Algorithme

1 Algorithme

On commence par calculer un couplage par un algorithme glouton puis on le rend maximum.
S'il sature tous les sommets de X, on cherche les arêtes à enlever.

```
algo principal(g)
Data: graphe(X,Y,E)
Result: graphe(X,Y,E)
initialization;
matchingEdges ← matching(g)
matchingEdges ← matchingMax(g)
if |matchingEdges| < |X| then
 | return(false)
end
RE ← RemoveEdge(g) return(RE)
```

1.1 Couplage maximum

On regarde tous les sommets de X non saturé.
On calcule une chaine augmentante à partir de ces sommets.
On améliore le couplage, c'est à dire que si une arête du couplage est dans la chaine augmentante, on l'enlève du couplage, sinon on la met dans le couplage.

```
matchingMax(g)
Data: graphe(X,Y,E)
Result: graphe(X,Y,E)
initialization;
foreach x ∈ X do
 | if x ∉ matchingNodes then
 | | AlternatingPath ← alternatingPath(g, x, matchingNodes)
 | | (matchingNodes, matchingEdges) ←
 | | upgradeMatching(graphe(X, Y, E), AlternatingPath)
 | end
end
```

Algorithm 1: Algorithme de la fonction couplage max

1.1.1 Couplage

On regarde chaque sommet de X, si un de ses voisins y n'est pas saturé, on met l'arête (x,y) dans le couplage sinon on passe au sommet suivant.

`matching(`g`)`
Data: g : graphe(X,Y,E)
Result: $matchingNodes$: Ensemble des sommets saturees
$matchedEdges$: Ensemble des arretes dans le couplage
initialization;
foreach $x \in X$ **do**
  if $\exists\ y \in neighboor(x) \setminus matchingNodes$ **then**
    $matchingNodes \leftarrow matchingNodes \cup \{x\} \cup \{y\}$
    $matchedEdges \leftarrow matchedEdges \cup \{(x, y)\}$
  end
end
return($matchingNodes$, $matchedEdges$)

Algorithm 2: Algorithme de la fonction couplage

1.1.2 Chaine augmentante

On fait un parcours en profondeur à partir d'un sommet x de X non sature. On finit le parcours une fois qu'on trouve un sommet y de Y non saturé. On renvoie la chaine entre x et y.

`alternatingPath(`g, x`)`
Data: graphe(X,Y,E)
$matchingNodes$: Ensemble des sommets saturees
x : sommet de depart
$visited$: Les sommets deja visites
Result:
$path$: La chaine augmentante
$stack$: Les futurs sommets a explorer
initialization;
$visited \leftarrow visited \cup \{x\}$
if $neighboor(x) \setminus visited = \emptyset$ **then**
  On revient en arriere
else
  $y \leftarrow FirstElement(Stack)$
  $remove(y, Stack)$
  $AlternatingPath_s \leftarrow y$
  if $y \notin matchingNodes$ **then**
    $return(AlternatingPath_s)$
  end
  $x \leftarrow$ voisin qui va bien de y
  $AlternatingPath_s \leftarrow x$
  $alternatingPath(g, x, matchingNodes, AlternatingPath_s, Stack)$
end

Algorithm 3: Algorithme de la fonction chaine augmentante

1.2 On enleve les arretes

On fait un parcours à partir de tous les sommets non saturés de Y, on marque toutes les arêtes parcourues comme used.
On marque toutes les arêtes appartenant à une composante fortement connexe comme used à l'aide de l'algorithme de tarjan.
On enlève toutes les arêtes qui ne sont pas marquées comme used et qui ne sont pas dans le couplage maximum calculé précédemment.

```
RemoveEdge(g)
Data: graphe(X,Y,E)
Result: graphe(X,Y,E)
initialization;
foreach y ∈ Y do
    if y ∉ Sat_s then
        Used ← dfs(g, y)
    end
end
P ← Tarjan(g) foreach C ∈ P do
    if |C| ≥ 1 then
        Used ← edge(C)
    end
end
foreach e ∈ E do
    if e ∉ Used then
        if e ∈ matchingEdges then
            vital ← e
        end
        RE ← e remove(e, g)
    end
end
return(RE)
```

1.2.1 Tarjan

`Tarjan(`g`)`
Data: graphe(X,Y,E)
Result: $Partition$: Les differentes composantes connexes
initialization;
$num \leftarrow 0$
$P \leftarrow \emptyset$
$Partition \leftarrow \emptyset$
foreach $x \in X$ **do**
    if $num(x)$ *non definit* **then**
        $Partition \leftarrow StrongConnect(x)$
    end
end
$return(Partition)$

Algorithm 4: Algorithme de la fonction parcours

1.2.2 Composante connexe

```
StrongConnect(x)
Data: graphe(X,Y,E)
Result: Partition
initialization;
y ← voisin qui va bien x
num(x) = num
num_accessible(x) = num
num = num + 1
num(y) = num
num_accessible(y) = num
num = num + 1
P ← x ∪ y
C ← ∅
foreach x ∈ voisin(y) do
    if num(x) non definit then
        StrongConnect(x)
        num_accessible(y) = min(num_accessible(y), num_accessible(x))
    else
        if x ∈ P then
            num_accessible(y) = min(num_accessible(y), num(x))
    end
end
if num_accessible(x) = num(x) then
    while y ∈ P ≠ x do
        remove y p
        C ← y
    end
    Partition ← C
end
return(partition)
```

Algorithm 5: Algorithme de Tarjan (composante connexe)