TIPE : Optimisation de flot sur un réseau de transport

Youssef IRHBOULA

2018-2019

On part d'une situation réelle : un réseau de transport est endommagé, on souhaite le réparer, plusieurs réparations sont possibles et le budget est limité.
Comment peut-on mettre en place un algorithme qui optimise la circulation dans ce réseau ?

Définition 1.1.1

Un graphe de flot est un graphe (S,A) dont deux sommets sont particuliers : la source s et le puits p. Il est muni d'une fonction de capacité c de $A \to \mathbb{R}_+$ et d'un flot.

Définition 1.1.2

Un flot est une fonction f de $A \to \mathbb{R}+$ qui vérifie les propriétés suivantes :

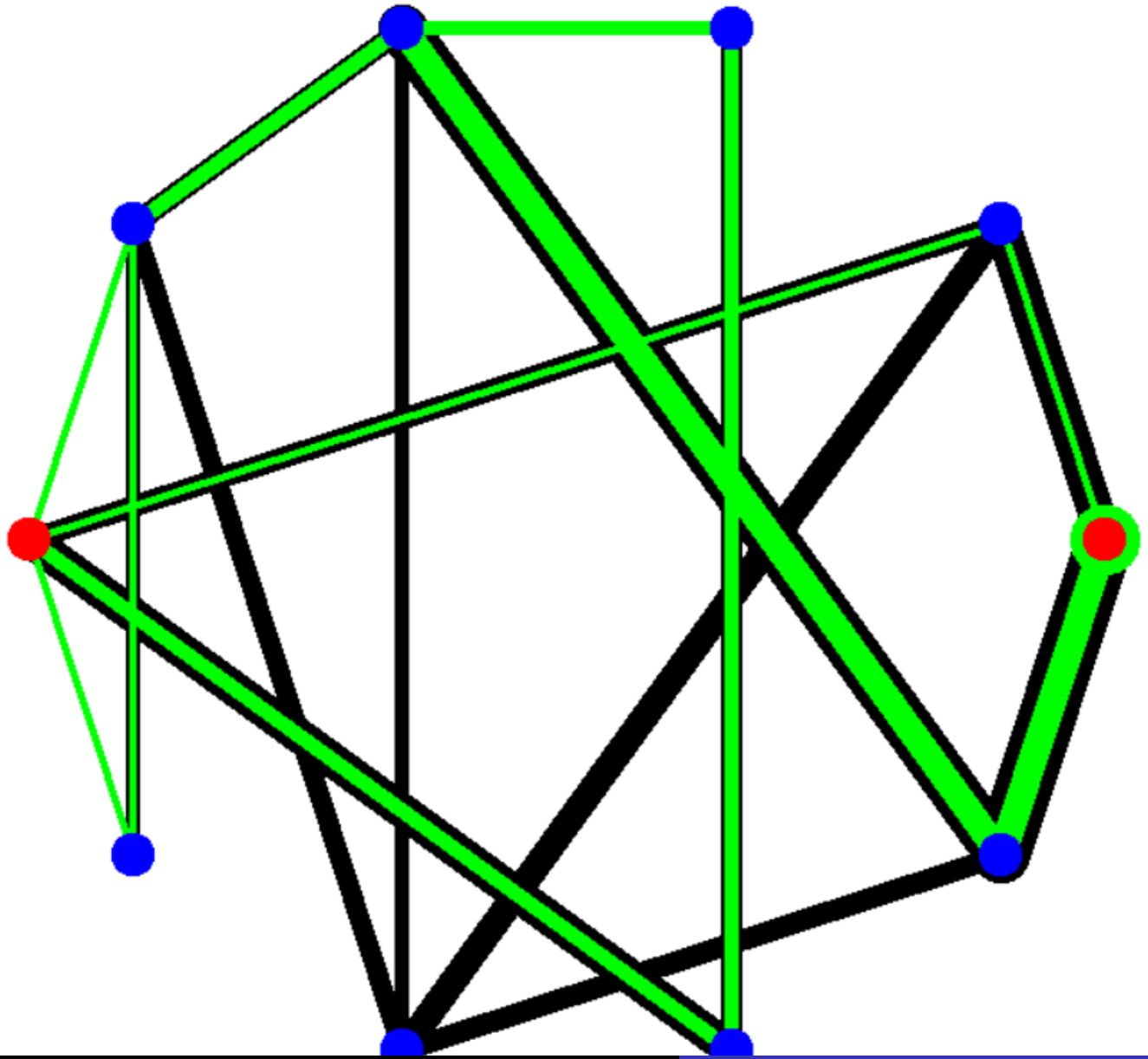
- *antisymétrie*
- *contrainte de capacité : $\forall\ u,v \in S,\ 0 \leq f(u,v) \leq c(u,v)$*
- *loi des noeuds (Kirchhoff) : $\forall\ u \in S$ -{s,p} ,*

$\sum_{v \in S} f(u,v) = \sum_{v \in S} f(v,u)$

Définition 1.1.3

On définit la valeur d'un flot, notée $|f|$, par :

$|f| = \sum_{v \in S} f(s,v)$



Définition 1.2.1

Pour G=(S,A,c,f) un graphe de flot donné, on définit son graphe résiduel $G_f = (S, A, c_f, 0)$ *où* $c_f(u, v) = c(u, v) - f(u, v)$.

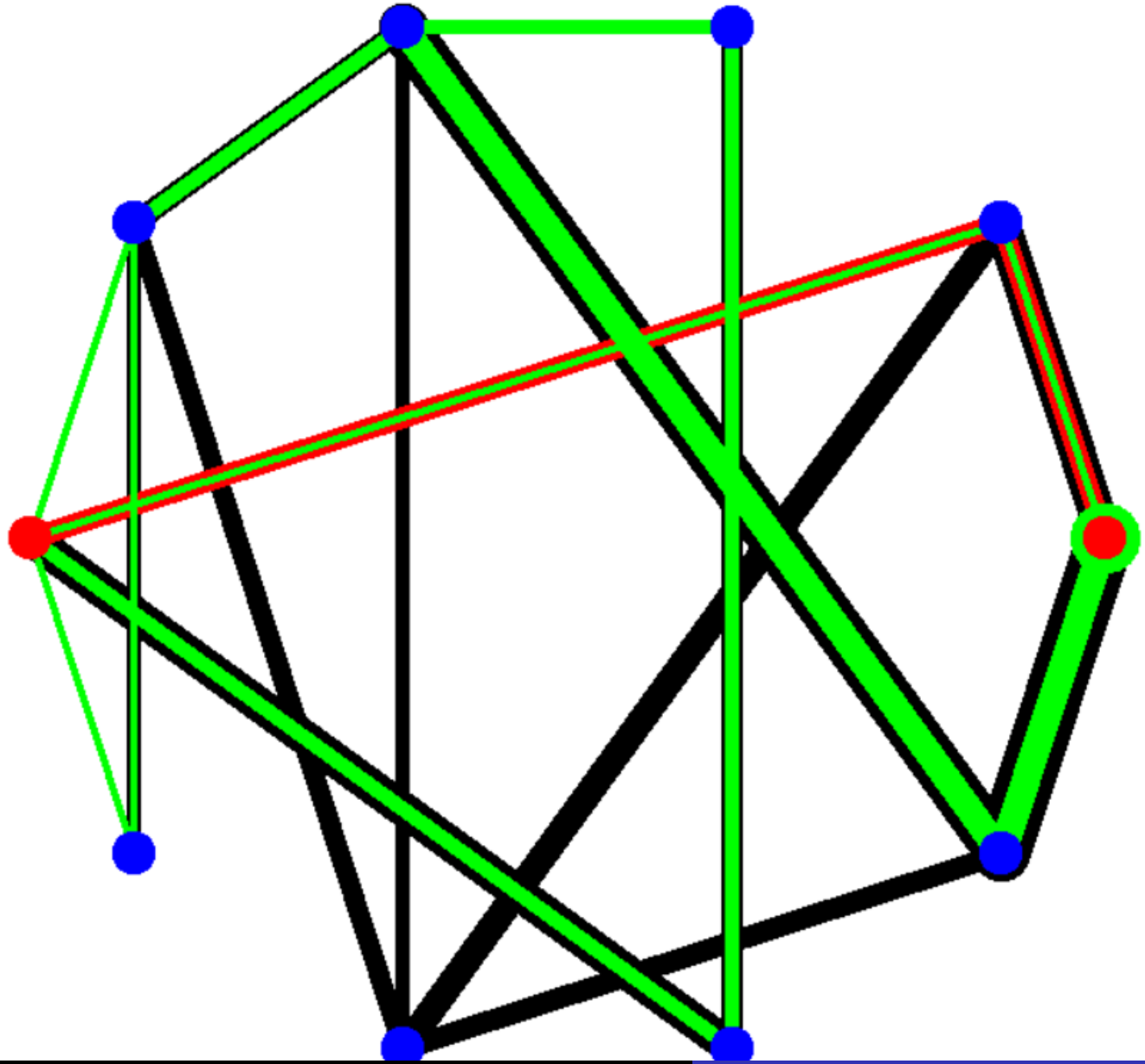
On vérifie aisément que G_f est bien un graphe de flot

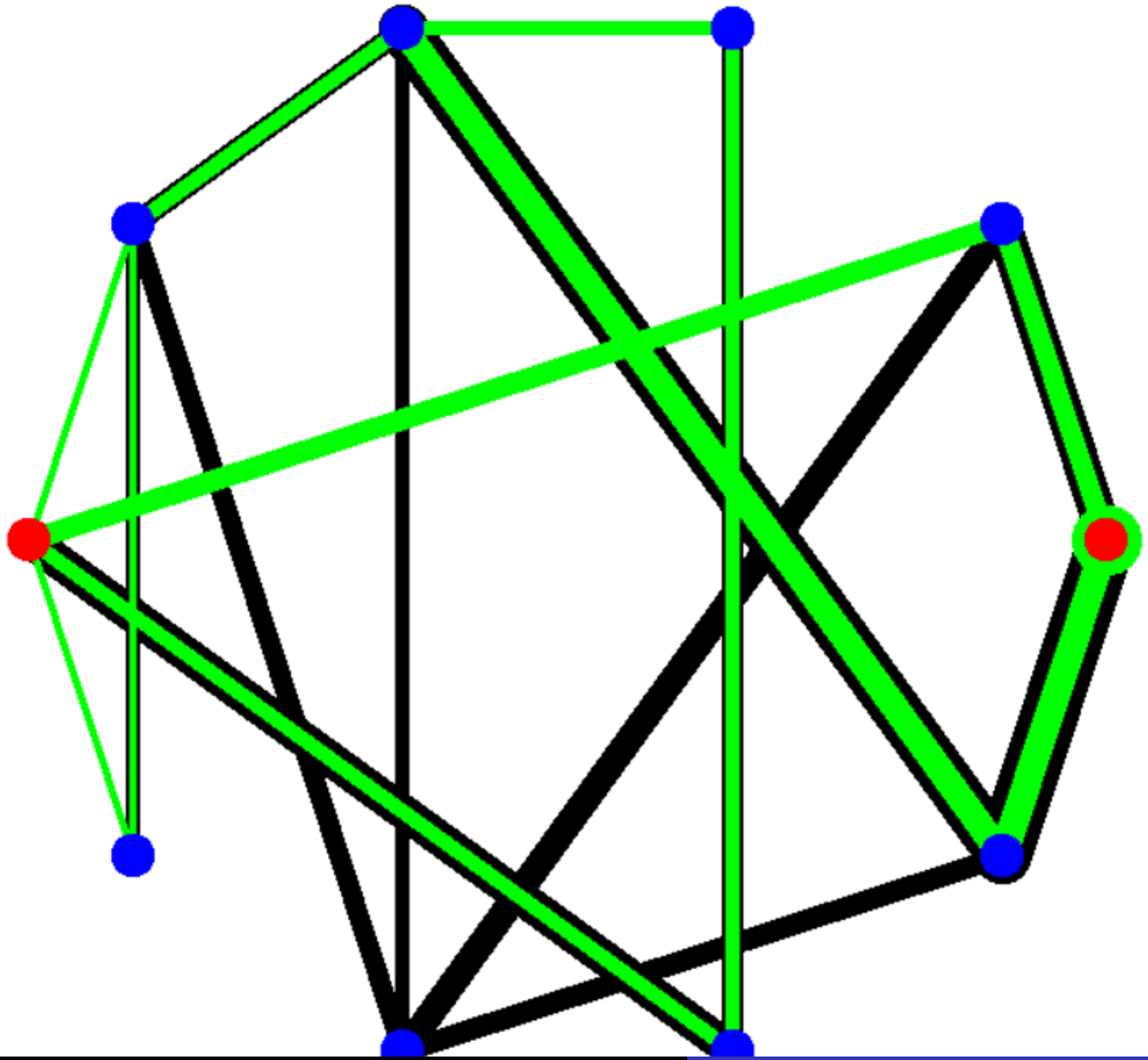
Proposition 1.2.2

Soit G=(S,A,c,f) un graphe de flot, et f' un flot sur G_f. *f+f' est un flot sur G et* $|f + f'| = |f| + |f'|$.

Définition 1.2.3

Un chemin r entre les sommets u et v est une suite finie d'arêtes (u_k, v_k) *telle que* $\forall k \leq n-1$ $u_{k+1} = v_k$, u_0=u *et* v_n=v.
On définit sa capacité par $c(r)$=$min_{0\leq k\leq n}(c(u_k, v_k))$





Proposition 1.2.4

Soit G un graphe doté du flot f. Soit r un chemin de s à p. On a alors f_r de S^2 dans $\mathbb{R}_+$ défini par :
f(u,v)=c(r) pour $(u,v) \in r$
f(u,v)=0 sinon

Théorème Maxflow/Mincut

Soit G=(S,A,c,f) un graphe de flot. S'équivalent :
1) f est un flot maximal
2) Il n'y a pas de chemin entre s et p dans G_f
3) Il existe une coupe de capacité $|f|$

Preuve en annexe

Données : G un graphe de flot
Résultat : f le flot maximal pour G

```
1 f ← 0
2 tant que il existe un chemin améliorant r faire
3 |   f ← f_r
4 fin
5 retourner f
```

Proposition 2.1.2

$\forall v \in S$
La longueur du plus court chemin de s à v dans G_f augmente de façon monotone avec les augmentations de flot

Théorème 2.1.3

Si la recherche du chemin se fait par un parcours en largeur, le nombre d'augmentations de flot est $O(|S||A|)$

Définition 2.2.1

Un préflot vérifie les mêmes propriétés que le flot sauf qu'au lieu d'avoir conservation du flot on a simplement un flot entrant supérieur au flot sortant :
$\forall u \in S$, $e(u)=\sum_{v \in S} f(u,v) \geq 0$
On appelle e(u) l'excédent de flot, et un sommet est dit débordant si $e(u)>0$.

Définition 2.2.2

On définit une fonction hauteur h dans un graphe $G=(S,A,c,f)$ *où f est un préflot par :*
$h(s)=S$, $h(p)=0$ *et h vérifie :* $\forall (u,v) \in A_f \ h(u) \leq h(v)+1$

Données : (u,v) une arête avec u débordant plus haut que v, $c_f(u,v) > 0$
Résultat : pousser de u vers v

1 $\Delta \leftarrow min(e(u), c_f(u,v)$
2 $e(u) \leftarrow e(u) - \Delta$
3 $e(v) \leftarrow e(v) + \Delta$
4 $f(u,v) \leftarrow f(u,v) + \Delta$
5 $f(v,u) \leftarrow f(v,u) - \Delta$

Données : u un sommet débordant de hauteur inférieure à tous ses voisins dans G_f
Résultat : met à jour la hauteur de u

1 $h(u) \leftarrow min_{(u,v)\in A_f}(h(v)) + 1$

Données : G=(S,A,c,**0**) un graphe de flot
Résultat : f un flot maximal sur G

initialisation
pour $u \in S$ **faire**
    $h(u), e(u) \leftarrow 0, c(s,u)$
    $f(s,u), f(u,s) \leftarrow c(s,u), -c(s,u)$
fin
tant que *pousser ou réétiqueter sont applicables* **faire**
    appliquer une poussée ou un réétiquetage possibles
fin

Borne pour les réétiquetages

Le nombre d'opérations de réétiquetages est un $O(|S|^2)$

Borne pour les poussées saturantes

Le nombre de poussées saturantes est un $O(|S||A|)$

Borne pour les poussées non saturantes

Le nombre de poussées non saturantes est un $O(|S|^2|A|)$

Ainsi si on implémente les opérations élémentaires en O(1) on a une complexité totale en $O(|S|^2|A|)$

On dispose d'une liste d'objets de poids et de valeurs données et d'un sac qui peut contenir un poids maximal. On cherche à maximiser la valeur contenue dans le sac.

Données : w,v listes des poids et valeurs des n objets, W_{max} le poids maximal

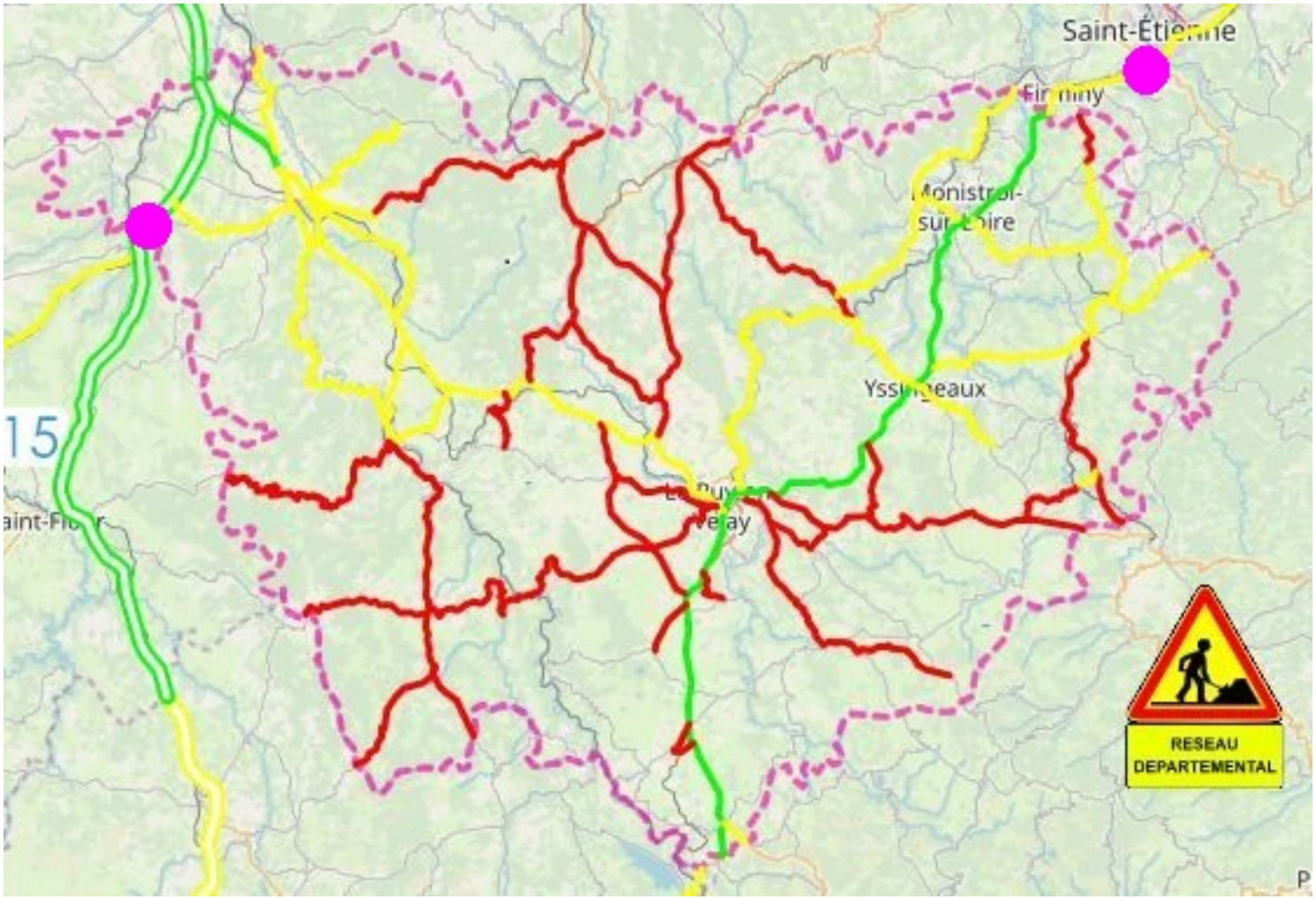
Résultat : la valeur maximale que peut contenir le sac

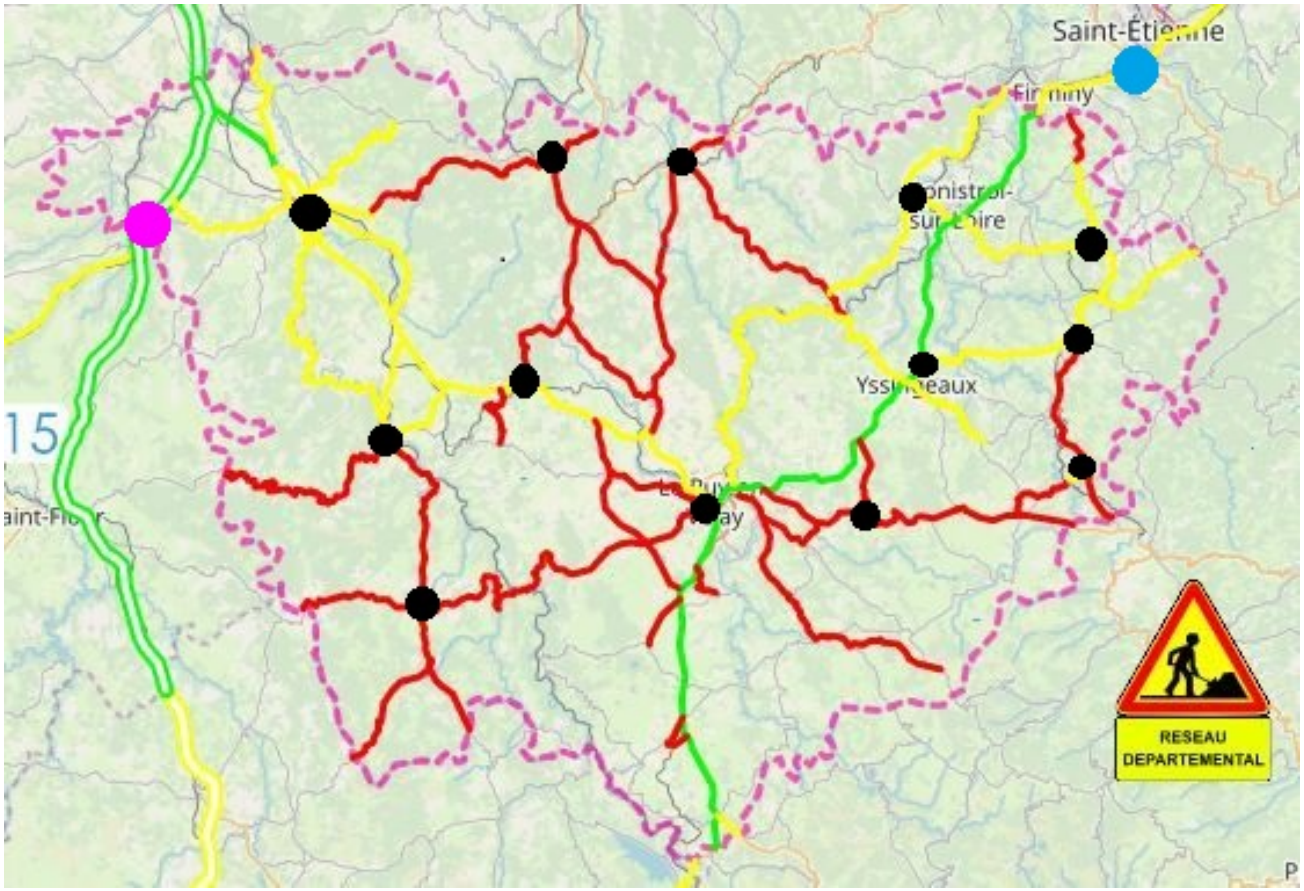
```
1  créer K une matrice (n+1)×(W_max+1) de zeros
2  pour i allant de 0 à n faire
3  |   pour j allant de 0 à W_max faire
4  |   |   si w(i)¿j alors
5  |   |   |   K(j,i+1)←K(j,i)
6  |   |   fin
7  |   |   sinon
8  |   |   |   K(j,i+1)←max(K(j,i),K(j-w(i),i)+v(i)))
9  |   |   fin
10 |   fin
11 fin
```

Correction et complexité

$\forall i, j$ K(i,j) donne la valeur maximale pour le poids maximal i et en les j premiers objets.
L'algorithme a une complexité temporelle en $O(nW_{max})$

Voici une carte montrant des routes enneigées. Ces routes relient Saint-Etienne à l'autoroute A75 en passant par différents villages.





Les usagers se déplacent dans un sens particulier (exemple : vacances).
On a donc bien une source et un puits.
En comptant le trafic en milliers, les arrêts et départs de villages sont nuls donc la conservation de flot est vérifiée.
Le modèle de graphe de flot est valide.

Les routes ont des capacités sont diminuées à cause des intempéries. Le déblaiement a un coût et apporte une capacité supplémentaire à l'arête déblayée. Le flot peut donc augmenter ce qui donne donc les valeurs des travaux.
Mais on ne peut appliquer directement l'algorithme du sac à dos car les valeurs ne sont pas additives.

On applique un algorithme analogue au sac à dos où la valeur est donnée par la valeur du flot maximal. Le calcul du flot en tenant compte des améliorations proposées s'implémente :

```
let upgradeflow_sum i j t g f=
    let f0=maxflow f g in
    let gi=g++t.(i)in
    let gij=gi++t.(j)in
    let fi=maxflow f0 gi in
    maxflow fi gij;;

let rec upgradeflow_sum2 t g f=function
    |[]->f
    |[x]->maxflow f (g++t.(x))
    |[i;j]->upgradeflow_sum i j t g f
    |i::q->upgradeflow_sum2 t (g++(t.(i)))
     (maxflow f (g++(t.(i)))) q;;
```

On a alors l'algorithme suivant :

```
let flowknapsack g t budget=
    let n= Array.length t in
    let kv= Array.make_matrix (budget+1) (n+1) (zero n) in
    let kf= Array.make_matrix (budget+1) (n+1) 0 in
    for i=0 to n-1 do
        for j=0 to budget do
            if t.(i)>j then
                kf.(j).(i+1)<-kf.(j).(i)
                kv.(j).(i+1)<-kv.(j).(i)
            else
                let f,v= upgradeflow_sum kf.(j-t.(i)), flowvalue f in
                if v>kv.(j).(i) then
                    kv.(j).(i+1)<-v
                    kf.(j).(i+1)<-f
                else
                    kf.(j).(i+1)<-kf.(j).(i)
                    kv.(j).(i+1)<-kv.(j).(i)
                done
        done;
        k.(budget).(n);;
```

Complexités

Le calcul initial du flot maximal se fait en $O(|S|^2|A|)$.
La complexité des boucles est en $O(nW_{max})$ augmentations de flot.
Chaque augmentation de flot se fera en utilisant l'algorithme de Ford-Fulkerson initialisé au flot précédent, qui sera mémoïsé dans K. Il y a donc au plus 1 chemin améliorant, ce qui donne une complexité en $O(|A|)$

On a alors une complexité totale en $O(|A|(|S|^2 + nW_{max}))$

Généralisation

Le problème de flot maximal sur un graphe à sources et/ou puits multiple est équivalent au problème de flot maximal sur un surgraphe de flot avec une supersource (resp. un superpuits) relié aux sources de départ (resp. aux puits) par des arêtes de capacité infinie.

Notre algorithme peut donc traiter les situations plus générales en construisant le surgraphe tout en gardant une même complexité asymptotique. (construction du graphe en $O(|S| + |A|)$)

Bibliographie :

◦ T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein : *Introduction to Algorithms third edition* : The MIT Press, 2009, Chapitre : V-Graph algorithms, 26-Maximum Flow

◦ L.R. Ford Jr, D.R. Fulkerson, : *Flows in Networks* : The Rand corporation, 1962

◦ S. Even : *Graph Algorithms* : Cambridge University Press, 1979, Chapitres : 5-Flows in Networks, 6-Application of Network Flow techniques

◦ A. V. Goldberg, E. Tardos, R.E. Tarjan : *Network Flow Algorithms* : Cornell University, 1989, Chapitre : 2-The Maximum flow problem

◦ J. Edmonds, R.M. Karp : *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems* : Journal of the Association for Computing Machinery, 1972

◦ R. Garfinkel, G.L. Nemhauser : *Integer Programming* : Springer, 1972, Chapitre : 13.5.4- Reformulations based on dynamic programming

Définition 1.3.1

Une coupe est une partition de S en deux ensembles E et P tels que $s \in E$ *et* $p \in P$
On définit alors la capacité d'une coupe :
$c(E,S)=\sum_{u\in E}\sum_{v\in P}c(u,v)$
On définit de même le flot à travers une coupe

Preuve Maxflow Mincut :

1)⇒2) si on a un chemin améliorant r la proposition 1.2.4 nous donne un flot sur G_f de valeur c(r)¿0, et la proposition 1.2.2 nous donne un flot' sur G tel que $|f| < |f'|$

2)⇒3) on prend E la composante connexe de s dans G_f

3)⇒1) on a $|f| \leq f(E,P) \leq c(E,P$ le cas d'égalité garantit que f est maximal

Correction de l'algorithme pousser-réétiqueter

La propriété "f est un préflot" est un invariant de boucle, f renvoyé est un flot maximal et h est une fonction de hauteur.

En sortie f est un préflot aux excédents nuls, c'est donc un flot. De plus si h est une fonction de hauteur alors il n'y a pas de chemin de s à t dans G_f donc f est maximal d'après le théorème Maxflow-Mincut.

```
let fordfulkerson pathsearch g f0=
    let f=matrixcopy f0 in
    let boo=ref true in
    let incrflow g f c=
        let rec aux=function
            |[x;y]->(g--f).(x).(y)
            |x::y::q->min ((g--f).(x).(y)) (aux(y::q))in
        let rec incrf f m=function
            |[x;y]->f.(x).(y)<-f.(x).(y)+m
            |x::y::q->f.(x).(y)<-f.(x).(y)+m;
                incrf f m (y::q)in
        incrf f (aux c) c in
    while !boo do
        let c,b=pathsearch(g--f) in
        incrflow g f c;
        boo:=b;done;
    f;;
```

```
let push_relabel g=
   let n=Array.length g in
   let h,e=Array.make n 0, Array.make n 0 in
   let f=Array.make_matrix  n n 0 in

   let push g f e i j=
      let d=min (g--f).(i).(j) e.(i) in
      if g.(i).(j)<>0 then
         f.(i).(j)<-f.(i).(j)+d
      else
         f.(j).(i)<-f.(j).(i)-d;
      e.(i)<-e.(i)-d;
      e.(j)<-e.(j)+d in

   let relabel g f h i=
      h.(i)<-1+ mint (g--f).(i) in

   let boo=ref true in
   while !boo do
      let b=ref false in
      for i=0 to n do
         if e.(i)>0 then
            for j=0 to n do
               if g.(i).(j)-f.(i).(j)>0 && h.(i)=h.(j)+1 then
                  push g f e i j; b:=true
                  done;
               if not !b then relabel g f h i; b:=true
            done;
         boo:=!b; done;
      f;;
```

```
let knapsack w v wmax =
    let n = Array.length w in
    let k = Array.make_matrix (wmax+1)(n+1)0 in
    for j = 0 to n - 1 do
        for p = 1 to wmax do
        if w.(j) > p then
            k.(p).(j+1)<-k.(p).(j)
        else
            k.(p).(j+1)<-max k.(p).(j)(k.(p-w.(j)).(j)+v.(j))
            done
    done;
    k.(wmax).(n);;
```