Couplages et flots

Jill-Jênn Vie Christoph Dürr

8 octobre 2015

Types de problèmes à SWERC 2014

- Exploration exhaustive (2)
- Plus courts chemins (2)
- Programmation dynamique (1)
- Recherche de motifs (1)
- **Couplages et flots** (1)
- Géométrie (2)
- Fast Fourier Transform WTF (1)

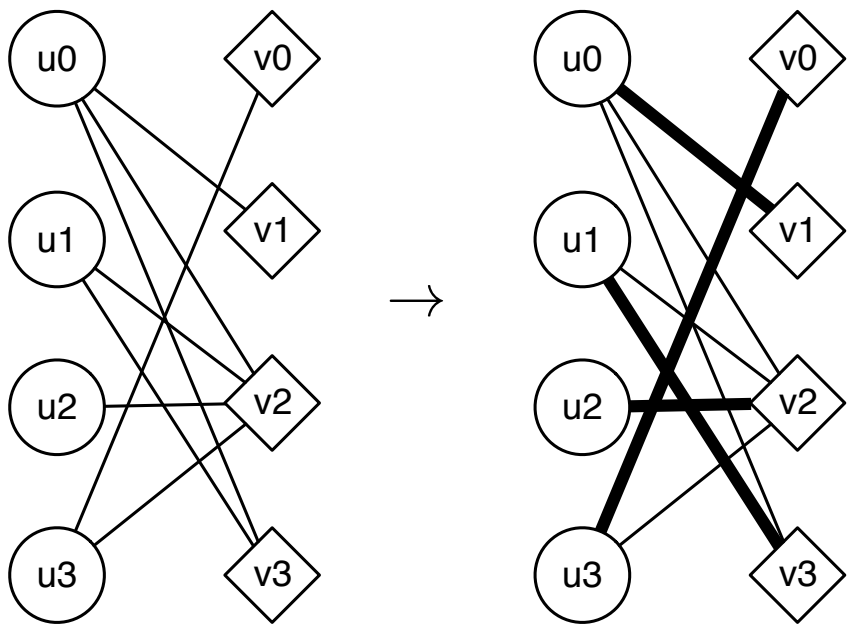
Types de problèmes à SWERC 2013

- Structures de données (tas, tables de hachage) (2)
- Plus courts chemins (1)
- Exploration exhaustive (1)
- Mathématiques (1)
- Recherche de motifs + mathématiques (1)
- Programmation dynamique (dont réécriture WTF) (2)
- **Couplages et flots** (1)
- Géométrie (1)

Couplage maximum dans un graphe biparti

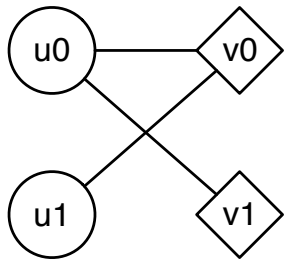
Entrée U : sommets de départ
V : sommets d'arrivée
E : arêtes

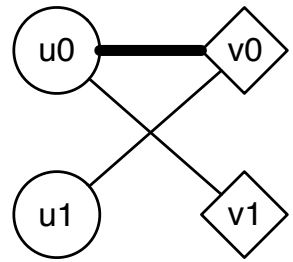
Sortie Un ensemble d'arêtes $M \subset E$ d'extrémités disjointes dans U et V, de cardinal maximum.

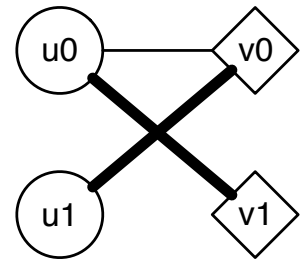


Idée

Améliorer progressivement un couplage de manière gloutonne.



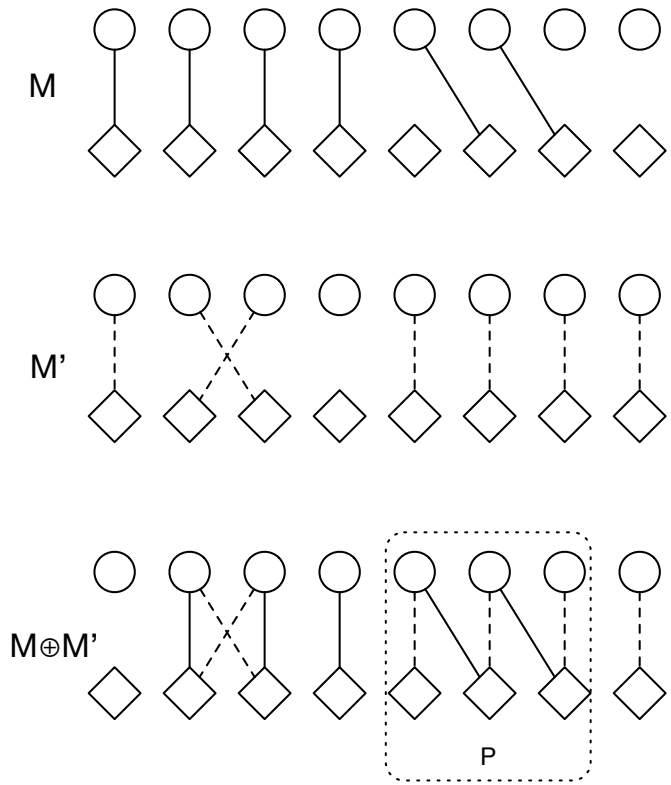




Observation clé

Un chemin augmentant est un chemin qui alterne entre pris / pas pris dans le couplage courant.

Si j'ai deux couplages M et M' tels que $|M| \leq |M'|$, il existe un chemin augmentant.



On cherche un tel chemin par DFS.

Code

```
bool augment(int u) {
    int v;
    for(int i = 0; i < graph[u].size(); ++i) {
        v = graph[u][i];
        if(!visit[v]) {
            visit[v] = true;
            if(match[v] == -1 || augment(match[v])) {
                match[v] = u;
                return true;
            }
        }
    }
    return false;
}
```

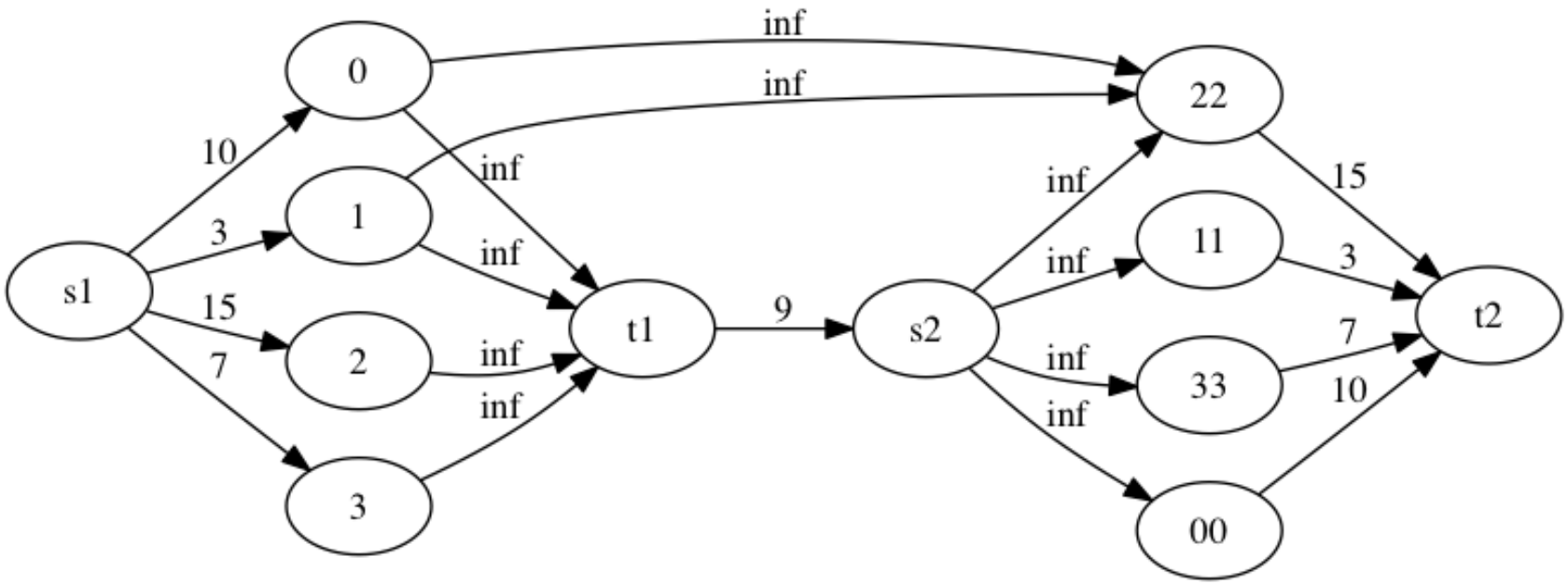
```
for(int i = 0; i < n; ++i)
    match[i] = -1;
for(int i = 0; i < n; ++i) {
    for(int j = 0; j < n; ++j)
        visit[j] = false;
    augment(i);
}
```

Flot maximum

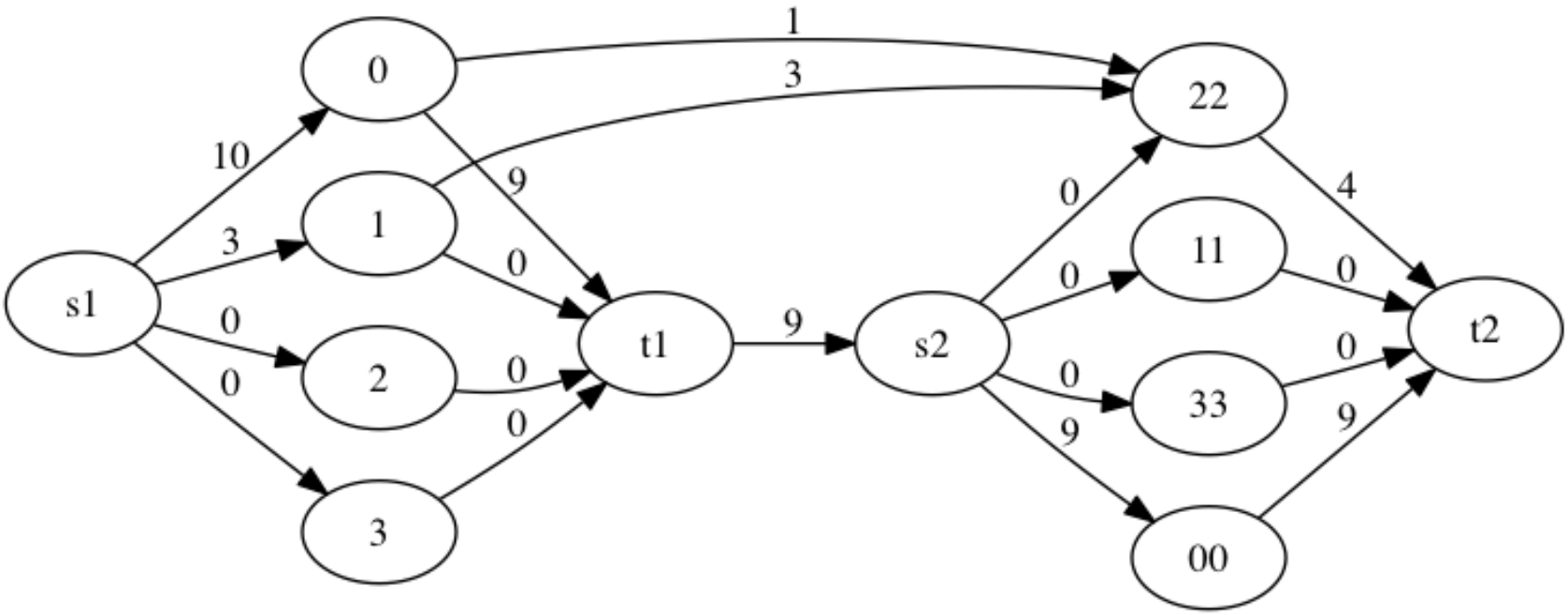
Entrée Un graphe orienté avec des capacités sur les arêtes

Sortie Un flot selon chaque arête, ne dépassant pas les capacités

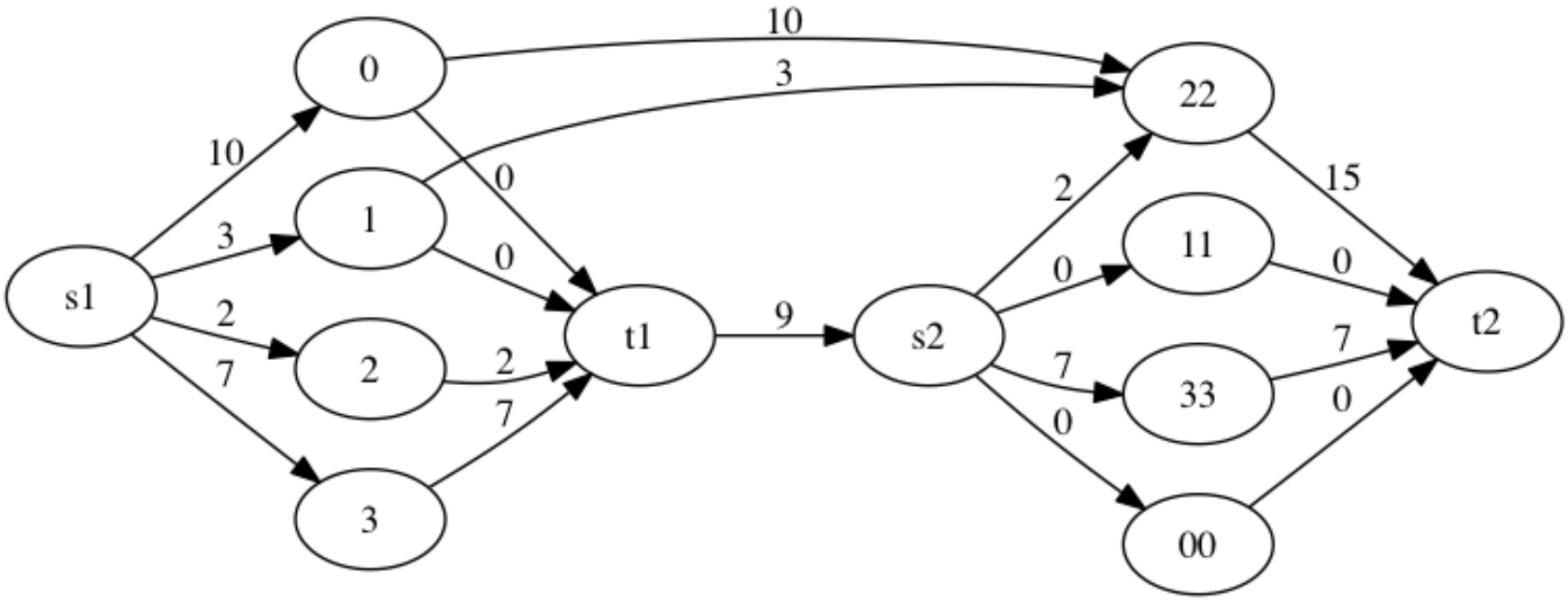
Contraintes En chaque nœud, le flot entrant = le flot sortant



Un exemple de flot



Un flot maximum



Algorithme de Ford-Fulkerson

```
bool augment(int u, int val) {
    visit[u] = true;
    if(u == 2 * n + 3)
        return val;
    int v, res, delta, cuv;
    for(int i = 0; i < graph[u].size(); ++i) {
        v = graph[u][i];
        cuv = capacity[u * SIZE + v];
        if(!visit[v] && cuv > flow[u * SIZE + v]) {
            res = min(val, cuv - flow[u * SIZE + v]);
            delta = augment(v, res);
            if(delta > 0) {
                flow[u * SIZE + v] += delta;
                flow[v * SIZE + u] -= delta;
                return delta;
            }
        }
    }
    return 0;
}
```

```
do {
    for(int i = 0; i <= t2; ++i)
        visit[i] = false;
} while(augment(s1, 1e9) > 0);
```