and only if for any constraint* C_i *included in* $\mathcal{X}'$*, the projection of* $L(\mathcal{X}')$ *onto* C_i *belongs to* R_i *(* $L(\mathcal{X}')[C_i] \in R_i$ *).*

A solution for the CSP is a consistent assignment of $\mathcal{X}$.

Definition 3

A Dynamic Constraint Satisfaction Problem P *is a sequence of static CSPs* $P_1, P_2, \ldots, P_i, \ldots$ *called instances of the DCSP, each resulting from a change in the preceding one:*
either a restriction (a new constraint is imposed on a pair of variables) or a relaxation (a constraint that was present in the CSP is removed).
So if $P_i = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R})$ *then* $P_{i+1} = (\mathcal{X}, \mathcal{D}, \mathcal{C}', \mathcal{R}')$ *, where* $\mathcal{C}' = \mathcal{C} \pm \{C_k\}$, *and* $\mathcal{R}' = \mathcal{R} \pm \{R_k\}$ *, where* R_k *is the relation attached to* C_k.
P_1 *may be defined as* $(\mathcal{X}, \mathcal{D}, \emptyset, \emptyset)$.

A DCSP is said to be binary if each of its static CSPs is binary.

We add to that framework what we call unary constraints: they consist in reducing domains.

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

MAINTENANCE OF SOLUTION IN A BINARY ACYCLIC DCSP

The problem treated in the present section is the following:
Given an instance P_i of a DCSP, s_i a solution of P_i and a new constraint C_k which disables s_i, compute a solution s_{i+1} for $P_{i+1} = P_i \cup \{C_k\}$ close to s_i if one exists.

In order to feature that notion of closeness, we may define a distance on the Cartesian product of all domains $D_1 \times \ldots \times D_n$ as following:

Definition 4

Let $\mathcal{X}'$ be a subset of $\mathcal{X}$.
Let s_1 and s_2 be two assignments of $\mathcal{X}$.
The distance between s_1 and s_2 depending on $\mathcal{X}'$ is defined by:

$$d_{\mathcal{X}'}(s_1, s_2) = \mid \{Y \in \mathcal{X}' / s_1[Y] \neq s_2[Y]\} \mid$$

This is the Hamming distance restricted to $\mathcal{X}'$.
The definition of $\mathcal{X}'$ allows us to use several criteria of closeness:

- if $\mathcal{X}' = \mathcal{X}$, s_1 is close to s_2, this means few values are different, with no preferences about the concerned variables.
- if $\mathcal{X}' \neq \mathcal{X}$, we have the possibility of keeping unchanged, or little changed, a determined subset of variables. But there may be many differences between s_1 and s_2 on the variables of $\mathcal{X} - \mathcal{X}'$; there is no preference concerning the values of these variables.

In the remainder of this section, we will only consider connected and acyclic graphs. The only new constraints we will consider are either unary constraints, or binary ones which do not modify the constraint-graph structure (those constraints consist in reducing a relation; they are modifications of existing constraints), in order to keep the property of acyclicity for the constraint-graph.

Addition of a unary constraint

Let $P_i = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R})$, *sol* a solution and $\mathcal{X}'$ a subset of $\mathcal{X}$.
If the added constraint involves the variable X, it necessarily consists in removing $sol[X]$ from D_X .

Principle

First step: each couple (Y, a) is associated with a number whose purpose is to count the minimal number of modifications to be done to *sol* because of the assignment of Y with a.

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

We note $\mathcal{V} = \{(Y, a)/Y \in \mathcal{X}$ and $a \in D_Y\}$ the set of all possible assignments;
$\mathcal{W} = \{(Y, a, Z)/Y$ and $Z \in \mathcal{X}, a \in D_Y$ and $\{Y, Z\} \in \mathcal{C}\}$.

These minimal numbers of modifications are kept in a table $\mathcal{N}$ indexed with $\mathcal{V}$. Initially for each couple (Y, a) of $\mathcal{V}$:

- $\mathcal{N}(Y, a) = 0$ if $sol[Y] = a$ and $Y \in \mathcal{X}'$
- $\mathcal{N}(Y, a) = 1$ if $sol[Y] \neq a$ and $Y \in \mathcal{X}'$
- $\mathcal{N}(Y, a) = 0$ if $Y \notin \mathcal{X}'$
- $\mathcal{N}(X, sol[X]) = \infty$, where X is the variable involved by the new constraint.

The constraint-graph is implicitly turned into a directed tree rooted at X.

The number of modifications is propagated from the leaves to the root according to the following rules:
Let $\{Z_1, \ldots, Z_k\}$ be the set of the successors of Y in the directed tree rooted at X.
For each value a of Y, we compute:

$$\mathcal{N}(Y, a) \longleftarrow \mathcal{N}(Y, a) + \sum_{Z \in \{Z_1, \ldots, Z_k\}} \mathop{\mathrm{Min}}_{((Y,a),(Z,b)) \in R_{YZ}} ((\mathcal{N}(Z, b))$$

Each value $N(Y, a)$ represents the minimal number of modifications to be done to *sol* (restricted to $\mathcal{X}'$) coming from the subtree rooted at Y and induced by the assignment of Y by a.

We will have a global knowledge about the question after the computation of all values $\mathcal{N}(X, b)$ where $b \in D_X$.

Second step: a new solution will be built by choosing first one value b of X such that $\mathcal{N}(X, b)$ is minimal and finite, then running through the directed tree from the root to the leaves and choosing for each variable the value which led to the construction of that minimal number of modifications.

That second part of computation can be prepared during the first one: if Z is a successor of Y in the directed tree, we have to keep for each value a of Y the value b of Z such that $((Y, a), (Z, b)) \in R_{YZ}$ and $\mathcal{N}(Z, b)$ minimal; the table $\mathcal{M}$ indexed with $\mathcal{W}$ is devoted to that task: $\mathcal{M}(Y, a, Z) = b$.

Remarks:

- a way of keeping in memory that a value a of a variable Y has been removed from the domain D_Y is to maintain those elements of the table $\mathcal{N}$ which are equal to ∞ during all the DCSP's resolution process;
- if a value a of Y has no support on Z (meaning there is no value b of Z such that$(((Y, a), (Z, b)) \in R_{YZ})$, then there is no solution involving (Y, a).

It will be specified by assigning ∞ to $N(Y, a)$.

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

In this algorithm, we denote as $\Gamma(Y)$ the set of the neighbours of Y in the constraint-graph.

First step: algorithm

```
{First part: initialization of the table N}
begin
  N(X, sol[X]) ⟵ ∞
  for every Y ∈ X do
    for every a ∈ D_Y do
      if N(Y, a) ≠ ∞ then
        if (Y ∈ X') and (sol[Y] ≠ a) then
          N(Y, a) ⟵ 1
        else
          N(Y, a) ⟵ 0
        endif
      endif
    endfor
  endfor
end

{Second part: propagation}
procedure Dismin (Y, Reach)
  begin
    for every Z ∈ Γ(Y) − Reach do
      Dismin(Z, Reach ∪ {Y})
    endfor
    for every a ∈ D_Y do
      if N(Y, a) ≠ ∞ then
        for every Z ∈ Γ(Y) − Reach do
          B ⟵ {b ∈ D_Z/((Y, a), (Z, b)) ∈ R_YZ}
          if B ≠ ∅ then
            b ⟵ c ∈ B/N(Z, c)minimal
            N(Y, a) ⟵ N(Y, a) + N(Z, b)
          else
            N(Y, a) ⟵ ∞
          endif
          if N(Y, a) ≠ ∞ then
            M(Y, a, Z) ⟵ b
          endif
        endfor
      endif
    endfor
  end
```

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Notice that $\Gamma(Y) - Reach$ is the set of successors of Y in the directed tree rooted at X.

Global variables: Tables $\mathcal{N}$ and $\mathcal{M}$
Call for the algorithm: $Dismin(X, \emptyset)$

Second step: computation of the new solution
Choose (X, b) such that $\mathcal{N}(X, b)$ is minimal and finite, if one exists; then run through the directed tree from the root to the leaves and choose for each successor of each variable Y the value kept in $\mathcal{M}(Y, a, Z)$, where a is the value of Y.

Complexity
Time
Each value is considered once during the initialization part: $O(nd^2)$.
Each constraint is considered twice:

- once when the informations go from the leaves to the root; each relation is entirely covered: $(n-1) \times O(d^2)$ or $O(nd^2)$.
- once during the computation of the solution: $(n-1) \times O(1)$ or $O(n)$.

Research of the minimal value $\mathcal{N}(X, b) : O(d)$
Construction of the solution: $O(n-1)$
So the complexity in time is $O(nd^2)$.
Space: the datastructures are

- $\mathcal{M}$ indexed with $\mathcal{W}$, where $|\mathcal{W}| = (n-1)d$. Notice that $\mathcal{M}$ is only partially used and may be replaced by a list; the space complexity remains the same in the worst case.
- $\mathcal{N}$ indexed with $\mathcal{V}$ where $|\mathcal{V}| = nd$

So the complexity in space is $O(nd)$.

Addition of a binary constraint which does not modify the constraint graph

Such an addition keeps the constraint-graph acyclic.

It consists in removing a tuple from the corresponding relation (the removed tuple must be marked).

Notice that it is not equivalent to the successive removing of these two values which are components of the tuple: if $((Y, a), (Z, b))$ is the removed tuple, it is possible that the closest solution computed involves (Y, a) or (Z, b).

The computation of a new solution close to the previous one follows the same rules as the former method; the directed tree is rooted indifferently at Y or Z.

In fact, computations on T_1 and on T_2 are independent of each other and of the constraint Y, Z (see Figure 1 below).

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

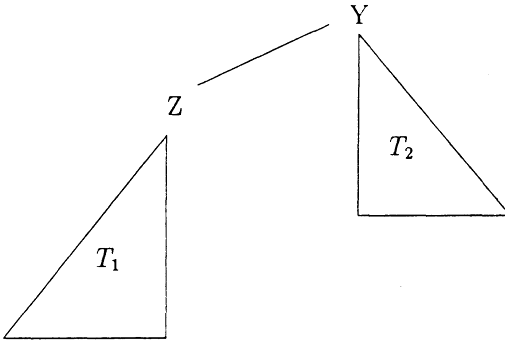

Figure 1: Modification of an existing constraint

The computation on Y, Z depends on the results of the computations on T_1 and on T_2 and on the connections between Y and Z, which are independent of the order upon Y and Z.

MAINTENANCE OF SOLUTION IN A CYCLIC BINARY DCSP

The former method is based on an information propagation from the leaves to the root of a directed tree, so that it cannot be applied to cyclic graphs. It can be easily shown that an application of this method to a cycle, where a starting node has been arbitrarily chosen, may lead to a loss of information.

Then the method needs an adaptation to the cyclic graphs (the constraint graph may be cyclic initially or because of the addition of a constraint).

The former method can be applied when removing the cycle by application of the cycle cutset decomposition technique (Dechter& Pearl [4], [3], and Jégou [7] for a linear algorithm which computes a small cycle cutset): a cycle-cutset of a graph is a set of vertices, the deletion of which makes the graph acyclic .

For any consistent assignment of the cycle cutset, the problem is divided into as many acyclic subproblems as there are connected components in the graph deprived of the cycle cutset.

The description of the induced trees may remove the cycle cutset, or duplicate it as many times as there are links to its elements. In the following, we will choose that second representation.

Given an assignment of the cycle cutset, the connections between the connected components have to be achieved through the cycle cutset: then the cycle cutset will be responsible for propagating inside one tree the information coming from the other trees.

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

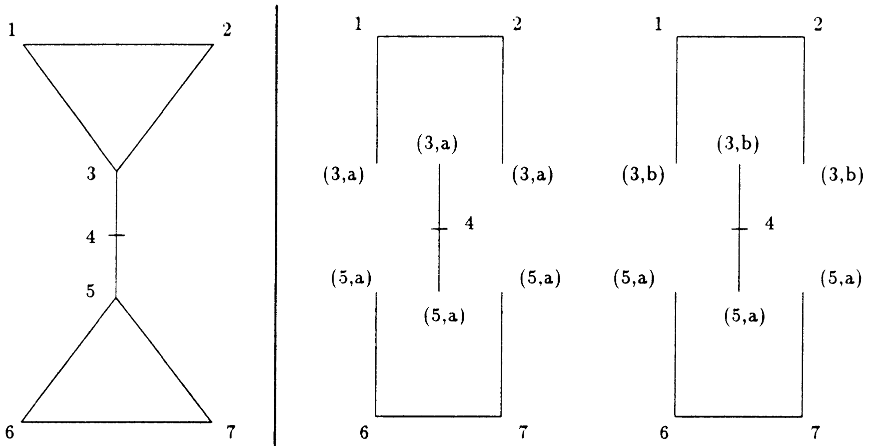

Figure 2: A cyclic constraint-graph and its decomposition by the cycle-cutset method

Example:
A cycle cutset may be $CC = \{3, 5\}$ with $D_3 = \{a, b\}$ and $D_5 = \{a\}$. Figure 2 shows a decomposition using the cycle-cutset method.

Addition of a unary constraint

If the modified variable X does not belong to the cycle cutset

The following instructions describe the work to be done once the cycle cutset has been assigned. They must be repeated for each one of the consistent assignments of the cycle cutset:

Part 1
For each connected component CO not containing X, we compute the assignment of CO that is the closest to the old solution *sol* (that means to $sol[CO]$), if possible.

In order to achieve that, we simply apply the method described in section 3, choosing as a root of the induced directed tree any element of the cycle cutset: the global information about that tree has to be assigned to a variable of the cycle cutset, in order to allow the propagation into the other trees.

It is necessary in that process to initialize $N(Y, a)$ to 0, for each variable Y belonging to the cutset (a being its value, which is unique because Y belongs to the cutset). This will avoid a modification to be done to *sol* to be counted several times, given that a variable of the cutset may be duplicated inside a connected component.

If one of the results received at the roots equals ∞, it is not possible to build a solution with that assignment of the cycle cutset; otherwise we must add them and add the number of modifications due to the assignment of the cycle cutset.

This last result will be assigned to one and only one variable of the cycle cutset in the connected component where X appears.

Part2

In the connected component where X appears, the process described in section 3 has to be run with X as a root.

If the modified variable X belongs to the cycle cutset

X may be duplicated when achieving the cycle cutset decomposition method.

To build up a new solution, we have to achieve the first part of the former process, extended to all connected components, with no exceptions. As a value a of X may appear in several assignments of the cycle cutset, we will know the final value of $\mathcal{N}(X, a)$ by computing the minimum of the different values of $\mathcal{N}(X, a)$ due to those assignments.

Example: $(\mathcal{X}' = \mathcal{X})$

The constraint-graph is the same as in Figure 2.
Domains and relations are defined as follows:
$D_1 = D_2 = D_6 = D_7 = \{b, c\}$
$D_3 = D_4 = D_5 = \{a, b\}$
$R_{12} = R_{67} = \{(b, b), (c, c)\}$
$R_{31} = R_{32} = R_{56} = R_{57} = \{(a, b), (a, c), (b, c)\}$
$R_{34} = \{(a, a), (a, b), (b, a), (b, b)\}$
$R_{54} = \{(a, a), (b, b)\}$

Previous solution:	1	2	3	4	5	6	7
	c	c	b	a	a	b	b

New constraint: (2,c) is removed

Cycle cutset: $\{3, 5\}$; it has four consistent assignments: $(3, a), (5, a)$
$(3, a), (5, b)$
$(3, b), (5, b)$
$(3, b), (5, b)$

We will only consider $(3, a), (5, a)$ and $(3, a), (5, b)$ because the two others belong to the globally inconsistent assignments of X (the computation would show it !).

In each connected component which does not include the variable 2, the root is a variable of the cycle-cutset.

In the connected component including the variable 2, only one vertex is in charge of the propagation of the information coming from the other connected component.

$\mathcal{N}(2,c) = \infty$
$\mathcal{N}(2,b) = 3$
$\mathcal{N}(1,b) = 2$ 1 2
$\mathcal{N}(1,c) = 1$
$\mathcal{N}(3,a) = 1$ $(3,a)$ $(3,a)$ $\mathcal{N}(3,a) = 0$

$\mathcal{N}(2,c) = \infty$
$\mathcal{N}(2,b) = 7$
$\mathcal{N}(1,b) = 6$ 1 2
$\mathcal{N}(1,c) = 5$
$\mathcal{N}(3,a) = 5$ $(3,a)$ $(3,a)$ $\mathcal{N}(3,a) = 0$

$\mathcal{N}(3,a) = 0$ $(3,a)$
$\mathcal{N}(4,a) = 0$ 4
$\mathcal{N}(4,b) = \infty$
$\mathcal{N}(5,a) = 0$ $(5,a)$

$(3,a)$ $\mathcal{N}(3,a) = 1$
4 $\mathcal{N}(4,a) = \infty$
$\mathcal{N}(4,b) = 1$
$(5,b)$ $\mathcal{N}(5,b) = 0$

$\mathcal{N}(5,a) = 0$ $(5,a)$ $(5,a)$ $\mathcal{N}(5,a) = 0$
$\mathcal{N}(6,b) = 0$ 6 7 $\mathcal{N}(7,b) = 0$
$\mathcal{N}(6,c) = 2$ $\mathcal{N}(7,c,) = 1$

$\mathcal{N}(5,b) = 2$ $(5,b)$ $(5,b)$ $\mathcal{N}(5,b) = 0$
$\mathcal{N}(6,b) = \infty$ 6 7 $\mathcal{N}(7,b) = \infty$
$\mathcal{N}(6,c) = 2$ $\mathcal{N}(7,c,) = 1$

Addition of the modifications due to the subtrees not containing 2: $\mathcal{N}(3,a) + \mathcal{N}(5,a) = 0$
We must add the modifications due to the assignment of the cutset (i.e. 1). The final result 1 is assigned to one and only one variable of the cutset in the subtree rooted at 2.

Addition of the modifications due to the subtrees not containing 2: $\mathcal{N}(3,a) + \mathcal{N}(5,B) = 3$
Modifications due to the assignment of the cutset: 2.
The final result 5 is assigned to one and only one variable of the cutset in the subtree rooted at 2.

Figure 3: Propagation through the connected components

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

The minimum is reached with $(3, a)$ $(5, a)$ and $(2, b)$.
We can build the new solution closest to the previous one ($distance = 3$).

New solution:	1	2	3	4	5	6	7
	b	b	a	a	a	b	b

Addition of a binary constraint which does not modify the constraint graph

We may use the same cycle-cutset as previously.

If none of the two variables of this constraint belongs to the cycle cutset, parts 1 and 2 of the process must be achieved (anyone one of those variables may be the root). If one or more of them belongs to the cycle cutset, only part 1 must be achieved.

Addition of a binary constraint which modifies the constraint graph

The cycle cutset must be computed again for it may have changed because of the new constraint.

Then the same process as previously can be achieved, and any one of the variables involved in the new constraint may be the root.

Complexity

Time :

Computation of the cycle cutset: $O(m + n)$

Computation for one assignment of cycle cutset: $O(md^2)$

If the size of the cutset is c, it has $O(d^c)$ assignments.

So the time complexity is $O(md^{c+2})$. It may be estimated after the computation of the cycle cutset.

Space:

if we duplicate the tables $\mathcal{N}$ and $\mathcal{M}$ the space complexity will turn into $O(nd^{c+1})$ for $\mathcal{N}$ and $O(md^{c+1})$ for $\mathcal{M}$.

But we may improve that by optimizing the process:

for each assignment of the cycle-cutset CC, we must compute tables $\mathcal{N}$ and $\mathcal{M}$ in order to build the new solution.

Once that solution has been built, there is no need to keep tables $\mathcal{N}$ and $\mathcal{M}$, then there is no need to duplicate them $O(d^c)$ times:

for the first assignment of CC, we have to keep the new solution, if one exists, and its distance to the solution *sol* of the previous problem.

For each of the next assignments of CC, we have to keep this one of the two new solutions which is the closest to *sol* and its distance to it.

So the space complexity is $O((m+n)^d)$ for tables $\mathcal{N}$ and $\mathcal{M}$ and $O(n+1)$ for the new solution.

CONCLUSION

We have shown that it is possible to optimize the computation of a new solution to a DCSP in terms of closeness to the former solution; the method is naturally efficient in the acyclic case, but we may not expect its extension to the cyclic case to be such efficient. So the next step of that study could be to try to add efficiency in terms of cost of the computation .

The described algorithms can be extended to other definitions of the closeness: we can assign, to each value of each domain and each tuple of each relation, a weight intented to express a cost or a preference; two solutions are close if the difference between their costs (or preferences) are low. It is then possible to maintain solutions of lower cost or of higher preference.

REFERENCES

[1] Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 221–226, Anaheim CA, 1991. American Association for Artificial Intelligence.

[2] R.P. Brazile and K.M Swigger. Gates : An airline gate assignment and tracking expert system. In *IEEE Expert*, 1988.

[3] Rina Dechter. Enhancement schemes for constraint processing : Back-jumping, learning and cutset decomposition. *Artificial Intelligence*, (41):273-312, 1990.

[4] Rina Dechter and Judea Pearl. The cycle-cutset method for improving search performance in ai applications. In *Proceedings of the Third IEEE Conference on AI applications*, pages 224–230, Orlando, Florida, 1987.

[5] Pascal Van Hentenryck. Incremental constraint satisfaction in logic programming. In *Seventh International Conference on Logic Programming*, pages 189–202, Jerasulem, Israel, 1990.

[6] P. Janssen, B. Nouguier, and M.C. Vilarem. Conception: une approche basée sur la satisfaction de contraintes. In *Les Systèmes Experts et leurs Applications, Neuvièmes journées Internationales*, pages 71–84, Avignon, France, 1 1989.

[7] Philippe Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution. Propagation dans les réseaux dynamiques.* PhD thesis, Université Montpellier 2, 1991.

[8] Rina and Avi Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the Sixth National Conference on Artificial*

Transactions on Information and Communications Technologies vol 2, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Intelligence, pages 37–42, St Paul MN, 1988. American Association for Artificial Intelligence.

[9] Louis I. Steinberg. Design as refinement plus constraint propagation. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 830–835. American Association for Artificial Intelligence, 1987.