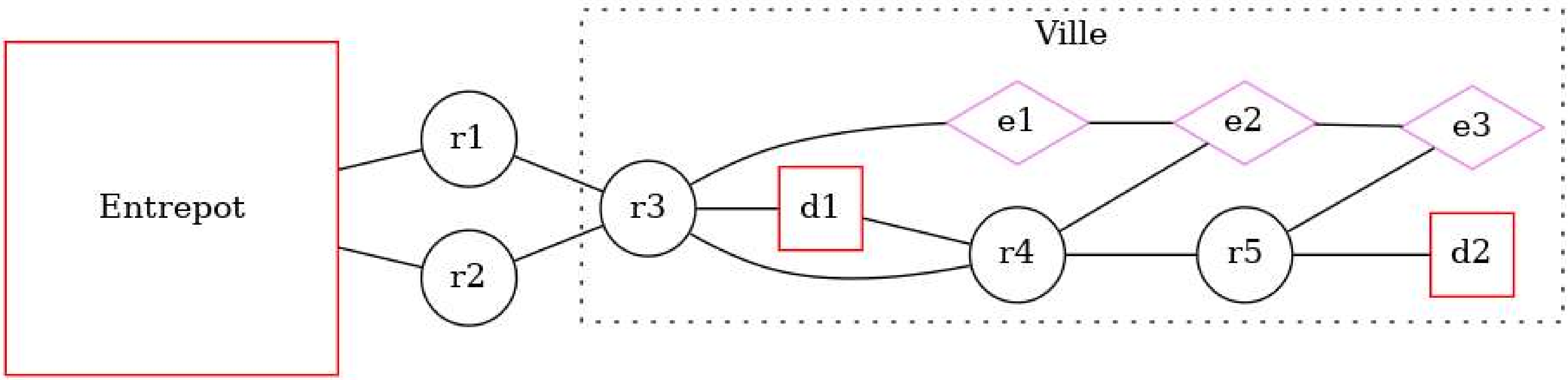
Distribution optimale de biens dans une ville

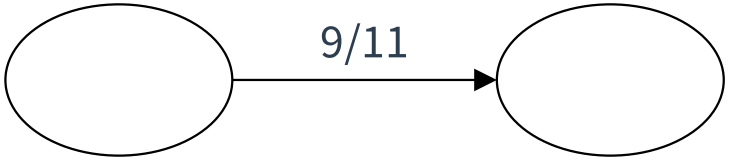
SENE Seydou Laara
Numéro de candidat : 35289

Présentation du modèle



Présentation du modèle

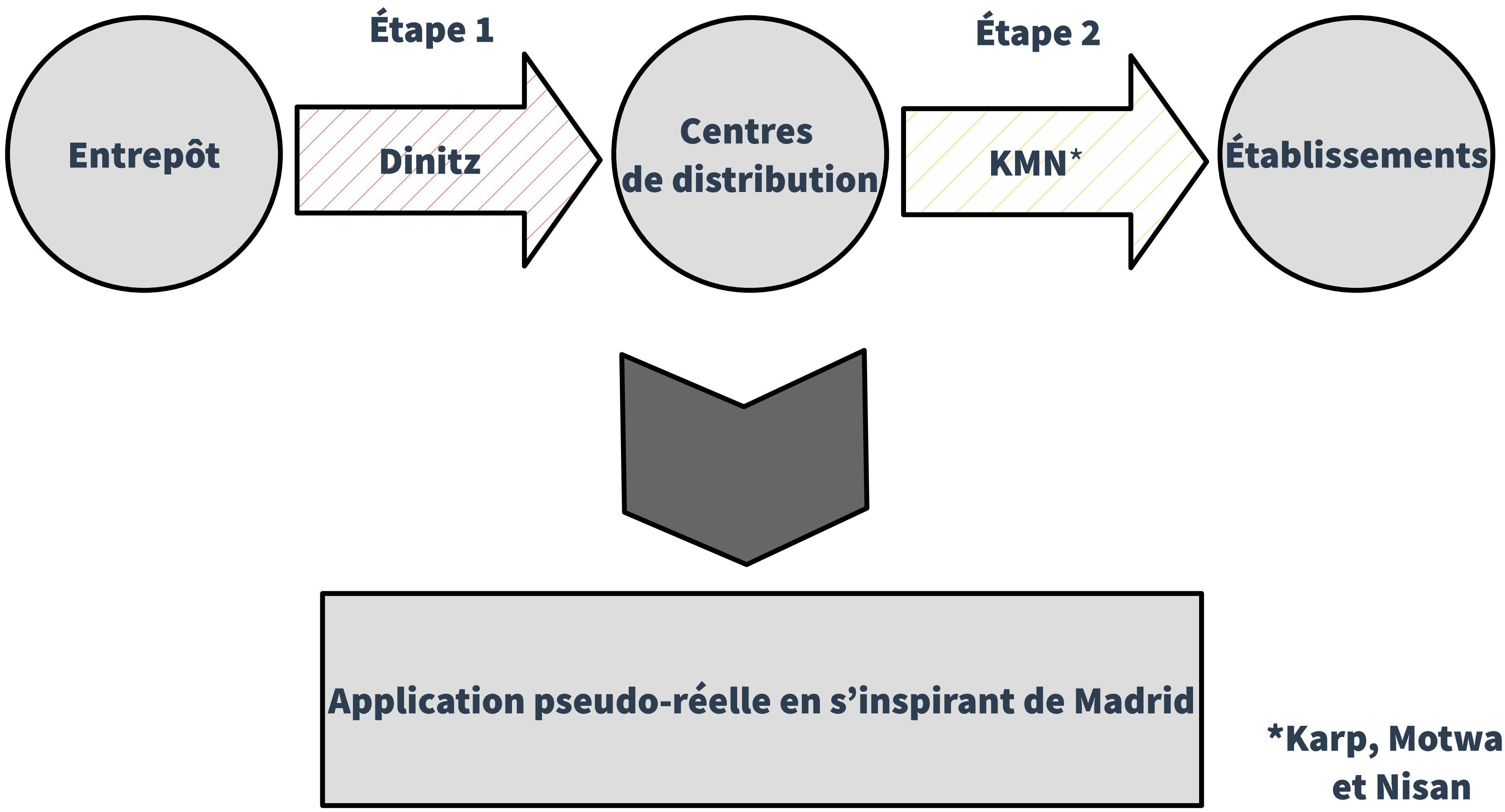
- Flot traversant une route = quantité arbitraire de marchandise traversant cette route.



- Problème du flot maximum :

  Étant donné les capacités de chaque route séparant les installations concernés, trouver une distribution de flot maximisant le flot total entre les installations sources et les installations puits.

Sommaire

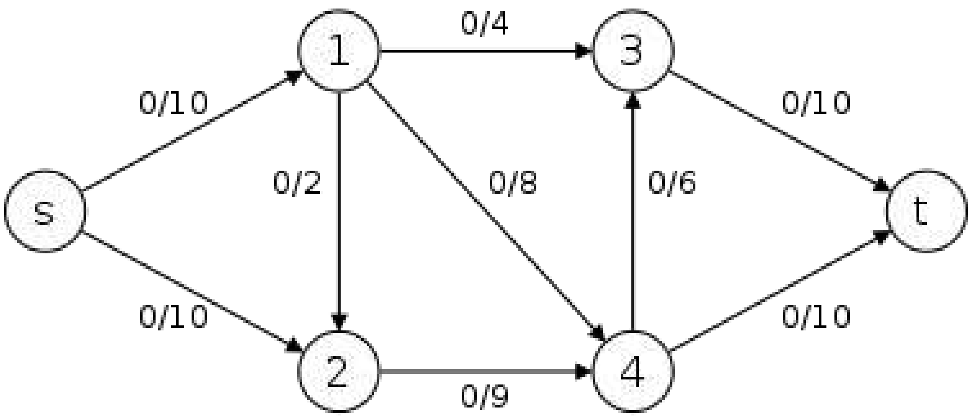


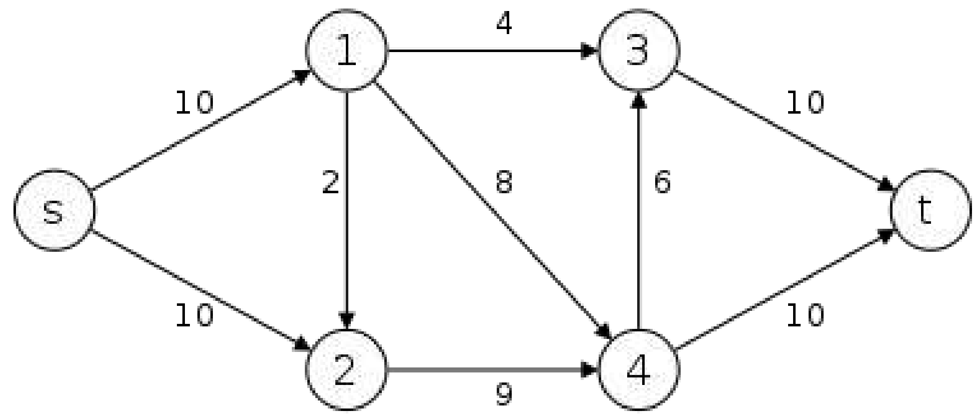
*Karp, Motwani et Nisan

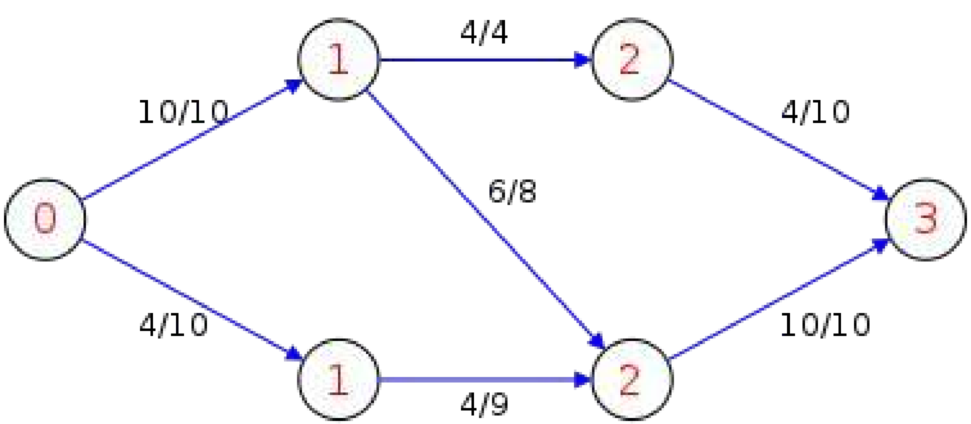
Algorithme de Dinitz

- Graphe orienté $G = (V, E, c, s, t)$
- Flot $f : E \rightarrow \mathbb{N}$ associé à G
- Capacité résiduelle $c_r(u, v) = c(u,v) - f(u, v)$
- Graphe résiduel $G_r = (V, E', c, s, t)$ où (u, v) appartient à E' uniquement si $c_r(u, v) > 0$
- Graphe des niveaux, qui traduit la distance de chaque nœud à la source s, dans G_r

Algorithme de Dinitz

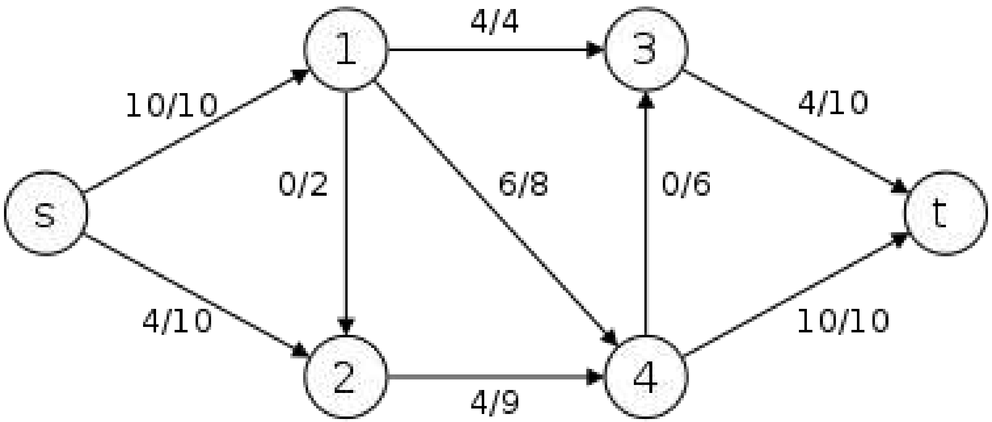


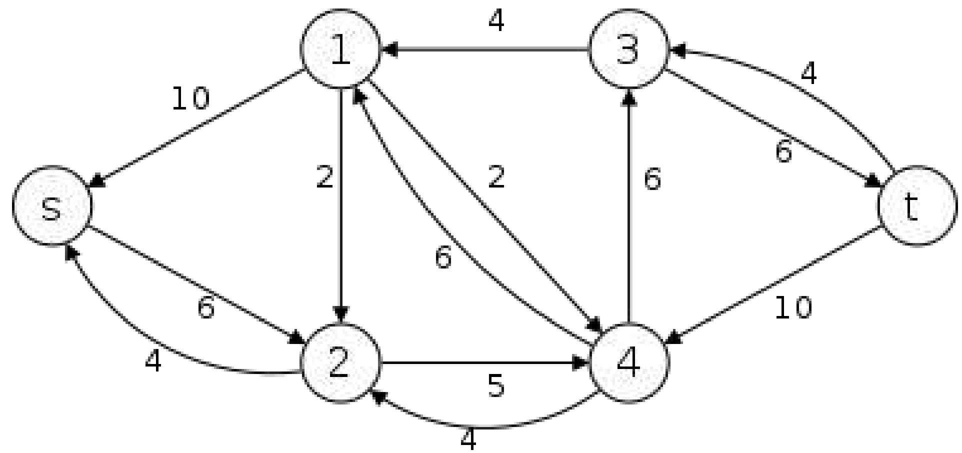


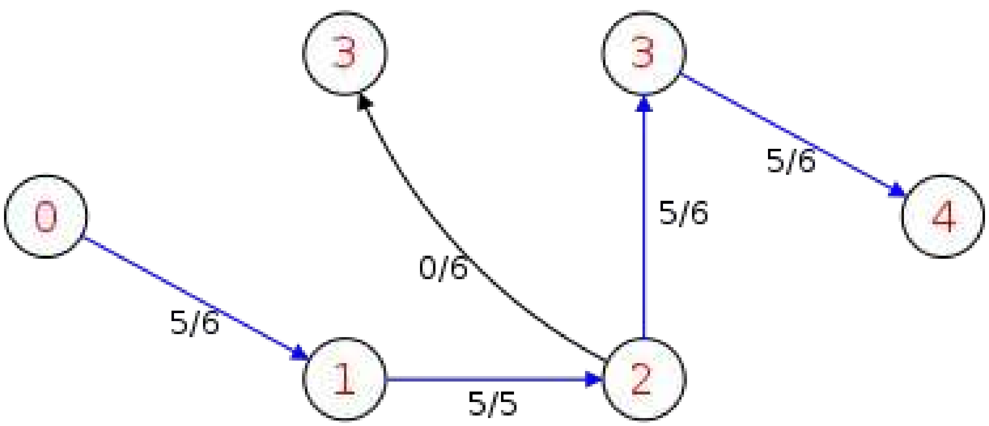


- {s, 1, 3, t} – Valeur 4
- {s, 1, 4, t} – Valeur 6
- {s, 2, 4, t} – Valeur 4
- Flot total = 14

Algorithme de Dinitz

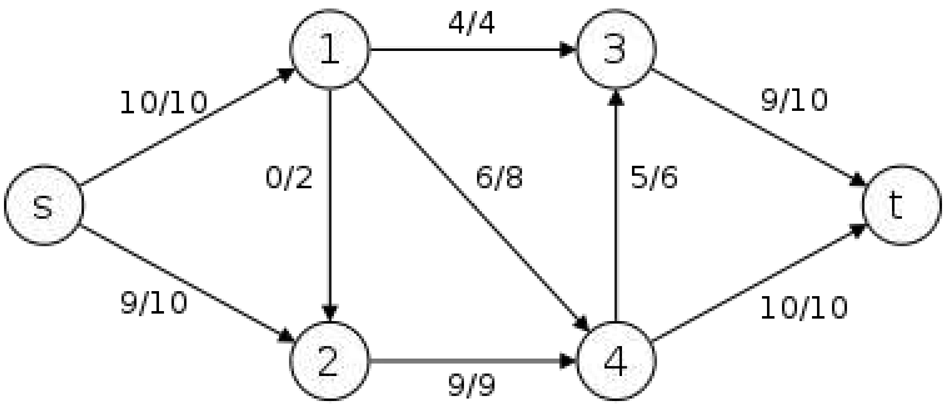


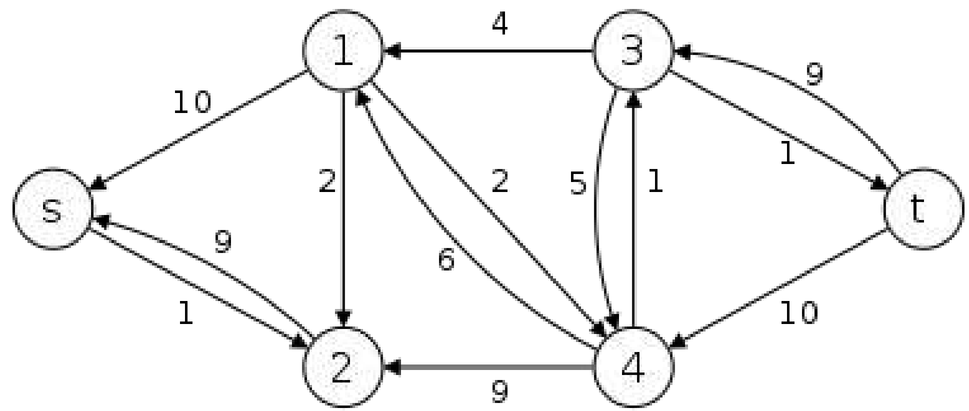


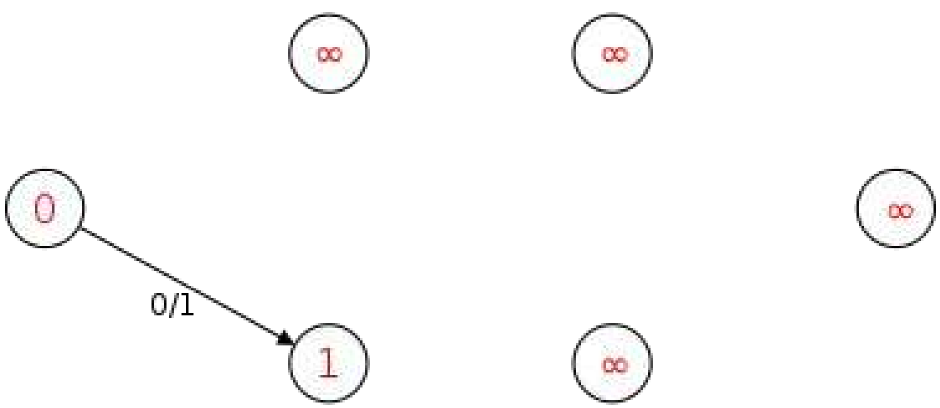


- {s, 2, 4, 3, t} – Valeur 5
- Flot total = 14 + 5 = 19

Algorithme de Dinitz







- **Flot total = 19**

Algorithme de Dinitz

```
type graph = (int, ((int, (int * int * int)) Hashtbl.t)) Hashtbl.t;;
```

- Complexité finale : $O(|V||E|^2)$
- 1000 sommets, 30 000 arêtes → Instantané.
- 20 000 sommets, 8 000 000 arêtes → 5,6 secondes.

```
let test = Dinitz.create_graph_n 20000;;
let nb_aretes_test = nb_aretes test;;
Sys.time ();;
Dinitz.dinitz test;;
Sys.time ();;
```

```
val test : Dinitz.graph = <abstr>
# val nb_aretes_test : int = 8155076
# - : float = 58.5486779999999953
# - : int = 14254
# - : float = 64.153947
# 
```

Algorithme de Karp, Motwani, Nisan

- Approximation du résultat en temps linéaire par rapport à la taille du graphe ($|G| = |V| + |E|$)
- Le processus de « mimicking » :

Déterminisation du problème

Résolution du problème déterminisé

Imitation de cette solution

Affinage de la solution

Algorithme de Karp, Motwani, Nisan
Plan d'attaque

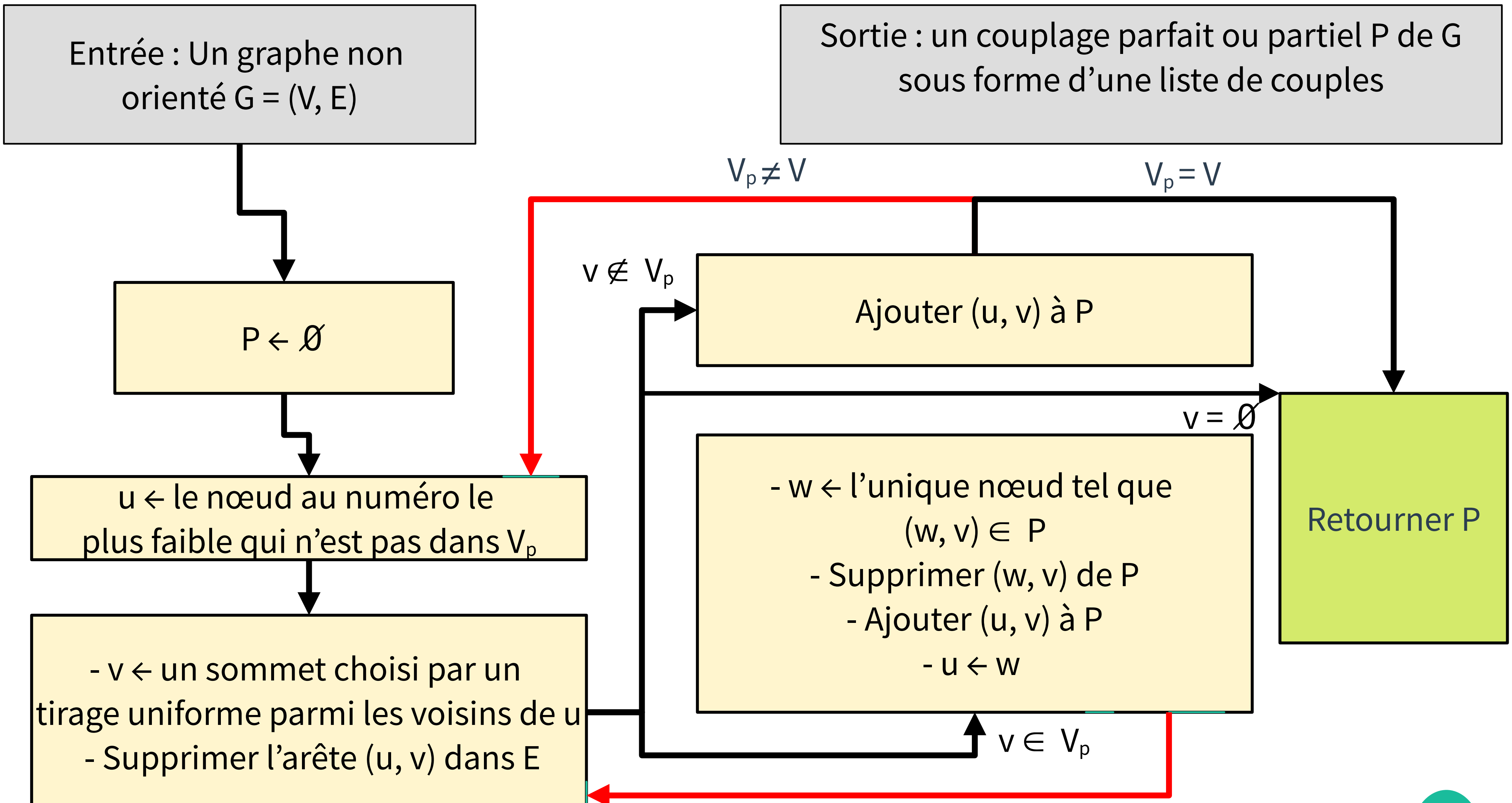
« Proposal Algorithm »

Matrices 0-1 & problème du transport sous capacité restreinte

« Fine-tuning Algorithm »

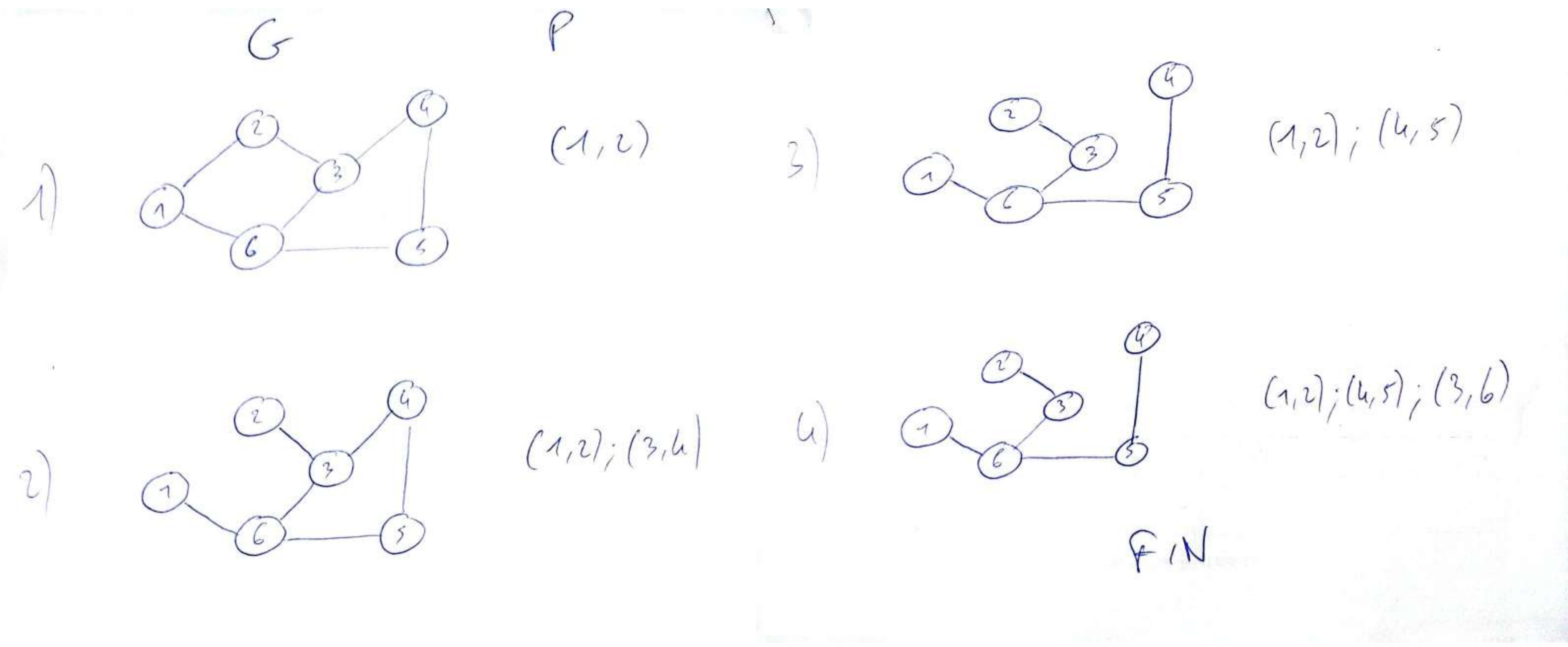
Algorithme de KMN

Algorithme de Karp, Motwani, Nisan
Proposal Algorithm



Algorithme de Karp, Motwani, Nisan

Proposal Algorithm



Algorithme de Karp, Motwani, Nisan

Proposal Algorithm

```
type graph = (int, ((int, (int * int * bool)) Hashtbl.t)) Hashtbl.t;;
```

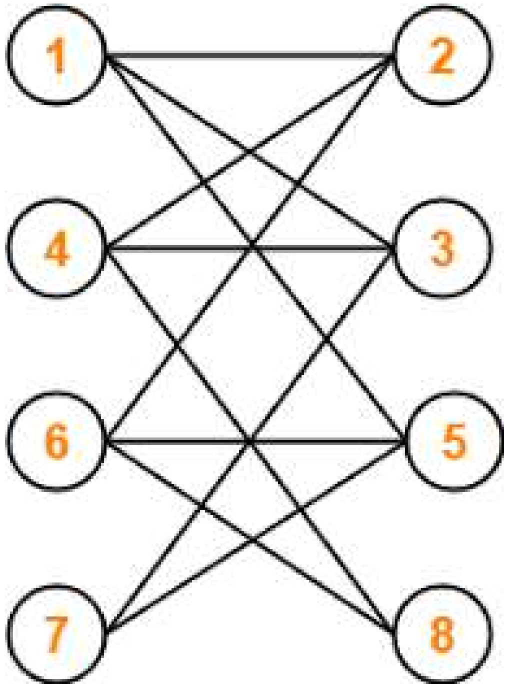
- Complexité : $O(|V|\log(|V|))$

Algorithme de Karp, Motwani, Nisan
Problème du transport sous capacité restreinte

- Problème du transport sous capacité restreinte, ou « Capacitated transportation problem » (CTP) :

  Étant donné un graphe biparti (sources et puits) et une distribution de ressources et de demande, trouver une distribution de flot maximisant le flot total entre les installations sources et les installations puits.

[3 ; 4 ; 2 ; 5], [6 ; 3 ; 2 ; 4]



Algorithme de Karp, Motwani, Nisan
Fine-tuning Algorithm

Entrée : Une instance G = (V, E, c, s t) du CTP vérifiant certaines conditions

Sortie : G accompagné d'un flot approximant une solution au CTP.

On construit M sous-graphes bipartis (B_k, $k < M$) de G

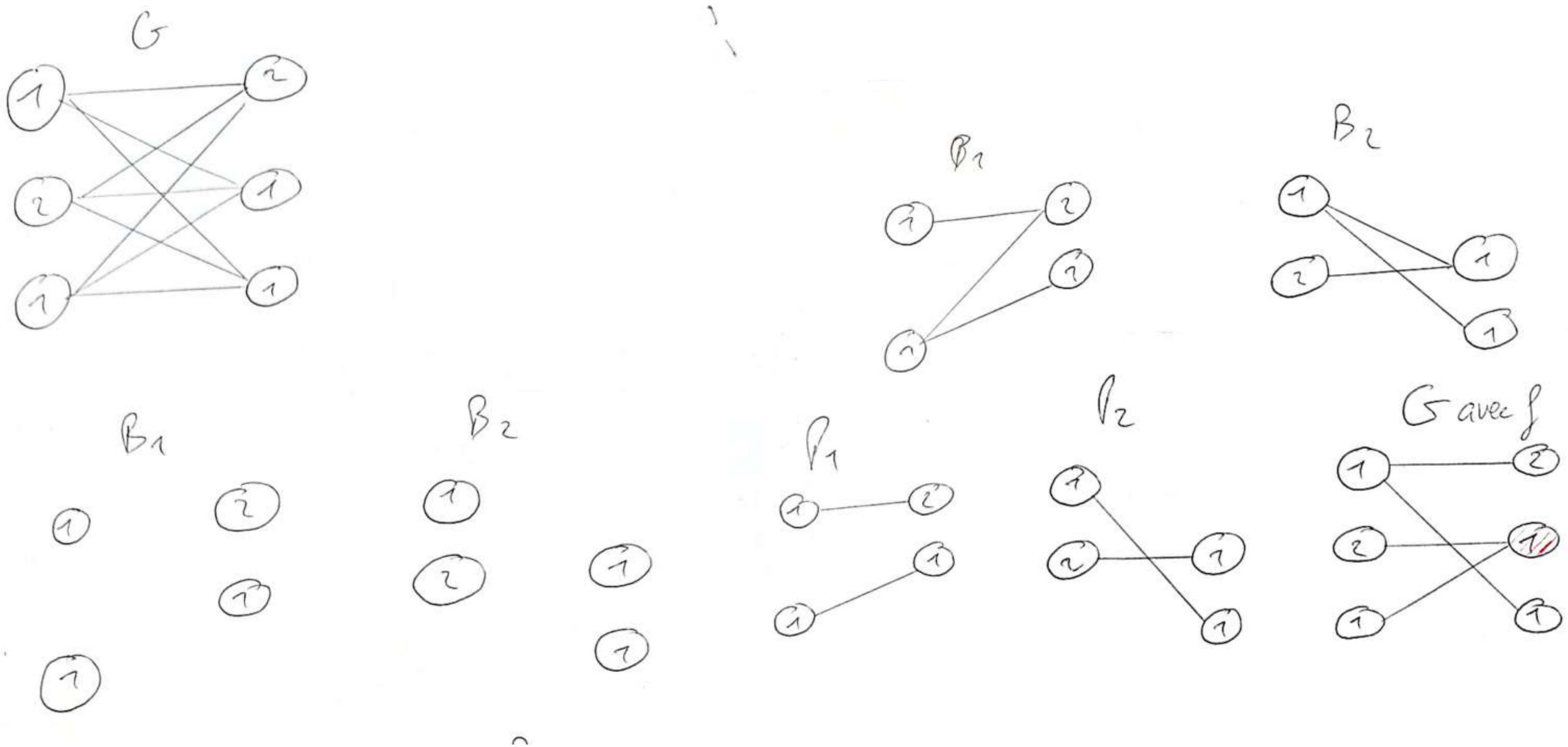
On « colore » chaque arête en lui associant un nombre choisi uniformément entre 0 et M - 1

$\forall$ (u, v) $\in$ E, l'ajouter à E_k uniquement si $u \in V_k$, $v \in V_k$, et (u, v) a la couleur k.

$\forall$ $k < M$, remplacer B_k par un couplage parfait P_k de ce graphe

Saturer en flot toutes les arêtes apparaissant dans l'un des graphes P_k.

Algorithme de Karp, Motwani, Nisan
Fine-tuning Algorithm



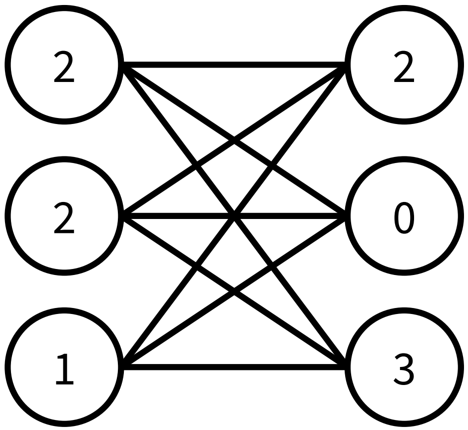
Algorithme de Karp, Motwani, Nisan
Fine-tuning Algorithm

- Implémentation accompagnée de problèmes, dont quelques uns qui sont compensés plus tard.
- Complexité linéaire en la taille du graphe (|V| +|E|)

Algorithme de Karp, Motwani, Nisan
Matrices 0-1, lien avec le CTP

- Matrice 0-1 = Matrice composée de zéros et uns
- Réalisabilité et réalisation d'une paire de vecteurs
- Dans un cas particulier, résoudre CTP se réduit à la réalisation des vecteurs formant la distribution associée

**([2 ; 2 ; 1], [2 ; 0 ; 3]) → [1 ; 0 ; 1
1 ; 0 ; 1
0 ; 0 ; 1]**



Algorithme de Karp, Motwani, Nisan
CTP Algorithm

Entrée : Une instance probabiliste G = (V, E, c, s, t) du CTP vérifiant certaines conditions

Sortie : G accompagné d'un flot approximant une solution au CTP.

Mimicking Method !

On associe des coefficients aux éléments de s et t liées à l'espérance des arêtes présentes

On obtient une version déterministe du problème que l'on résout avec une réalisation de s et t

On sature toutes les arêtes correspondant à des 1 dans la matrice 0-1 obtenue.
Cela donne une nouvelle matrice 0-1, on note les vecteurs somme associés a' et b'

Compenser l'excès de flot dans le graphe en utilisant l'algorithme d'affinage, avec pour Ressources « a' - a » et pour demande « b' - b »

Complexité linéaire en la taille du graphe

Algorithme de Karp, Motwani, Nisan
Max flow Algorithm

Entrée : Une instance probabiliste $G = (S, T, I, E, c, s, t)$ du problème de flot maximum, avec conditions

Sortie : G accompagné d'un flot approximant une solution au problème du flot maximum.

On sature toutes les arêtes de S x I et de I x T.

$\forall$ v $\in$ I, on pose $D_v = c(S, v) - c(v, T)$. On pose $\mathbf{I^+} = \{v \in I \mid D_v > 0\}$, $\mathbf{I^-}$, $\mathbf{D^+}$ la somme des Dv pour v $\in$ $\mathbf{I^+}$, $\mathbf{D^-}$.

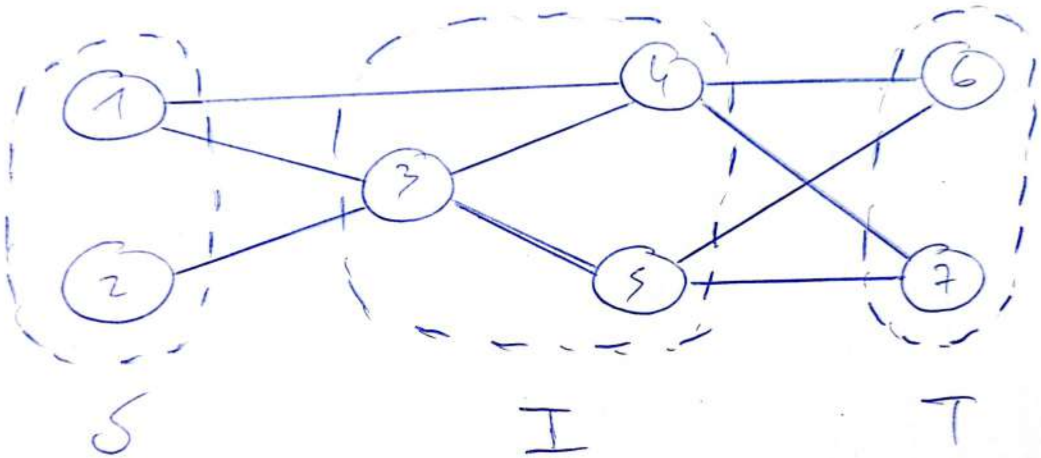
On suppose $c(S, I) \leq c(I, T)$. Ainsi $\mathbf{D^+ + D^- \leq}\ 0$. On réduit le flot entre T et $\mathbf{I^-}$ jusqu'à ce que $\mathbf{D^+ + D^- =}\ 0$, ce qui met à jour $\mathbf{D^-}$ et certaines valeurs de Dv.

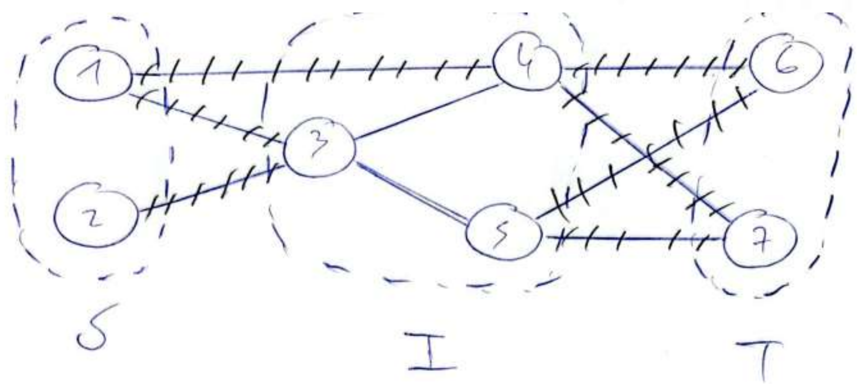
On considère le graphe biparti composé de $\mathbf{I^+}$ et $\mathbf{I^-}$, avec pour ressources et demande les valeurs de Dv. On applique l'algorithme concernant le CTP dessus.

Complexité linéaire en la taille du graphe

Algorithme de Karp, Motwani, Nisan

Max flow Algorithm





$D_3 = 2$; $D_4 = 1 - 2 = -1$; $D_5 = -2$

Algorithme de Karp, Motwani, Nisan

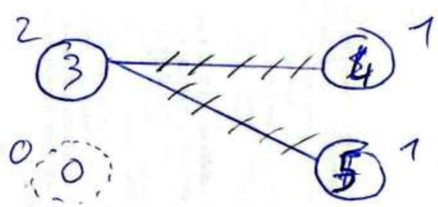
Max flow Algorithm

- $I^+ = \{3\}$; $I^- = \{4, 5\}$;
  $D^+ = 2$; $D^- = -3$

- $D^+ + D^- = -1$

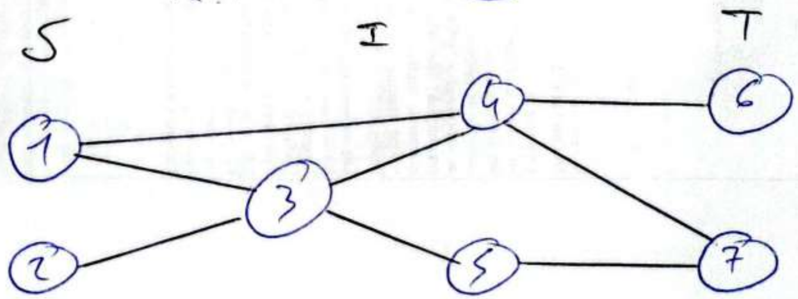
- On retire le flot de $(5, 6) \rightarrow D_5 = -1$
  et $D^+ + D^- = 0$.

- CTP sur



Trivial.

- Résultat :
(Flot seulement)



Algorithme de Karp, Motwani, Nisan

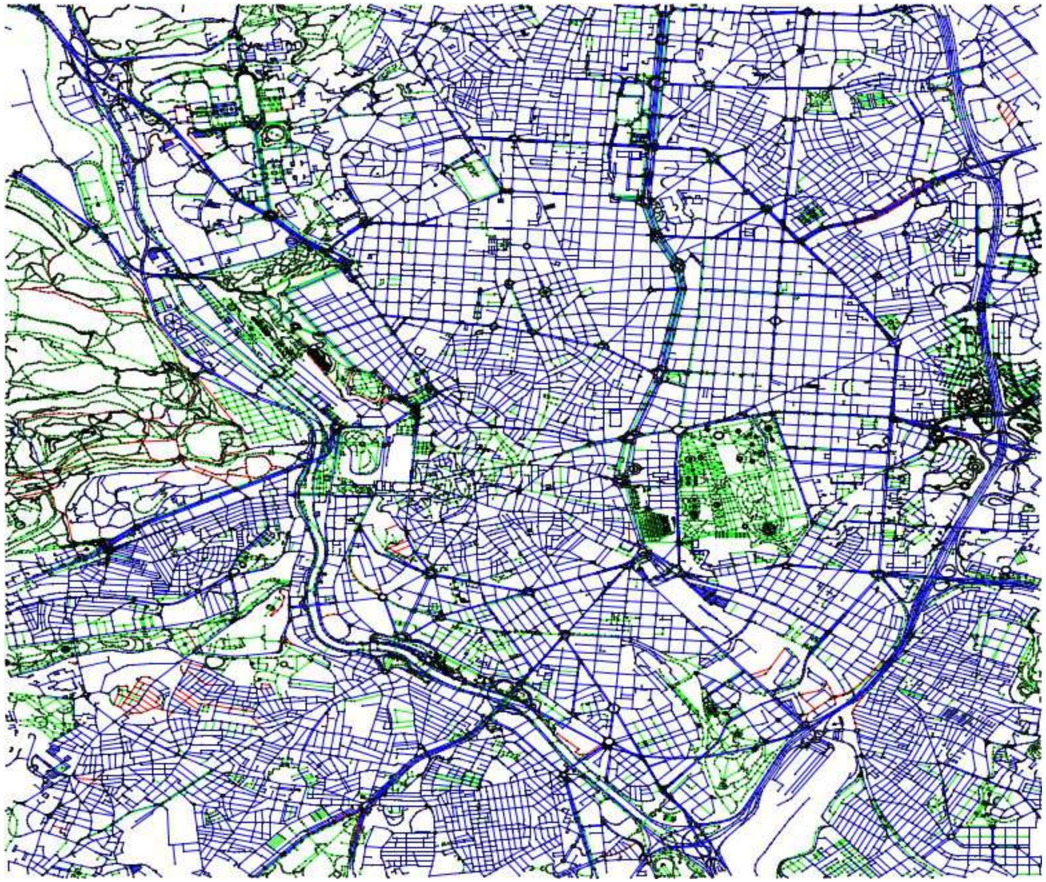
Max flow Algorithm

- Complexité linéaire en la taille du graphe
- Pour 1000 nœuds et 1 500 000 arêtes, temps d'éxécution de 14,1 secondes.

```
let ex_1 = Dinitz.create_graph_n 10000;;
let ex_2 = create_max_flow 600 200;;
nb_aretes ex_1;;
nb_aretes ex_2;;
Sys.time ();;
Dinitz.dinitz ex_1;;
Sys.time ();;
kmn ex_2 600 200;;
Sys.time ();; (* Presque 3x plus long *)
```

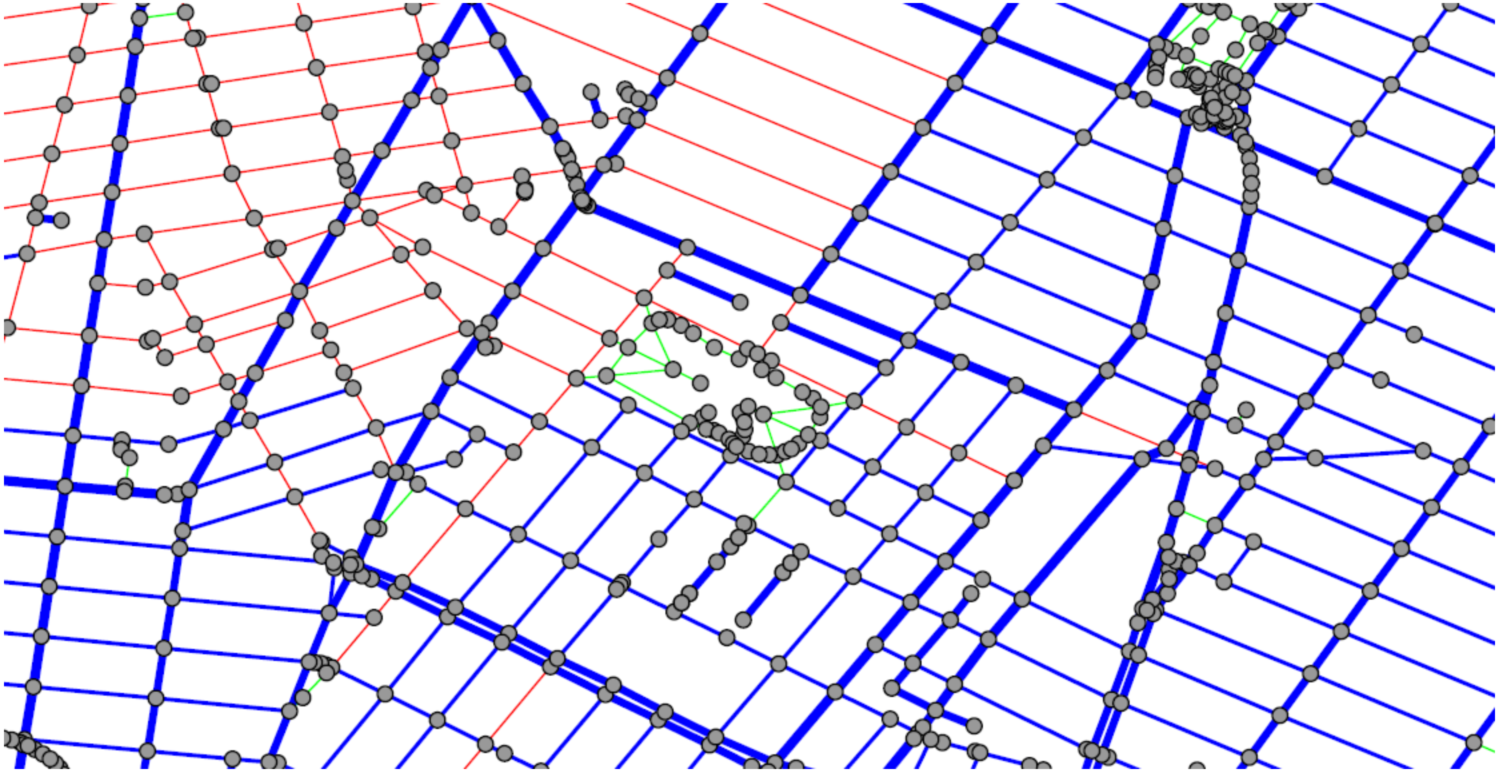
```
# - : int = 2097514
# - : int = 1498348
# - : float = 13.192584
# - : int = 7018
# - : float = 18.008174
# - : int = 4793
# - : float = 31.6615850
```

Pseudo-application



Igniacio Ronaldo

Pseudo-application

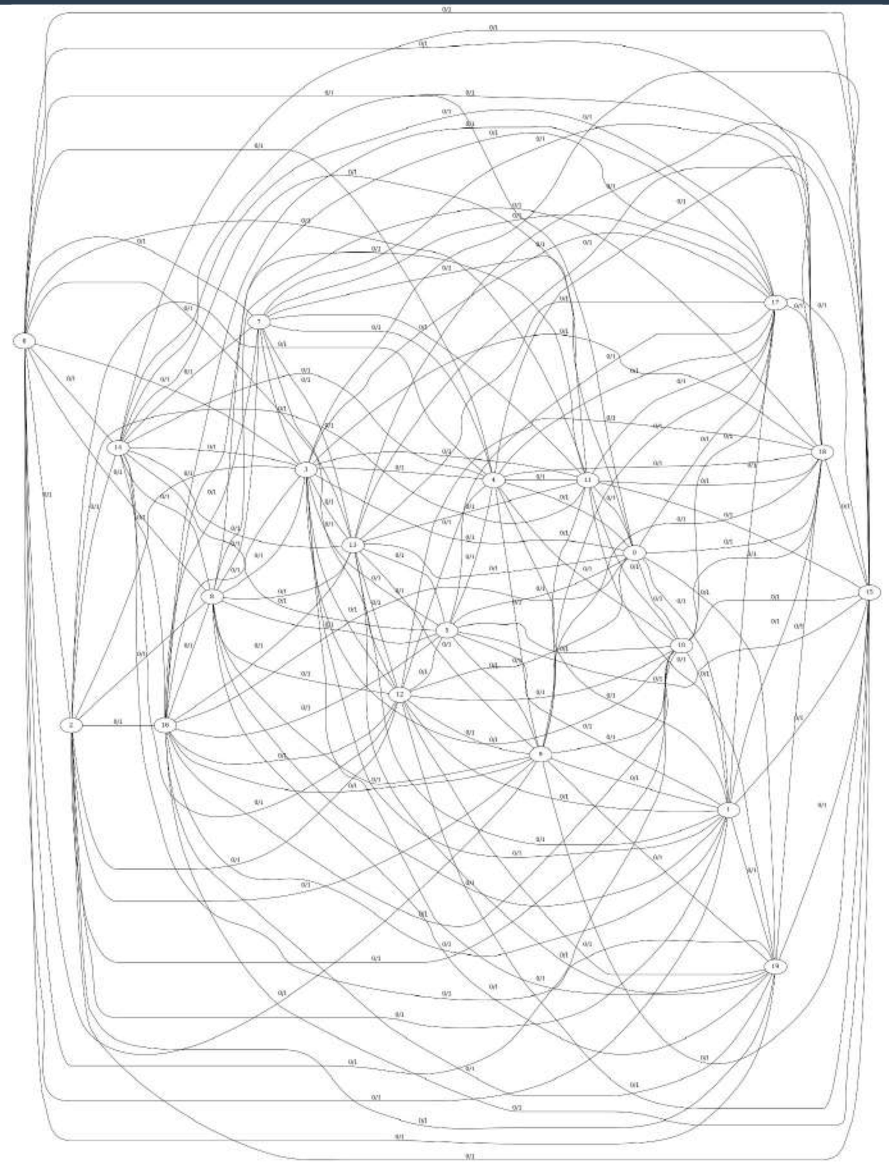


Igniacio Ronaldo

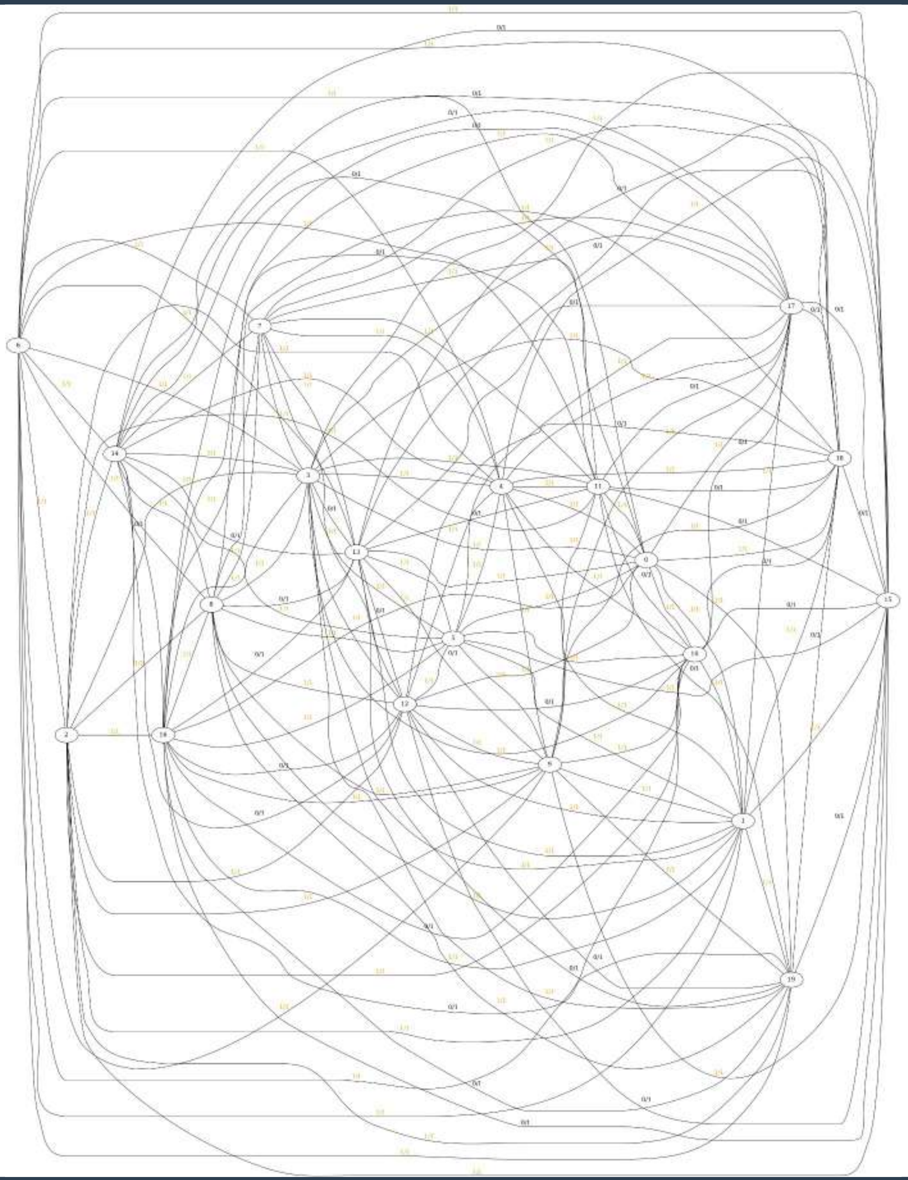
Pseudo-application

- Parsing très spécifique depuis GEXF
- Problèmes avec ce graphe
- Finalement j'ai réalisé un algorithme simple qui prend en compte de manière très approximative l'allure du graphe importé
- Flot moyen autour de 6000 pour 200 établissements pouvant commander, autour de 4500 pour 100 établissements.

Pseudo-application



Pseudo-application



Bibliographie

1 – Richard M. Karp, Rajeev Motwani, Noam Nisan - « Probabilistic Analysis of Network Flow Algorithms » - https://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/CSD-88-392.pdf (1993)

2 - Yefim Dinitz - « Dinitz' Algorithm : The Original Version and Even's Version » - https://doi.org/10.1007/11685654_10 (2006)

3 - AntsRoute (société commerciale) - Comment l'optimisation de tournées de livraison améliore vos performances en 2021 ? https://antsroute.com/blog/comment-optimisation-de-tournees-de-livraison-ameliore-vosperformances-en-2021/ Vu le 09/01/23

4 - Kardinal (société commerciale) - Quelle complexité mathématique pour l'optimisation de tournées ? - https://kardinal.ai/fr/mathematiques-et-optimisation-de-tournees/ Vu le 23/01/23

5 - OpenStreetMap – Données - https://www.openstreetmap.fr/donnees/ Vu le 09/01/23

Annexes

```
senseyyy@KindASUS:~/tipe$ ls
'#code.ml#'     dinitz.cmo    ex3.dot          'karp_motwani_nisan - Copie.ml'     maxflow-master   prioQueue.mli
'#prez.dot#'    dinitz.ml     ex3.dot.png       karp_motwani_nisan.ml              prez.dot         test.txt
 GRAF-007.c     dinitz.mli    ex4.dot           madrid.dot                         prez.png
 Graphviz       ex1.dot       ex4.dot.png       madrid.dot.png                     prez.svg
 Makefile       ex1.dot.png   export_graph.ml   madrid.png                         prioQueue.cmi
 dinitz         ex2.dot       idk.dot           madrid_highway.gexf                prioQueue.cmo
 dinitz.cmi     ex2.dot.png   idk.dot.png       madrid_highway.gexf:Zone.Identifier  prioQueue.ml
senseyyy@KindASUS:~/tipe$
```

```
senseyyy@KindASUS:~/tipe/maxflow-master/maxflow-master/src$ ls
CMakeCache.txt  Makefile    cmake_install.cmake    data_structures  ex_kmn.txt      lib                maxflow.cpp
CMakeFiles      algorithms  command_line_parser.h  ex.txt           gitignore.txt   madrid_graphe.txt  measure.h
CMakeLists.txt  build       common_types.h         ex_dinitz.txt    graph_loader.h  maxflow
senseyyy@KindASUS:~/tipe/maxflow-master/maxflow-master/src$
```

Annexes

```
senseyyy@KindASUS:~/tipe/maxflow-master/maxflow-master/src$ ./maxflow din -f ex_dinitz.txt
solver:         din
filename:       ex_dinitz.txt
flow:           918284
time read:      1508 ms
time init:      0 ms
time solve:     81 ms
# of threads:   1
```

Annexes

https://github.com/ignacioarnaldo
https://github.com/Zagrosss/maxflow

Code

```
type graph = (int, ((int, (int * int * int)) Hashtbl.t)) Hashtbl.t;;

let nb_vertex = Random.int 1500 + 101;;
let avg_degree = nb_vertex / 100 + 5;;
(* un conteneur stocke entre 100 et 24K EVP *)
let avg_capacity = 200;;

let new_graph () = Hashtbl.create nb_vertex;;

let has_vertex graph v = Hashtbl.mem graph v;;

let add_vertex graph v = Hashtbl.add graph v (Hashtbl.create nb_vertex);;

let remove_vertex graph v = Hashtbl.remove graph v;;

let has_edge graph v1 v2 =
  let h = Hashtbl.find graph v1 in
  Hashtbl.mem h v2
;;
```

Code

```
let add_edge graph v1 v2 c =
  let h = Hashtbl.find graph v1 in
  Hashtbl.add h v2 (c, 0, c);
  let h = Hashtbl.find graph v2 in
  Hashtbl.add h v1 (0, 0, 0)
;;

let capacity graph v1 v2 =
  let h = Hashtbl.find graph v1 in
  let c, _, _ = Hashtbl.find h v2 in
  c
;;

let flow graph v1 v2 =
  let h = Hashtbl.find graph v1 in
  let _, f, _ = Hashtbl.find h v2 in
  f
;;

let residu graph v1 v2 =
  let h = Hashtbl.find graph v1 in
  let _, _, r = Hashtbl.find h v2 in
  r
;;
```

```
let remove_edge graph v1 v2 =
  let h = Hashtbl.find graph v1 in
  Hashtbl.remove h v2
;;

let fullremove_edge graph v1 v2 =
  let h = Hashtbl.find graph v1 in
  Hashtbl.remove h v2;
  let h = Hashtbl.find graph v2 in
  Hashtbl.remove h v1
;;

Hashtbl.iter;;

let neighbours graph v =
  let h = Hashtbl.find graph v in
  let l = ref [] in
  let add t (c, f, r) = l := (t, c, f, r) :: !l in
  Hashtbl.iter add h;
  !l
;;
```

Code

```
let create_graph () =
  Random.self_init ();
  let g = new_graph () in
  for i = 0 to nb_vertex - 1 do
    add_vertex g i
  done;
  let vu = Array.make nb_vertex false in
  for i = 0 to avg_degree do
    let k = Random.int (nb_vertex / 2) + 1 in
    if not vu.(k) then begin
        vu.(k) <- true;
        add_edge g 0 k ((Random.int avg_capacity + 1) * k) end
  done;
  for j = 1 to nb_vertex - 2 do
    let vu = Array.make nb_vertex false in
    for i = 0 to avg_degree do
      let k = Random.int (nb_vertex - 1) + 1 in
      if not vu.(k) && not (has_edge g k j) && k <> j then begin
          vu.(k) <- true;
          add_edge g j k ((Random.int avg_capacity + 1) * k) end
    done
  done;
  g
;;
```

Code

```
let affiche_graphe g =
  let print_edges u v =
    Printf.printf "%d ->" u;
    let print t (c, f, r) =
      if c > 0 then
        Printf.printf " (%d|%d,%d,%d)" t c f r in
    Hashtbl.iter print v;
    Printf.printf "\n"
  in Hashtbl.iter print_edges g
;;

(*affiche_graphe (create_graph ());;*)

Hashtbl.replace;;

let graphe_residuel g =
  let residu u h =
    let res_aux v (c, f, r) =
      if c - f > 0 then
        Hashtbl.replace h v (c, f, c - f)
      else
        Hashtbl.remove h v
    in Hashtbl.iter res_aux h
  in Hashtbl.iter residu g
;; (* C = O(|A|) *)
(* Servait juste à comprendre, add_flow fait tout *)
```

Code

```ocaml
let distances g =
  let n = Hashtbl.length g in
  let d = Array.make n (-1) in
  let vu = Array.make n false in
  let q = Queue.create () in
  Queue.push 0 q;
  d.(0) <- 0;
  while not (Queue.is_empty q) do
    let t = Queue.pop q in
    let rec explore_voisins l =
      match l with
      | [] -> ()
      | (v, _, _, r) :: qu ->
         if r > 0 && not vu.(v) then begin
              d.(v) <- d.(t) + 1;
              vu.(v) <- true;
              Queue.push v q end;
         explore_voisins qu
    in explore_voisins (neighbours g t)
  done;
  d
;;
```

Code

```
let rec add_flow graph v1 v2 f =
  if capacity graph v1 v2 = 0 then
    add_flow graph v2 v1 f
  else begin
      let h = Hashtbl.find graph v1 in
      let c, ff, r = Hashtbl.find h v2 in
      if r - f > 0 then
        Hashtbl.replace h v2 (c, ff + f, r - f)
      else
        Hashtbl.remove h v2;
      let h = Hashtbl.find graph v2 in
      let c, ff, r = Hashtbl.find h v1 in
      if r + f > 0 then
        Hashtbl.replace h v1 (c, ff - f, r + f)
      else
        Hashtbl.remove h v1 end
;;
```

Code

```
let flot_bloquant g fmax d =
  let n = Hashtbl.length g in
  let bloquant = ref false in
  let parcours g =
    let vu = Array.make n false in
    let parent = Array.make n (-1) in
    let p = Stack.create () in
    let chemin_st = ref false in
    Stack.push 0 p;
    while not (Stack.is_empty p) && not !chemin_st do
      let u = Stack.pop p in
      if u = (n - 1) then begin
          chemin_st := true;
          let c_limitante = ref max_int in
          let rec determination_cl s =
            if s > 0 then begin
                let sp = parent.(s) in
                c_limitante := min !c_limitante (residu g sp s);
                determination_cl sp end
          in let rec mise_a_jour s =
              if s > 0 then begin
                  let sp = parent.(s) in
                  add_flow g sp s !c_limitante;
                  mise_a_jour sp end
            in determination_cl u;
               fmax := !fmax + !c_limitante;
```

Code

```
                   add_flow g sp s !c_limitante;
                   mise_a_jour sp end
             in determination_cl u;
                fmax := !fmax + !c_limitante;
                mise_a_jour u;
                Stack.push u p
        end;
      vu.(u) <- true;
      let rec explore_voisins l =
        match l with
        | [] -> ()
        | (v, _, _, _) :: q ->
           if d.(v) > d.(u) && not vu.(v) then begin
               Stack.push v p;
               parent.(v) <- u end;
           explore_voisins q
      in explore_voisins (neighbours g u)
    done;
    if Stack.is_empty p then bloquant := true
  in parcours g;
     if !bloquant then
       !bloquant
     else begin
         while not !bloquant do
           parcours g
         done;
         false
       end
;;
```

Code

```
let dinitz g =
  let g_copy = Hashtbl.copy g in
  let fmax = ref 0 in
  let puit_accessible = ref true in
  while !puit_accessible do
    let d = distances g_copy in
    puit_accessible := not (flot_bloquant g_copy fmax d)
  done;
  !fmax
;;

Printexc.record_backtrace true;;

(*let ex = create_graph ();;
affiche_graphe ex;;
dinitz ex;;
affiche_graphe ex;;*)
```

Code

```
let export_graphiz g name show_hidden =
  let f = open_out name in
  output_string f "graph {\n  rankdir=LR;\n";
  let added_edges = Hashtbl.create 100 in
  let export_edge u v (capacity, flow, exists) =
    if not (Hashtbl.mem added_edges (u, v)) && not (Hashtbl.mem added_edges (v, u)) then
      if exists || show_hidden then
        begin
          Hashtbl.add added_edges (v, u) ();
          Printf.fprintf f "  %d -- %d [label=\"%d/%d\"" u v flow capacity;
          if capacity = flow then output_string f ", fontcolor=\"orange\"";
          if (not exists) && show_hidden then output_string f ", color=\"red\"";
          output_string f "];\n"
        end
  in
  Hashtbl.iter (
      fun u voisins ->
      Hashtbl.iter (export_edge u) voisins
    ) g;
  output_string f "}\n";
  close_out f
;;
```

Code

```
let export_dimacs g name =
  let f = open_out name in
  let n = Hashtbl.length g in
  let a = nb_aretes g in
  Printf.fprintf f "p max %d %d\n" n a;
  Printf.fprintf f "n 1 s\n";
  Printf.fprintf f "n %d t\n" n;
  let arcs u h =
    let arcs_aux v (c, _, _) =
      Printf.fprintf f "a %d %d %d\n" (u + 1) (v + 1) c
    in Hashtbl.iter arcs_aux h
  in Hashtbl.iter arcs g
;;

let test = open_in "test.txt";;
let testt = Scanf.Scanning.from_channel test;;
Scanf.sscanf "heyH\" 3" "%s %d" (fun x d -> Printf.printf "%d" d);;
```

Code

```
let nb_vertex = 100;;
let avg_degree = nb_vertex / 100 + 3;;
(* un conteneur stocke entre 100 et 24K EVP *)
let avg_capacity = 200;;
let p = 0.75;;

let l_n = 3;;
let u_n = 5;;
let d_n =
  let nb = float_of_int nb_vertex in
  (nb /. (2. *. sqrt (log10 nb)));;

let e_n =
  let nb = float_of_int nb_vertex in
  nb /. (2. *. log10 nb);;

let bound = int_of_float (p *. d_n);;

let swap tab i j =
  let temp = tab.(i) in
  tab.(i) <- tab.(j);
  tab.(j) <- temp
;;

let fisher_yates n =
  let tab = Array.init n (fun x-> x) in
  for i = 0 to (n - 2) do
    let k = i + 1 + Random.int (n - i - 1) in
    swap tab i k
  done;
  tab
;;
```

```
let fine_tuning_distrib () =
  let pdb = Array.make (nb_vertex * 2) (-1) in
  for i = 0 to nb_vertex - 1 do
    let d = u_n - Random.int (u_n - l_n + 1) in
    pdb.(i) <- d
  done;
  let redistrib = fisher_yates nb_vertex in
  for i = nb_vertex to 2 * nb_vertex - 1 do
    pdb.(i) <- pdb.(redistrib.(i - nb_vertex))
  done;
  pdb
;;

let ctp_distrib () =
  let pdb = Array.make (nb_vertex * 2) (-1) in
  for i = 0 to nb_vertex - 1 do
    let d = 1 + Random.int bound in
    pdb.(i) <- d
  done;
  let redistrib = fisher_yates nb_vertex in
  for i = nb_vertex to 2 * nb_vertex - 1 do
    pdb.(i) <- pdb.(redistrib.(i - nb_vertex))
  done;
  pdb
;;
```

Code

```
exception Found of int;;

let give_me h = (* les manips de fou *)
  try
    Hashtbl.iter (fun u _ -> raise (Found u)) h;
    0
  with Found x -> x
;;

let copie h =
  let hh = Hashtbl.create (Hashtbl.length h) in
  Hashtbl.iter (fun x y -> Hashtbl.add hh x (Hashtbl.copy y)) h;
  hh
;;
```

Code

```
let create_bipartite distrib =
 (*Random.self_init();*)
 let g = new_graph () in
 for i = 0 to nb_vertex * 2 - 1 do
   add_vertex g i
 done;
 for i = 0 to nb_vertex - 1 do
   for j = 0 to nb_vertex - 1 do
     if Random.int 4 > 0 then begin
         let k = Random.int avg_capacity + 1 in
         add_edge g i (j + nb_vertex) k;
         add_edge g (j + nb_vertex) i k end
   done
 done;
 g
;;
```

Code

```
let create_max_flow n r =
  let g = new_graph () in
  for i = 0 to 2 * r + n - 1 do
    add_vertex g i
  done;
  for i = 0 to 2 * r + n - 1 do
    for j = i + 1 to 2 * r + n - 1 do
      if Random.int 4 > 0 then begin
          add_edge g i j 1;
          add_edge g j i 1 end
    done
  done;
  g
;;
```

Code

```
let import_gexf name =
  let gexf = Scanf.Scanning.from_channel (open_in name) in
  let graph = Hashtbl.create 100000 in
  let sommets = Array.make 58335 0 in
  for i = 1 to 10 do
    Scanf.bscanf gexf "%s@>" (fun x -> ())
  done;
  for i = 0 to 58334 do
    Scanf.bscanf gexf "%s@\"" (fun x -> ());
    Scanf.bscanf gexf "%d\"" (fun x -> add_vertex graph i; sommets.(x) <- i);
    Scanf.bscanf gexf "%s@n" (fun x -> ());
    Scanf.bscanf gexf "%s@n" (fun x -> ());
  done;
  Scanf.bscanf gexf "%s@>" (fun x -> ());
  for i = 0 to 68448 do
    Scanf.bscanf gexf "%s@o%s@\"%d\"%s@\"%d\"%s@\"%d"
      (fun _ _ u _ v _ c -> add_edge graph sommets.(u) sommets.(v) c);
    Scanf.bscanf gexf "%s@g" (fun x -> ());
    Scanf.bscanf gexf "%s@g" (fun x -> ());
    Scanf.bscanf gexf "%s@g" (fun x -> ());
  done;
  graph
;;
```

Code

```
exception No_match;;

let select graph u =
  let neighbours = neighbours graph u in
  if neighbours = [||] then
    raise No_match
  else
    let n = Array.length neighbours in
    let k = Random.int n in
    let v, _, _ = neighbours.(k) in
    delete graph u v;
    v
;;

(* renvoie true s'il a réussi à trouver un perfect matching, renvoie false sinon et garde quand meme
un partial matching = perfect sur un sous-ensemble des noeuds. Le graphe est modifié *)

let cleaning graph =
  let clean u h =
    let clean_aux v (c, f, b) =
      if not b then
        Hashtbl.remove h v
    in Hashtbl.iter clean_aux h
  in Hashtbl.iter clean graph;
     graph
;;
```

Code

```
let perfect_matching graph = (* beaucoup plus subtile à faire en temps raisonnable que je ne le pensais, plein de détails *
  let p = copie graph in
  delete_all p;
  let n = Hashtbl.length graph in
  let k = ref n in
  let remaining = Hashtbl.create n in
  let couples = Hashtbl.create n in(* là uniquement pour que trouver w soit en tenmps constant *)
  Hashtbl.iter (fun u _ -> Hashtbl.add remaining u ()) graph;
  try while !k <> 0 do
    let u = give_me remaining in
    let v = select graph u in
    if Hashtbl.mem remaining v then begin (* Donc v n'est pas dans p !*)
      add p u v;
      Hashtbl.remove remaining u;
      Hashtbl.remove remaining v;
      Hashtbl.add couples u v;
      Hashtbl.add couples v u;
      k := !k - 2 end
    else begin
        let w = Hashtbl.find couples v in
        delete p v w;
        add p u v;
        Hashtbl.remove remaining u;
        Hashtbl.add remaining w ();
        Hashtbl.replace couples v u;
        Hashtbl.remove couples w;
        Hashtbl.add couples u v end
  done;
  cleaning p
  with No_match -> cleaning p
;;
```

Code

```
let fine_tuning bipart distrib sup nb_vertex =
  let a = ref 0 in
  let a_sum = Array.make (nb_vertex + 1) 0 in
  let b = ref 0 in
  let b_sum = Array.make (nb_vertex + 1) 0 in
  for i = 0 to nb_vertex - 1 do
    a := !a + distrib.(i);
    b := !b + distrib.(i + nb_vertex);
    a_sum.(i+1) <- !a;
    b_sum.(i+1) <- !b;
  done;
  let belongs_to = Array.make_matrix (2 * nb_vertex) sup false in
  for i = 0 to nb_vertex - 1 do
    for j = a_sum.(i) + 1 to a_sum.(i+1) do
      belongs_to.(i).(j mod sup) <- true
    done;
    for j = b_sum.(i) + 1 to b_sum.(i + 1) do
      belongs_to.(i + nb_vertex).(j mod sup) <- true
    done;
  done;
  let sous_graphs = Array.init sup (fun _ -> Hashtbl.create (nb_vertex / sup)) in
  let coloring u h =
    let color_aux v (c, f, b) =
      let k = Random.int sup in
      if belongs_to.(u).(k) && belongs_to.(v).(k) && u <nb_vertex && v >= nb_vertex then begin
          if not (Hashtbl.mem sous_graphs.(k) u) then
            add_vertex sous_graphs.(k) u;
          if not (Hashtbl.mem sous_graphs.(k) v) then
            add_vertex sous_graphs.(k) v;
          let h_k = Hashtbl.find sous_graphs.(k) u in
```

Code

```
        add_vertex sous_graphs.(k) v;
      let h_k = Hashtbl.find sous_graphs.(k) u in
      Hashtbl.replace h_k v (c, f, b);
      let h_k = Hashtbl.find sous_graphs.(k) v in
      Hashtbl.replace h_k u (c, f, b) end in
  Hashtbl.iter color_aux h in
Hashtbl.iter coloring bipart;
for i = 0 to sup - 1 do
  sous_graphs.(i) <- perfect_matching sous_graphs.(i);
  (*affiche_graphe sous_graphs.(i);
  Printf.printf "\n\n"*)
done;
let union = Hashtbl.create (2 * nb_vertex) in
for i = 0 to 2 * nb_vertex - 1 do
  add_vertex union i
done;
let give g =
  let give_aux u h =
    let h_union = Hashtbl.find union u in
    let aux v (c, f, b) =
      Hashtbl.replace h_union v (c,f,b)
    in Hashtbl.iter aux h
  in Hashtbl.iter give_aux g
in for i = 0 to sup - 1 do
     give sous_graphs.(i)
   done;
   let saturate u h =
     let h_bipart = Hashtbl.find bipart u in
     let sat_aux v _ =
       let (c, f, b) = Hashtbl.find h_bipart v in
       Hashtbl.replace h_bipart v (c, c, b) in
     Hashtbl.iter sat_aux h in
   Hashtbl.iter saturate union
;;
```

Code

```
let rev_compare (x, y) (z, t) = z - x;;

let realization r c =
  let m = Array.length r in
  let n = Array.length c in
  let res = Array.make_matrix m n 0 in
  let ordre = Array.init n (fun i -> (c.(i), i)) in
  Array.sort rev_compare ordre;
  for i = 0 to m - 1 do
    let count = ref (r.(i) - 1) in
    while !count >= 0 do
      res.(i).(snd ordre.(!count)) <- 1;
      ordre.(!count) <- (fst ordre.(!count) - 1, snd ordre.(!count));
      decr count
    done;
    Array.sort rev_compare ordre
  done;
  res
;;

realization [|2; 2; 1|] [|1; 2; 2|];;
```

Code

```
let deterministic_relaxation distrib nb_vertex =
  let c = 1. /. p in
  let new_db = Array.make (2 * nb_vertex) (-1) in
  for i = 0 to 2 * nb_vertex - 1 do
    let db_i = float_of_int distrib.(i) in
    new_db.(i) <- int_of_float (Float.round (c *. db_i +. e_n))
  done;
  new_db
;;

(* let distribb = ctp_distrib ();;
deterministic_relaxation distribb;; *)
```

Code

```
let undirected_transportation bipart distrib nb_vertex =
  let copy = copie bipart in
  let distrib_det = deterministic_relaxation distrib nb_vertex in
  let s' = Array.sub distrib_det 0 nb_vertex in
  let t' = Array.sub distrib_det nb_vertex nb_vertex in
  let d_sol = realization s' t' in (* solution de la version determinisee du probleme *)
  let s_m = Array.copy s' in (* s_moins *)
  let t_m = Array.copy t' in
  for i = 0 to nb_vertex - 1 do
    for j = 0 to nb_vertex - 1 do
      if d_sol.(i).(j) = 1 then
        let h = Hashtbl.find bipart i in
        if Hashtbl.mem h (j + nb_vertex) then begin
          let (c, _, b) = Hashtbl.find h (j + nb_vertex) in
          Hashtbl.replace h (j + nb_vertex) (c, c, true);
          Hashtbl.replace (Hashtbl.find bipart (j + nb_vertex)) i (c, c, true);
          end
        else begin
            s_m.(i) <- s_m.(i) - 1;
            t_m.(j) <- t_m.(j) - 1
          end
        (*if has_edge bipart i (j + nb_vertex) then begin
          let c = capacity bipart i (j + nb_vertex) in
          let h = Hashtbl.find bipart i in
          Hashtbl.replace h (j + nb_vertex) (c, c, true);
          Printf.printf "ok\n" end
        else begin
          s_m.(i) <- s_m.(i) - 1;
          t_m.(j) <- t_m.(j) - 1
          end*)
    done
  done;
  (*affiche graphe bipart;*)
```

Code

```
  done;
  (*affiche_graphe bipart;*)
  let ex_s = Array.init nb_vertex (fun i -> s_m.(i) - distrib.(i)) in (* traduit le flot en excès issu des sommets de S *)
  let ex_t = Array.init nb_vertex (fun i -> t_m.(i) - distrib.(nb_vertex + i)) in
  fine_tuning copy (Array.append ex_s ex_t) (2 * bound) (nb_vertex);
  (*affiche_graphe copy;*)
  let final_step u h =
    let final_aux v (c, f, b) =
      let f' = flow copy u v in
      Hashtbl.replace h v (c, flow bipart u v - f', b)
    in Hashtbl.iter final_aux h
  in Hashtbl.iter final_step bipart
;;

(*let distrib_ = ctp_distrib ();;
let ex_ = create_bipartite distrib_;;
affiche_graphe ex_;;
undirected_transportation ex_ distrib_ nb_vertex;;
affiche_graphe ex_;;
verif_graph ex_ distrib_;; *)
```

Code

```
let saturate graph u v =
  let h = Hashtbl.find graph u in
  let (c,f,b) = Hashtbl.find h v in
  Hashtbl.replace h v (c, c, b);
  let h = Hashtbl.find graph v in
  let (c, f, b) = Hashtbl.find h u in
  Hashtbl.replace h u (c, c, b)
;;

let desaturate graph u v =
  let h = Hashtbl.find graph u in
  let (c,f,b) = Hashtbl.find h v in
  Hashtbl.replace h v (c, 0, b);
  let h = Hashtbl.find graph v in
  let (c, f, b) = Hashtbl.find h u in
  Hashtbl.replace h u (c, 0, b)
;;

let delta g u n r =
  let v = List.map (fun (x, y, z) -> x, y) (neighbours_list g u) in
  let rec aux l acc =
    match l with
      | [] -> acc
      | t :: q -> if fst t < r then
                    aux q (acc + snd t)
                  else if fst t < 2 * r then
                    aux q (acc - snd t)
                  else
                    aux q acc
  in aux v 0
;;
```

Code

```
let bipart_subgraph_and_distrib g n r i_minus i_plus =
  let m = 2 * r + n in (* bipart_subgraph *)
  let minus_list = ref [] in
  let plus_list = ref [] in
  for i = 2 * r to m - 1 do
    if i_minus.(i) then
      minus_list := i :: !minus_list;
    if i_plus.(i) then
      plus_list := i :: !plus_list
  done;
  let k = min (List.length !minus_list) (List.length !plus_list) in
  let bipart = Hashtbl.create (2 * k) in
  for i = 0 to 2 * k - 1 do
    add_vertex bipart i
  done;
  let give u h =
    let give_aux v (c, f, b) =
      if i_minus.(u) && i_plus.(v) && Hashtbl.mem bipart v && Hashtbl.mem bipart u then
        add_edge bipart u v c in
    Hashtbl.iter give_aux h in
  Hashtbl.iter give g;
  (* distrib *)
  let distrib = Array.make (2 * k) (-1) in
  let rec fill l1 l2 l =
    match l1, l2 with
      | l1, l2 when l = k -> ()
      | t1 :: q1, t2 :: q2 -> begin
          distrib.(l) <- delta g t1 n r;
          distrib.(l + k) <- - (delta g t2 n r);
          fill q1 q2 (l + 1) end
      | _, _ -> failwith "impossible" in
  fill !plus_list !minus_list 0;
  bipart, distrib, k
;;
```

Code

```
let flot g =
  let f_tot = ref 0 in
  let flot_aux u h =
    let aux v (_, f, _) =
      f_tot := !f_tot + f
    in Hashtbl.iter aux h
  in Hashtbl.iter flot_aux g;
     !f_tot
;;

let kmn g n r =
  let m = 2 * r + n in
  for i = 0 to 2 * r - 1 do
    let v = Array.map (fun (x, y, z) -> x) (neighbours g i) in
    Array.iter (saturate g i) v
  done;
  let i_plus = Array.make m false in
  let i_minus = Array.make m false in
  let delta_plus = ref 0 in
  let delta_minus = ref 0 in
  for v = 2 * r to m - 1 do
    let delta_v = delta g v n r in
    if delta_v < 0 then begin
      i_minus.(v) <- true;
      delta_minus := !delta_minus + delta_v
      end
    else if delta_v > 0 then begin
      i_plus.(v) <- true;
```

Code

```
let flot g =
  let f_tot = ref 0 in
  let flot_aux u h =
    let aux v (_, f, _) =
      f_tot := !f_tot + f
    in Hashtbl.iter aux h
  in Hashtbl.iter flot_aux g;
     !f_tot
;;

let kmn g n r =
  let m = 2 * r + n in
  for i = 0 to 2 * r - 1 do
    let v = Array.map (fun (x, y, z) -> x) (neighbours g i) in
    Array.iter (saturate g i) v
  done;
  let i_plus = Array.make m false in
  let i_minus = Array.make m false in
  let delta_plus = ref 0 in
  let delta_minus = ref 0 in
  for v = 2 * r to m - 1 do
    let delta_v = delta g v n r in
    if delta_v < 0 then begin
      i_minus.(v) <- true;
      delta_minus := !delta_minus + delta_v
      end
    else if delta_v > 0 then begin
      i_plus.(v) <- true;
```

Code

```
    else if delta_v > 0 then begin
        i_plus.(v) <- true;
        delta_plus := !delta_plus + delta_v
      end
  done;
  let diff = ref (!delta_plus + !delta_minus) in
  if !diff <= 0 then begin
      let u = ref (2 *r) in
      while (!diff < 0 && !u < m) do
        if i_minus.(!u) then begin
          let v = Array.map (fun (x, y, z) -> x) (neighbours g !u) in
          let equilibrage x =
            if !diff < 0 && x >= n && x < 2 * n && delta g x n r <= 0
```

Code

```
    let equilibrage x =
      if !diff < 0 && x >= n && x < 2 * n && delta g x n r <= 0
      then (desaturate g !u x; incr diff)
    in Array.iter equilibrage v end;
   incr u
  done;
  let bipart, distrib, k = bipart_subgraph_and_distrib g n r i_plus i_minus in
  undirected_transportation bipart distrib k ;
  let give u h =
   let h_g = Hashtbl.find g u in
   let give_aux v (c, f, b) =
    Hashtbl.replace h_g v (c,f,b) in
   Hashtbl.iter give_aux h in
  Hashtbl.iter give bipart;
 end
else begin
  diff := - !diff;
  let u = ref (2 *r) in
  while (!diff < 0 && !u < m) do
   if i_minus.(!u) then begin
    let v = Array.map (fun (x, y, z) -> x) (neighbours g !u) in
    let equilibrage x =
      if !diff < 0 && x < n && delta g x n r <= 0
      then (desaturate g !u x; incr diff)
    in Array.iter equilibrage v end;
```

Code

```
        let equilibrage x =
          if !diff < 0 && x < n && delta g x n r <= 0
          then (desaturate g !u x; incr diff)
        in Array.iter equilibrage v end;
      incr u
    done;
    let bipart, distrib, k = bipart_subgraph_and_distrib g n r i_plus i_minus in
    undirected_transportation bipart distrib k;
    let give u h =
      let h_g = Hashtbl.find g u in
      let give_aux v (c, f, b) =
        Hashtbl.replace h_g v (c,f,b) in
      Hashtbl.iter give_aux h in
    Hashtbl.iter give bipart;
  end;
flot g
;;
```

Code

```
(*let ex__ = create_max_flow (3 * nb_vertex) nb_vertex;;
nb_aretes ex__;;
(*affiche_graphe ex__;;*)
kmn ex__ (3 * nb_vertex) nb_vertex;;
Sys.time ();; *)
(*affiche_graphe ex__;;
let _ex = Dinitz.create_graph ();;
Dinitz.dinitz _ex;; *)
let madrid = import_gexf "madrid_highway.gexf";;
nb_aretes madrid;;
(*let madrid_d = import_gexf_dinitz "madrid_highway.gexf";;
Dinitz.dinitz madrid_d;;*)
kmn madrid (11667 * 4) 11667;;
(*affiche_graphe madrid;;*)
(*export_graphiz madrid "madrid.dot" true;;*)

let int_sqrt n = int_of_float (sqrt (float_of_int n));;

int_sqrt 216;;
```

Code

```
let algo_final graph demande_max = (* 10h de fonctionnement *)
  let n = Hashtbl.length graph in
  let flot_total = ref 0 in
  for h = 0 to 9 do
    flot_total := !flot_total + Dinitz.dinitz (Dinitz.create_graph_n (n / 10));
    for m = 0 to 6 do
      let demande = Random.int (demande_max - demande_max / 10) + (demande_max / 10) in
      flot_total := !flot_total + kmn (create_max_flow (int_sqrt n) demande) (int_sqrt n) demande
    done
  done;
  !flot_total
;;
```

Conclusion