Optimiser les réseaux de contraintes temporelles

Maëlle Gautrin

Nous nous intéresserons ici à la résolution des problèmes contraints dans le temps : *Temporal Constraint Satisfaction Problems* (TCSPs). Nous chercherons à modéliser ce problème par un graphe. Puis nous nous intéresserons à un cas particuler de TCSP : les *Simple Temporal Problems* (STPs) en étudiant un algorithme de recherche du plus court chemin dans un graphe. Enfin, nous déduirons de la résolution des STPs, une méthode de résolution et une méthode d'approximation de la solution.

1 Modélisation d'une situation

Soit X_0 et X_1 deux instants distincts. On dit que ces instants sont contraints dans le temps s'il existe un ensemble d'intervalle de temps $\{[a_1;b_1];[a_2;b_2];...;[a_p;b_p]\}$ tel que $X_1 - X_0 \in [a_1;b_1] \cup [a_2;b_2] \cup ... \cup [a_p;b_p]$, ce qui signifie que X_0 et X_1 sont distants d'une durée comprise dans un des intervalles de l'ensemble $\{[a_1;b_1];[a_2;b_2];...;[a_p;b_p]\}$. Cet intervalle est appelé une *contrainte* sur le couple (X_0,X_1).

Un *réseau de contraintes* est un ensemble de variables $\{X_0;X_1;...;X_n\}$ muni d'un ensemble de contraintes $(i_1,j_1,C_1);...;(i_p,j_p,C_p)$ tel que $\forall\, k \in [\![1,p]\!]$,$C_k$ soit une contrainte sur les évenements X_{i_k} et X_{j_k}. Un réseau est donc la représentation d'une série d'évenements contraints dans le temps que l'on appelle : *Temporal Constraint Satisfaction Problems* (TCSPs). Pour simplifier notre problème, nous supposerons, quitte à regrouper les intervalles, que pour chaque contrainte d'un TCSP, les intervalles qui la composent sont deux à deux disjoints.

Un n-uplet $\{t_0;...;t_{n-1}\}$ est une solution de ce réseau si l'affectation $\{X_0 = d_0; X_1 = d_1;...;X_n = d_{n-1}\}$ satisfait toutes les contraintes. Un réseau est dit *cohérent* s'il admet une solution. On dit que v est une solution possible pour X_i s'il existe une solution $\{t_0;...;t_{n-1}\}$ telle que t_i=v. De la même manière, on dit que d est une solution possible de la contrainte entre X_i et X_j si il existe une solution $\{t_0;...;t_{n-1}\}$ tel que t_j - t_i=d. En réduisant chaque intervalle de contrainte à l'ensemble des valeurs qui sont des solutions possibles pour cette contrainte, on obtient le *réseau minimal* du TCSP.

Il est possible de représenter un réseau de contraintes par un *graphe orienté de contraintes*. Pour cela, on affilie à chaque variable temporelle un sommet.

Quand X_i et X_j sont contraints dans le temps, l'ensemble des intervalles les contraignant est représenté par une arête entre X_i et X_j étiquetée par cet ensemble.

Sur l'exemple de la *Figure 1.* On considère 4 instants : X_0, X_1, X_2 et X_3. X_1 survient entre 0 et 10 minutes ou entre 15 et 20 minutes après X_0. X_2 a lieu entre 15 et 20 minutes après X_0. X_3 survient entre 20 et 35 minutes après X_1 mais aussi entre 20 et 25 minutes après X_2

A partir d'un TCSP, l'objectif est tout d'abord de déterminer si le réseau est cohérent, puis de déterminer son réseau minimal et ainsi d'en déduire une solution.

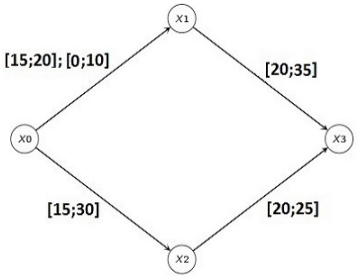


FIGURE 1 – Exemple de graphe de contraintes.

2 Etude d'un cas particulier : les STPs

Dans le cas général, décider de la cohérence d'un TCSP est un problème NP-complet. C'est pourquoi nous nous intéressons d'abord à un cas particulier de celui-ci : le *Simple Temporal Problem*, STP. Ici,

une contrainte est réduite à un seul intervalle.

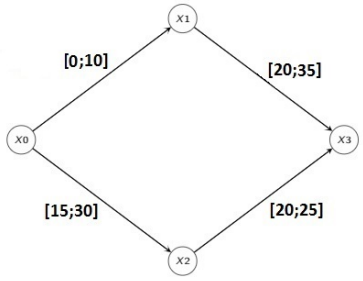


Figure 2 – Exemple d' un graphe de contrainte d'un STP.

2.1 Modélisation par une matrice

On considère un STP ainsi que X_i et X_j deux variables de ce STP contraintes par un intervalle [a,b]. On a l'inégalité donc :
a$\leq X_j$ - $X_i \leq$ b.

Cette inégalité peut aussi se mettre sous la forme :

$$\begin{cases} X_j - X_i \leq b \\ X_i - X_j \leq -a \end{cases}$$

De cette manière, on modifie le graphe de contrainte du TCSP en étiquetant l'arête entre X_i et X_j par la valeur a telle que $X_j - X_i \leq$a. Ce nouveau graphe est appelé *graphe de distances*. Ainsi, les graphes de la *Figure 2* et de la *Figure 3* représentent le même STP.

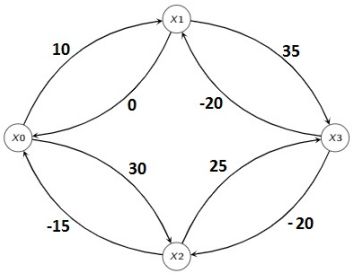


Figure 3 – Graphe de distance du TCSP de la *figure 2*.

A partir de cette représentation du STP, on peut définir sa *matrice de distances* $(m_{i,j})_{i,j\in[0;n-1]}$ associée telle que $m_{i,j}$ est l'étiquette de l'arête entre X_i et X_j dans la représentation sous forme de graphe de distances.

Or pour un STP donné, tous les instants ne sont pas nécessairement contraints deux à deux. Le graphe de distances associé n'est donc pas un graphe complet, et par conséquent la matrice n'est pas entièrement remplie. Pour pouvoir utiliser cette matrice, on la complète en donnant à deux sommets X_i et X_j non contraints, la contrainte $[-\infty;+\infty]$ ce qui n'enlève ni ne rajoute aucune information.

$$\begin{pmatrix} 0 & 10 & 30 & \infty \\ 0 & 0 & \infty & 35 \\ -15 & \infty & 0 & 25 \\ \infty & -20 & -20 & 0 \end{pmatrix}$$

FIGURE 4 – Matrice du STP de la *Figure 2*.

2.2 Utilisation de l'algorithme de Floyd-Warshall

Avec cette modélisation d'un STP, rechercher son réseau minimal revient à une recherche de plus court chemin du graphe associé à ce STP. On va pour cela adapter l'algorithme de Floyd-Warshall, qui consiste en une recherche de plus court chemin dans un graphe, en un *algorithme de cohérence de chemin (Path-Consistency, PC)* pour résoudre notre problème.

Nous avons besoin, pour cet algorithme de définir deux opérations sur nos contraintes :

— Intersection : $\oplus$ définie telle que $[a_1,b_1] \oplus [a_2,b_2] = [\max(a_1;a_2),\min(b_1;b_2)]$.

— Composition : $\otimes$ définie telle que $[a_1,b_1] \otimes [a_2,b_2] = [a_1+a_2,b_1+b_2]$

L'objectif est ici de réduire au maximum la taille de l'intervalle de chaque contrainte. L'algorithme PC consiste à confronter les contraintes entre elles. Par exemple, pour réduire l'intervalle de contrainte $[a_{i,j};b_{i,j}]$ entre X_i et X_j, on choisit un autre sommet X_k : appelé *sommet pivot*. On note $[a_{i,k};b_{i,k}]$ (resp. $[a_{k,j};b_{k,j}]$) l'intervalle de contrainte entre X_i et X_k (resp. entre X_k et X_j) et on compare les deux "chemins". Ainsi : $[a_{i,j};b_{i,j}] \leftarrow [a_{i,j};b_{i,j}] \oplus ([a_{i,k};b_{i,k}] \otimes [a_{k,j};b_{k,j}])$. Concrètement, s'il faut au minimum $a_{i,k}$ minutes entre X_i et X_k et qu'il faut au minimum $a_{k,j}$ minutes entre X_k et X_j alors il faudra au minimum $\max(a_{i,k}+a_{k,j},a_{i,j})$ minutes entre X_i et X_j.

En répétant cette opération avec tous les sommets pivots, l'intervalle $[a_{i,j};b_{i,j}]$ est réduit au maximum. C'est cette étape qui justifie l'intéret de la matrice de distances car elle permet d'accéder rapidement à chaque contrainte et à ne pas distinguer le cas où il existe une contrainte entre X_i et X_j et le cas où il n'y en a pas. Ce raisonnement devient ainsi une suite d'opérations sur les coefficients de la matrice.

Pour notre algorithme PC, tout comme dans l'algorithme de Floyd-Warshall, on fixe d'abord un sommet pivot puis on compare les contraintes imposées à chaque couple de variables X_i et X_j avec la composition des contraintes imposées entre X_i et X_k et entre X_k et X_j.

$$\begin{pmatrix} 0 & 10 & 25 & 45 \\ 0 & 0 & 15 & 35 \\ -15 & -5 & 0 & 25 \\ -35 & -25 & -20 & 0 \end{pmatrix}$$

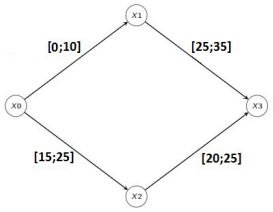


Figure 5 – Réseau minimal du STP de la *Figure 2*.

Cet algorithme PC est en complexité temporelle $\mathcal{O}(n^3)$ avec n le nombre de variables temporelles car on fait varier k,i et j entre 0 et n-1.

Algorithm 1 Cohérence de chemin (PC)

Entrée : matrice ▷ matrice= matrice de distance
Pour k allant de 0 à (n-1) :
  Pour i,j allant de 0 à (n-1) :
    $C_{i,j} \leftarrow C_{i,j} \oplus (C_{i,k} \otimes C_{k,j})$
  Fin pour
Fin pour
Si $C_{i,j}=\emptyset$:
  Failwith "réseau incohérent"
Fin si

Cet algorithme donne aussi la cohérence du réseau. En effet, en reprenant les notations précédentes, si en confrontant les contraintes, on obtient un intervalle $[a_{i,j};b_{i,j}]$ tel que $b_{i,j} < a_{i,j}$, cela signifie que les contraintes sont incompatibles et donc que le réseau est incohérent.

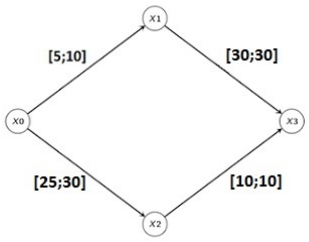
2.3 Obtention d'une solution particulière

Après avoir determiné qu'il existait des solutions à notre STP, c'est à dire qu'il était cohérent puis obtenu le réseau minimal de ce STP. Nous voudrions en exhiber une solution précise, c'est à dire que chaque intervalle ne contienne qu'une seule valeur ou bien que sa matrice de distances soit une matrice antisymétrique.

Nous allons introduire un ordre de priorité sur les variables. Pour simplifier le raisonnement, nous voudrons pour chaque couple de variables, ou bien maximiser la durée entre ces deux évènements, ou bien la minimiser. On va ainsi définir quels intervalles de temps on veut minimiser ou maximiser en priorité. On représente cet ordre de priorité par une liste l=$[(i_1,j_1,x_1);...;(i_p,j_p,x_p)]$ avec x$\in \{0;1\}$ definie telle que : $\forall t \in [\![1;p]\!]$, x_t=0 s'il faut minimiser l'intervalle entre X_{i_t} et X_{j_t} et x_t=1 s'il faut la maximiser. Cette liste va permettre d'automatiser l'obtention d'une solution particulière. Mais il n'est pas possible, simplement à partir du réseau minimal, de choisir la valeur minimale ou maximale de

chaque contrainte. En effet, même si chaque valeur de chaque intervalle est solution, toutes les valeurs ne sont pas compatibles entre elles. Pour pouvoir fixer une à une les valeurs de chaque contrainte, nous réduisons le réseau (avec l'algorithme de cohérence de chemin) à chaque fois que nous avons fixé une valeur. Ainsi, le passage dans l'algorithme supprime toutes les valeurs des autres contraintes incompatibles avec la valeur fixée.

Cherchons une solution particulière du STP de la *Figure 2* : on va par exemple choisir de maximiser en priorité l'intervalle entre X_0 et X_1 puis de maximiser celui entre X_1 et X_2 et de minimiser l'intervalle entre X_0 et X_2 avant de minimser celui entre X_2 et X_3. Ces conditions se traduisent donc par la liste l=[(0,1,1) ;(1,3,1) ;(0,2,0),(2,3,0)]. On obtient ainsi la solution donnée *Figure 6*.



$$\begin{pmatrix} 0 & 10 & 30 & 40 \\ -5 & 0 & 20 & 30 \\ -25 & -20 & 0 & 10 \\ -35 & -30 & -10 & 0 \end{pmatrix}$$

Figure 6 – Solution particulière du STP de la *Figure 2*

3 Résolution du cas général

3.1 Méthode naïve

A partir de cette méthode de résolution du STP, une méthode se déduit naturellement pour le cas général : à partir d'un TCSP, on sélectionne un intervalle par contrainte ce qui nous donne un STP, puis on résout ce STP. On répéte ce processus avec toutes les combinaisons d'intervalles. L'ensemble des solutions du TCSP est donc l'union des solutions de chaque STP. Mais cette méthode fait exploser la complexité de la résolution. En effet, en considérant un TCSP à p contraintes et une moyenne de k intervalles par contraintes, il faut résoudre k^p STP.

3.2 Approximation de la solution

On étudie maintenant une méthode d'approximation du réseau minimal d'un TCSP. L'idée est de mettre en place un algorithme qui supprime pour chaque contrainte les intervalles dont toutes valeurs ne peuvent pas être des solutions.

Pour cela, on va transformer un TCSP en STP en choisissant pour chaque contrainte de la forme C_k={$[a_{1,k};b_{1,k}];...;[a_{p,k};b_{p,k}]$} un unique intervalle C'_k la représentant. On cherche m_k et M_k respectivement le minimum et le maximum de la contrainte qu'on définit tels que : m_k=min{$a_{i,k} \backslash i \in [1,p]$} et M_k=max{$b_{i,k} \backslash i \in [1,p]$}.
On appelle cette opération *Relaxation des contraintes*. De cette manière, l'intervalle qui représente la contrainte C_k sera C'_k=$[m_k ;M_k]$. C'_k contient donc plus de valeurs que l' union des intervalles de la contrainte C_k.

On considère le STP : $\{X_0;X_1;...;X_{n-1}\}$ muni de ses contraintes $\{C'_1 ;...C'_p\}$ tel que $\forall k \in [\![1;p]\!]$, C'_k est la contrainte C_k relaxée comme vu ci-dessus. En réduisant la taille des intervalles de contraintes, c'est à dire en obtenant le réseau minimal de ce STP grâce à la méthode de cohérence de chemin décrite dans la *Partie 2*, il est évident que l'on réduit aussi la taille d'au moins un des intervalles de chaque contrainte de notre TCSP de départ. En effet $\forall k \in [\![1;p]\!]$, C'_k=$[m_k ;M_k]$ est tel que m_k=min{$a_{i,k}$ $\backslash i \in [1,p]$} et M_k=max{$b_{i,k}$ $\backslash i \in [1,p]$} donc il existe un intervalle $[a_{i_0,k} ;b_{i_0,k}]$ et $[a_{i_1,k} ;b_{i_1,k}]$ tel que $a_{i_0,k}$=m_k et $b_{i_1,k}$=M_k. Si le filtrage par arc cohérence réduit C'_k=$[m_k ;M_k]$ en C''_k=$[m'_k ;M'_k]$ alors l'ensemble des valeurs appartenant à $[m_k ;m'_k]$∪[M'$_k$;M_k] sont des valeurs impossibles. Pour réduire la taille des intervalles du TCSP de départ, on peut alors intersecter les contraintes de ce TCSP avec les contraintes de son STP réduit, la complexité de cette méthode de réduction est en $\mathcal{O}(n^3 + ap)$, avec n le nombre de variables du TCSP, a le nombre de contraintes et k le nombre maximum d'intervalles par contrainte. A partir de cette réduction, 3 possibilités se présentent :

— Cas n°1 : $\exists k \in [\![1;p]\!]$ tel que $M'_k < a_{i_0,k}$ alors l'intervalle $[a_{i_0,k}\,;b_{i_0,k}]$ est un intervalle de contraintes impossible donc on l'exclut de nos solutions. Ainsi, si on répète l' opération de relaxation des contraintes sur le TCSP de départ dont la contrainte C_k est privée de l'intervalle $[a_{i_0,k}\,;b_{i_0,k}]$ alors le maximum de la contrainte C_k devient un réel strictement inférieur à $a_{i_0,k}$ ce qui justifie de recommencer ce processus dans le but de réduire encore plus les intervalles de contraintes du TCSP.
— Cas n°2 : $\exists k \in [\![1;p]\!]$ tel que $m'_k > a_{i_1,k}$ analogue au cas n°2.
— Cas n°3 : $\forall k \in [\![1;p]\!]$,M'$_k$ $\leq$a$_{i_0,k}$ et $a_{i_1,k}$ $\leq$m'$_k$ alors $\forall k \in [\![1\,;\mathrm{k}]\!] l'intervalle[\mathrm{a}_{i_0,k}\,;b_{i_0,k}]$ est simplement réduit à $[a_{i_0,k}\,;M'_k]$ et l'intervalle $[a_{i_1,k}\,;b_{i_1,k}]$ à $[m'_k\,;b_{i_1,k}]$. Le minimum et le maximum de chaque contrainte restent donc inchangés en relaxant les contraintes du TCSP dont les intervalles sont réduits. Ce qui signifie que ré-appliquer cette méthode de réduction d'intervalle n'excluera aucune valeur supplémentaire.

De cette méthode se déduit un algorithme appelé "Réduction supérieur-inférieur". Le principe de cet algorithme est à partir d'un TCSP de répéter le processus précédent jusqu'à se trouver dans le *cas n°3*. Comme chaque répétition (sauf la dernière) supprime au moins un intervalle, le nombre d'itérations est inférieur au nombre d'intervalles total du TCSP, c'est à dire inférieur à $a \times p$. Ces intervalles étant en nombre fini, la terminaison de la "Réduction supérieur-inférieur" est prouvée. Ainsi, la complexité de cet algorithme est en $\mathcal{O}(n^3ap + a^2p^2)$.

Algorithm 2 Réduction inférieur-supérieur

```
Entrée : reseau
res3 ← reseau
Tant que res3≠reseau ou ∃ k tel que max(C_k)<min(C_k) :
        reseau ← res3
        res1 ← relaxation(res)
        res2 ← PC(res)
        res2 ← res ⊕ res2
Fin tant que
Si res3 ≠ reseau :
        Failwith"incohérent"
Fin si
```

4 Implémentation en Ocaml

4.1 Représentation des TCSP

Pour coder un TCSP, on va utiliser le type int_inf= E of int| Inf pour représenter une durée finie ou infinie entre deux évenements. Le type utilisé pour coder les TCSP en Ocaml est le type : (int*int*(int_inf*int_inf) list) list. Une liste représentant un TCSP est telle que chaque élément de celle-ci est sous la forme : (i,j,[(E(a_1),E(b_1)) ;... ;(E(a_p),E(b_p))]) et indique que la contrainte entre X_i et X_j est l'ensemble $\{[a_1;b_1];[a_2;b_2];...;[a_p;b_p]\}$. Pour avoir accès au nombre de sommets n sans avoir à parcourir toute la liste en faisant une recherche de maximum, on met comme premier élément de la liste (n,n,[ ]). Ainsi dans le TCSP de l'exemple de la *Figure 1* est codé par la liste :
l=[(4,4,[ ]) ;(0,1,[(E(0),E(10)) ;(E(15),E(20))]) ;(0,2,[(E(15),E(30))]) ;(1,3,[(E(20),E(35))]) ;(2,3,[(E(20),E(25))])].

4.2 Algorithme PC

Chaque contrainte d'un STP est par définition codée par une liste d'un seul élément. Pour simplifier son utilisation, on codera un STP par le type (int*int*int_inf*int_inf) list. De cette manière le STP de la *Figure 2* est codé par la liste :
l=[(4,4,E(4),E(4)) ;(0,1,(E(0),E(10))) ;(0,2,E(15),E(30)) ;(1,3,E(20),E(35)) ;(2,3,E(20),E(25))].

La seule difficulté d'implémentation de cet algorithme est la détection de l'incohérence du réseau. En effet, même si les fonctions d'intersection et de composition se codent simplement grâce à des fonctions auxiliaires donnant le minimum et le maximum de deux éléments du type int_inf, détecter la cohérence en utilisant ce même type nécessite de faire la distinction entre les infinis représentatnt

des $+\infty$ et ceux représentant des -∞. Pour pouvoir comparer $m_{i,j}$ et -mj,i il faut ainsi considérer le cas où $m_{j,i}$=∞.

4.3 Approximation de la solution

L'implémentation en Ocaml se fait ici en utilisant 4 références de réseaux : res pour le TCSP qu'on veut réduire, res1 pour le STP obtenu en relaxant ses contraintes, res2 pour le réseau minimal de ce STP et res3 pour l'intersection entre res et res2.

Pour détecter les changements, on utilise une fonction auxiliaire qui regarde si le minimum ou le maximum a été modifié. C'est aussi cette fonction auxiliaire qui detecte si le réseau est incohérent, c'est à dire si le minimum est superieur au maximum. L'intersection entre res2 et res se fait grâce à une fonction intersection_reseau qui intersecte chaque intervalle de contrainte du TCSP avec l'intervalle unique de la contrainte du STP.

5 Annexes

5.1 Implémentation

5.1.1 Définition des fonctions de base avec le type int_inf

```
type int_inf= Inf|E of int;;


let add x y=match x,y with (*additionne x et y*)
  |Inf,_->Inf
  |_,Inf->Inf
  |E(a),E(b)->E(a+b);;

let mini x y = match x,y with (*renvoye le minimum entre x et y*)
  |Inf,_->y
  |_,Inf->x
  |E(a),E(b)->E(min a b);;

let maxi x y =match x,y with (*renvoye le maximum entre x et y*)
  |Inf,_->Inf
  |_,Inf->Inf
  |E(a),E(b)->E(max a b);;

let minore x y =match x,y with (*indique si x inferieur    y*)
  |Inf,Inf->true
  |Inf,_->false
  |_,Inf->true
  |E(a),E(b)->a<(b);;

let majore x y=match x,y with (*indique si x superieur    y*)
  |Inf,_->true
  |_,Inf->false
  |E(a),E(b)-> a>b;;

let oppose x= match x with (*renvoye l'oppos  de x si x est un entier*)
  |E(a)->E(-a)
  |_-> failwith"infini";;
```

5.1.2 Algorithme PC

```
(* composition et intersection de 2 contraintes *)
let composition it jt is js graphe=
  (add (graphe.(jt).(it)) (graphe.(js).(is))),add (graphe.(it).(jt)) (graphe.(is).(js)
    );;

let intersection reseau (i,j) (a,b)=
  (mini (reseau.(j).(i)) a), (mini reseau.(i).(j) b);;
(**)

(* Pour passer d'une liste representatnt un STP a sa matrice de distances *)
let matrice_adj l=
  let (n,_,_,_)::q=l in
  let m=Array.make_matrix n n Inf in
  let rec aux l= match l with
    |[]->()
    |(r,t,y,u)::p-> m.(r).(t)<-u;
        m.(t).(r)<-oppose(y);
        aux p;
  in
  for i=0 to n-1 do
    m.(i).(i)<-E(0);
  done;
  aux q;
  m,q;;
(**)


(* recherche du reseau minimal pour un STP *)
```

```
let rec pc_failure i j t=
  if i>=j then
    match t.(i).(j),t.(j).(i) with
    |E(a),_ -> minore t.(j).(i) (E(-a))
    |Inf,_ -> false
  else
    match t.(i).(j),t.(j).(i) with
    |Inf,_-> false
    |_,Inf-> false
    |E(a),E(b)-> (-b)>a
;;

let pc graphe=
  let n= Array.length graphe in
  let t= Array.copy graphe  in
  let b=ref true in
    for k= 0 to n-1 do
      for i=0 to n-1 do
        for j=0 to n-1 do
          let intetcomp=intersection t (i,j) (composition i k k j t) in
          t.(i).(j)<-snd(intetcomp);
          t.(j).(i)<-(fst(intetcomp));
          if pc_failure i j t then failwith"non coherent"
        done;
      done;
  done;
  t;;
(**)
```

5.1.3 Recherche d'une solution particulière

```
(*fonction pour fixer une valeur dans un intervalle en le minimisant ou en le
    maximisant*)
let minimiser_valeur graphe i j=
  if (minore (graphe.(i).(j))( E(0))) then
    graphe.(j).(i)<- (oppose graphe.(i).(j))
  else graphe.(i).(j)<- (oppose graphe.(j).(i));
  graphe;;

let maximiser_valeur graphe i j=
  if (minore (graphe.(i).(j))( E(0))) then
    graphe.(i).(j)<- (oppose graphe.(j).(i))
  else graphe.(j).(i)<- (oppose graphe.(i).(j));
  graphe;;
(**)

let rec solution graphe l = match l with
  |[]-> graphe
  |t::q -> let (it,jt,e)=t in
            if e=0 then  solution (pc (minimiser_valeur graphe it jt)) q
            else solution (pc (minimiser_valeur graphe it jt)) q
;;
```

5.1.4 Approximation de la solution d'un TCSP

```

(* renvoye le minimum /maximum d'une liste de contraintes*)

let max_contrainte c =
  let rec aux c max = match c with
    |[]-> max
    |(_,b)::q -> aux q (maxi max b)
  in
  let (_,b)::q=c in
  aux q b;;

let min_contrainte c=
  let rec aux c min = match c with
```

```
    |[]-> min
    |(a,_)::q -> aux q (mini min a)
  in
  let (a,_)::q=c in
  aux q a;;
(**)

(*Op ration de relaxation sur toutes les contraintes d'un TCSP pris en parametre*)
let relaxation res=
  let rec remplir_contr q= match q with
      []->[]
    |(a,b,l)::h-> (a,b,(min_contrainte l),(max_contrainte l))::(remplir_contr h)
  in
  remplir_contr res;;


(*intersection des r seau de contraintes*)
let rec intersection_contrainte_liste a b l= match l with
  |[]-> []
  |(min,max)::q->((maxi a min),(mini b max))::(intersection_contrainte_liste a b q);;

let rec intersection_reseau res2 res= match res2,res with
  |[],[]->[]
  |(m1,m2,a,b)::q1,(_,_,l)::q2->(m1,m2,intersection_contrainte_liste a b l)::(
    intersection_reseau q1 q2);;
(**)

  (* Pour passer de la matrice de distances d'un STP minimal    une liste tout en
    conservant l'ordre de contraites d'une liste l *)
let rec liste_matrice m l=match l with
  |[]->[]
  |(a,b,_)::q->(a,b,oppose(m.(b).(a)),m.(a).(b))::(liste_matrice m q);;
(**)

(*fonction de condition d'arret de la boucle "while*)
let rec detecte_changement res res3=match res,res3 with
  |[],[]->false
  |(_,_,l)::q,(_,_,l3)::q3-> (
      let min3= min_contrainte l3 in
      let min=min_contrainte l in
      let max3=max_contrainte l3 in
      let max=max_contrainte l in
      if (majore min3 max3) then failwith"incoh rent"
      else (min)<>(min3) && (max)<>(max3) && not(detecte_changement q q3))
;;
(**)

let approximation reseau=
  let (n,_,_)::rese=reseau in
  let res= ref rese in
  let res1=ref [] in
  let res2= ref [] in
  let res3=ref rese in
  let b= ref true in
  while !b do
    res:=(!res3);
    res1:=relaxation (!res);
    let m,q=matrice_adj((n,n,E(n),E(n))::!res1) in
    let graphe= pc m in
    res2:=liste_matrice graphe (!res);
    res3:=intersection_reseau (!res2) (!res) ;
    b:= detecte_changement (!res) (!res3);
  done;
  (!res3);;
```