Projet : Problème d'activation de capteurs pour surveillance de zones

1 Présentation

La vidéo-surveillance peut être réalisée avec des capteurs vidéo sans fil possédant des ressources très limitées (batteries à durée de vie limitée). L'idée est de déployer un ensemble de capteurs vidéo sur les zones à surveiller et de proposer un ordonnancement adaptatif de l'activité des capteurs de manière à maintenir la couverture de chaque zone tout en augmentant la durée de vie du réseau. Prenons un exemple simple pour mieux comprendre, imaginons un réseau de 4 capteurs pour surveiller 3 zones. Chaque capteur a une durée de vie spécifiée en unité de temps. Le tableau suivant donne les zones surveillées par chaque capteur.

Capteur	Zones couvertes	Durée de vie (en unité de temps)
s_1	z_1, z_2	$T_1 = 6$
s_2	z_2, z_3	$T_2 = 3$
s_3	z_3	$T_3 = 2$
s_4	z_1, z_3	$T_4 = 6$

On désigne par configuration valide un ensemble de capteurs vidéo qui permet de surveiller l'ensemble des zones. Dans cet exemple, les configurations $(2,4)$,$(1,2)$,$(1,3)$,$(1,4)$,$(1,2,3)$,$(1,2,4)$,$(1,3,4)$,$(2,3,4)$ sont des configurations valides. Les 4 premières sont des configurations valides élémentaires : toutes les zones sont surveillées et il n'y a pas de capteur superflu activé. Cela signifie que si on désactive un capteur de la configuration, certaines zones ne sont pas couvertes.

Supposons que les 4 premières configurations $u_1 = (2,4)$,$u_2 = (1,2)$,$u_3 = (1,3)$,$u_4 = (1,4)$ sont alternativement mises en service pendant un intervalle de temps t_{u_1},t_{u_2}, t_{u_3}, t_{u_4},pour surveiller sans interruption les 3 zones. Nous cherchons à maximiser la somme $t_{u_1} + t_{u_2} + t_{u_3} + t_{u_4}$ pour maximiser la durée de vie du réseau de surveillance tout en respectant les contraintes d'énergie des capteurs. Pour le capteur s_1 par exemple qui appartient aux configurations u_2, u_3 et u_4, on doit avoir $t_{u_2} + t_{u_3} + t_{u_4} \leq T_1$. Le problème d'ordonnancement adaptatif de l'activité des capteurs de surveillance revient à résoudre le programme linéaire (P) :

$$\begin{cases} \max \; t_{u_1} + t_{u_2} + t_{u_3} + t_{u_4} \\ s.c. \\ \qquad\quad t_{u_2} + t_{u_3} + t_{u_4} \leq 6 \\ t_{u_1} + t_{u_2} \qquad\quad \leq 3 \\ \qquad\quad t_{u_3} \qquad\quad \leq 2 \\ t_{u_1} \qquad\quad + t_{u_4} \leq 6 \end{cases}$$

La solution optimale du problème obtenue par la méthode du simplexe est $t^*_{u_1} = 2.5, t^*_{u_2} = 0.5, t^*_{u_3} = 2.0, t^*_{u_4} = 3.5$. Le réseau de surveillance a une durée de vie optimale de 8.5 unités de temps. La configuration 1 est activée pendant 2.5 unités de temps : les capteurs 2 et 4 sont activés durant cette période et les autres sont en veille. Puis la configuration 2 est activée pendant 0.5 unités de temps : les capteurs 1 et 2 sont activés durant cette période et les autres sont en veille. Et ainsi de suite. Au total, le capteur 1 sera activé pendant $t^*_{u_2} + t^*_{u_3} + t^*_{u_4} = 6$ unités de temps. Toute l'énergie de sa batterie sera consommée. Le capteur 2 sera actif pendant $t^*_{u_1} + t^*_{u_2} = 3$ unités de temps. Toute l'énergie de sa batterie sera consommée. Le capteur 4 sera actif pendant $t^*_{u_1} + t^*_{u_4} = 6$ unités de temps. Toute l'énergie de sa batterie sera consommée. Le capteur 3 sera actif pendant $t^*_{u_3} = 2$ unités de temps. Il lui reste de l'énergie.

Le but de ce projet est de spécifier et programmer un ensemble de procédures qui permettent le traitement informatique du problème d'ordonnancement adaptatif expliqué ci-dessus.

2 Travail demandé

2.1 Partie 1 : manipulation des données

On vous demande d'écrire une ou plusieurs procédures capables de gérer les données du problème. Ces données peuvent être lues dans un fichier ou rentrées par l'utilisateur au clavier. Bien entendu, il faudra généraliser l'exemple présenté ci-dessus en considérant un nombre M de zones à surveiller, un nombre N de capteurs. La liste des zones surveillées et la durée de vie pour chaque capteur peuvent être choisies aléatoirement ou fixées par l'utilisateur.

2.2 Partie 2 : construction de configurations élémentaires

Vous devez proposer une ou plusieurs heuristiques permettant de construire un ensemble de configurations élémentaires.

2.3 Partie 3 : écriture et résolution du programme linéaire

Une fois les configurations élémentaires construites, on vous demande d'écrire le programme linéaire correspondant au problème d'ordonnancement et de le résoudre avec un solveur de programme linéaire (par exemple GLPK - logiciel libre - sous linux). On vous demande d'écrire ensuite la solution optimale.

2.4 Partie 4 : analyse des résultats

Pour un même problème, vous devez examiner l'influence du nombre et du type des configurations élémentaires choisies sur la durée de vie du réseau.

3 Obligations et recommandations

Vous devrez rédiger un rapport de 1 page recto/verso qui traitera toutes les questions. Vous êtes libres d'enrichir à votre guise le projet à condition que toutes les questions posées soient déjà traitées. Ce projet est à rendre pour le lundi 26 Mars 2010 à 10H30 (Groupe A2) et mardi 27 mars (Groupe A1). Vous présenterez votre travail lors d'une soutenance de 15 minutes (10 minutes de présentation).

4 Utilisation du solveur GLPK

Vous pouvez utiliser le solveur pour résoudre un programme linéaire écrit dans un fichier sous un format donné (ex format CPLEX).

4.1 Création du fichier au format CPLEX

Pour résoudre un programme linéaire avec le solveur GLPK, vous devez d'abord écrire ce programme dans un fichier d'extension .lp (nomdufichier.lp)avec le formalisme suivant :
Maximize $t_1 + t_2 + t_3 + t_4$

Subject To

$t_2 + t_3 + t_4 <= 6.000000$
$t_1 + t_2 <= 3.00000$
$t_3 <= 2.00000$
$t_1 + t_4 <= 6.000000$

END

4.2 Commandes en ligne

Pour exécuter la résolution du programme linéaire écrit dans le fichier programme.lp, il faut taper sous unix la commande :
glpsol –cpxlp programme.lp -o solution
La solution du problème est sauvegardée dans le fichier "solution".

4.3 Commandes intégrées dans un programme en langage C

Pour appeler une fonction du solveur GLPK dans un programme C,
- il faut inclure le .h :#include "glpk.h"
- écrire le programme dans un fichier d'extension .lp

```
const char *intlpfilename="nomdufichier.lp";
FILE *ofile;

if((ofile=fopen(intlpfilename, "w"))==NULL){
cerr«"error! could not create "«intlpfilename«endl;
return false;
}

fprintf(ofile, "Maximize ");
for(u=0;u<Nbconfig;u++) fprintf(ofile," t_fseek(ofile, -1, 1);
fprintf(ofile,"\n \n Subject To\n \n");
fprintf(ofile, "\n \\* Contraintes temporelles————————————————————\n");
for(k=0;k<Nbcapteurs;k++)
{...}

fprintf(ofile, "\nEND\n");
fclose (ofile);
```

- résoudre le programme linéaire

```
LPX *lp;
lp =lpx_create_prob();
lpx_set_prob_name(lp, "PL");
lp = lpx_read_cpxlp(intlpfilename);
lpx_simplex(lp);
```

- récupérer la solution

```
for(u=0;u<Nbconfig;u++) {Valprimal[u]=lpx_get_col_prim(lp,u+1)};

lpx_delete_prob(lp);
```

4.4 Interprétation de la solution

Les premières lignes du fichier "solution" sont :
Problem :
Rows : 4
Columns : 4
Non-zeros : 8
Status : OPTIMAL
Objective : obj = 4 (MAXimum)

- Rows désigne le nombre de contraintes
- columns désigne le nombre de variables
- Non-Zeros désigne le nombre de coeeficients non nuls dans la matrice des contraintes
- Status indique si une solution optimale a été trouvée

- Objective donne la valeur de la fonction objectif à l'optimalité

Le fichier "solution" contient deux tableaux dont voici l'interprétation des colonnes.

- Pour la solution
  - no numero de la variable
  - column name : nom de la variable
  - St : statut (en base B, hors base HB)
  - Activity : valeur prise par la variable
  - Lower Bound : borne inférieure de la variable 0
  - Upper Bound : borne supérieure (non precisée)
  - Marginal : si une variable x_i entre en base, la fonction objectif varie de marginal*x_i.
- Pour les contraintes
  - no numero de la contrainte
  - row name : nom de la contrainte
  - St : statut (B : contrainte non saturée-lache, NU : contrainte saturée)
  - Activity : valeur du membre de gauche de l'inegalité
  - Lower bound : y en a pas,
  - Upper bound : valeur du second membre de l'inegalité
  - Marginal : c'est nul pour les contraintes lâches, sinon si on augmente le second membre b_i d'une contrainte saturée i d'une quantite t_i, alors la fonction objectif varie de t_i*marginal.