BONNAL Valentin
RAIMONDI Gautier

UNIVERSITÉ DE FRANCHE-COMTÉ

Master 1 Informatique

Graph
Problème de flots max

11-12-18

Table des matières

1. Introduction

Cette seconde partie du projet a pour objectif de mettre en place une gestion des flots sur le projet précédemment développé qui consistait à la représentation en la représentation de graph sous forme de liste chainée.

La gestion des flots devant permettre la découverte de flots maximal pour les graphs renseignés, cette recherche se fait par l'utilisation d'heuristiques destinée à détecter « la meilleure » chaine améliorante à un état E jusqu'à atteindre le débit maximal du graph. Les différentes heuristiques, qui permettent cette recherche, ayant chacune leur limite, il nous a été proposé d'en implémenter plusieurs afin de comparer leurs résultats et ainsi leur efficacité en fonction des cas particuliers et des types de graph qu'il est possible de rencontrer.

2. Fonctionnement

Pour définir le flot max, nous disposons de deux méthodes destinées à mettre à jour les graphs résiduels avec les nouveaux flots prétendus maximaux d'un état E.

```
int flot_max(struct Graph g, int source, int puit, struct amel (*f)(struct Graph,int**,int,int)){
        …

        while (true){
                struct amel chaine_amel = f(g,flot,source,puit);
                if (chaine_amel.succes){
                        struct val_flot val = maj_flot(F, flot, g, chaine_amel);
                        for (i = 0;i<N;i++){
                                for (j = 0;j<N;j++){
                                        flot[i][j] = val.flot[i][j];
                                }
                        }
                        F = val.val;
                } else {
                        break;
                }
        }
        …

        return F;
}
```

Cette méthode a comme objectif pour chaque chaine améliorante réussite/existante de mettre à jour le graph via l'appel à maj_flot(), et refaire appel à une heuristique pour trouver une autre chaine améliorante, et ainsi de suite.

À noter que la méthode heuristique est passée en paramètre de la méthode flot max ce qui permet une indépendance entre ces deux.

3. Heuristiques

La première heuristique réalise un parcours en largeur pour essayer de trouver la meilleure chaine améliorante.

```
struct amel bfsMethod(struct Graph g, int** flot,  int source, int puit, int** (*f)(struct Graph,int**,int,int)){
	int N = g.nbMaxNodes+1;
	int *chn = malloc(N * sizeof(int));
	int *orig = malloc(N * sizeof(int));
	int *ec = malloc(N * sizeof(int));
	int *marque = malloc(N * sizeof(int));

	...

	int succes = 0;
	int head = 1;
	int queue = 1;
	chn[1] = source;
	orig[1] = NAN;
	ec[1] = INFINITY;
	marque[source] = 1;

	while (!succes && queue-head>=0){
		int nc = chn[head];
		struct cchainedList** ls = g.list;
		struct cchainedList* l = ls[nc];
		l = l->next;
		while (l->value != -1){
			int nn = l->value;
			if (!marque[nn] && flot[nc][nn] < l->secondValue){
				marque[nn] = 1;
				chn[queue+1] = nn;
				queue++;
				orig[queue] = nc;
				ec[queue] = MIN(ec[head],l->secondValue-flot[nc][nn]);
				if (nn==puit){
					succes = 1;
					break;
				}
			}
			l = l->next;
		}
		head++;
	}

	...
}
```

Les noeuds visités sont marqués dans le tableau marque, le delta est lui stocké dans le tableau ec et l'origine, ou nœud parent, sont stockés dans le tableau. En fonction du succès

ou de l'échec de cette recherche, la recherche de flots max s'arrête, estimant qu'il n'y a plus de chemin possible.

La deuxième catégorie d'heuristique correspond à l'utilisation d'algorithmes afin de trouver un plus court chemin entre la source et le puit dans le graphe résiduel et d'en déduire une chaîne améliorante. On utilise pour cela la fonction suivante :

```
struct amel shortestMethod(struct Graph g, int** flot, int source, int puit, int** (*f)(struct Graph,int**,int,int)){
	int N = g.nbMaxNodes+1;
	int *chn = malloc(N*sizeof(int));
	int *orig = malloc(N*sizeof(int));
	int *ec = malloc(N*sizeof(int));
	int *marque = malloc(N*sizeof(int));

	...

	ec[0] = INFINITY;
	int ** chm = f(g, flot, source, puit);
	i = 0;
	while (chm[i][0] != -1){
		i++;
	}
	i--;
	int im = i+1;
	while (i>=0){
		chn[im-i]=chm[i][0];
		orig[im-i]=chm[i+1][0];
		ec[im-i]=MIN(ec[im-i-1],chm[i][1]);
		i--;
	}
	...

}
```

Le succès de la recherche de la chaîne améliorante se traduit alors par chm[0][0] != -1 et le delta est calculé puis stocké dans ec, comme le minimum entre le flot sauvegardé jusqu'ici et le flot disponible dans l'arrête à venir.

Deux différents algorithmes de recherche de chemin ont alors été implémentés : Djikstra et Floyd-Warshall, ils sont passés en paramètre de la fonction par (*f).

4. Résultats et Comparaison

Le programme permet également de comparer les différents temps d'exécution pour chaque heuristique, on obtient alors les résultats suivants : Si la méthode BFS s'exécute en *x* ms, Djikstra s'exécute en environ 2.5*x* ms et Floyd-Warshall en 5*x* ms.