Vérification formelle et optimisation de l'allocation de registres

Benoît Robillard

Laboratoire CEDRIC - Équipes CPR et OC

CNAM - ENSIIE

Soutenance de thèse

30 Novembre 2010

Compilation et allocation de registres

Machine

Code exécutable

Compilation et allocation de registres

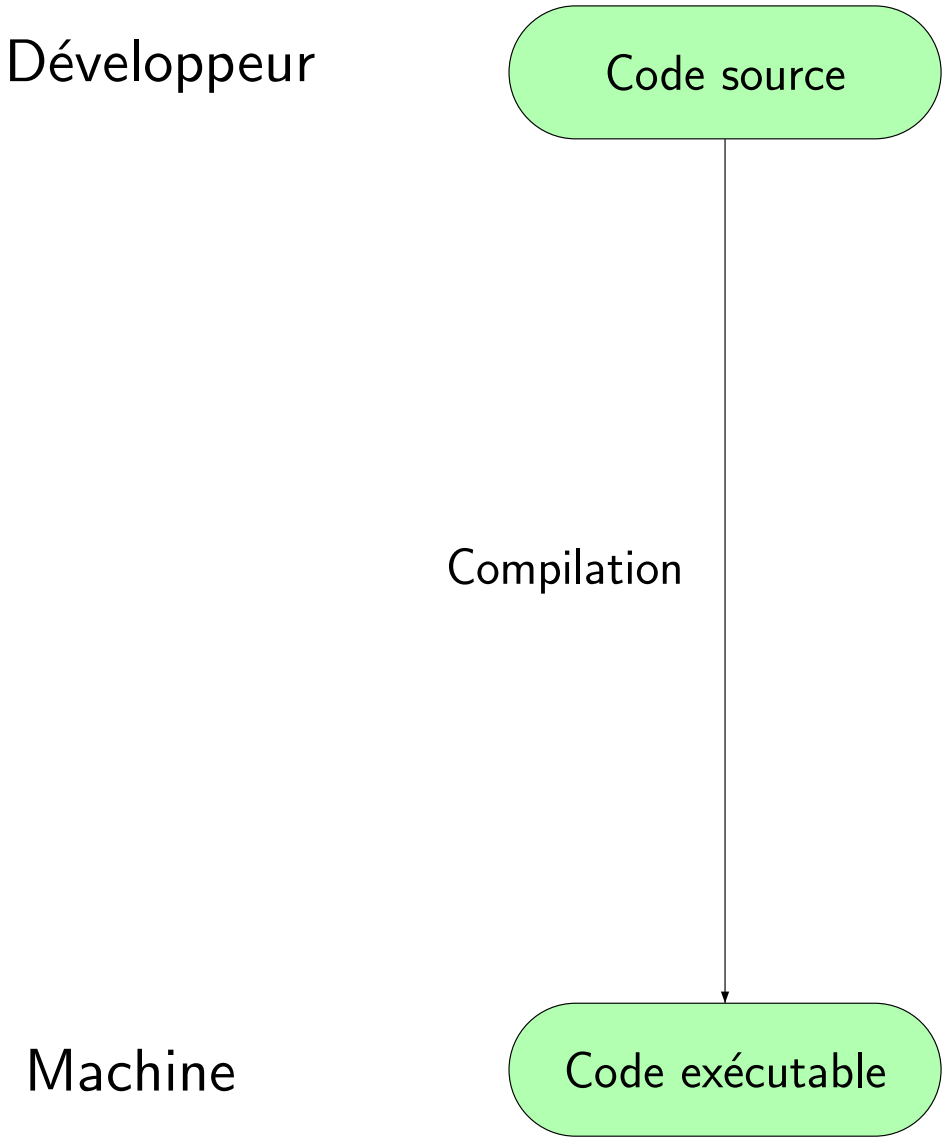
Développeur

Code source

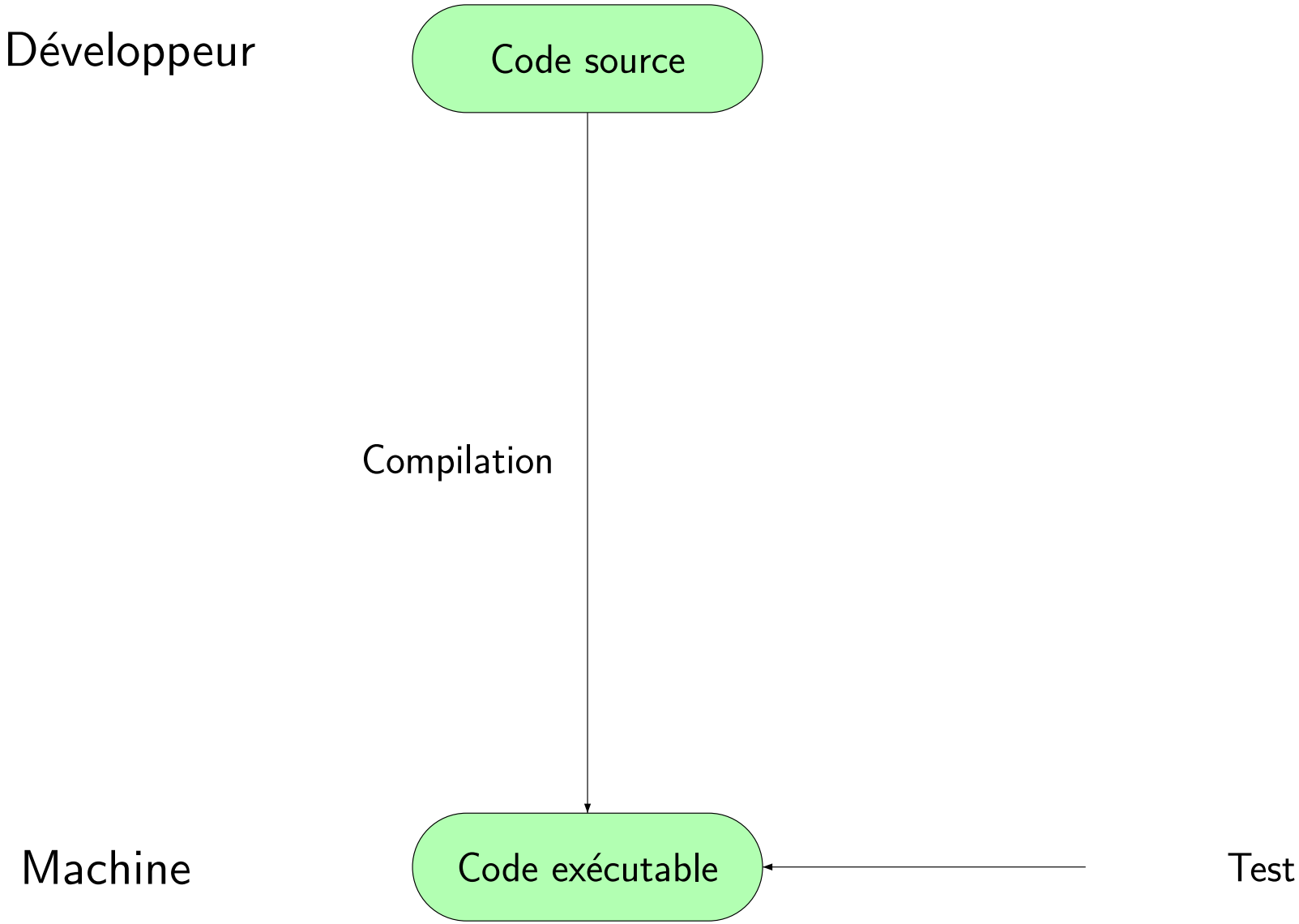
Machine

Code exécutable

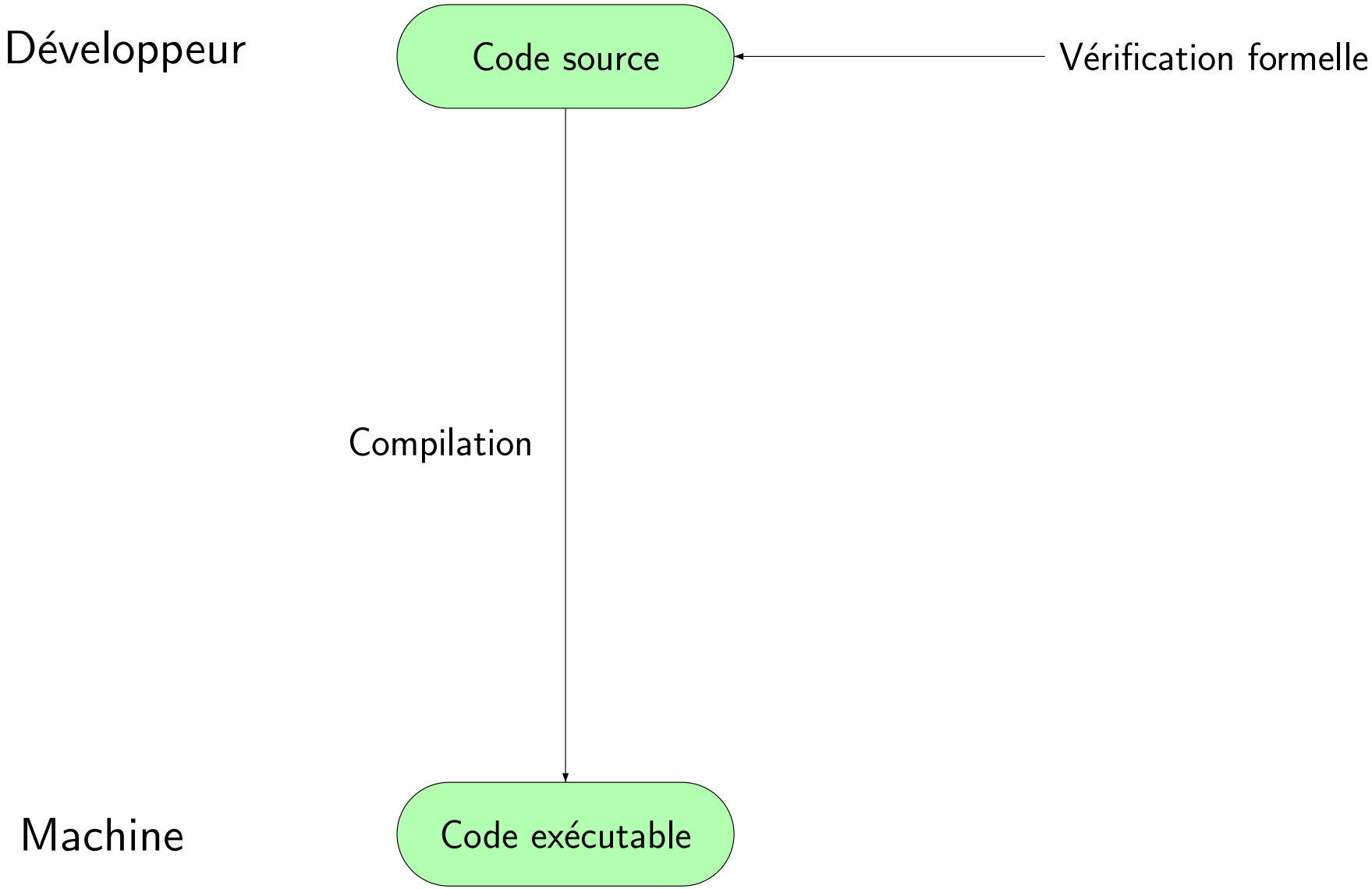
Compilation et allocation de registres



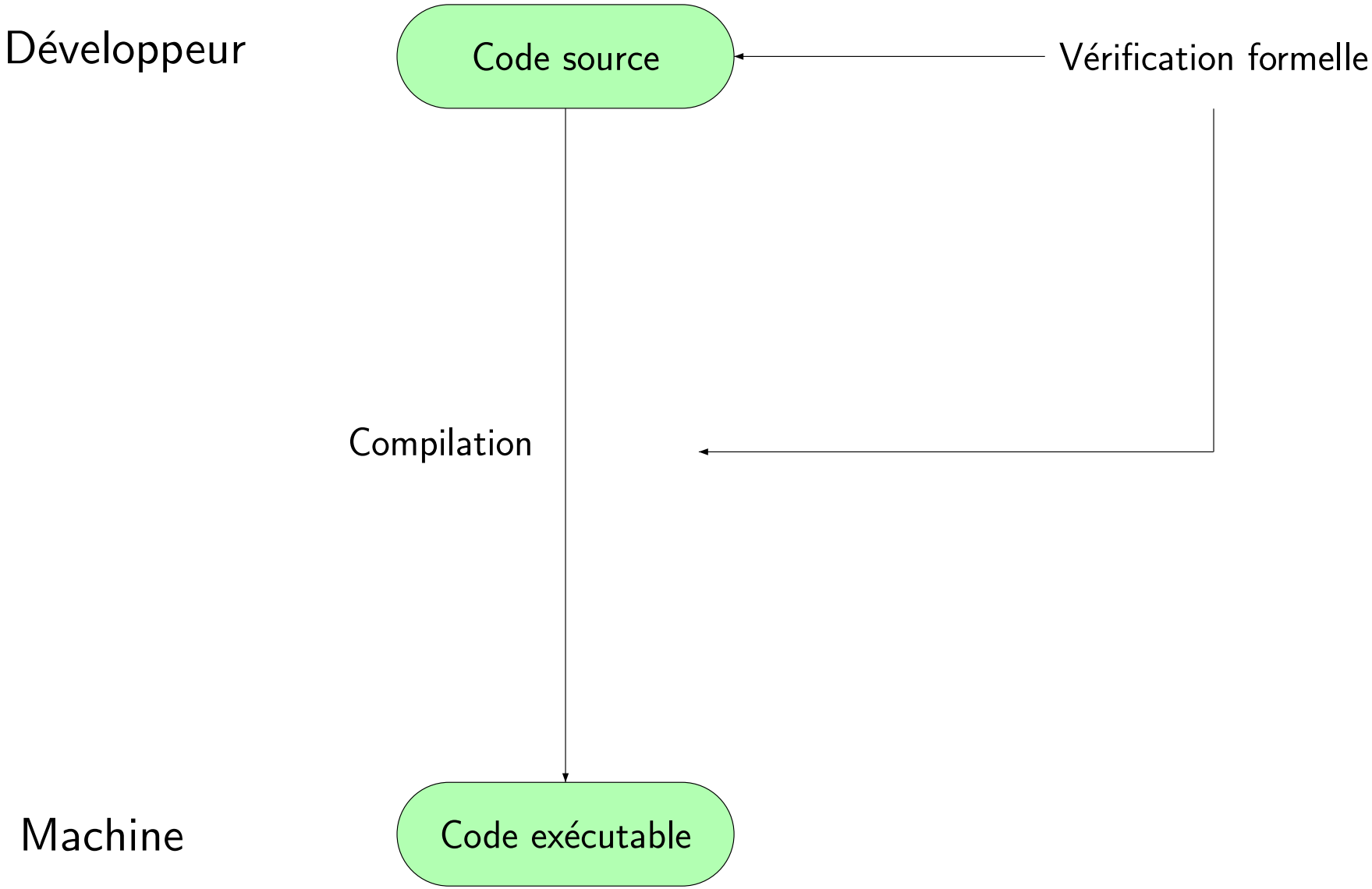
Compilation et allocation de registres



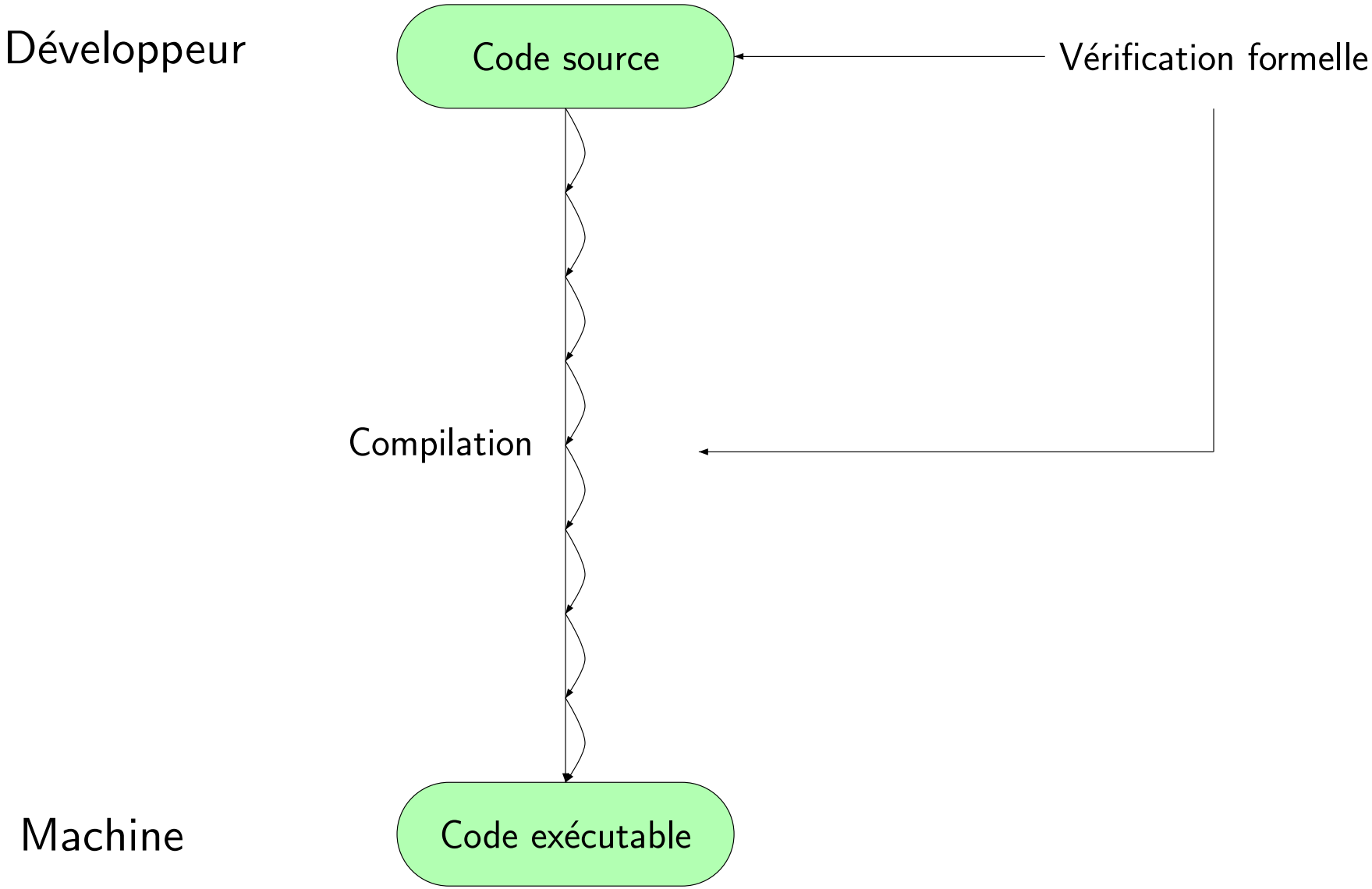
Compilation et allocation de registres



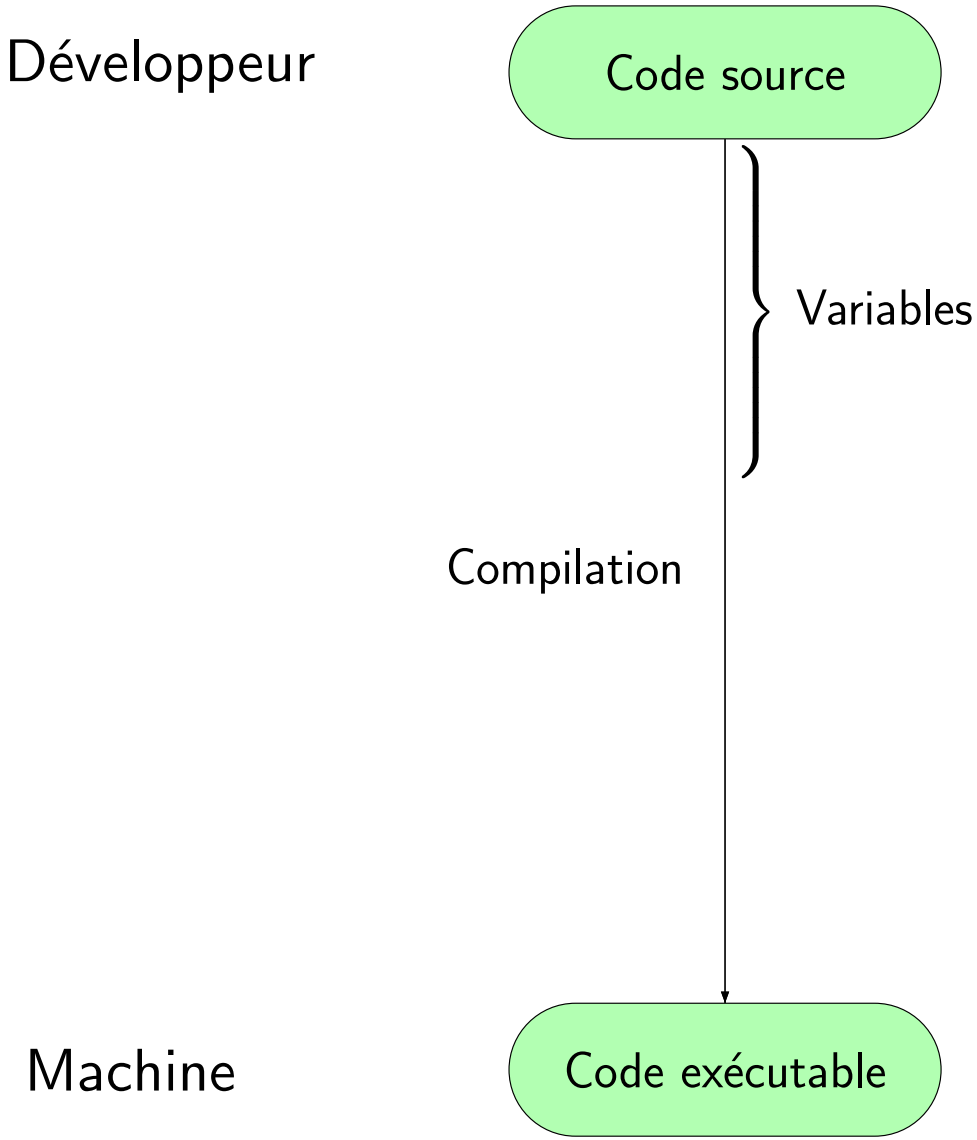
Compilation et allocation de registres



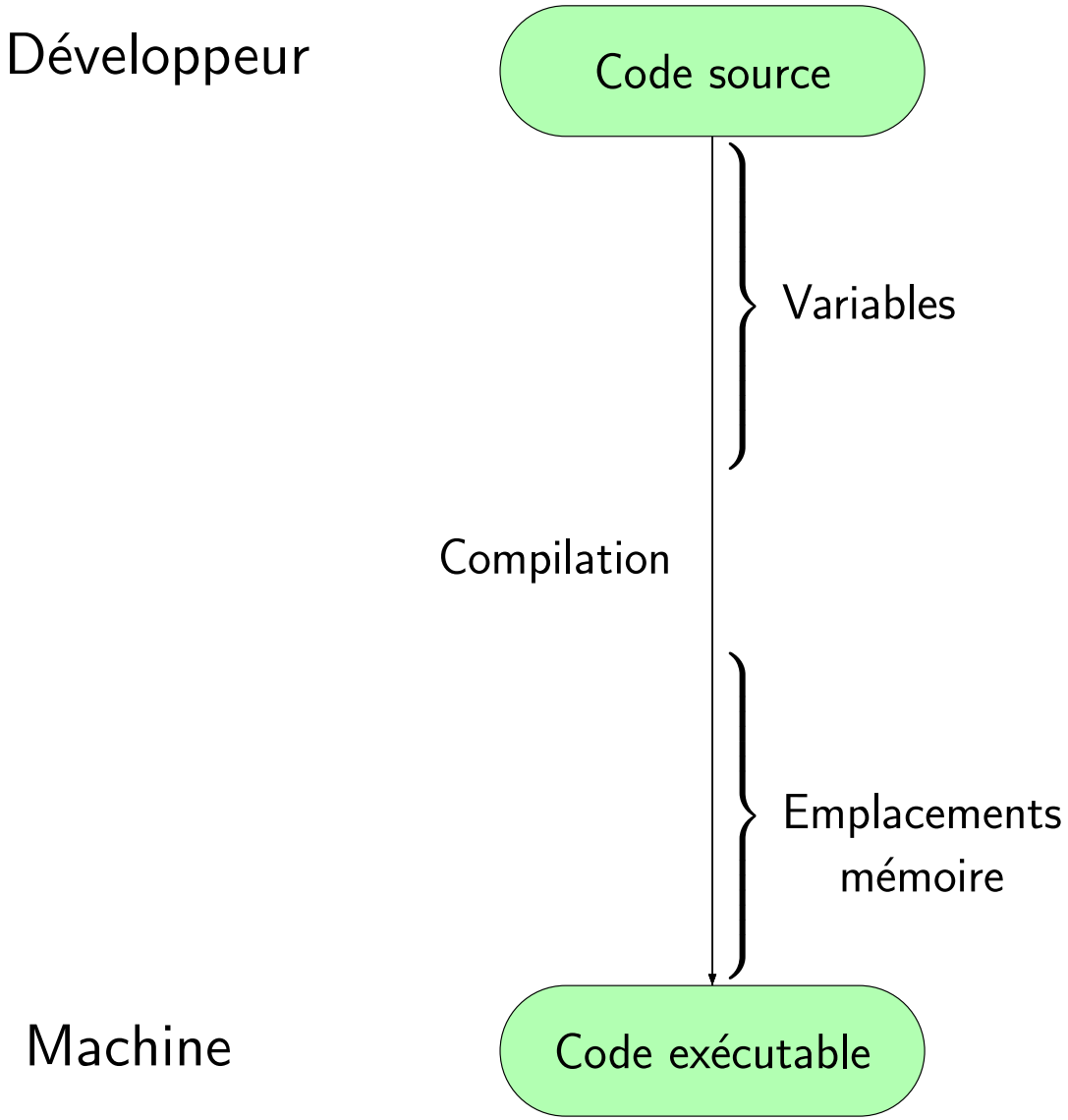
Compilation et allocation de registres



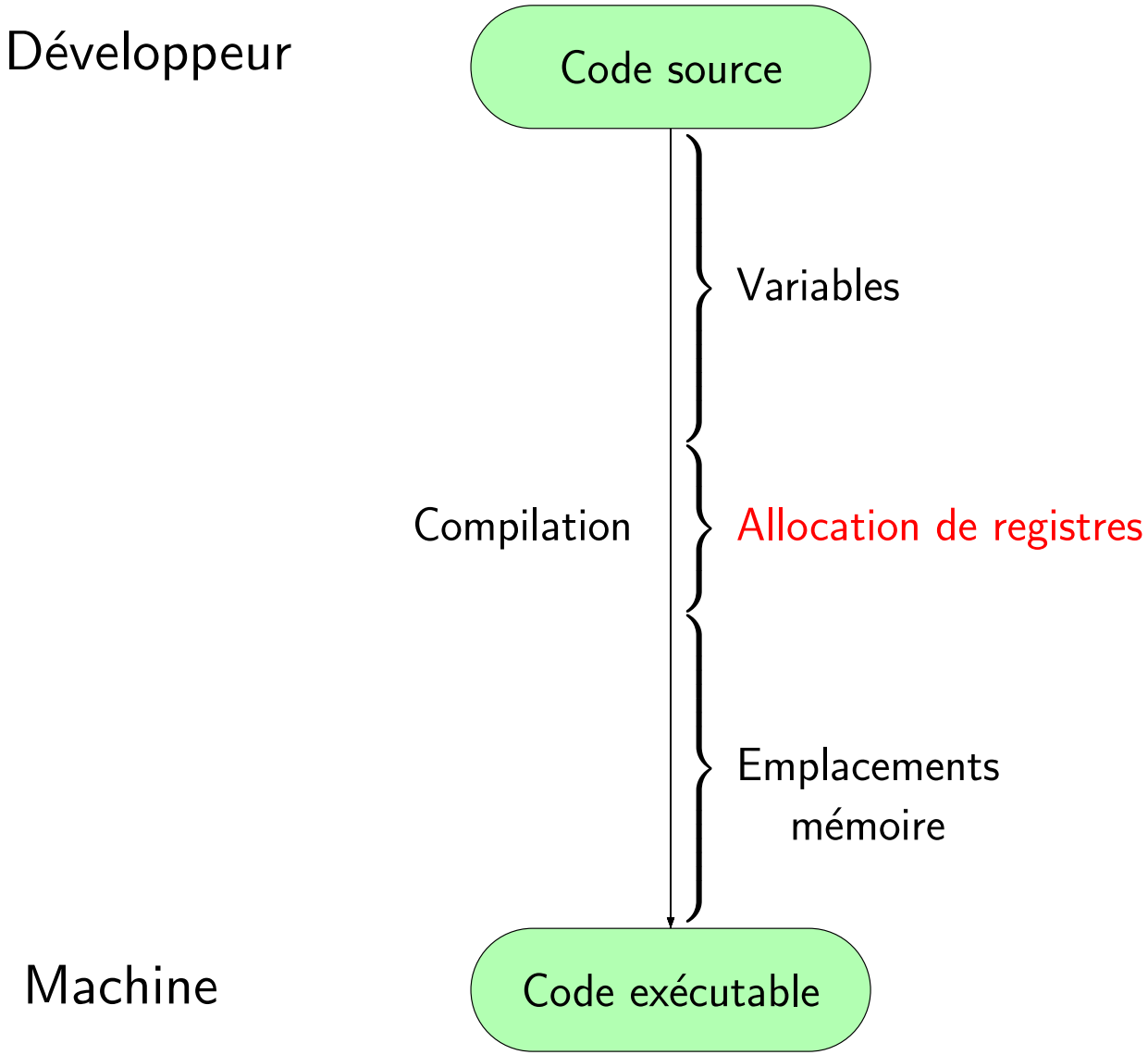
Compilation et allocation de registres



Compilation et allocation de registres



Compilation et allocation de registres



L'allocation de registres

Deux types d'emplacements mémoire

1. Les registres, accès rapide, nombre limité
2. La pile, accès lent, très grande capacité

L'allocation de registres

Deux types d'emplacements mémoire

1. Les registres, accès rapide, nombre limité
2. La pile, accès lent, très grande capacité

Allocation de registres

Transition d'un nombre quelconque de variables à un nombre fixé de registres

L'allocation de registres

Deux types d'emplacements mémoire

1. Les registres, accès rapide, nombre limité
2. La pile, accès lent, très grande capacité

Allocation de registres

Transition d'un nombre quelconque de variables à un nombre fixé de registres

Question

Quelle variable placer dans quel emplacement mémoire à tout moment de l'exécution du programme ?

But

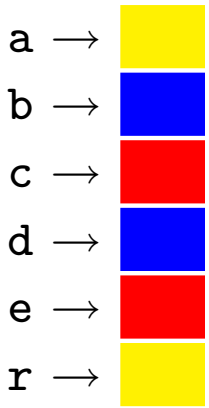
Utiliser au maximum les registres et au minimum la pile

Deux allocations de registres

```
int a = 9;
int b = 5;
int c = a + b;
int d = b + c;
int e = c + d;
int r = a + e;
```

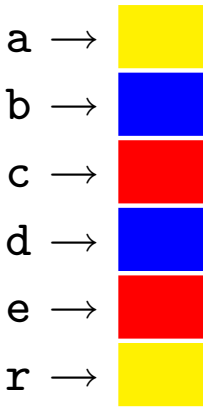
Deux allocations de registres

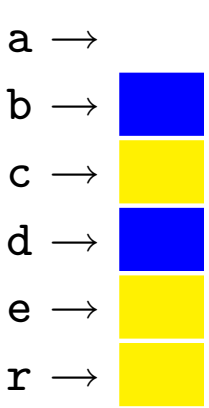
```
int a = 9;
int b = 5;
int c = a + b;
int d = b + c;
int e = c + d;
int r = a + e;
```



Deux allocations de registres

```
int a = 9;
int b = 5;
int c = a + b;
int d = b + c;
int e = c + d;
int r = a + e;
```





Graphe d'interférence-affinité

- Sommets = Variables du programme
- Deux ensembles d'arêtes :
  1. Interférence $(v_1, v_2) \Rightarrow v_1$ et v_2 dans des registres différents
  2. Affinité $(v_1, v_2) \Rightarrow v_1$ et v_2 opérandes d'une instruction `move`

Graphe d'interférence-affinité

- Sommets = Variables du programme
- Deux ensembles d'arêtes :
  1. Interférence $(v_1, v_2) \Rightarrow v_1$ et v_2 dans des registres différents
  2. Affinité $(v_1, v_2) \Rightarrow v_1$ et v_2 opérandes d'une instruction `move`

Modèle

Une couleur = Un registre

Sommets liés par une interférence $\Rightarrow$ couleurs différentes

Sommets liés par une affinité $\Rightarrow$ préférablement de même couleur

Allocation de registres par coloration

Définition : Coloration de graphe d'interférence-affinité

Une coloration d'un graphe d'interférence-affinité est une fonction qui affecte à chaque sommet (au plus) une couleur, telle que deux sommets liés par une interférence ont des couleurs différentes. Une p-coloration est une coloration utilisant au plus p couleurs.

Allocation de registres par coloration

Définition : Coloration de graphe d'interférence-affinité

Une coloration d'un graphe d'interférence-affinité est une fonction qui affecte à chaque sommet (au plus) une couleur, telle que deux sommets liés par une interférence ont des couleurs différentes. Une p-coloration est une coloration utilisant au plus p couleurs.

K-coloration de G = Affectation valide des registres

Allocation de registres par coloration

Définition : Coloration de graphe d'interférence-affinité

Une coloration d'un graphe d'interférence-affinité est une fonction qui affecte à chaque sommet (au plus) une couleur, telle que deux sommets liés par une interférence ont des couleurs différentes. Une p-coloration est une coloration utilisant au plus p couleurs.

K-coloration de G = Affectation valide des registres

But

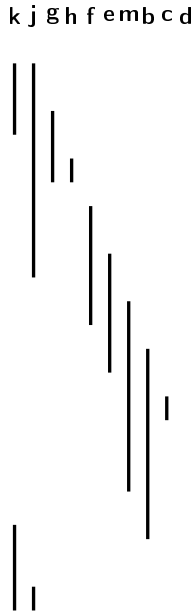
Trouver une K-coloration de G maximisant la somme des poids des affinités satisfaites

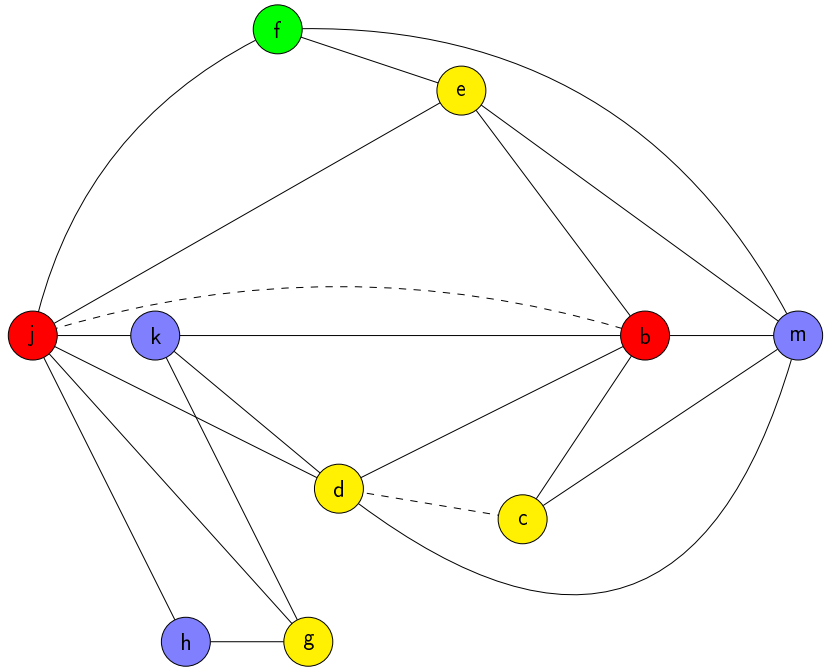
Exemple (K = 4)

Entrée : k j

(p_0)	g = mem[j+12]
(p_1)	h = k - 1
(p_2)	f = g * h
(p_3)	e = mem[j+8]
(p_4)	m = mem[j+16]
(p_5)	b = mem[f]
(p_6)	c = e + 8
(p_7)	d = c
(p_8)	k = m + 4
(p_9)	j = b
(p_{10})	

Sortie : d j k





Quatre composantes de l'allocation de registres

Vidage (« spilling »)

Quelles variables en registre ?

Déterminer un sous-graphe K-colorable de taille maximale

Quatre composantes de l'allocation de registres

Vidage (« spilling »)

Quelles variables en registre ?

Déterminer un sous-graphe K-colorable de taille maximale

Fusion

Quelles instructions `move` supprimer du code ?

Fusionner un maximum d'affinités

Quatre composantes de l'allocation de registres

Vidage (« spilling »)

Quelles variables en registre ?

Déterminer un sous-graphe K-colorable de taille maximale

Fusion

Quelles instructions `move` supprimer du code ?

Fusionner un maximum d'affinités

Affectation des registres

Quelles variables dans quels registres ?

Colorer le graphe avec K couleurs

Quatre composantes de l'allocation de registres

Vidage (« spilling »)

Quelles variables en registre ?

Déterminer un sous-graphe K-colorable de taille maximale

Fusion

Quelles instructions `move` supprimer du code ?

Fusionner un maximum d'affinités

Affectation des registres

Quelles variables dans quels registres ?

Colorer le graphe avec K couleurs

Insertion du « spill code »

Comment modifier le programme ?

Ne faire apparaître que des emplacements mémoire (load,store)

Problématique

Comment gagner confiance en l'allocation de registres et ses optimisations pour les systèmes embarqués critiques ?

Comment définir des allocateurs de registres encore plus optimisants ?

Problématique

Comment gagner confiance en l'allocation de registres et ses optimisations pour les systèmes embarqués critiques ?

1. Vérification formelle d'une heuristique optimisante
   Compilateur CompCert

Comment définir des allocateurs de registres encore plus optimisants ?

Problématique

Comment gagner confiance en l'allocation de registres et ses optimisations pour les systèmes embarqués critiques ?

1. Vérification formelle d'une heuristique optimisante
   Compilateur CompCert

Comment définir des allocateurs de registres encore plus optimisants ?

Optimisation de la fusion

Problématique

Comment gagner confiance en l'allocation de registres et ses optimisations pour les systèmes embarqués critiques ?

1. Vérification formelle d'une heuristique optimisante
   Compilateur CompCert

Comment définir des allocateurs de registres encore plus optimisants ?

Optimisation de la fusion
Gestion des affinités globale et spécifique

Problématique

Comment gagner confiance en l'allocation de registres et ses optimisations pour les systèmes embarqués critiques ?

1. Vérification formelle d'une heuristique optimisante
   Compilateur CompCert

Comment définir des allocateurs de registres encore plus optimisants ?

   Optimisation de la fusion
   Gestion des affinités globale et spécifique

2. Forme SSA
3. Forme élémentaire

Vérification formelle de l'Iterated Register Coalescing (IRC)

1. L'allocation de registres de CompCert
2. L'heuristique
3. Implantation fonctionnelle de l'IRC
4. Les graphes d'interférence-affinité
5. Correction de l'algorithme
6. Statistiques numériques

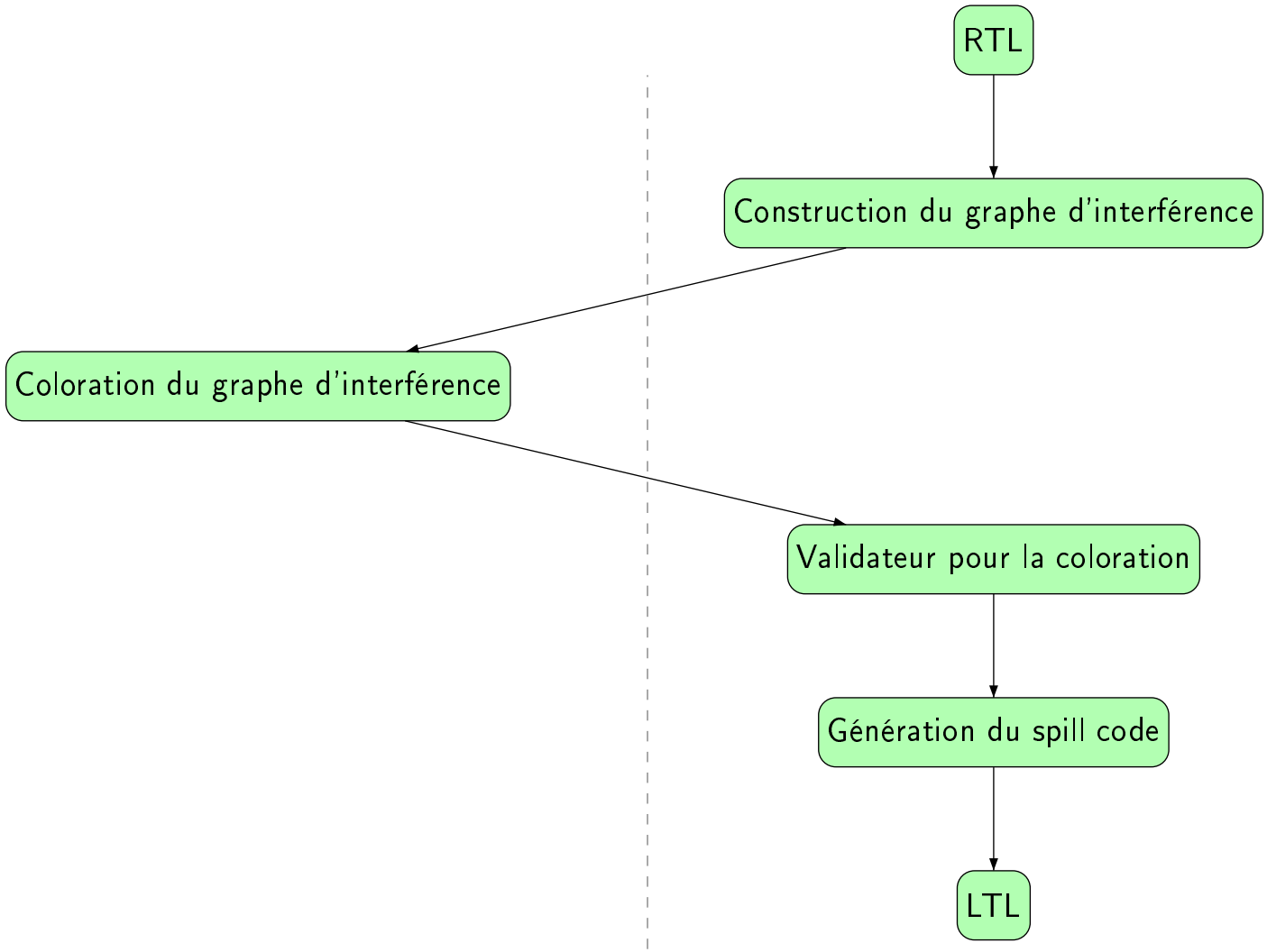
Le compilateur CompCert

Compilateur de C $\longrightarrow$ PowerPC/ARM/x86, spécifié, développé, prouvé correct dans le système Coq

Allocation de registres : Iterated Register Coalescing (IRC) [GA96]

IRC efficace sur les architectures dotées de nombreux registres comme PowerPC/ARM

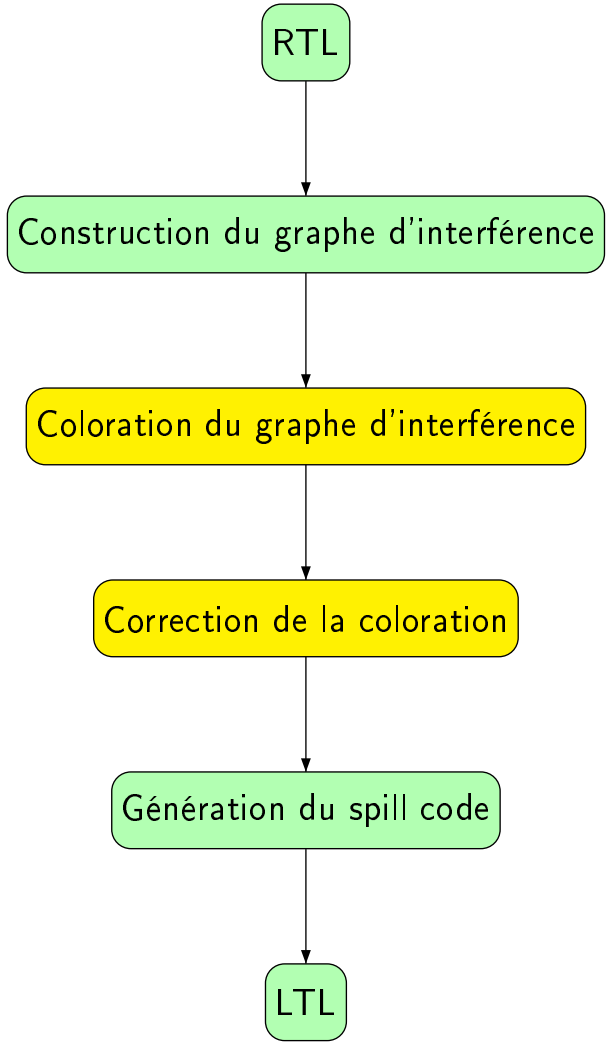
L'allocation de registres de CompCert



Développé en Caml et non prouvé

Développé et prouvé dans le système Coq

L'allocation de registres de CompCert

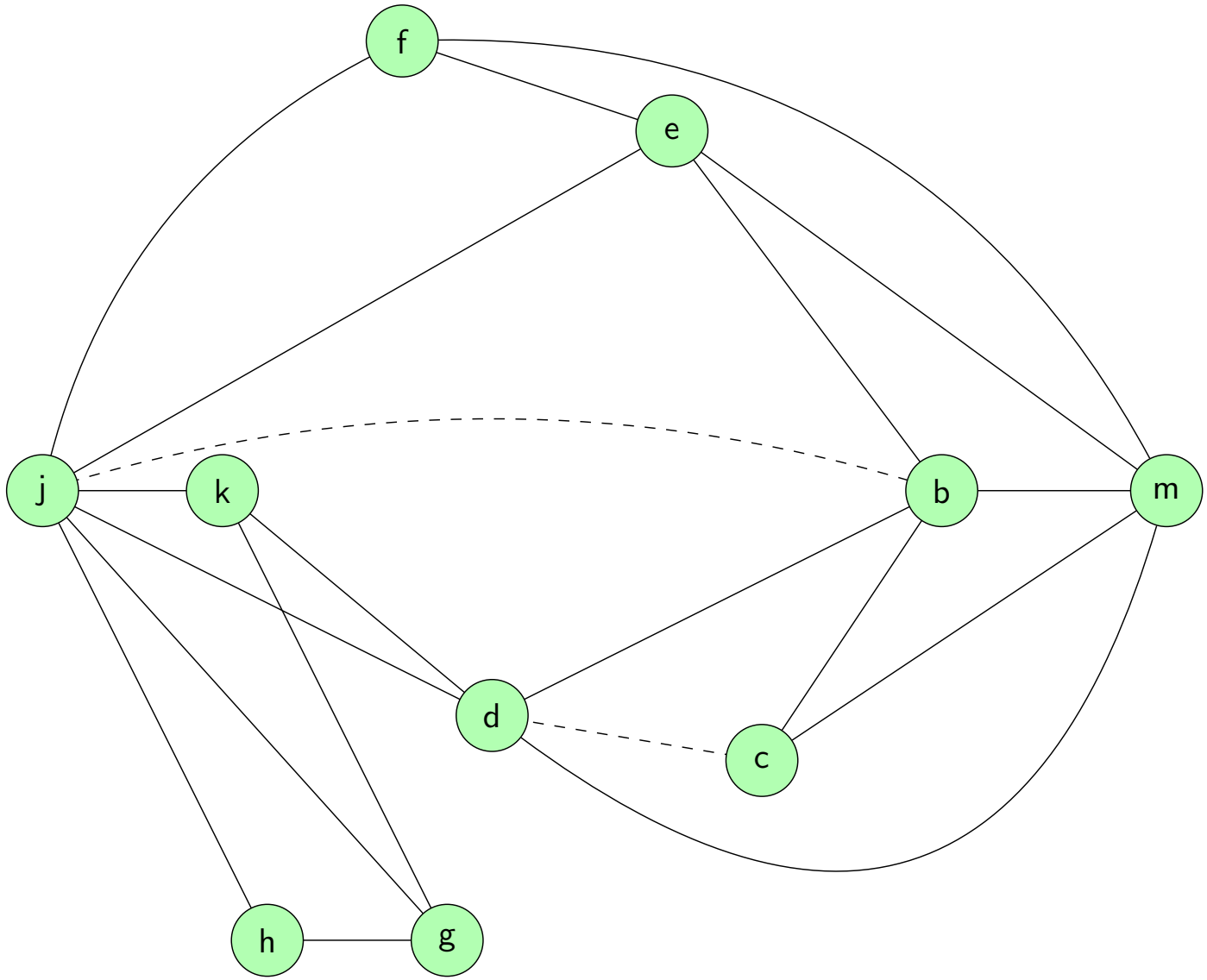


Développé en Caml et non prouvé

Développé et prouvé dans le système Coq

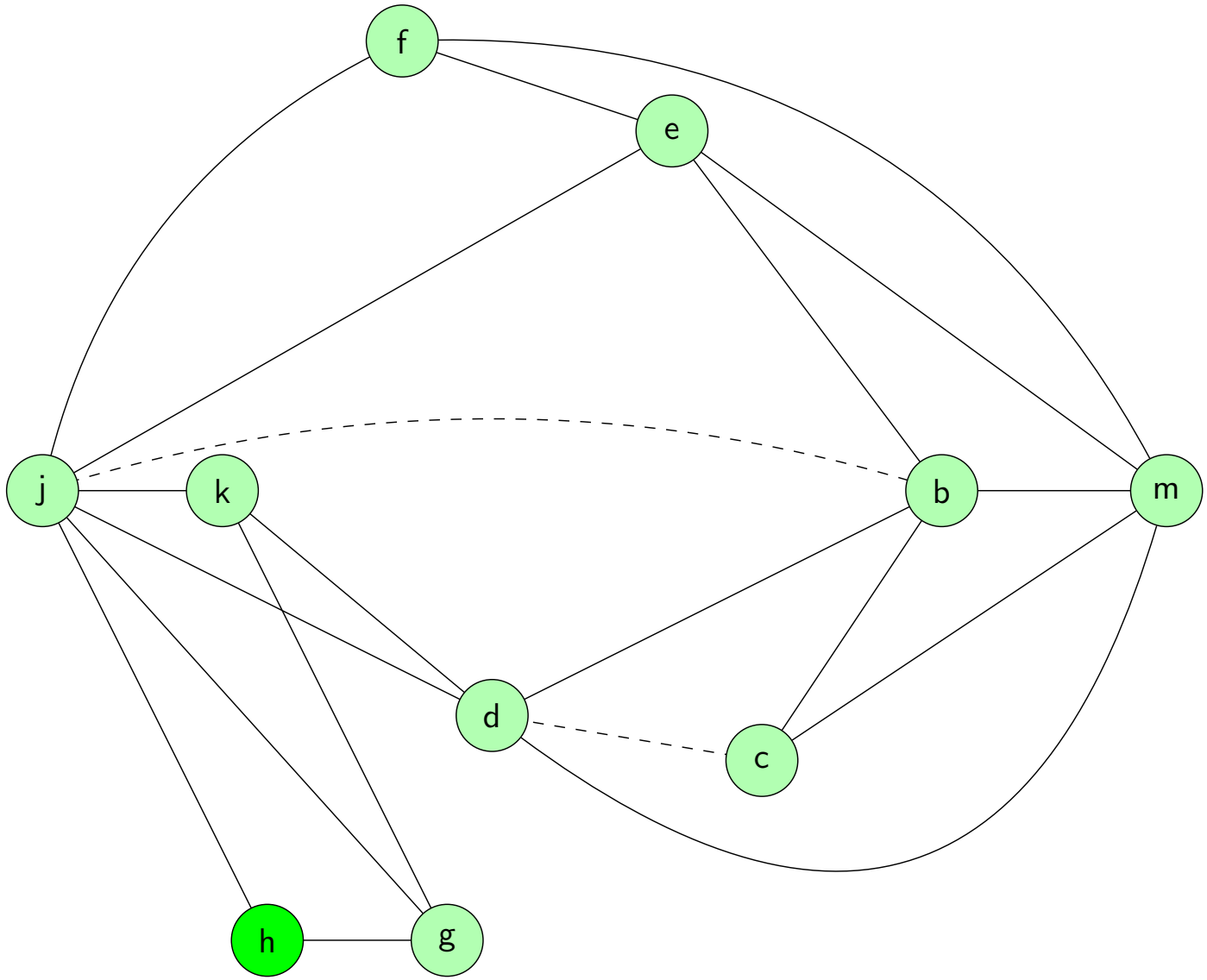
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



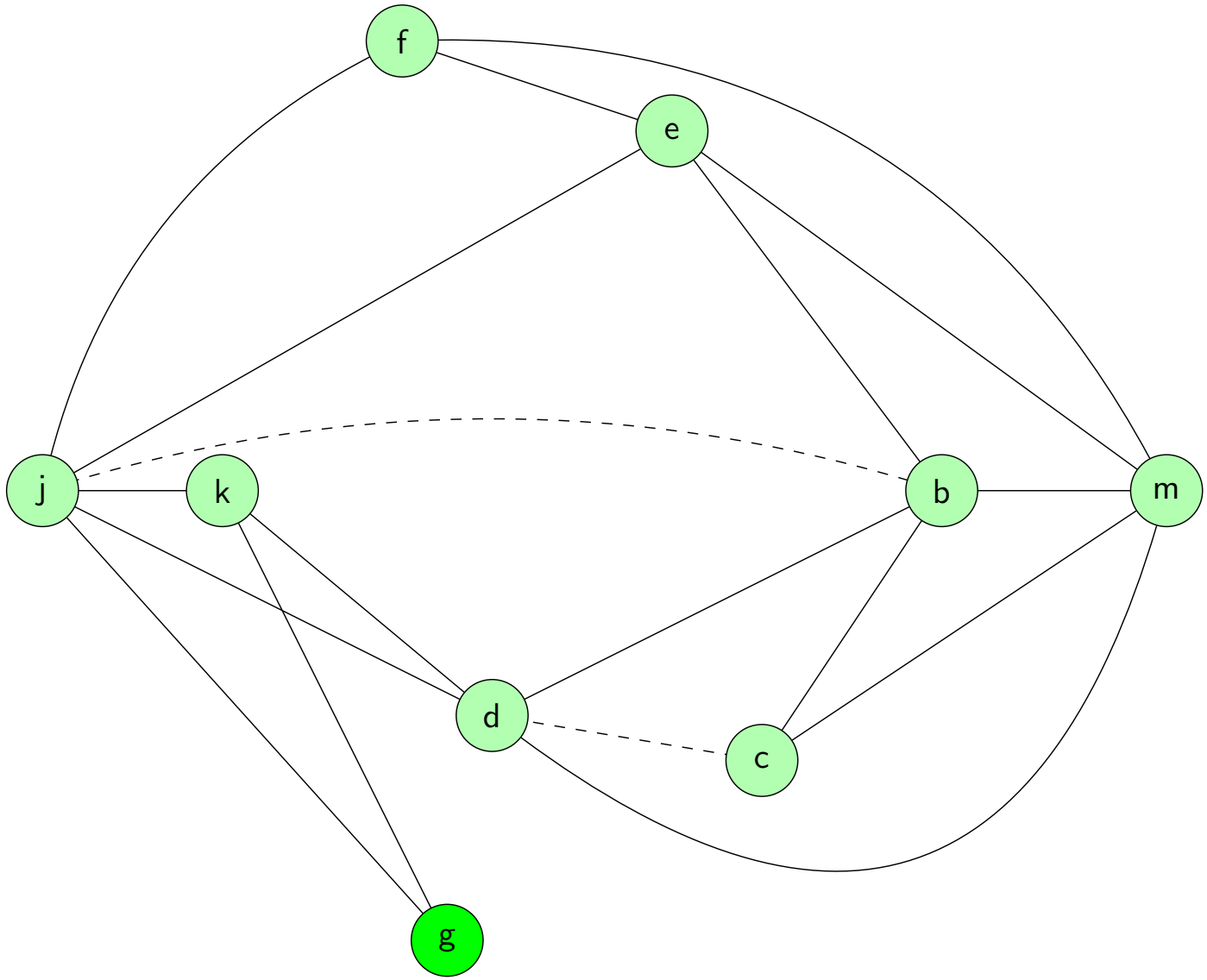
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



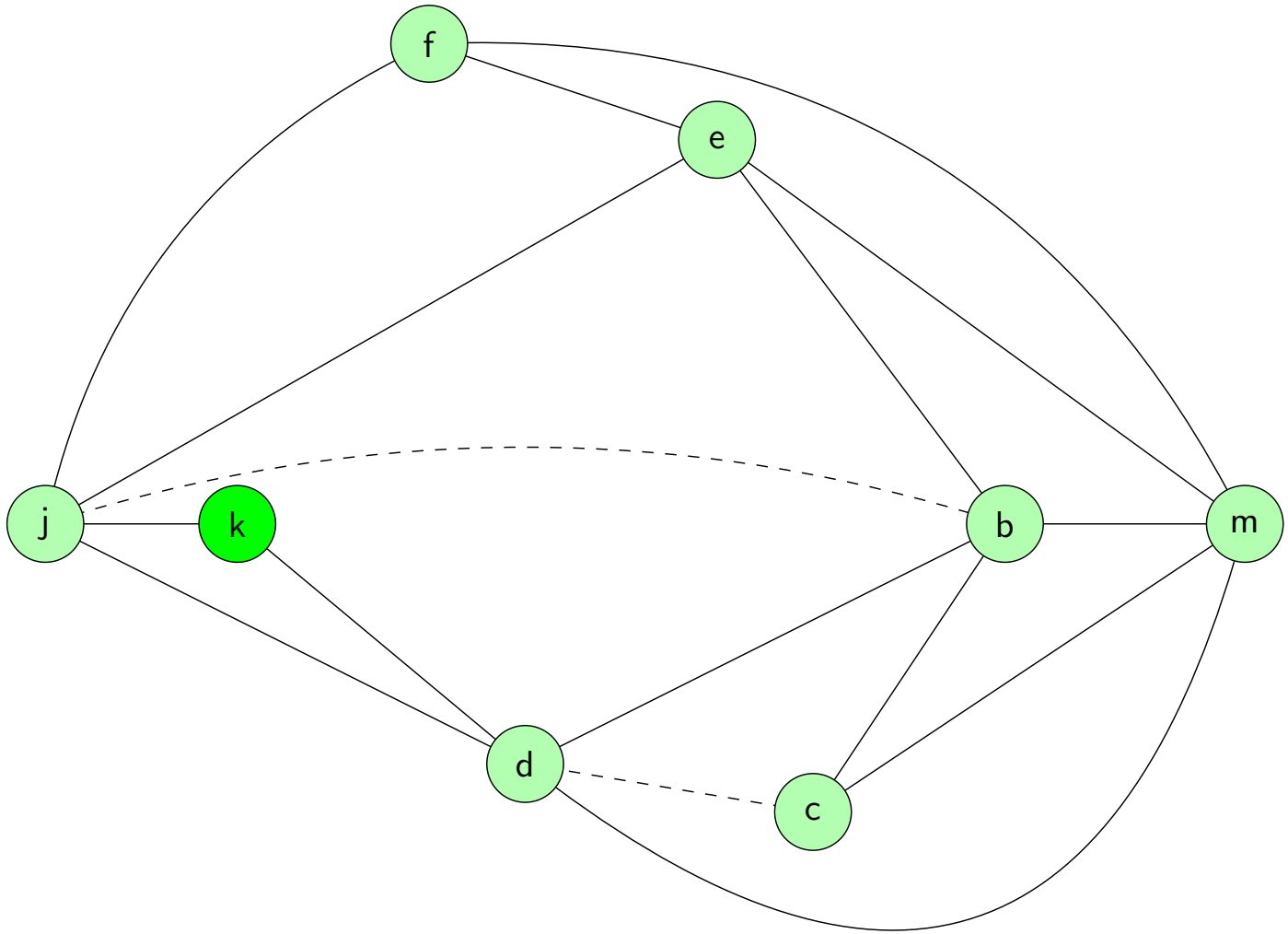
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



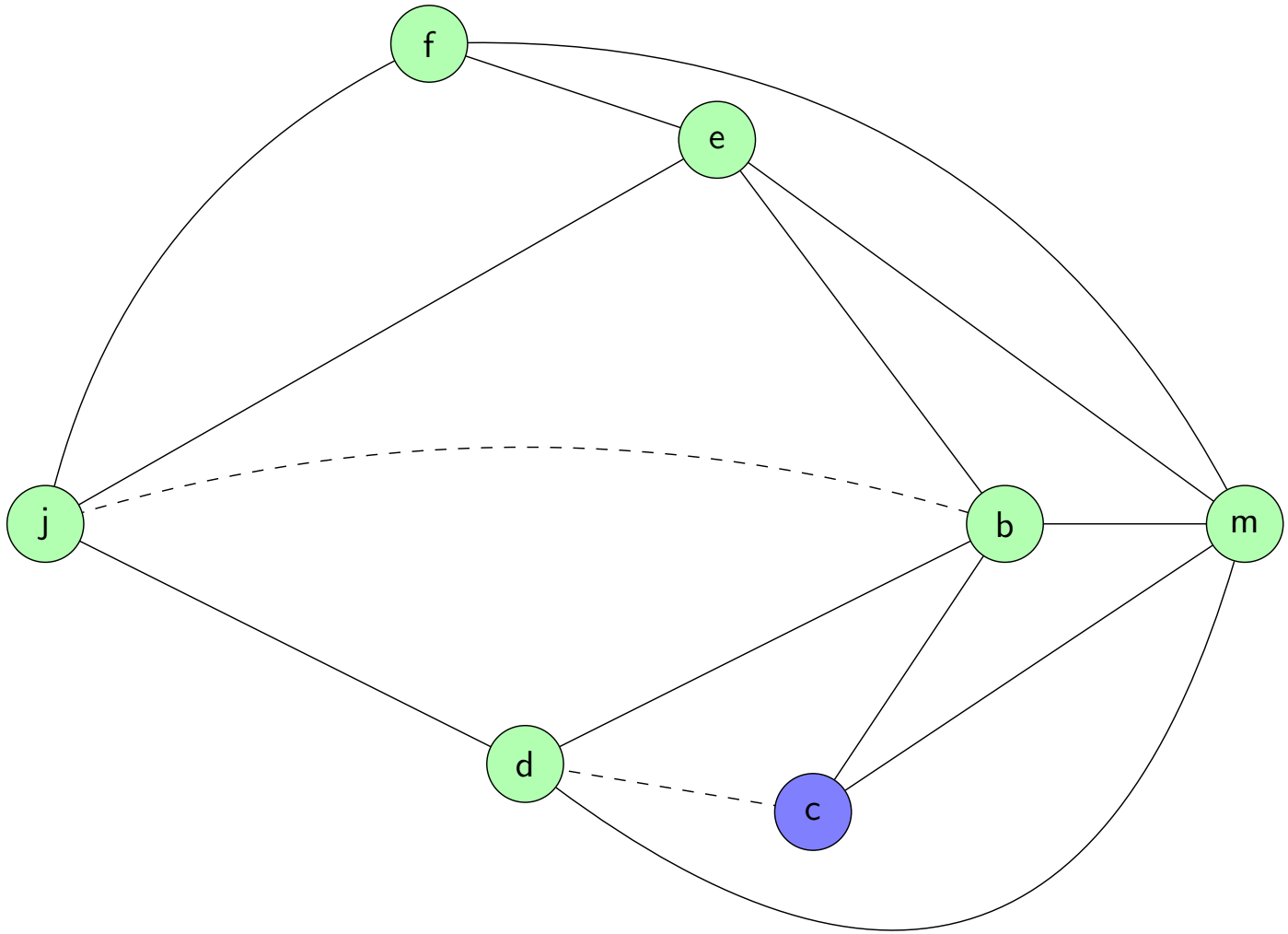
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



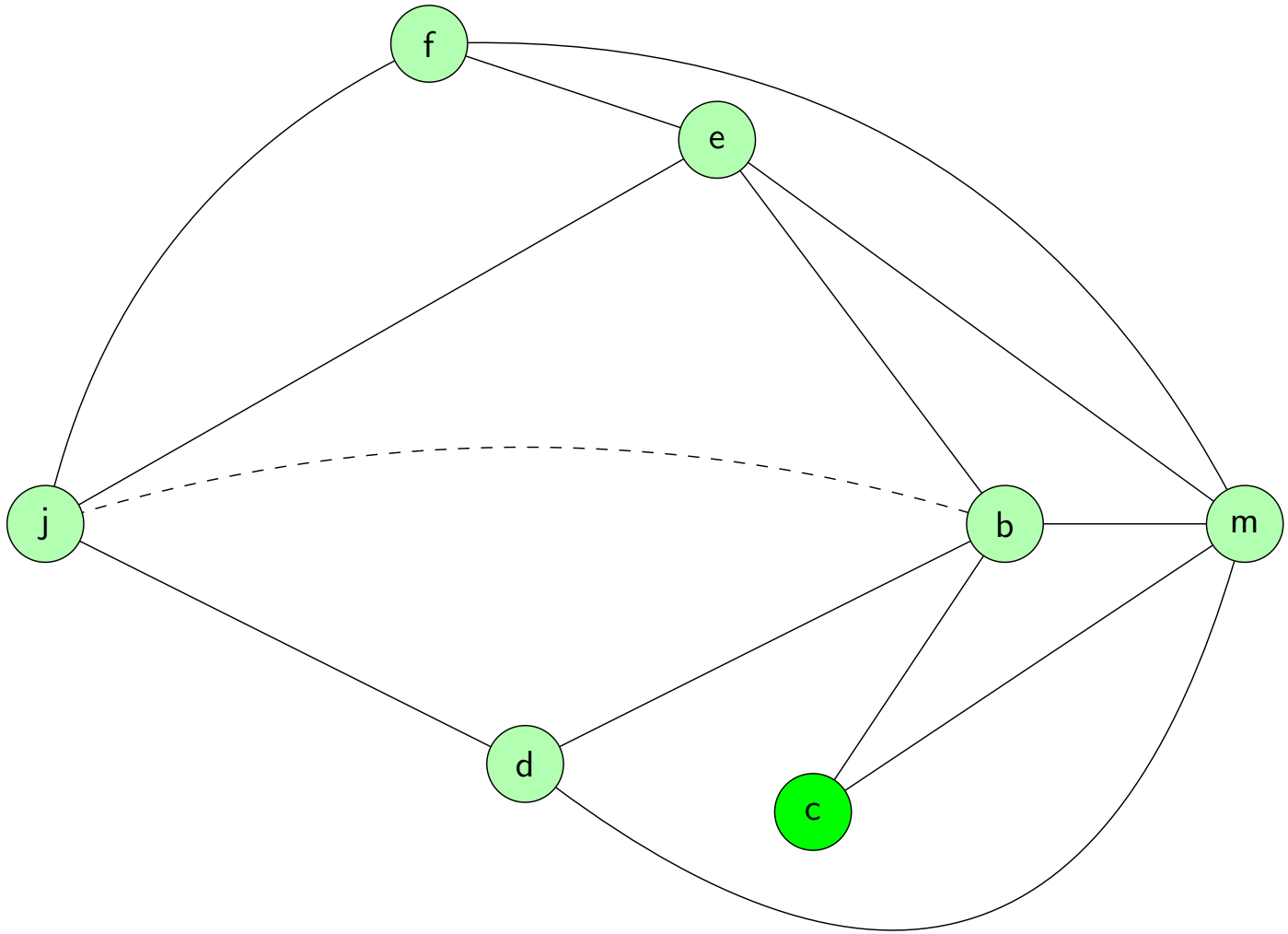
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



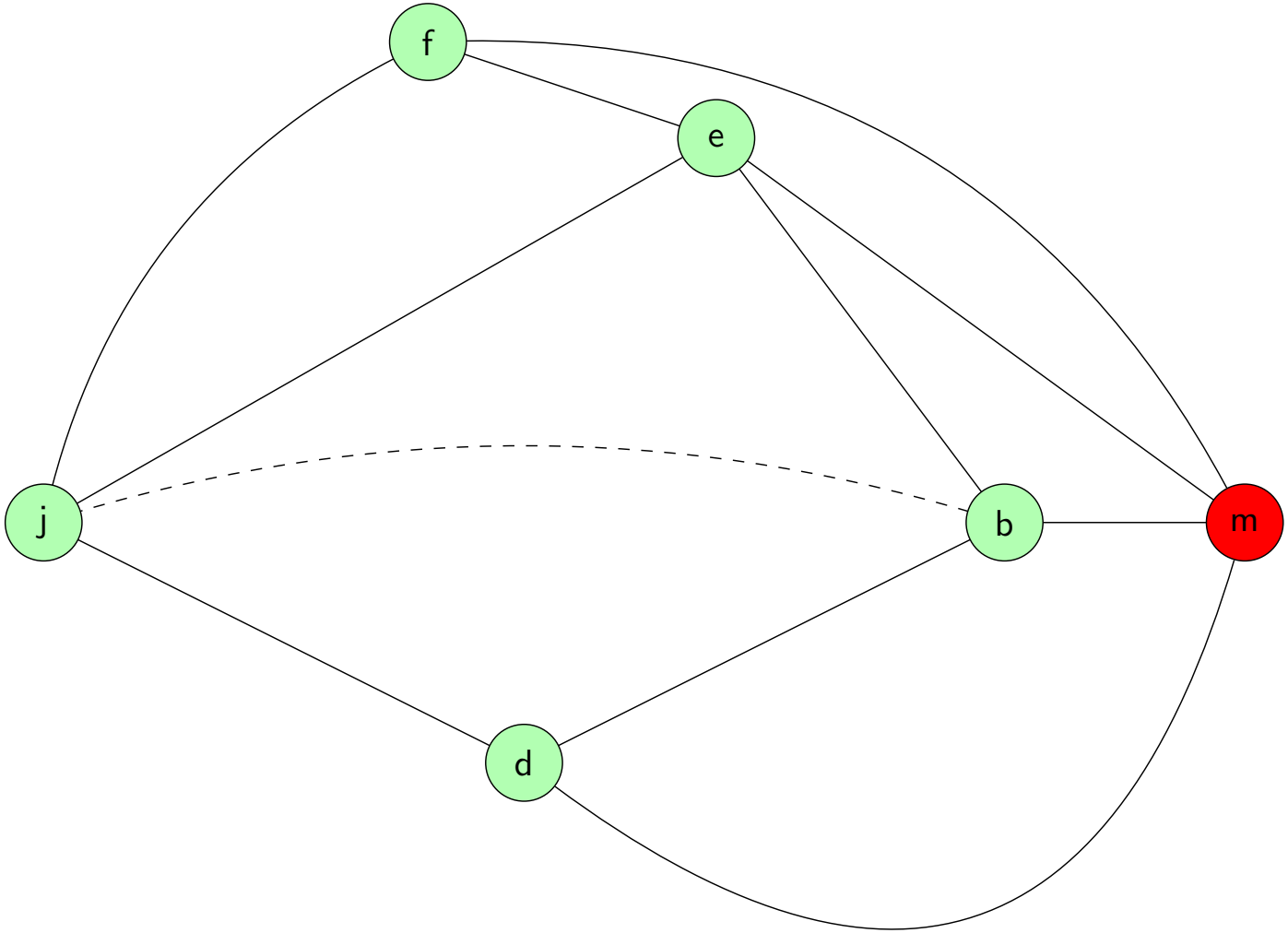
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



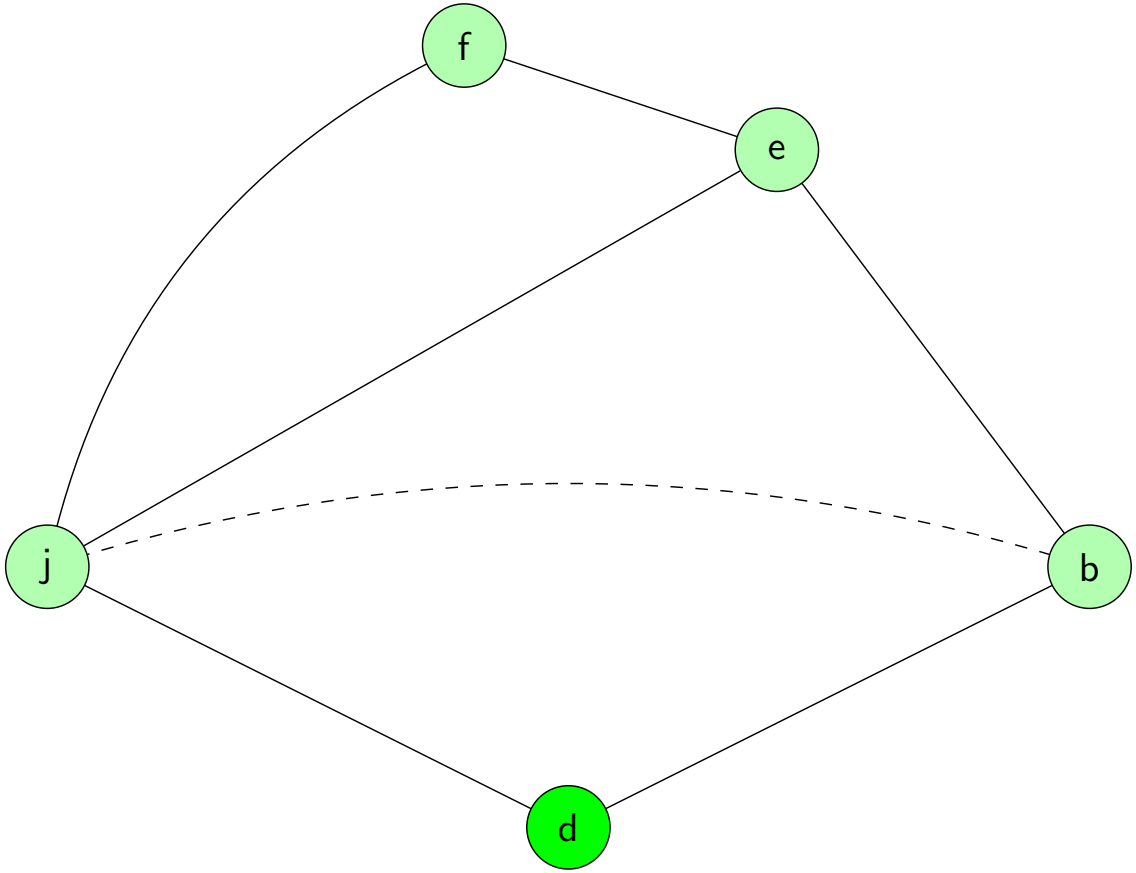
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



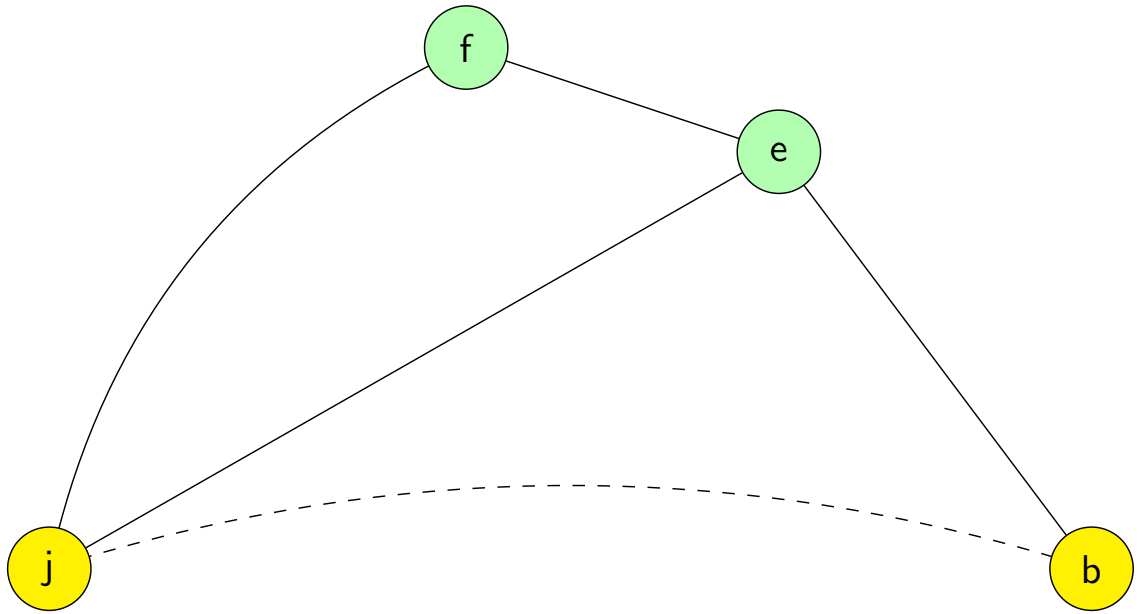
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



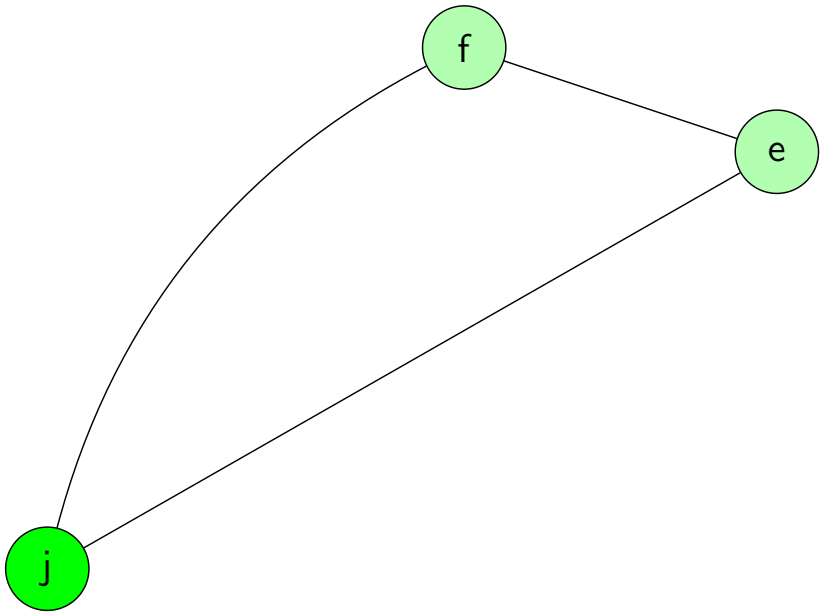
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



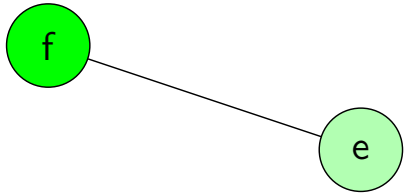
L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage



L'heuristique par l'exemple (K = 3)

simplification fusion gel vidage

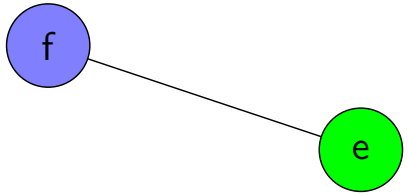


L'heuristique par l'exemple (K = 3)

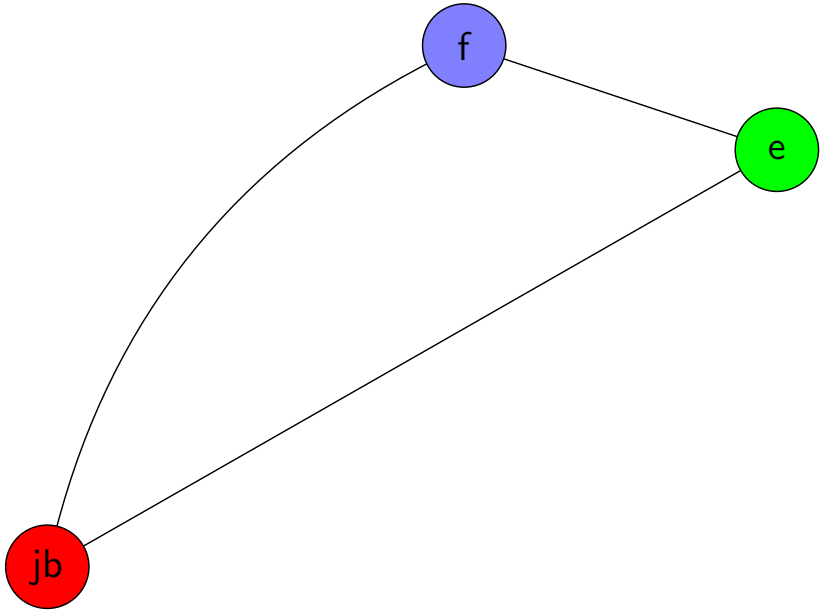
L'heuristique par l'exemple (K = 3)

L'heuristique par l'exemple (K = 3)

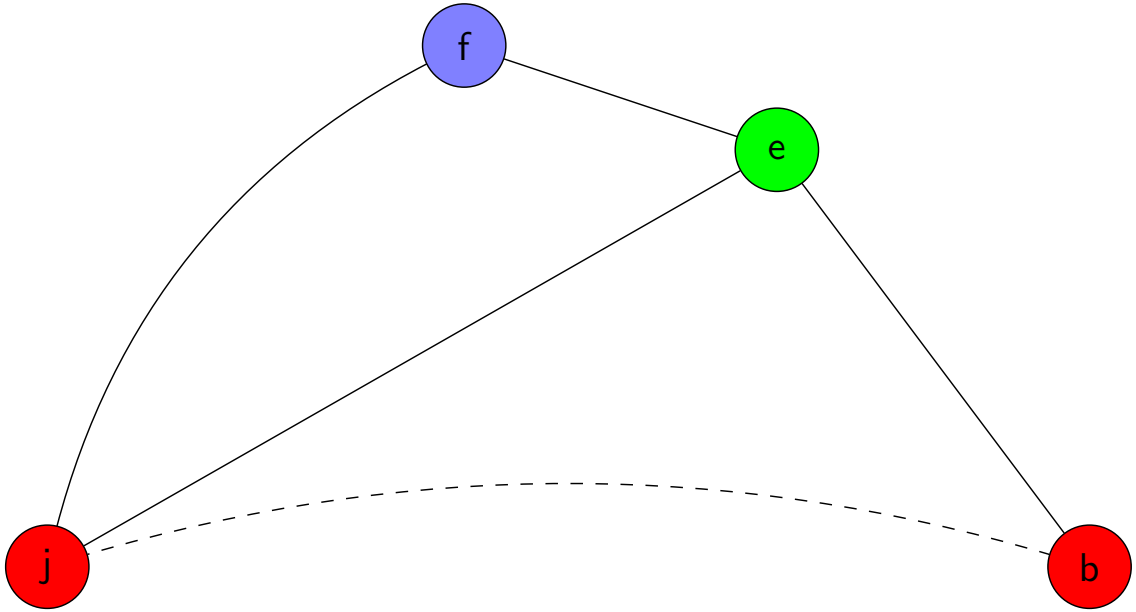
L'heuristique par l'exemple (K = 3)



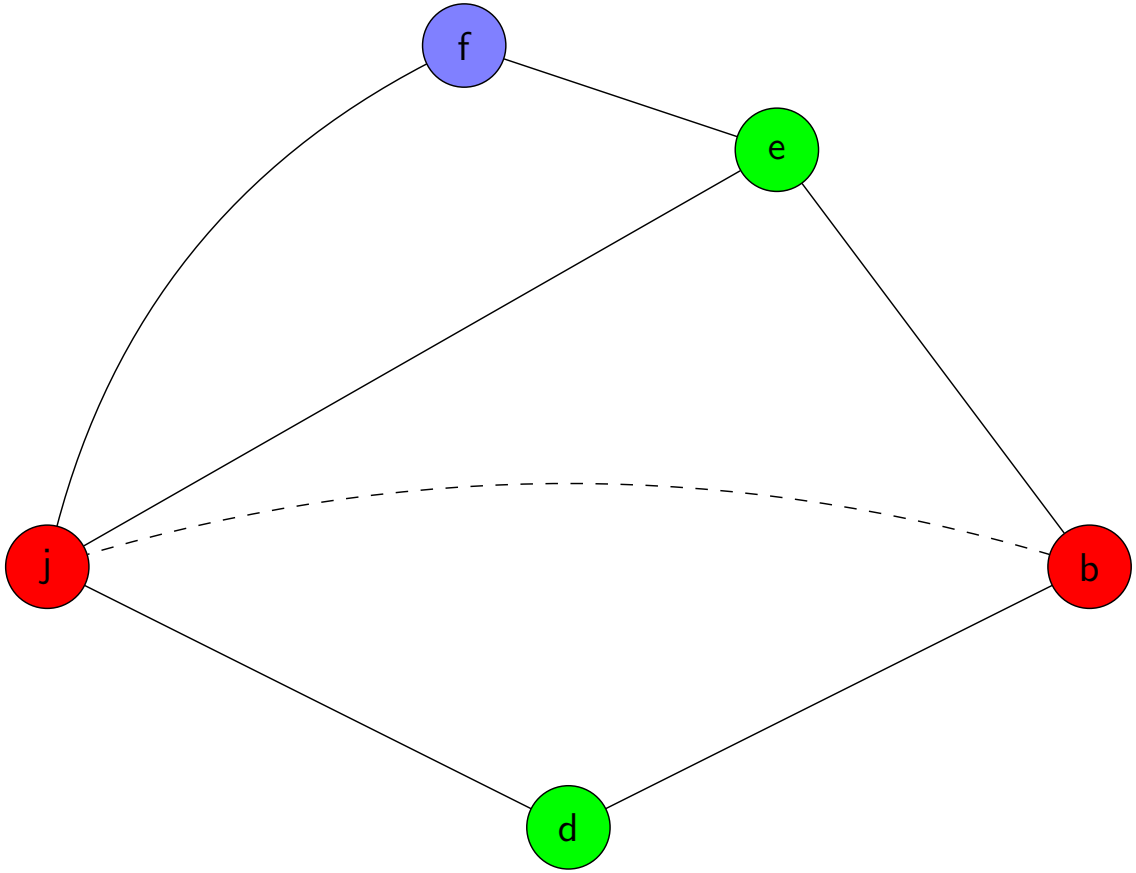
L'heuristique par l'exemple (K = 3)



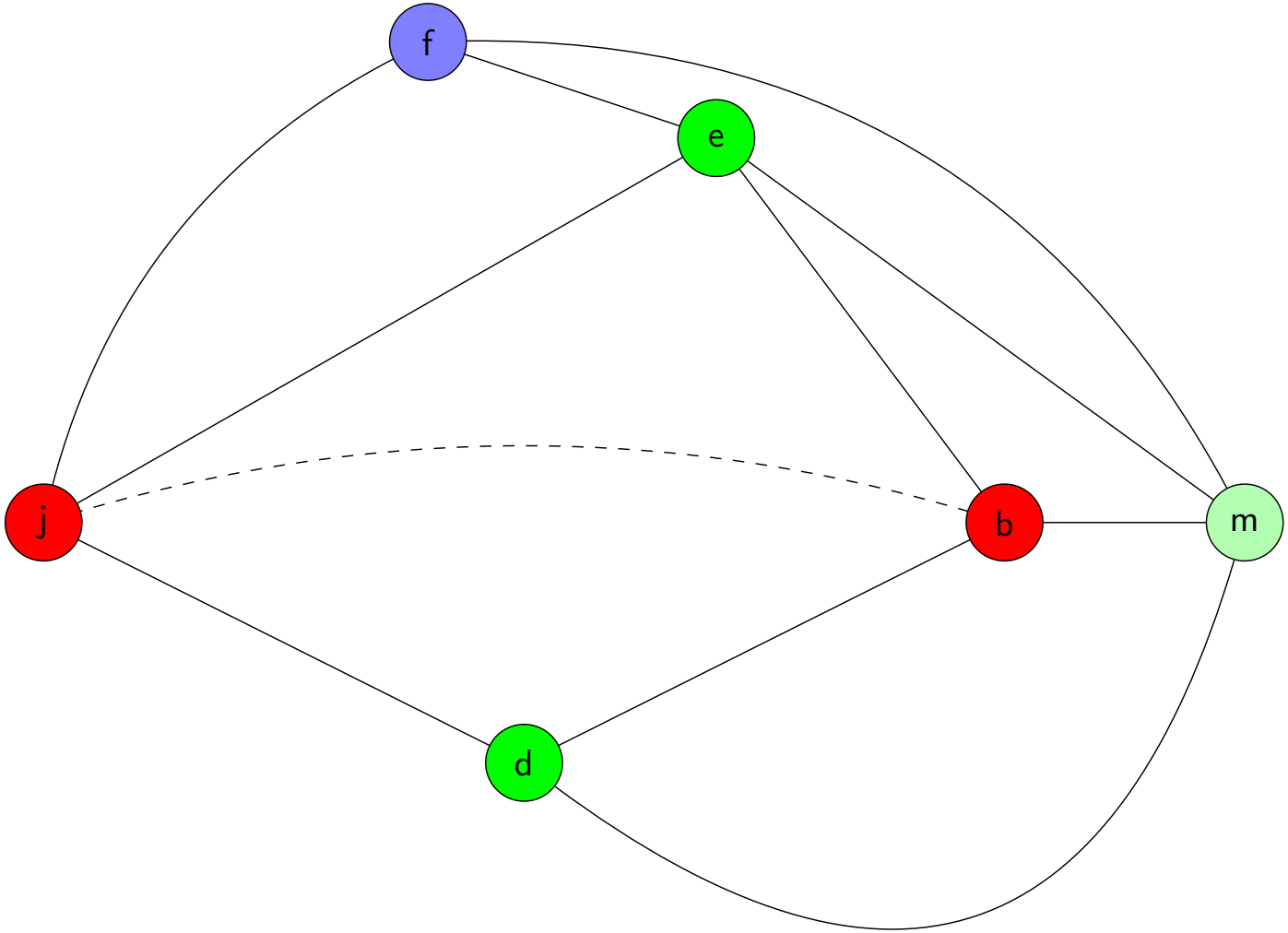
L'heuristique par l'exemple (K = 3)



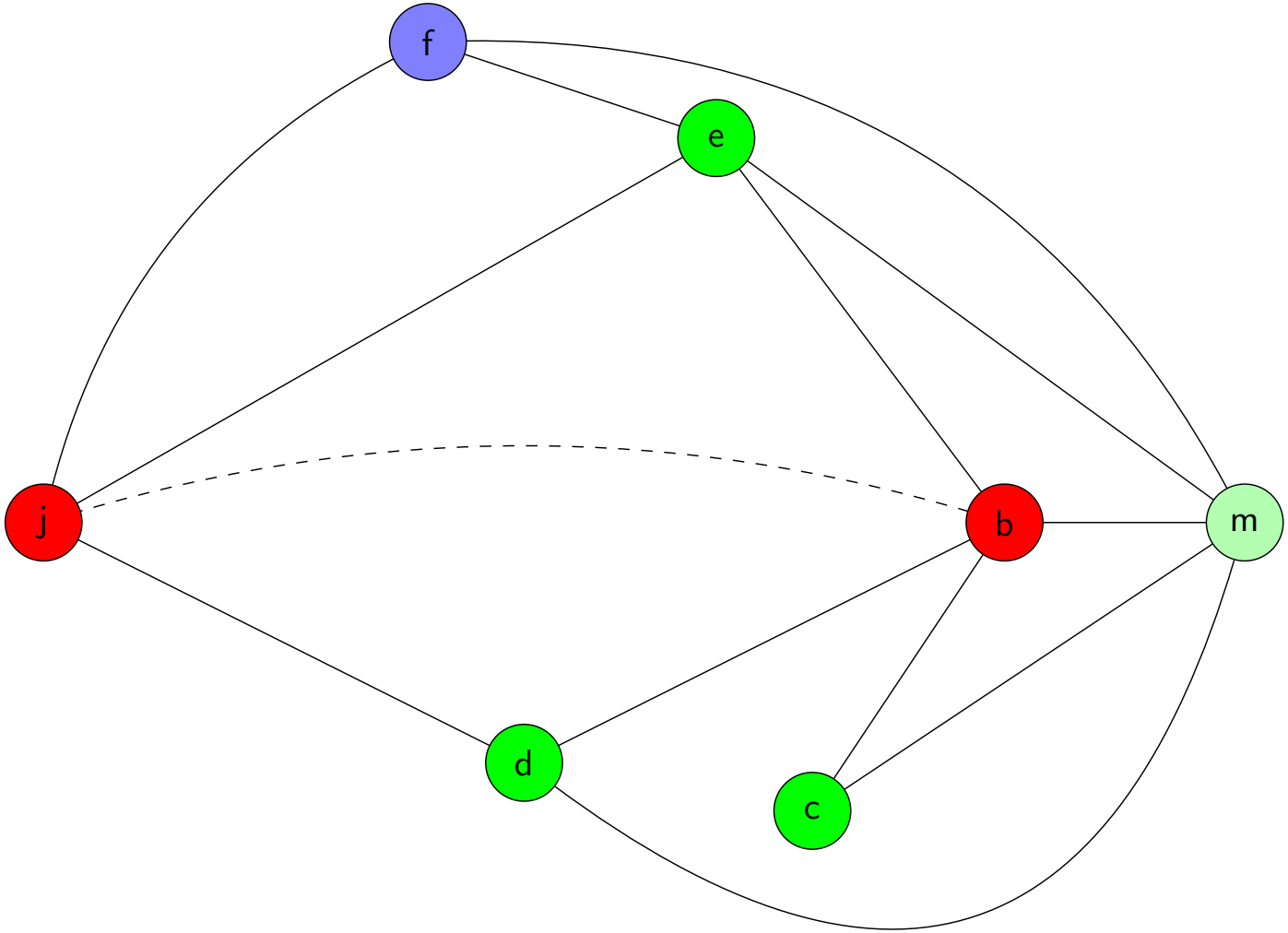
L'heuristique par l'exemple (K = 3)



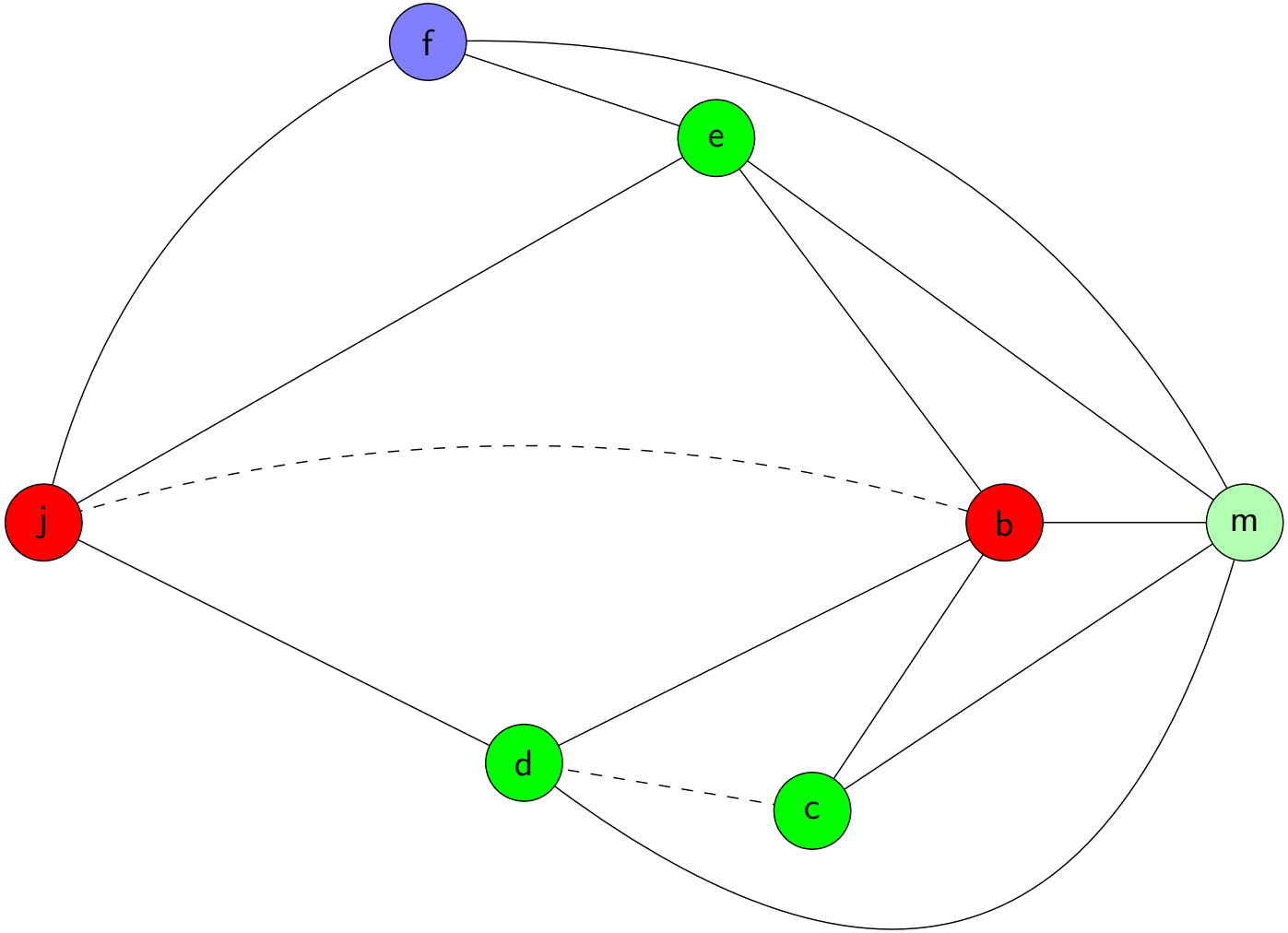
L'heuristique par l'exemple (K = 3)



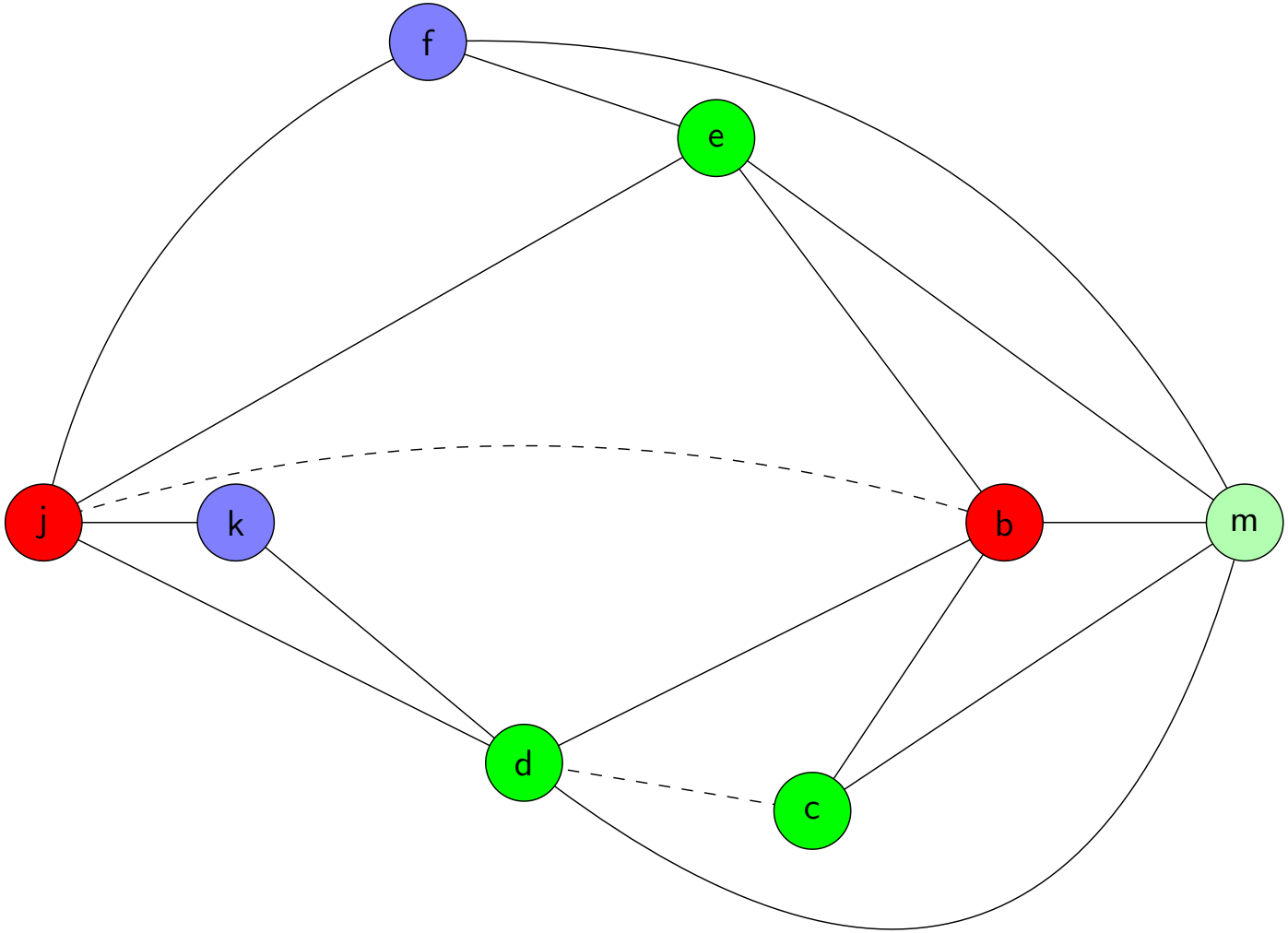
L'heuristique par l'exemple (K = 3)



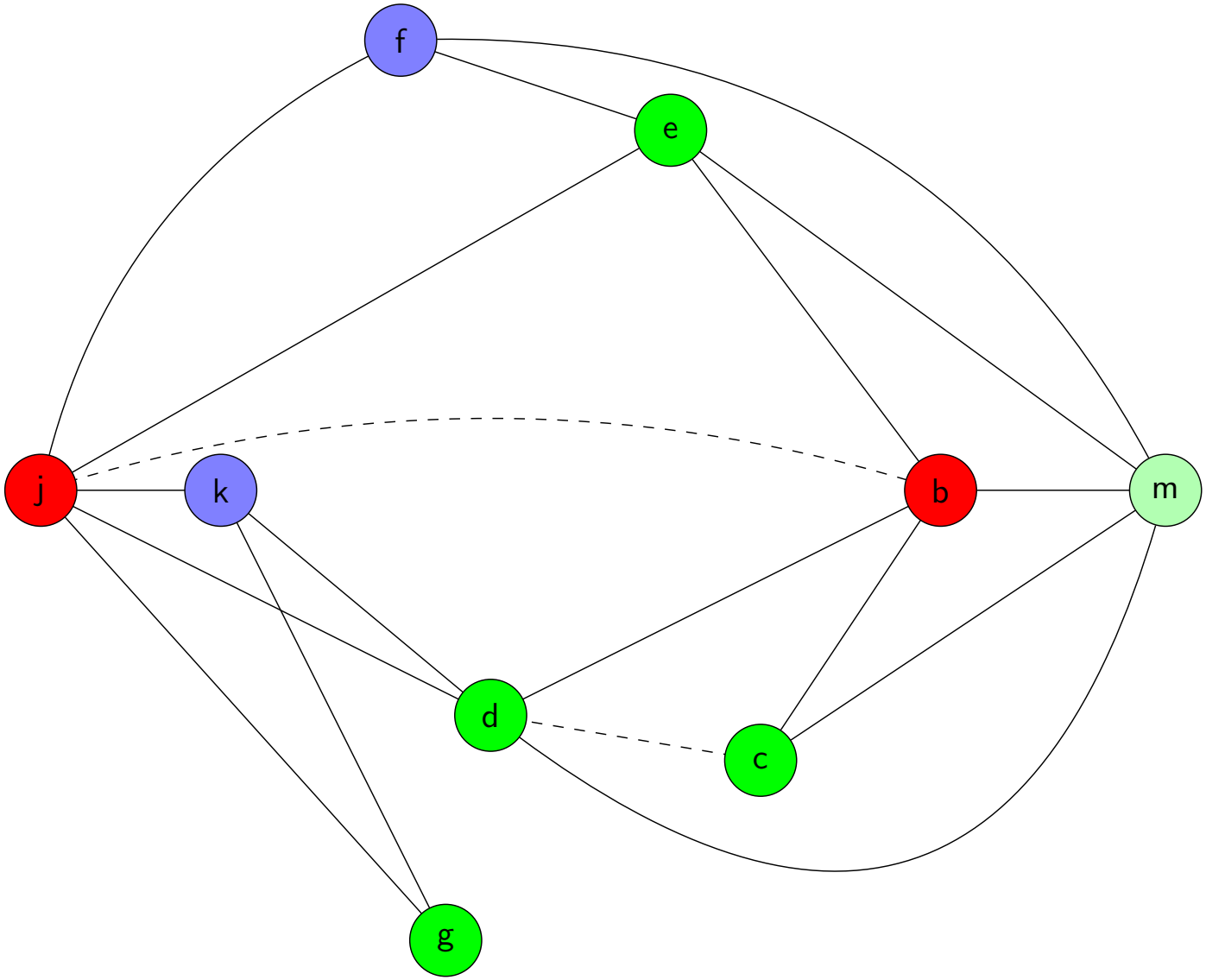
L'heuristique par l'exemple (K = 3)



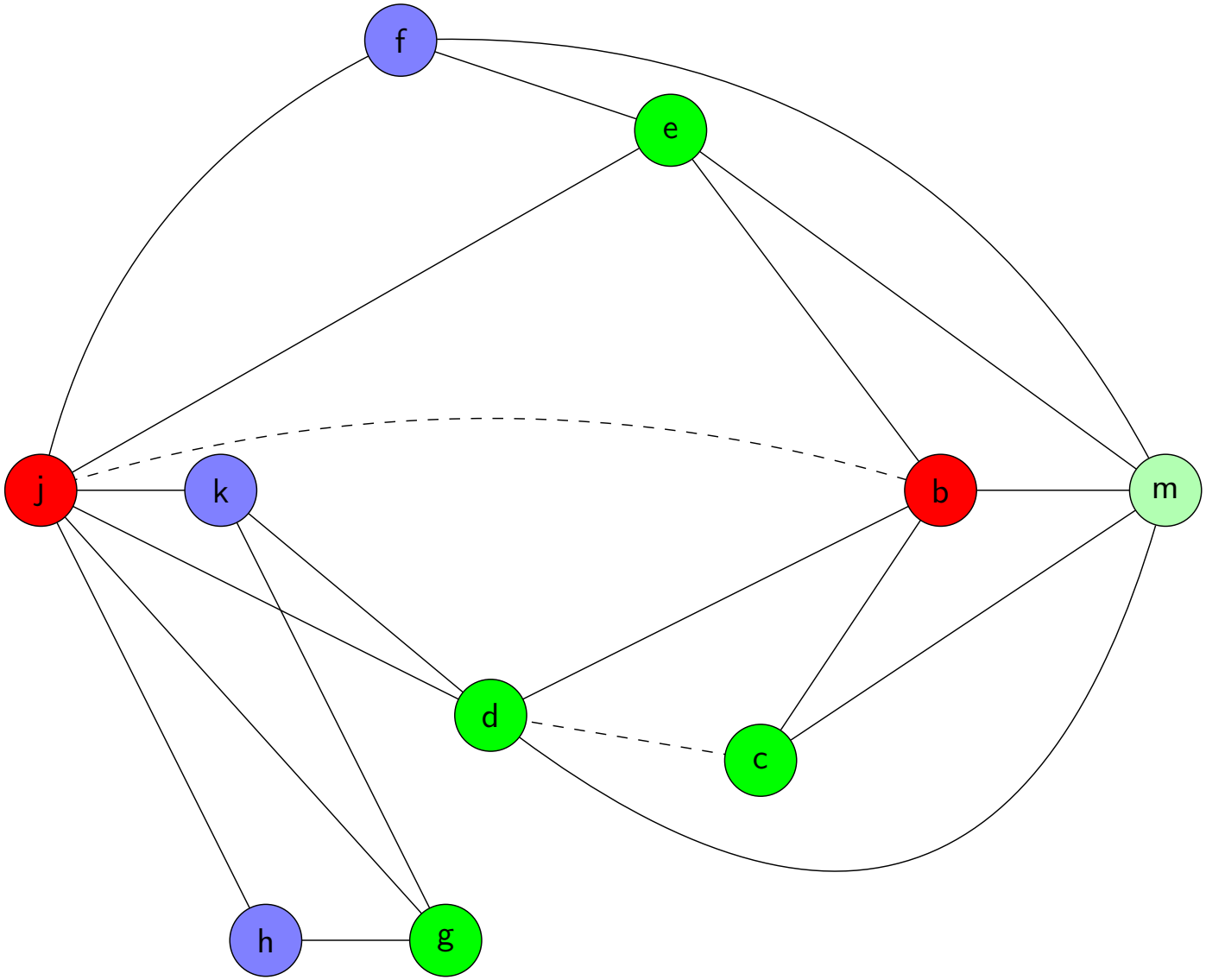
L'heuristique par l'exemple (K = 3)



L'heuristique par l'exemple (K = 3)



L'heuristique par l'exemple (K = 3)



La fonction IRC

```
1  : Algorithm IRC g : Coloring :=
2  : match simplify g with
3  : |⌊(r, g')⌋ → available_coloring g r (IRC g')
4  : |  ∅     → match coalesce g with
5  :    |⌊(e, g')⌋ → complete_coloring e (IRC g')
6  :    |  ∅  → match freeze g with
7  :       |⌊g'⌋ → IRC g'
8  :       | ∅  → match spill g with
9  :          |⌊(r, g')⌋ → available_coloring g r (IRC g')
10 :          |  ∅ → precoloring g
11 :        end
12 :      end
13 :   end
14 : end.
```

Les ensembles de travail

Efficacité de l'IRC liée aux ensembles de travail :

1. `simplifyWL` ensemble des sommets de bas degré, sans affinité, non précolorés
2. `freezeWL` ensemble des sommets de bas degré, avec affinité, non précolorés
3. `spillWL` ensemble des sommets de haut degré, non précolorés
4. `movesWL` ensemble des affinités

Définition d'un invariant : `WL_invariant`

Structure de données et fonctions auxiliaires

```
Record irc_graph := Make_IRC_Graph {
  gph : Graph.t;
  wl : WL;
  pal : ColorSet.t;
  k : nat;
  Hk : ColorSet.cardinal pal = k;
  Hwl : WL_invariant gph pal wl }.
```

Structure de données et fonctions auxiliaires

```
Record irc_graph := Make_IRC_Graph {
  gph : Graph.t;
  wl : WL;
  pal : ColorSet.t;
  k : nat;
  Hk : ColorSet.cardinal pal = k;
  Hwl : WL_invariant gph pal wl }.
```

Les fonctions auxilaires :

1. vérifient si l'opération est possible
2. modifient le graphe
3. ajustent les ensembles de travail
4. construisent une preuve de préservation de l'invariant

Formalisation des graphes d'interférence-affinité

Définition modulaire : interface + implantation

Formalisation des graphes d'interférence-affinité

Définition modulaire : interface + implantation

Interface

Graphes = triplets d'ensembles : sommets, interférences, affinités
$\Rightarrow$ expressivité

Propriétés axiomatiques, fonction de modification usuelles en allocation de registres, fonctions de calcul des voisinages, des degrés,...

Formalisation des graphes d'interférence-affinité

Définition modulaire : interface + implantation

Interface

Graphes = triplets d'ensembles : sommets, interférences, affinités
$\Rightarrow$ expressivité

Propriétés axiomatiques, fonction de modification usuelles en allocation de registres, fonctions de calcul des voisinages, des degrés,...

Implantation

Graphes = fonctions : sommets $\rightarrow$ ensembles d'adjacence $\Rightarrow$ efficacité du calcul des voisinages (complexité logarithmique)

Terminaison de notre implantation de l'IRC

Définition d'une mesure `irc_measure` = 2*n-card(simplifyWL)

Théorème : Terminaison

Theorem `terminate_IRC : ∀ g g' r e,`
`simplify g = ⌊(r, g')⌋ ∨`
`coalesce g = ⌊(e, g')⌋ ∨`
`freeze g = ⌊g'⌋ ∨`
`spill g = ⌊(r, g')⌋ ⇒`
`irc_measure g' < irc_measure g.`

Correction de notre implantation de l'IRC

Théorème : Correction

```
Theorem proper_coloring_IRC : ∀ g palette,
proper_coloring (precoloring g) g palette ⇒
proper_coloring (IRC g palette) g palette.
```

Induction fonctionnelle

⇒ 5 sous-buts : 4 appels récursifs et 1 cas de base

- Simplification et vidage : correction de la fonction `available_coloring`
- Fusion : correction du critère de fusion
- Gel : conservation de la coloration par suppression des affinités
- Cas de base : correction de la précoloration

Statistiques de développement

Implantation modulaire de l'IRC

Propriétés sur les graphes d'interférence-affinité (550 lignes)

Mise à jour des ensembles de travail (3300 lignes)

Correction (650 lignes)

Terminaison (300 lignes)

Ratio preuve/extraction = 8

Implantation modulaire des graphes d'interférence-affinité

Spécification des graphes (100 lignes)

Implantation des graphes (4000 lignes)

Bilan et perspectives

Bilan

Implantation fonctionnelle de référence de l'IRC

Mise à jour incrémentale et efficace des ensembles de travail

Preuves de terminaison et de correction

Spécification et implantation des graphes d'interférence-affinité

Intégration à un prototype de CompCert

Généralisation de la démarche à une classe d'algorithmes

Perspectives

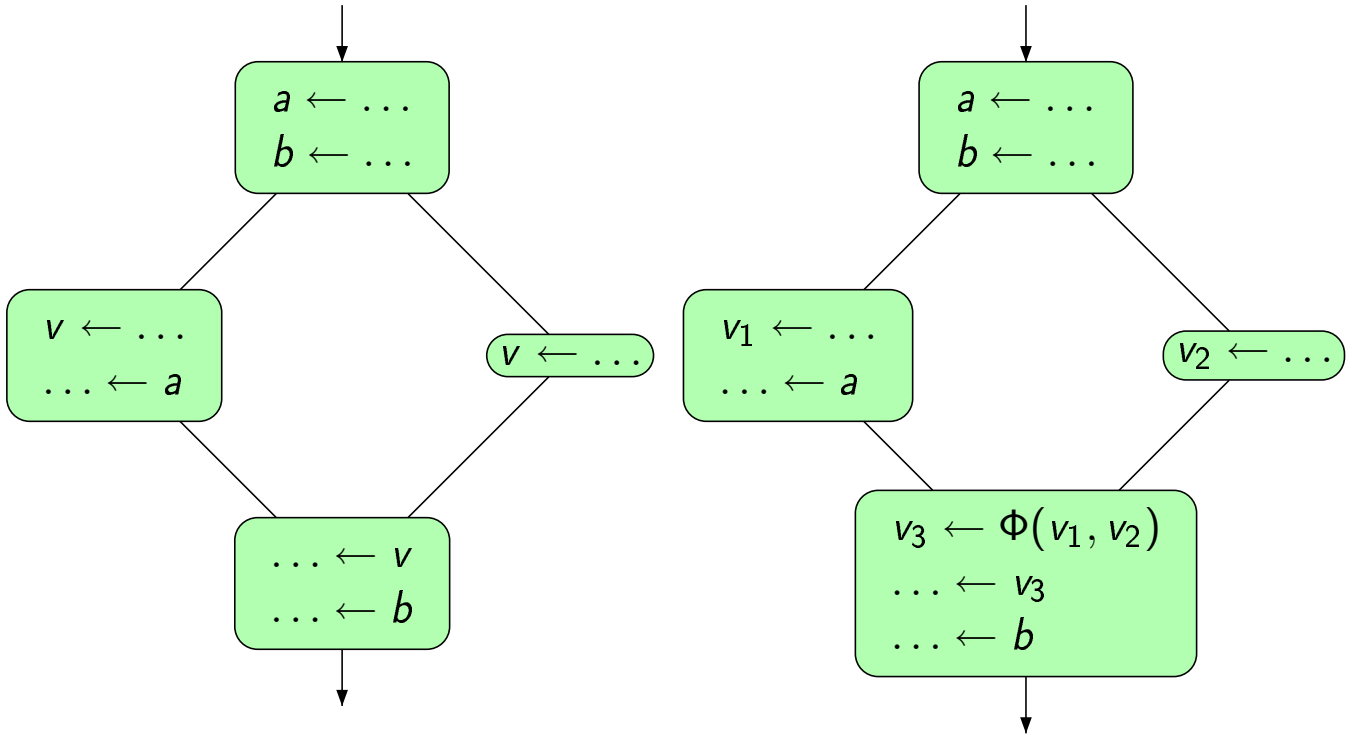
Optimisation de la mise à jour des candidats à la fusion

Fusion optimale dans les graphes SSA par PL

1. Graphes d'interférence-affinité SSA
2. Les affinités source de complexité
3. Fusion dans les $(K-1)$-arbres partiels
4. Fusion dans les graphes SSA

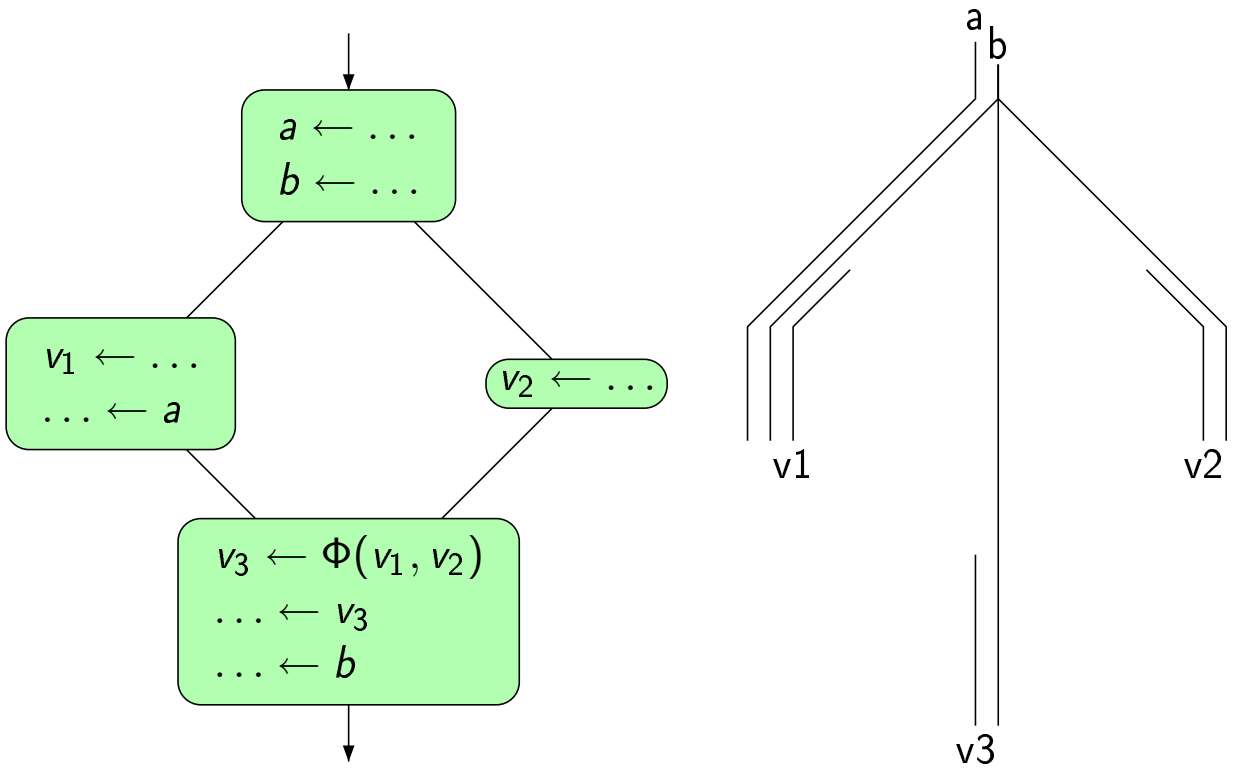
Définition : Forme SSA

Un programme est sous forme SSA si chaque variable est définie textuellement une seule fois.



Définition : Graphes SSA

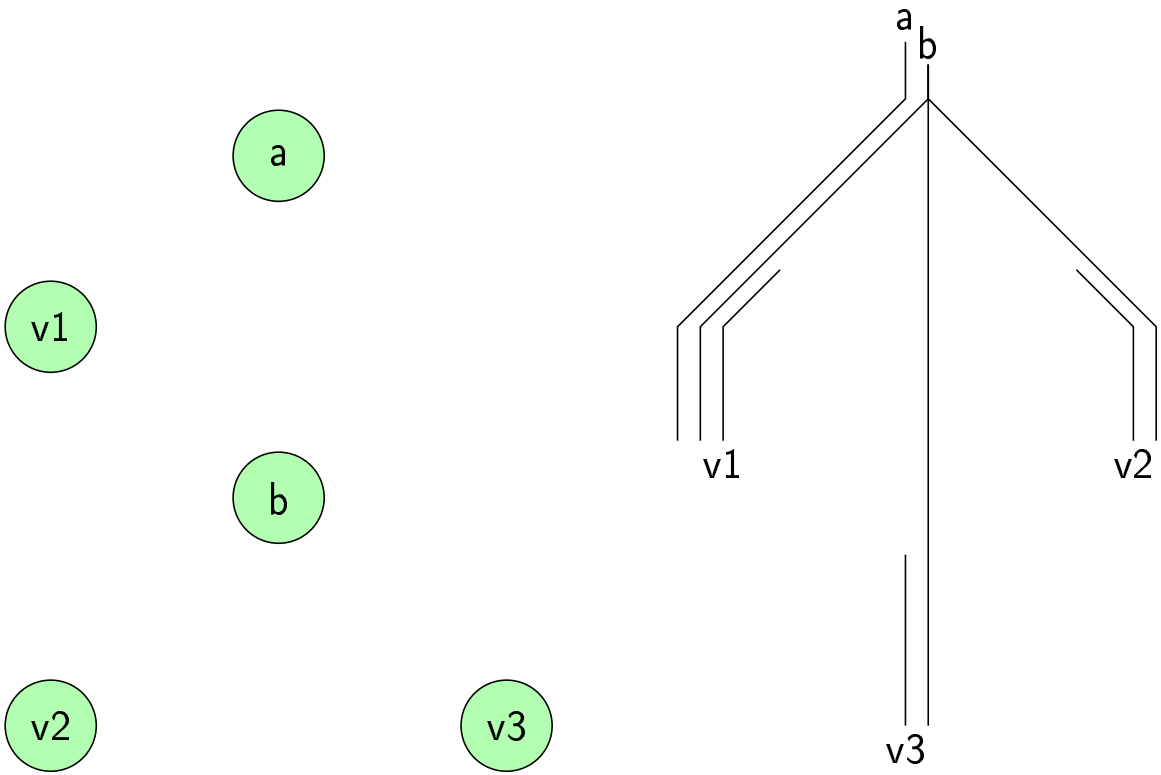
Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].



Graphes SSA et arbre de dominance

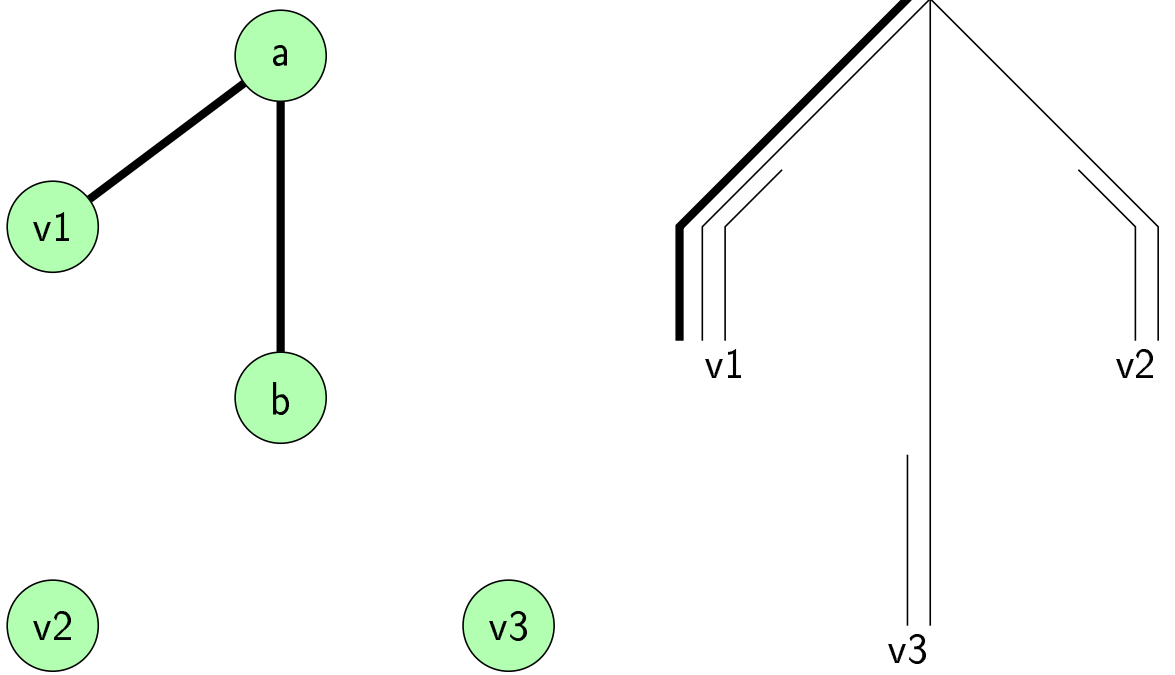
Définition : Graphes SSA

Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].



Définition : Graphes SSA

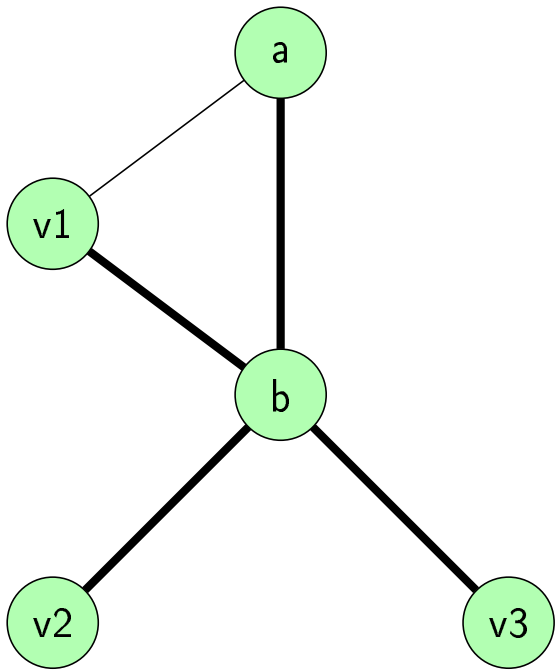
Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].

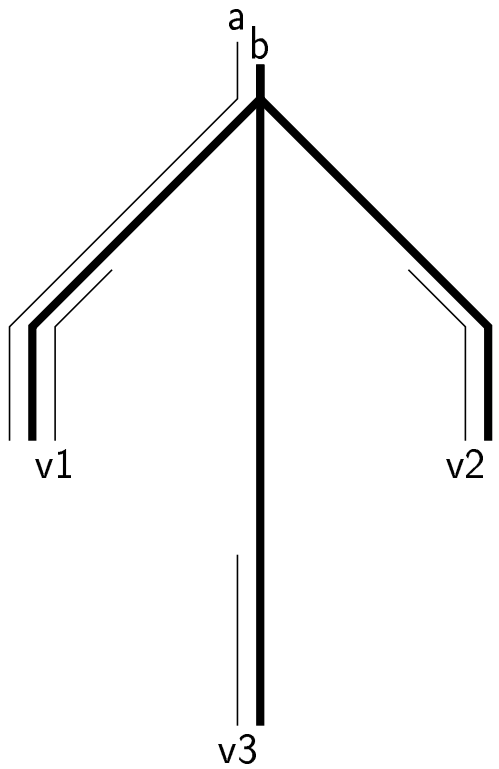


Graphes SSA et arbre de dominance

Définition : Graphes SSA

Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].

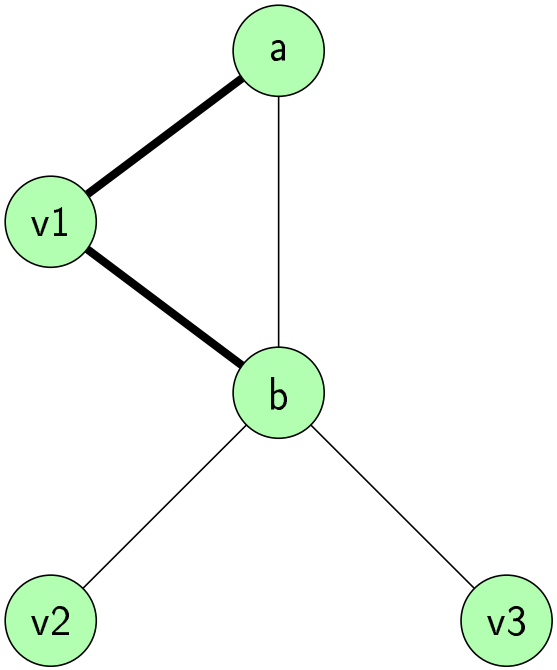


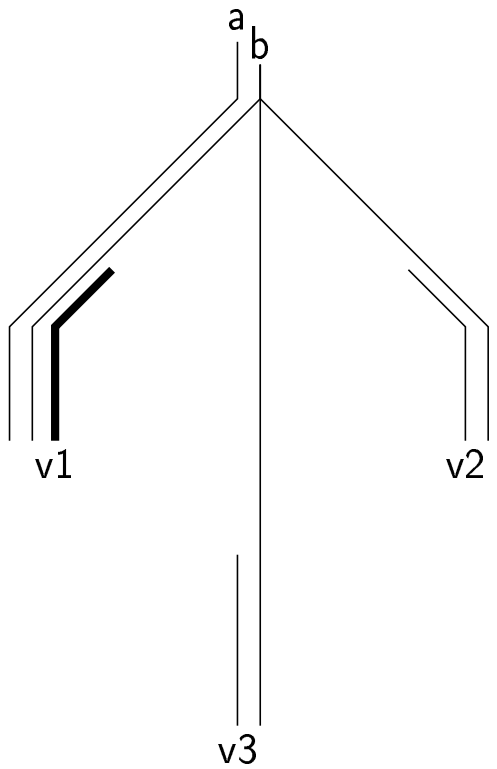


Graphes SSA et arbre de dominance

Définition : Graphes SSA

Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].

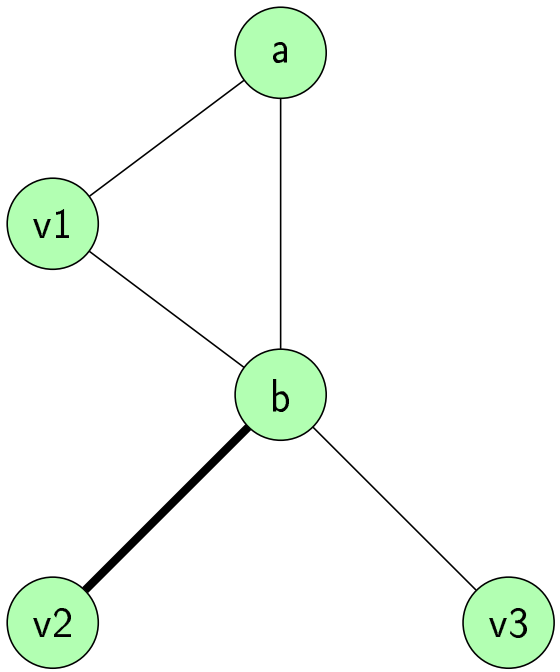


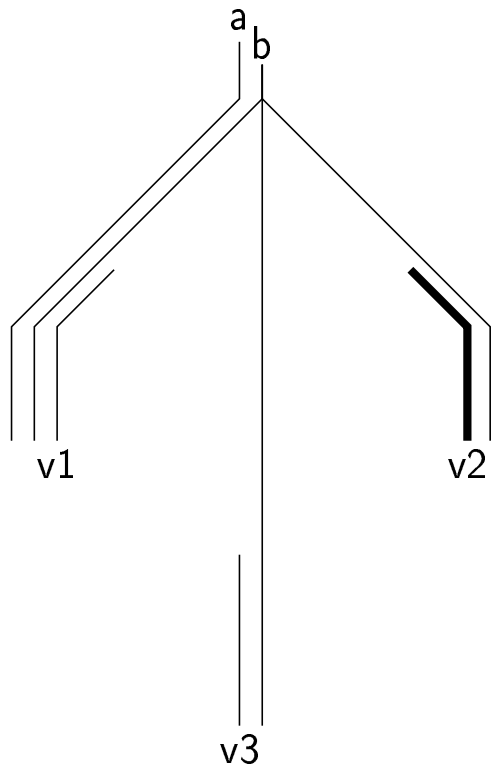


Graphes SSA et arbre de dominance

Définition : Graphes SSA

Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].

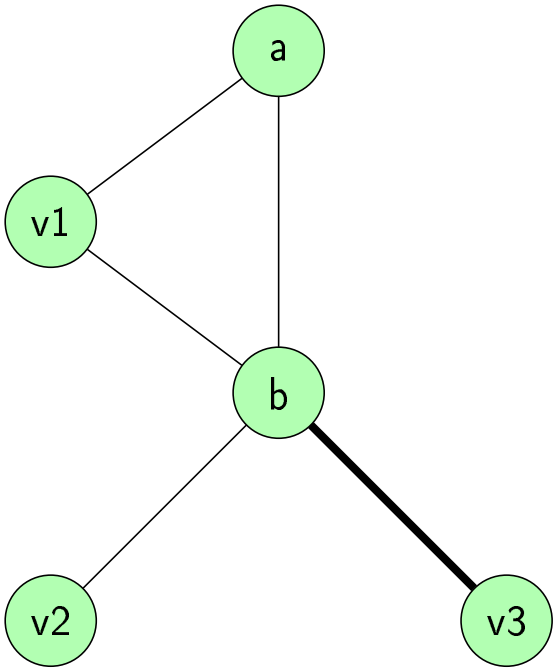


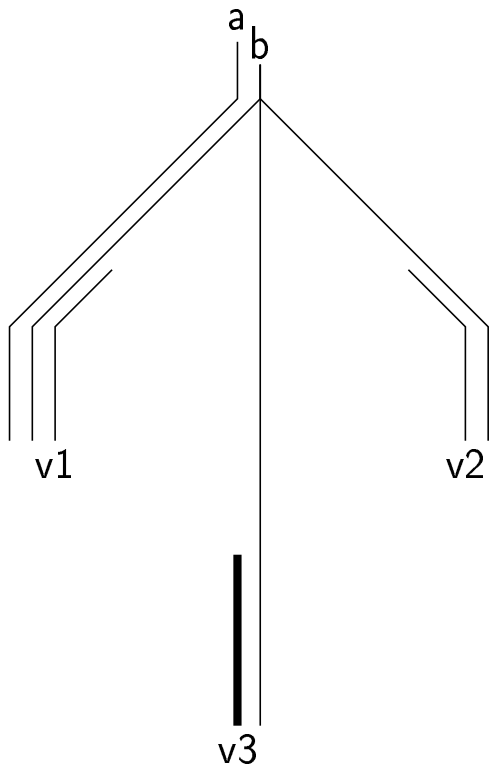


Graphes SSA et arbre de dominance

Définition : Graphes SSA

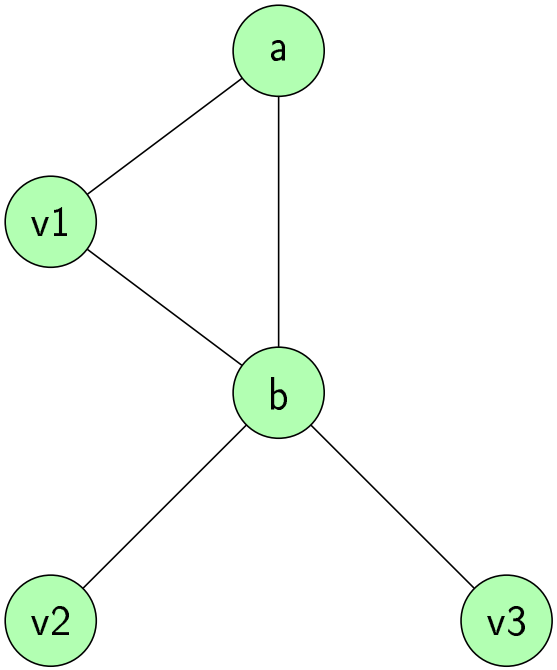
Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].

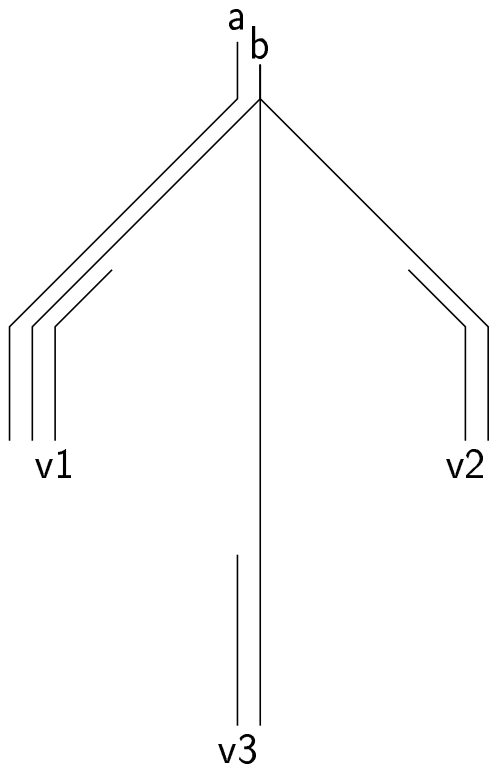




Définition : Graphes SSA

Les durées de vie d'un programme sous forme SSA forment des sous-arbres d'un arbre, appelé arbre de dominance du programme [BDGR05, Hac05].





Les graphes triangulés

Définition : Caractérisation des graphes triangulés

Un graphe est triangulé si et seulement s'il est le graphe d'intersection d'une famille de sous-arbres d'un arbre. On appelle décomposition arborescente la représentation d'un graphe sous cette forme.

Conséquences de SSA

Les interférences d'un programme SSA forment un graphe triangulé [BDGR05, Hac05]

Nombre de couleurs nécessaire connu en temps polynomial

Nouvelles heuristiques pour le vidage plus précises

Séparation en deux phases : vidage puis fusion/affectation

Affinités issues des Φ-fonctions $\Rightarrow$ problème de fusion

Notre démarche

Approche

Montrer que les affinités sont une source de complexité majeure sous forme SSA

Utiliser la configuration globale des affinités et non un critère local pour la fusion : intérêt de la largeur d'arbre

Séparer la fusion de l'affectation aux registres : allocation de registres en trois phases

Les affinités : source de complexité majeure

Théorème : Complexité de la fusion/affectation pour K=3 dans les graphes 2-colorables

Le problème de fusion/affectation est NP-difficile pour K=3 dans les graphes 2-colorables. (réduction du problème de K-coloration de graphe)

Les affinités : source de complexité majeure

Théorème : Complexité de la fusion/affectation pour K=3 dans les graphes 2-colorables

Le problème de fusion/affectation est NP-difficile pour K=3 dans les graphes 2-colorables. (réduction du problème de K-coloration de graphe)

Théorème : Complexité de la fusion/affectation pour K=2 dans les graphes 2-colorables

Le problème de fusion/affectation est NP-difficile pour K=2 dans les graphes 2-colorables. (réduction du problème 3-SAT)

Les affinités : source de complexité majeure

Théorème : Complexité de la fusion/affectation pour K=3 dans les graphes 2-colorables

Le problème de fusion/affectation est NP-difficile pour K=3 dans les graphes 2-colorables. (réduction du problème de K-coloration de graphe)

Théorème : Complexité de la fusion/affectation pour K=2 dans les graphes 2-colorables

Le problème de fusion/affectation est NP-difficile pour K=2 dans les graphes 2-colorables. (réduction du problème 3-SAT)

Interprétation

Choisir les affinités à satisfaire est difficile même si colorer le graphe et déterminer s'il reste K-colorable est trivial !

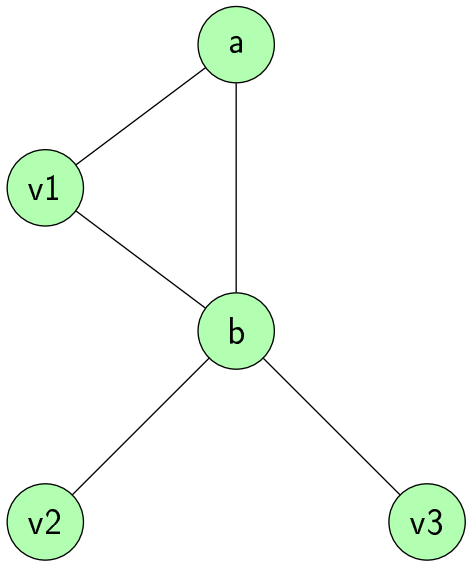
Largeur d'arbre

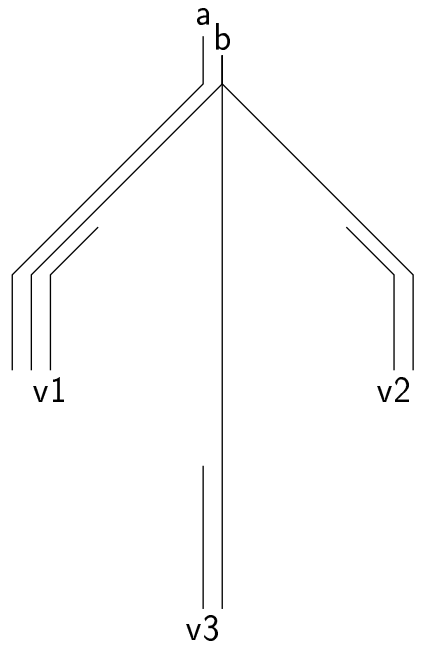
Définition : Triangulation d'un graphe

Une triangulation de G est un graphe triangulé contenant G.

Définition : Largeur d'arbre

La largeur d'une décomposition arborescente est la taille de l'intersection max. des sous-arbres, moins 1. La largeur d'arbre d'un graphe est le min. des largeurs d'arbre de ses triangulations.

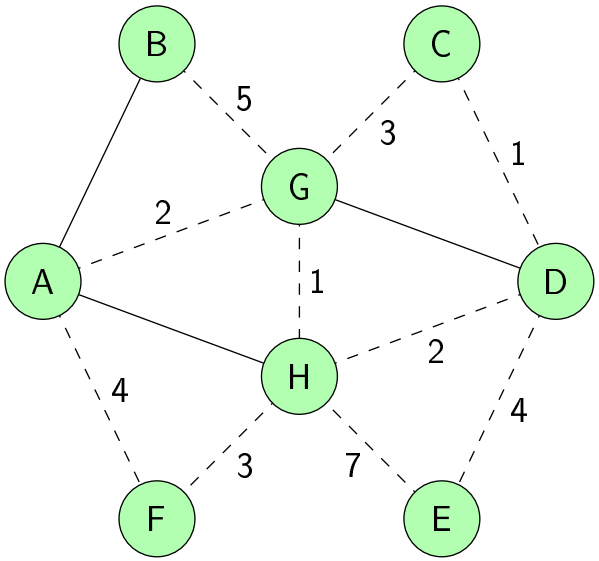




Fusion dans les $(K-1)$-arbres partiels

Théorème : Équivalence des fusions conservative et agressive

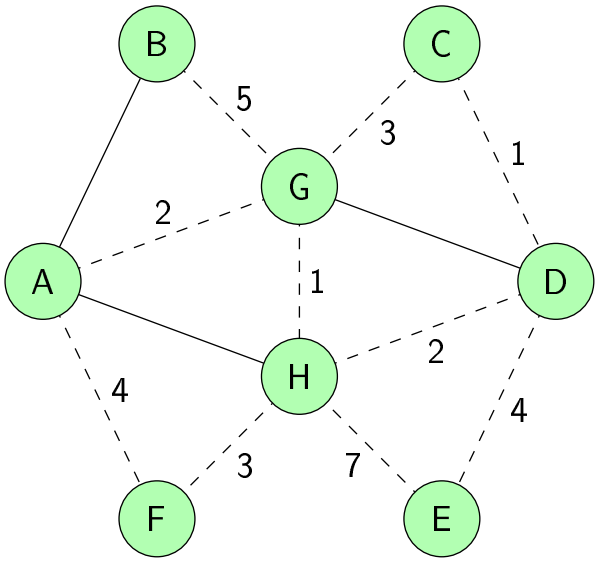
Si la largeur d'arbre d'un graphe d'interférence-affinité est inférieure à $(K-1)$, alors la fusion de n'importe quelle affinité est conservative et fait diminuer la largeur d'arbre du graphe.

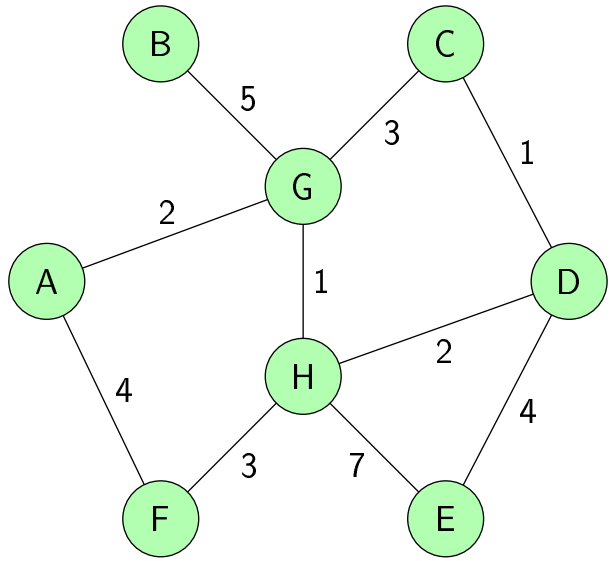


Fusion dans les $(K-1)$-arbres partiels

Théorème : Équivalence des fusions conservative et agressive

Si la largeur d'arbre d'un graphe d'interférence-affinité est inférieure à $(K-1)$, alors la fusion de n'importe quelle affinité est conservative et fait diminuer la largeur d'arbre du graphe.



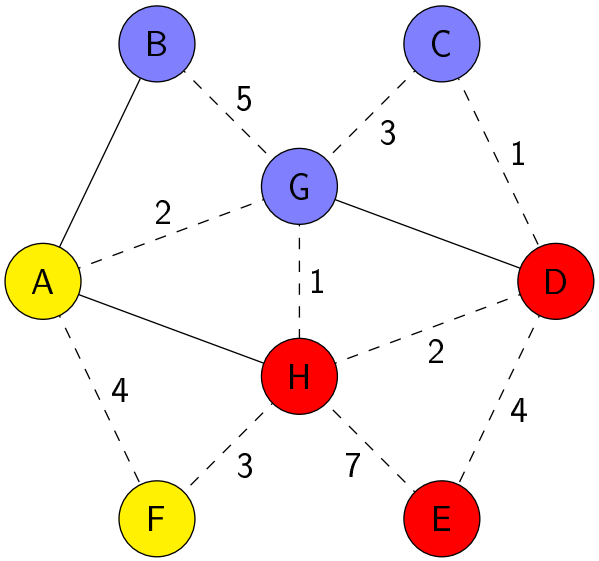


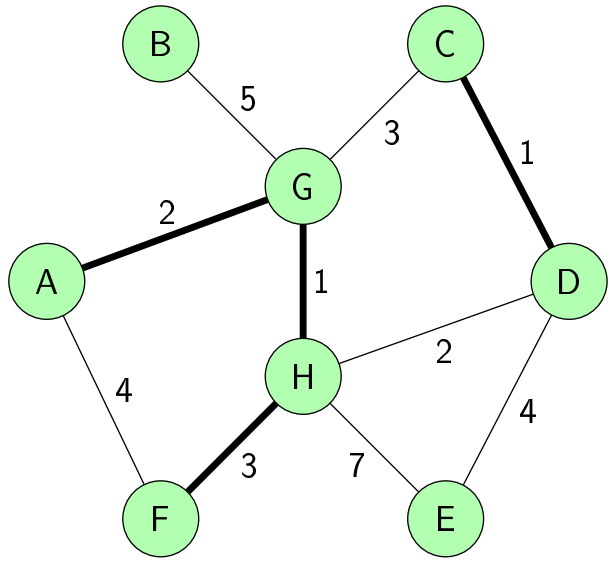
T = {(A,B),(A,H),(G,D)}

Fusion dans les $(K-1)$-arbres partiels

Théorème : Équivalence des fusions conservative et agressive

Si la largeur d'arbre d'un graphe d'interférence-affinité est inférieure à $(K-1)$, alors la fusion de n'importe quelle affinité est conservative et fait diminuer la largeur d'arbre du graphe.





T = {(A,B),(A,H),(G,D)}

Fusion/multicoupe dans les $(K-1)$-arbres partiels

Théorème : Fusion dans les $(K-1)$-arbres partiels

Le problème de fusion dans les graphes de largeur d'arbre inférieure à $(K-1)$ est un problème de multicoupe minimum.

Fusion/multicoupe dans les $(K-1)$-arbres partiels

Théorème : Fusion dans les $(K-1)$-arbres partiels

Le problème de fusion dans les graphes de largeur d'arbre inférieure à $(K-1)$ est un problème de multicoupe minimum.

$$(PMCM)\begin{cases} Min & \sum_{e\in A} w_e y_e \\ s.c. & \\ \forall (i,j)\in T, \forall p\in P(i,j), & \sum_{e\in p} y_e \geq 1 \\ \forall e, & y_e \in \{0,1\} \end{cases}$$

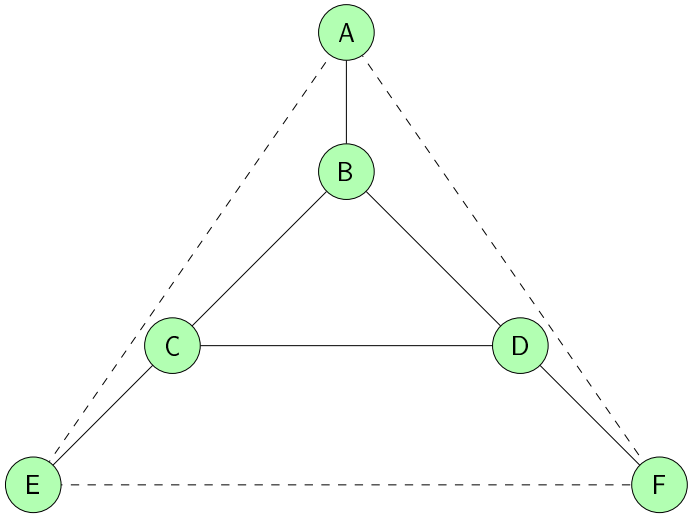
Approche

1. Trianguler le graphe d'interférence-affinité avec des affinités de poids nul
2. Déterminer un ensemble d'affinités dont la fusion fait décroître la largeur d'arbre en dessous de $K-1$ par PL

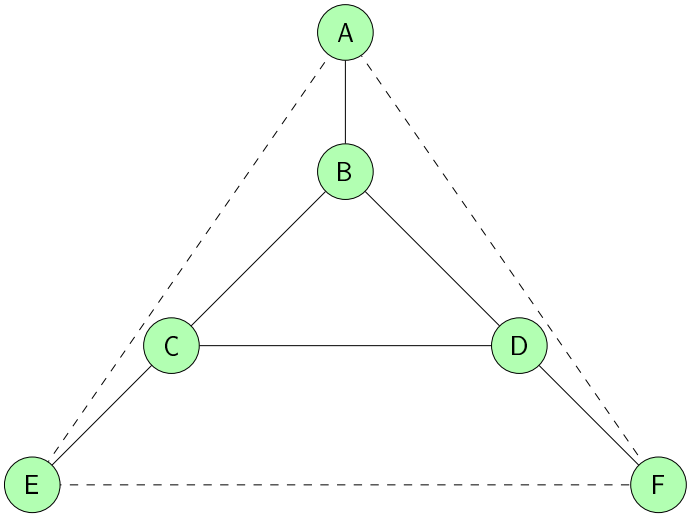
$$(P)\begin{cases} \textit{Min} & \sum_{e\in A} w_e y_e & & \\ \textit{s.c.} & & & \\ \forall (i,j)\in I, \forall p\in P(i,j), & \sum_{e\in p} y_e & \geq & 1 \\ \forall a, \forall i\in C_a - I_a, & \sum_{j\in I_a - N(i)} y_e & = & |I_a - N(i)| - 1 \\ \forall e\in A, & y_e\in\{0,1\} & & \end{cases}$$

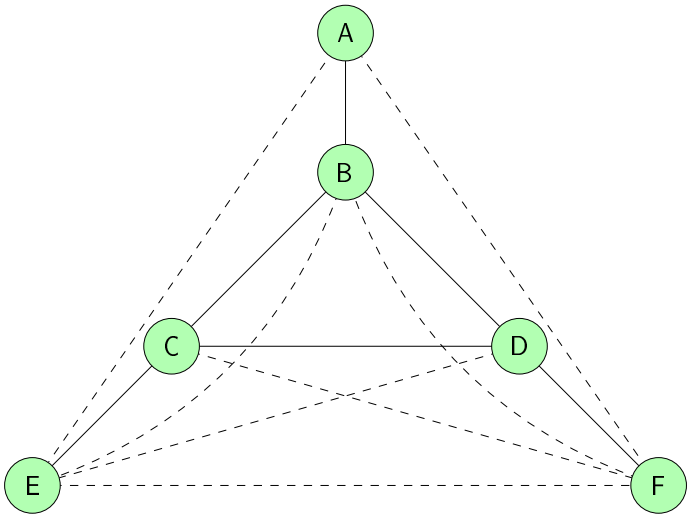
3. Colorer le graphe d'interférence-affinité (triangulé, sans affinité) avec un algorithme polynomial

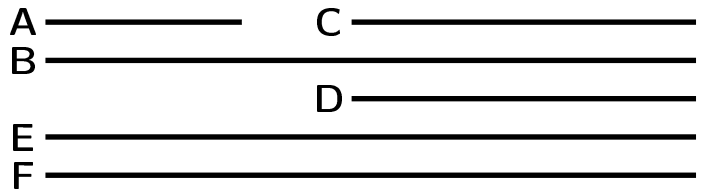
Exemple (étape 1)



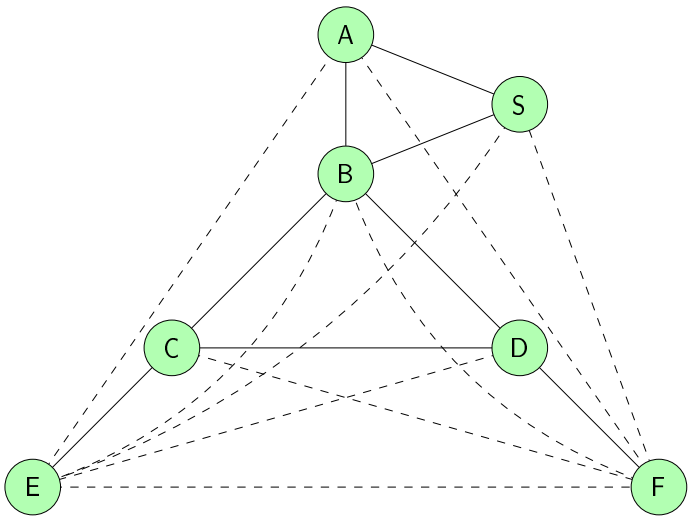
Exemple (étape 1)





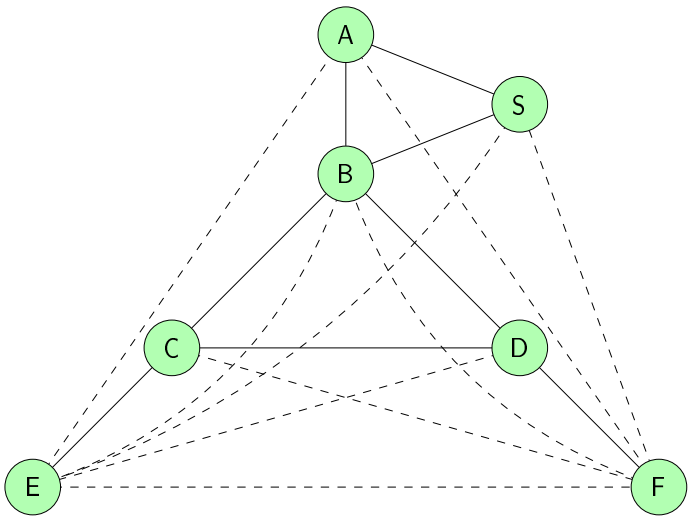


Exemple (étape 2)



$$
(P)\ \begin{cases}
\textit{Min} & y_{AE} + y_{AF} + y_{EF} & & \\
s.c. & & & \\
(* \textit{ chemins } AB\ *) & & & \\
y_{AE} + y_{BE} & & \geq & 1 \\
y_{AE} + y_{EF} + y_{BF} & & \geq & 1 \\
y_{AF} + y_{BF} & & \geq & 1 \\
y_{AF} + y_{EF} + y_{BE} & & \geq & 1 \\
(* \textit{ chemins } BC\ *) & & & \\
y_{BF} + y_{CF} & & \geq & 1 \\
y_{BE} + y_{EF} + y_{CF} & & \geq & 1 \\
y_{BE} + y_{AE} + y_{AF} + y_{CF} & & \geq & 1 \\
(* \textit{ chemins } BD\ *) & & & \\
y_{BE} + y_{DE} & & \geq & 1 \\
y_{BF} + y_{EF} + y_{DE} & & \geq & 1 \\
y_{BF} + y_{AF} + y_{AE} + y_{DE} & & \geq & 1 \\
(* \textit{ chemins } CD\ *) & & & \\
y_{CF} + y_{EF} + y_{DE} & & \geq & 1 \\
y_{CF} + y_{AF} + y_{AE} + y_{DE} & & \geq & 1 \\
(* \textit{ chemins } CE\ *) & & & \\
y_{CF} + y_{EF} & & \geq & 1 \\
(* \textit{ chemins } DF\ *) & & & \\
y_{DE} + y_{EF} & & \geq & 1 \\
(* \textit{ clique } ABEFS\ *) & & & \\
y_{AE} + y_{BE} + y_{ES} & & = & 2 \\
y_{AF} + y_{BF} + y_{ES} & & = & 2 \\
(* \textit{ clique } BCDEF\ *) & & & \\
y_{BE} + y_{DE} & & = & 1 \\
y_{BF} + y_{CF} & & = & 1 \\
\forall e \in A, & y_e \in \{0,1\} & &
\end{cases}
$$

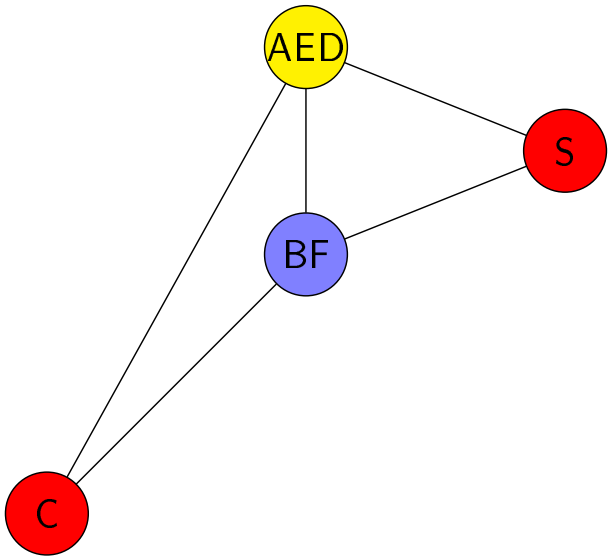
Exemple (étape 2)



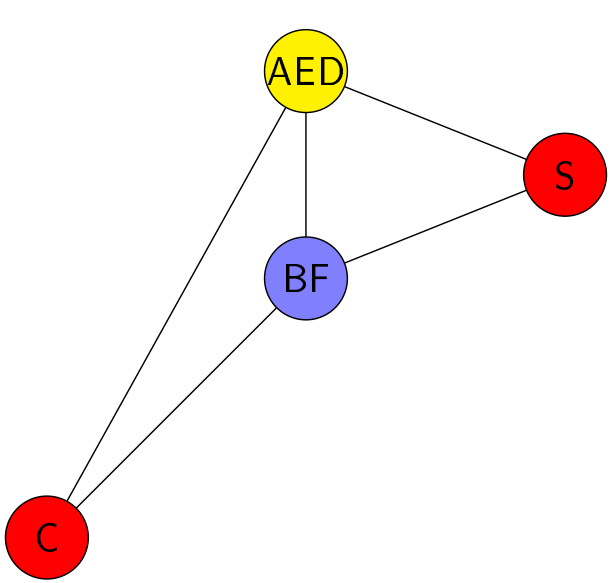
9 variables, 24 contraintes au lieu de 21 variables, 33 contraintes et 20 coupes [GH07]

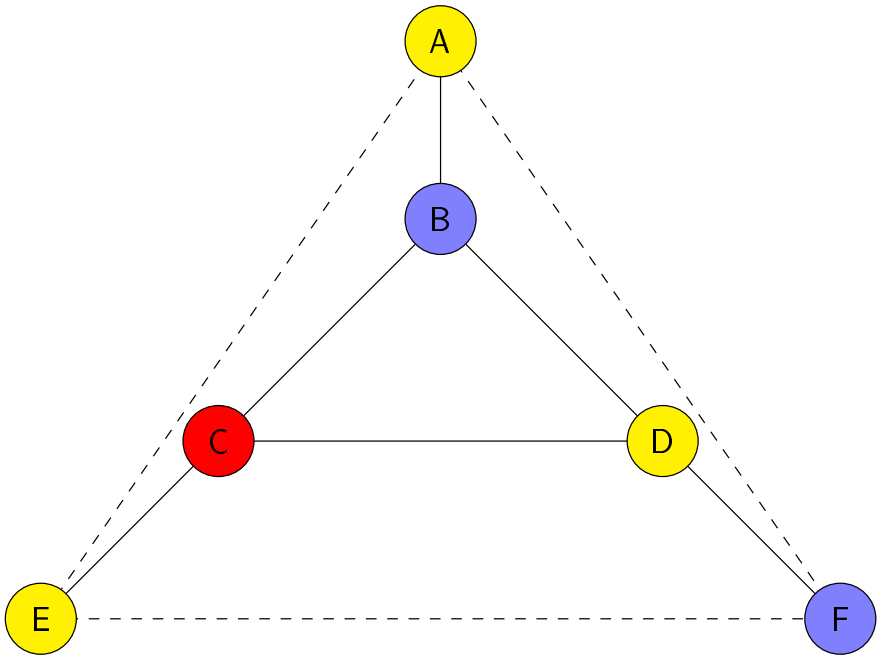
$$(P)\quad \begin{array}{lll}
\textit{Min} & y_{AE}+y_{AF}+y_{EF} & \\
s.c. & & \\
(*\ \textit{chemins } AB\ *) & & \\
y_{AE}+y_{BE} & \geq & 1 \\
y_{AE}+y_{EF}+y_{BF} & \geq & 1 \\
y_{AF}+y_{BF} & \geq & 1 \\
y_{AF}+y_{EF}+y_{BE} & \geq & 1 \\
(*\ \textit{chemins } BC\ *) & & \\
y_{BF}+y_{CF} & \geq & 1 \\
y_{BE}+y_{EF}+y_{CF} & \geq & 1 \\
y_{BE}+y_{AE}+y_{AF}+y_{CF} & \geq & 1 \\
(*\ \textit{chemins } BD\ *) & & \\
y_{BE}+y_{DE} & \geq & 1 \\
y_{BF}+y_{EF}+y_{DE} & \geq & 1 \\
y_{BF}+y_{AF}+y_{AE}+y_{DE} & \geq & 1 \\
(*\ \textit{chemins } CD\ *) & & \\
y_{CF}+y_{EF}+y_{DE} & \geq & 1 \\
y_{CF}+y_{AF}+y_{AE}+y_{DE} & \geq & 1 \\
(*\ \textit{chemins } CE\ *) & & \\
y_{CF}+y_{EF} & \geq & 1 \\
(*\ \textit{chemins } DF\ *) & & \\
y_{DE}+y_{EF} & \geq & 1 \\
(*\ \textit{clique } ABEFS\ *) & & \\
y_{AE}+y_{BE}+y_{ES} & = & 2 \\
y_{AF}+y_{BF}+y_{ES} & = & 2 \\
(*\ \textit{clique } BCDEF\ *) & & \\
y_{BE}+y_{DE} & = & 1 \\
y_{BF}+y_{CF} & = & 1 \\
\forall e \in A, & y_e \in \{0,1\} &
\end{array}$$

Exemple (étape 3)



Exemple (étape 3)





Bilan et perspectives

Bilan

Approche plus globale de la gestion des affinités

Séparation des phases de fusion et d'affectation des registres

Équivalence fusion/multicoupe dans les $(K-1)$-arbres partiels

Approche optimale de la fusion dans les graphes SSA par PL

Affectation des registres issue du processus de fusion polynomiale

Perspectives

Implantation C de l'algorithme

Validation de la PL

Fusion optimale dans les graphes éclatés par recomposition des durées de vie

1. Découpage des durées de vie et graphes éclatés
2. Recomposition des durées de vie
3. Décomposition via les cliques de séparation
4. Évaluation expérimentale

Vidages inutiles et découpage des durées de vie

Une variable = Un emplacement mémoire durant toute l'exécution

Vidages inutiles et découpage des durées de vie

Une variable = Un emplacement mémoire durant toute l'exécution

Une solution : Le découpage des durées de vie [CS98]

Découper la durée de vie d'une variable en plusieurs durées de vie chacune matérialisée par une nouvelle variable

$\Rightarrow$ Allocation de registres plus précise mais graphe d'interférence-affinité plus grand

Découpage extrême des durées de vie

Découpage extrême des durées de vie = découpage partout

- Ouvre la voie à l'optimalité
- Problème de fusion/affectation difficile
- Optimal Coalescing Challenge (OCC)

Fusion optimale par PL atteint ses limites pour certaines instances de l'OCC

Approche

Idée

Beaucoup de points de découpage utiles pour le vidage mais inutiles pour la fusion

Comment les détecter avant de procéder à la fusion ?

Approche

Idée

Beaucoup de points de découpage utiles pour le vidage mais inutiles pour la fusion

Comment les détecter avant de procéder à la fusion ?

Approche

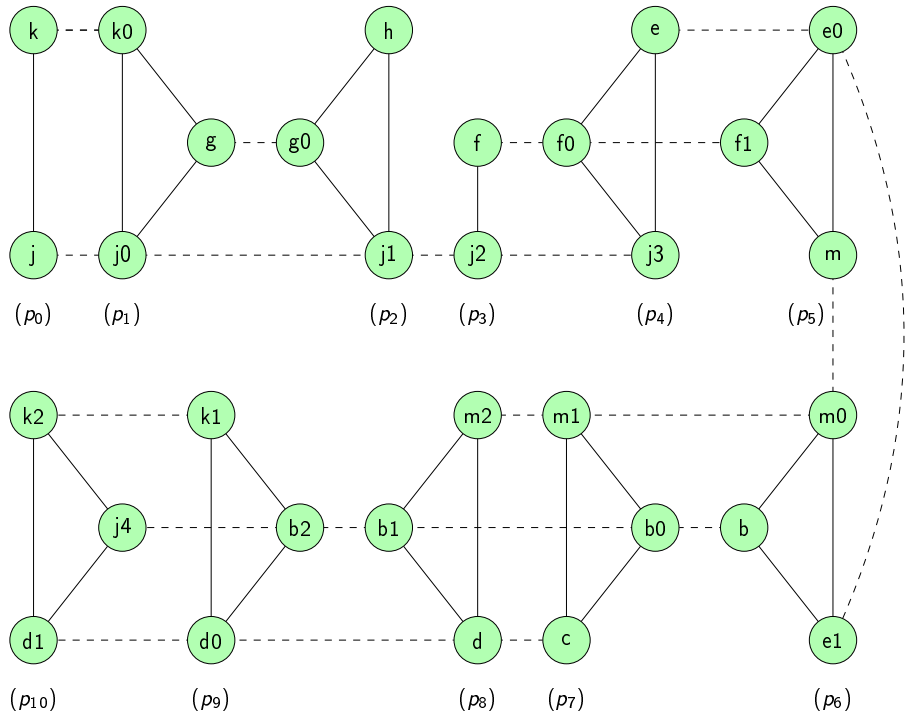
- Étudier les propriétés des graphes éclatés
- Identifier les points de découpage inutiles pour la fusion
- Élaborer une réduction des graphes rendant la fusion soluble

Exemple de graphe éclaté

Live-in : k j

(p_1)	k0 = k ‖ j0 = j ‖ g = mem[j+12]
(p_2)	j1 = j0 ‖ g0 = g ‖ h = k0 - 1
(p_3)	j2 = j1 ‖ f = g0 · h
(p_4)	f0 = f ‖ j3 = j2 ‖ e = mem[j2+8]
(p_5)	e0 = e ‖ f1 = f0 ‖ m =mem[j3+16]
(p_6)	e1 = e0 ‖ m0 = m ‖ b =mem[f1]
(p_7)	b0 = b ‖ m1 = m0 ‖ c = e1 + 8
(p_8)	b1 = b0 ‖ m2 = m1 ‖ d = c
(p_9)	b2 = b1 ‖ d0 = d ‖ k1 = m2 + 4
(p_{10})	d1 = d0 ‖ k2 = k1 ‖ j4 = b2

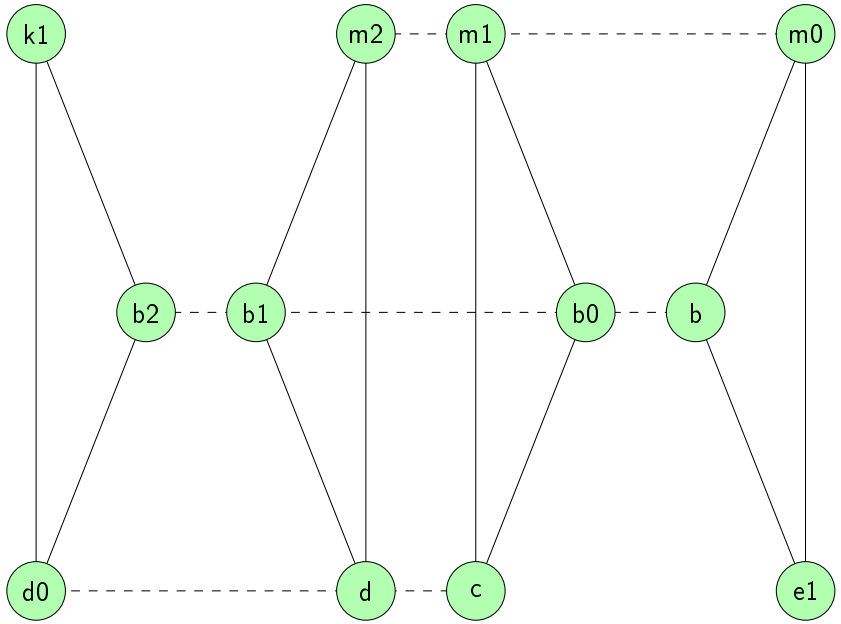
Live-out : d1 k2



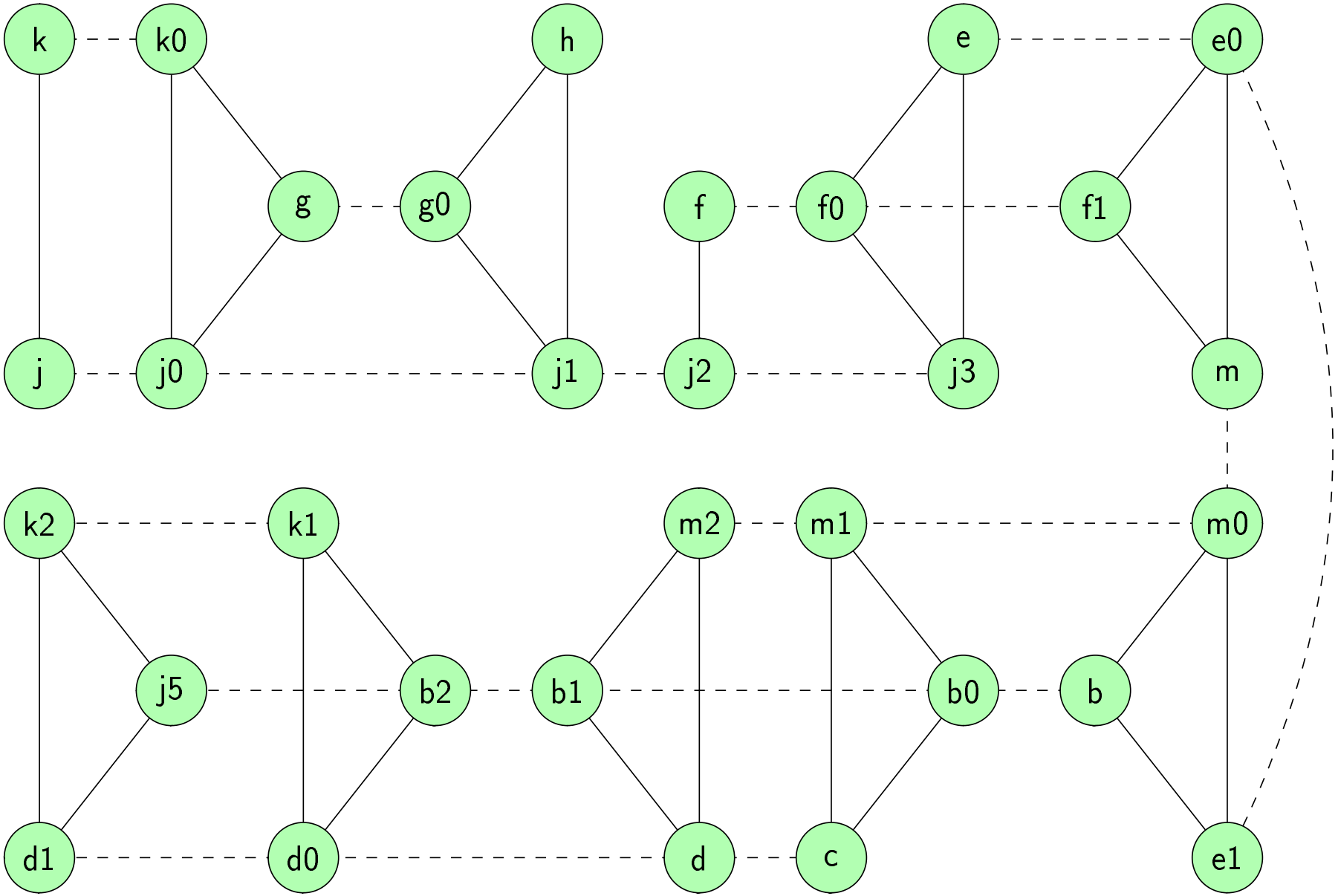
Cliques parallèles

(p_6) $\quad$ e1 = e0 $\|$ m0 = m $\|$ b = mem[f1]
(p_7) $\quad$ b0 = b $\|$ m1 = m0 $\|$ **c = e1 + 8**
(p_8) $\quad$ b1 = b0 $\|$ m2 = m1 $\|$ **d = c**
(p_9) $\quad$ b2 = b1 $\|$ d0 = d $\|$ k1 = m2 + 4

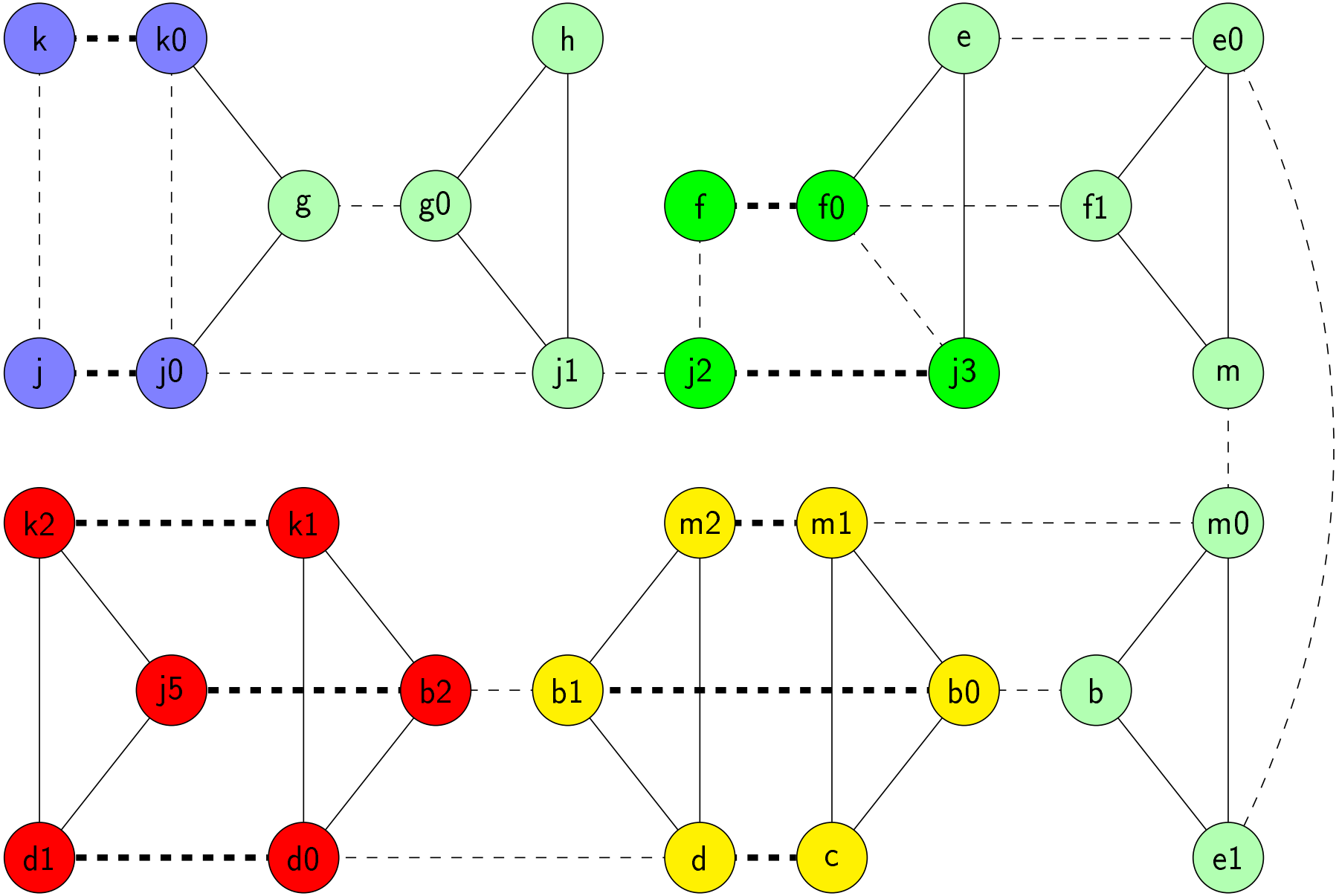
(p_6) $\quad$ e1 = e0 $\|$ m0 = m $\|$ b = mem[f1]
(p_7, p_8) $\quad$ b1 = b $\|$ m2 = m0 $\|$ **d = e1 + 8**
(p_9) $\quad$ b2 = b1 $\|$ d0 = d $\|$ k1 = m2 + 4



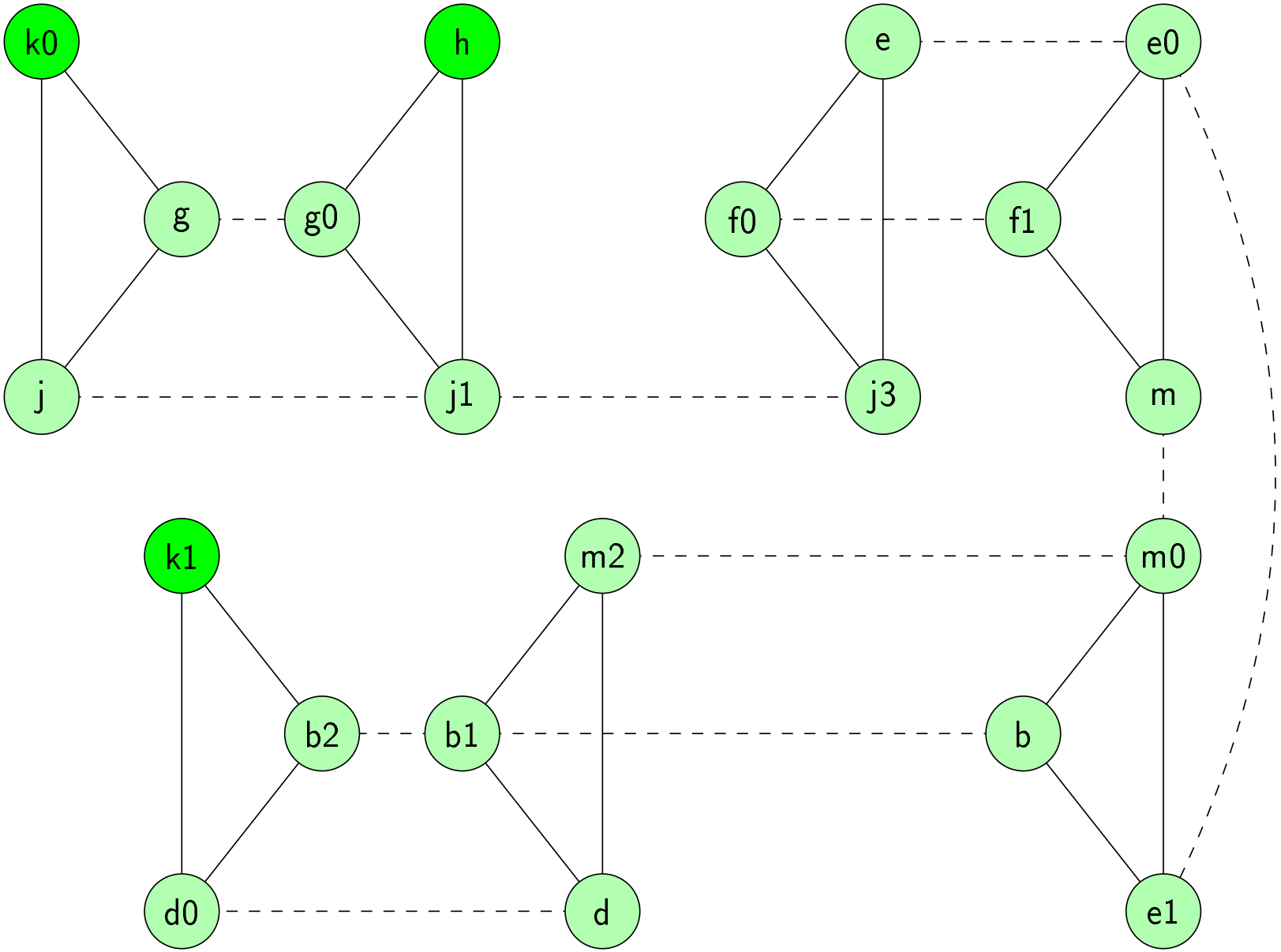
Algorithme appliqué à l'exemple



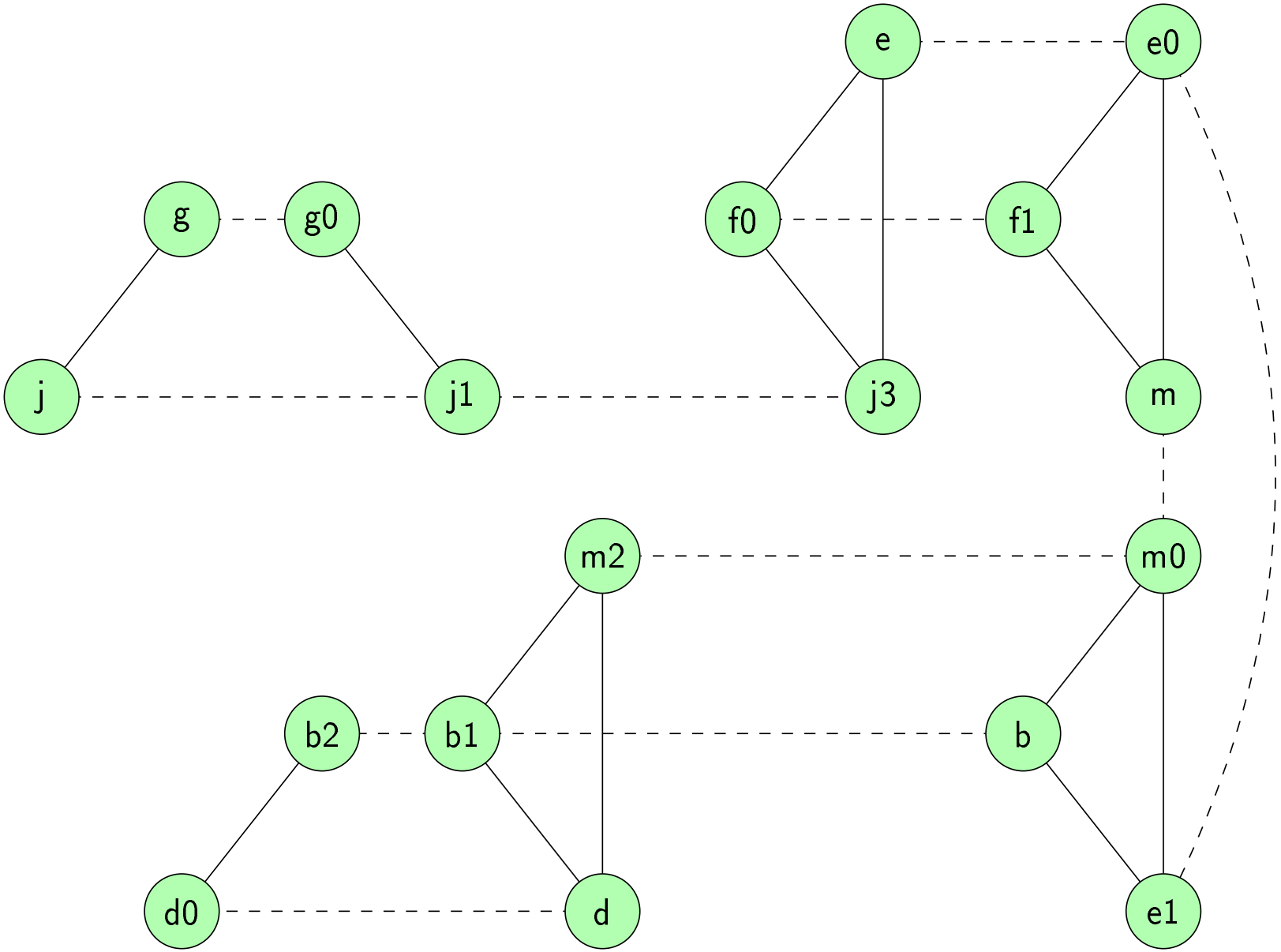
Algorithme appliqué à l'exemple



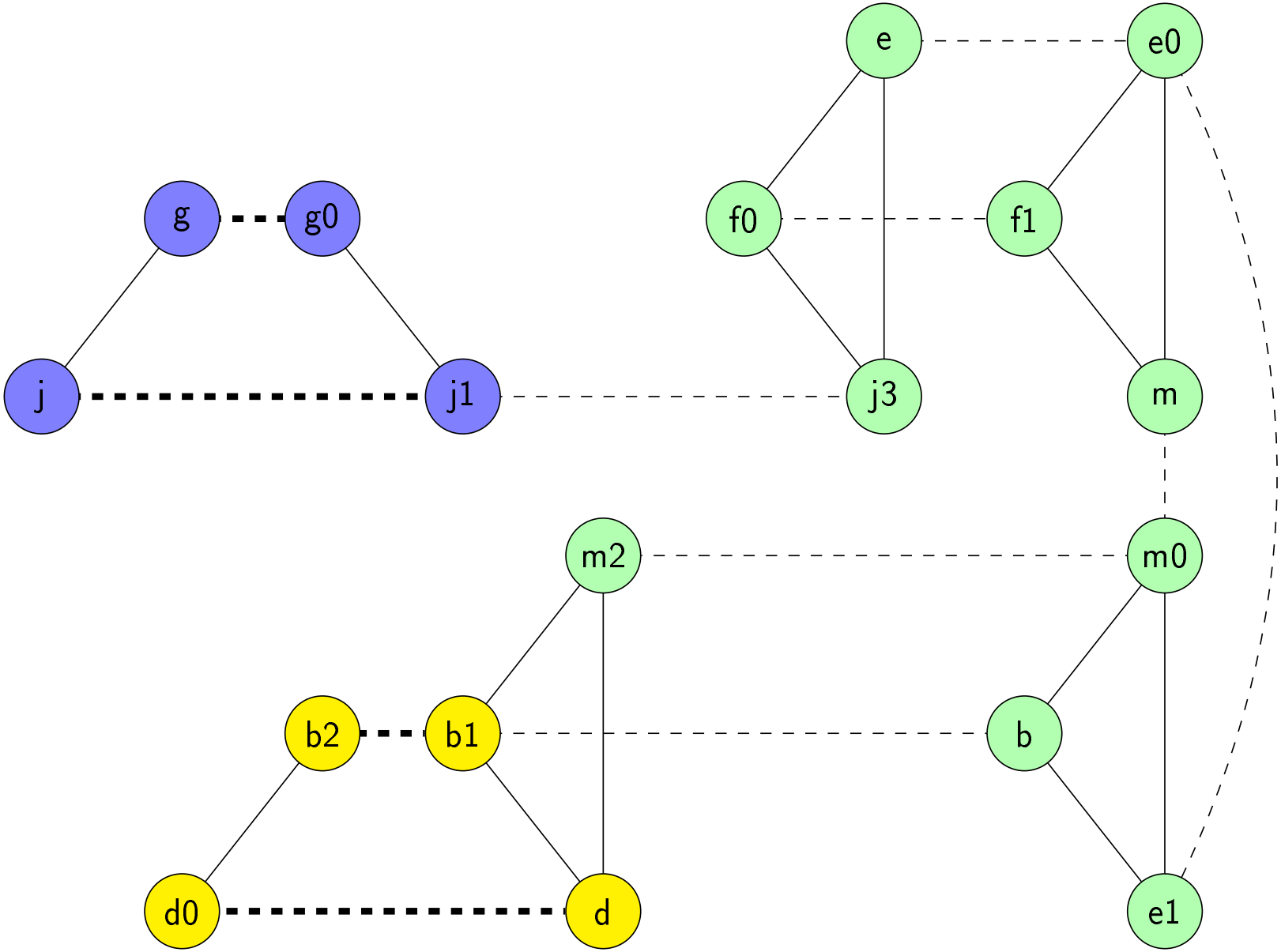
Algorithme appliqué à l'exemple



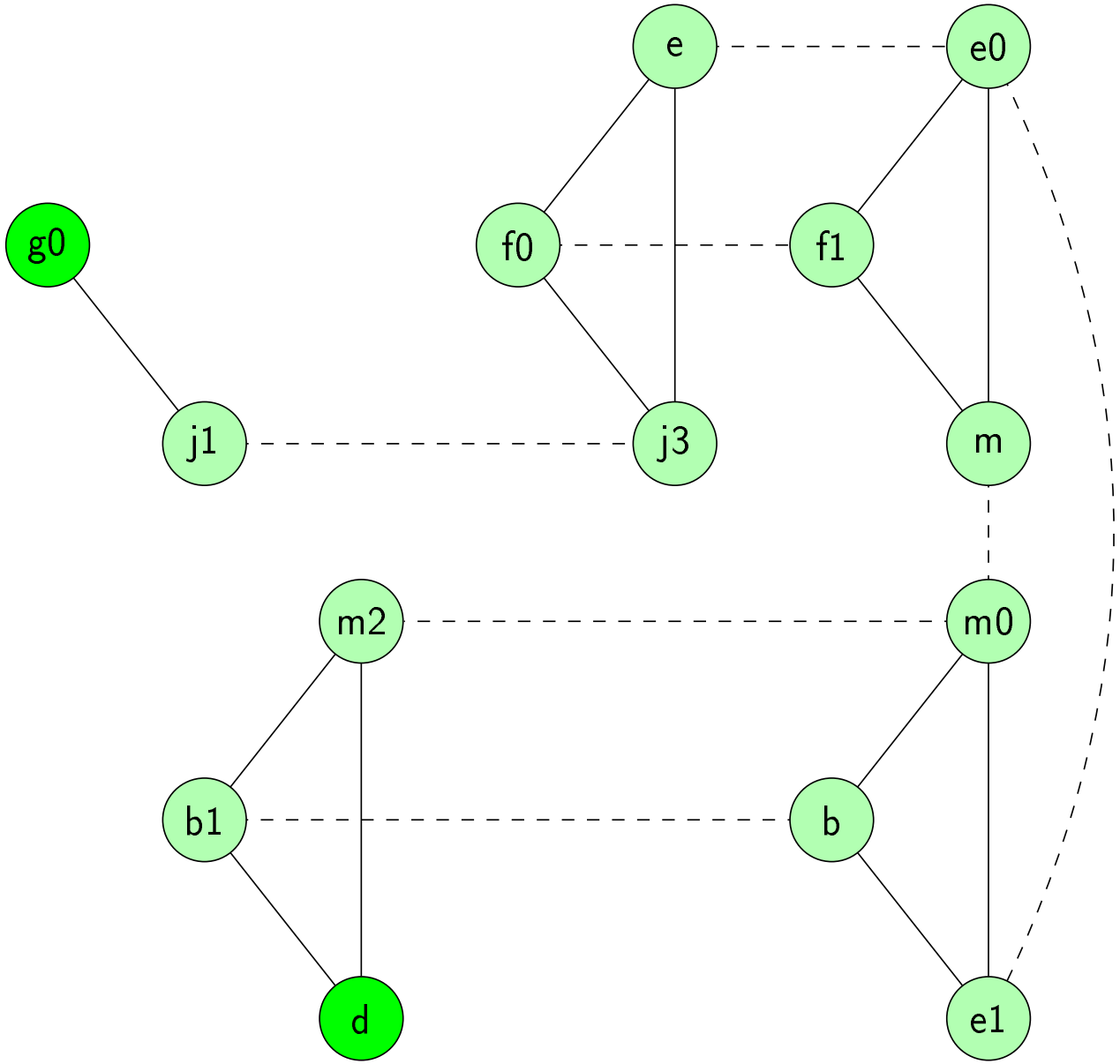
Algorithme appliqué à l'exemple



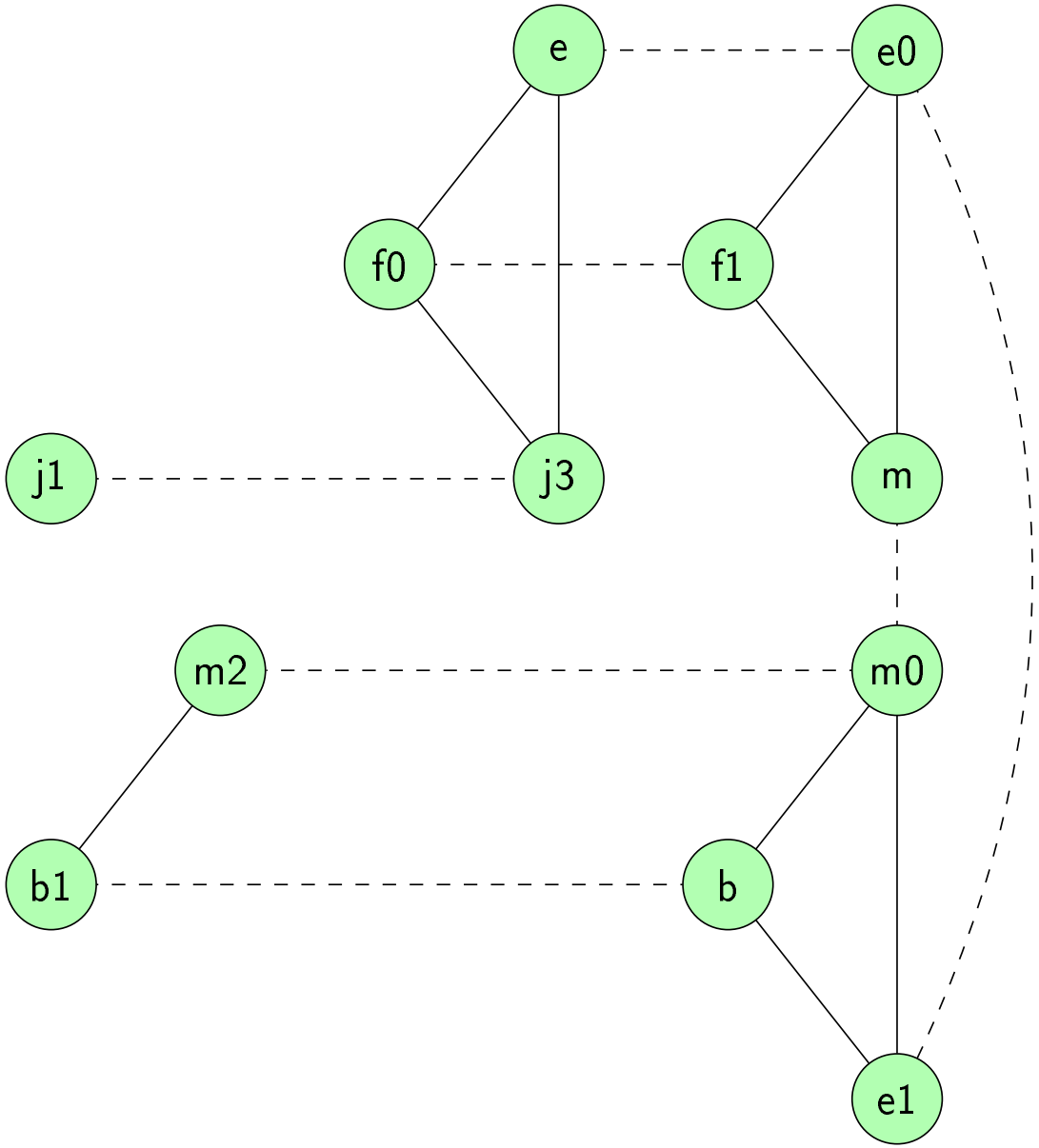
Algorithme appliqué à l'exemple



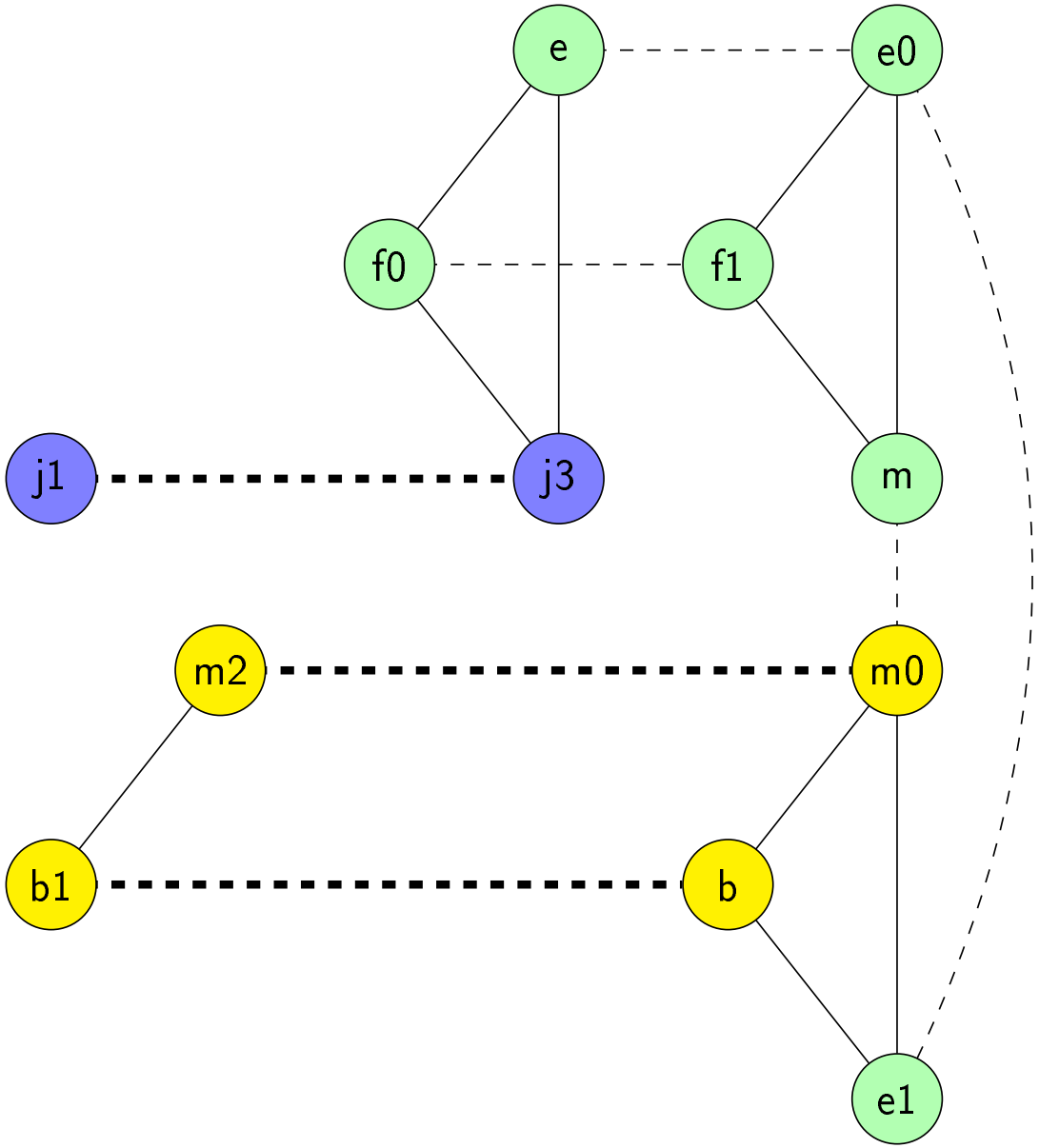
Algorithme appliqué à l'exemple



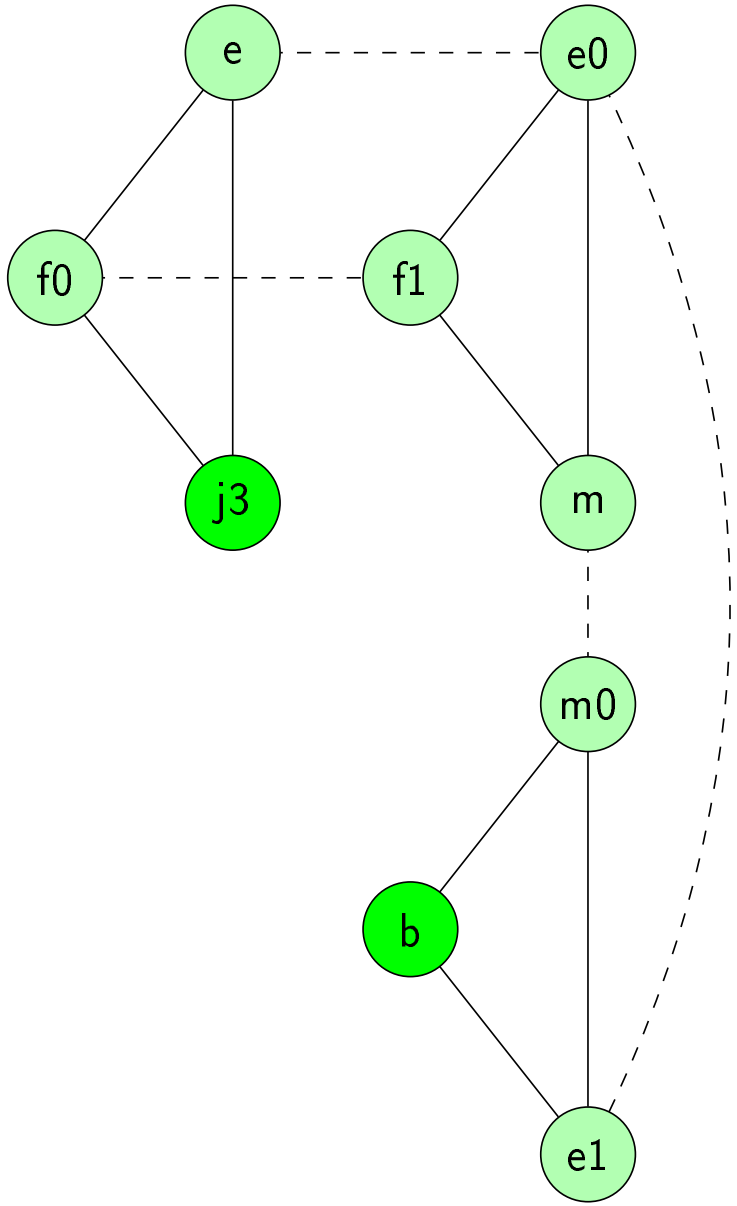
Algorithme appliqué à l'exemple



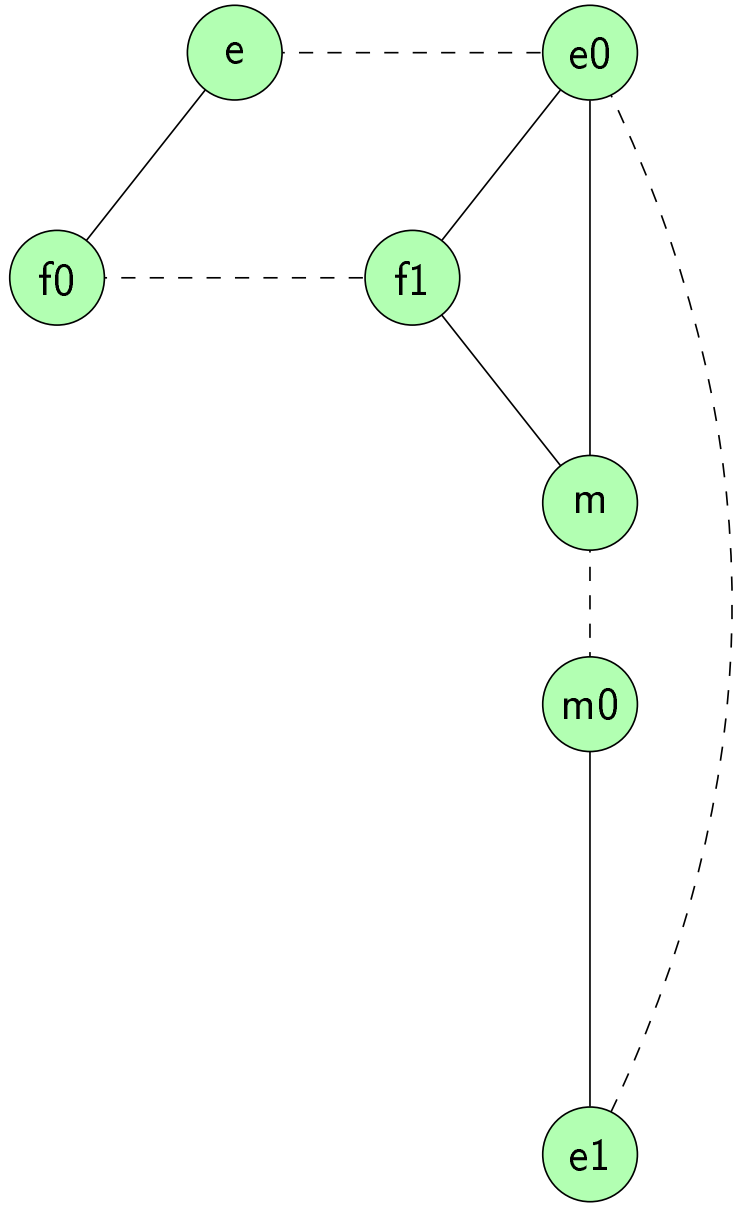
Algorithme appliqué à l'exemple



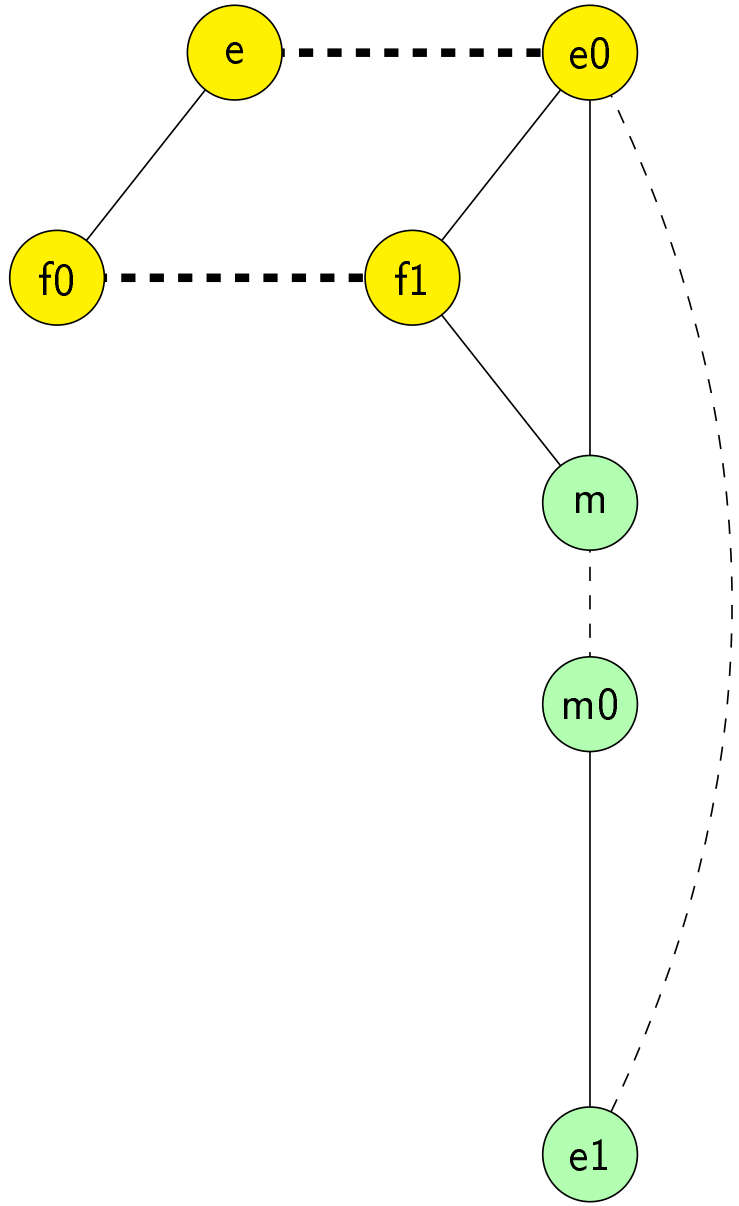
Algorithme appliqué à l'exemple



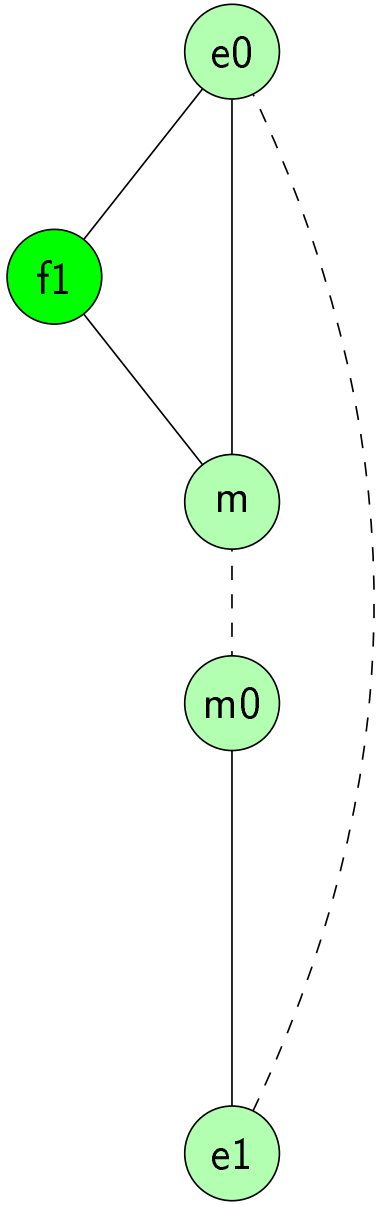
Algorithme appliqué à l'exemple



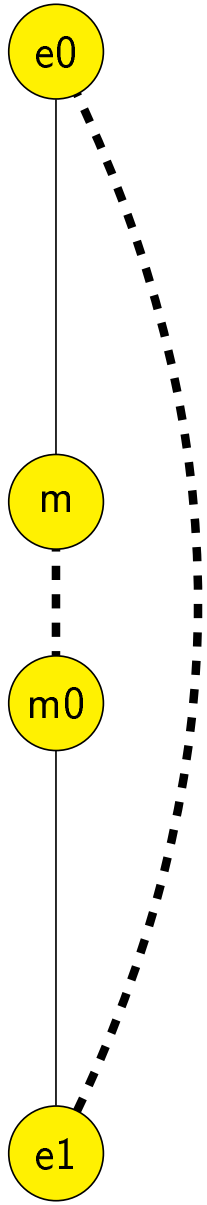
Algorithme appliqué à l'exemple



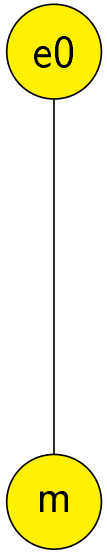
Algorithme appliqué à l'exemple



Algorithme appliqué à l'exemple

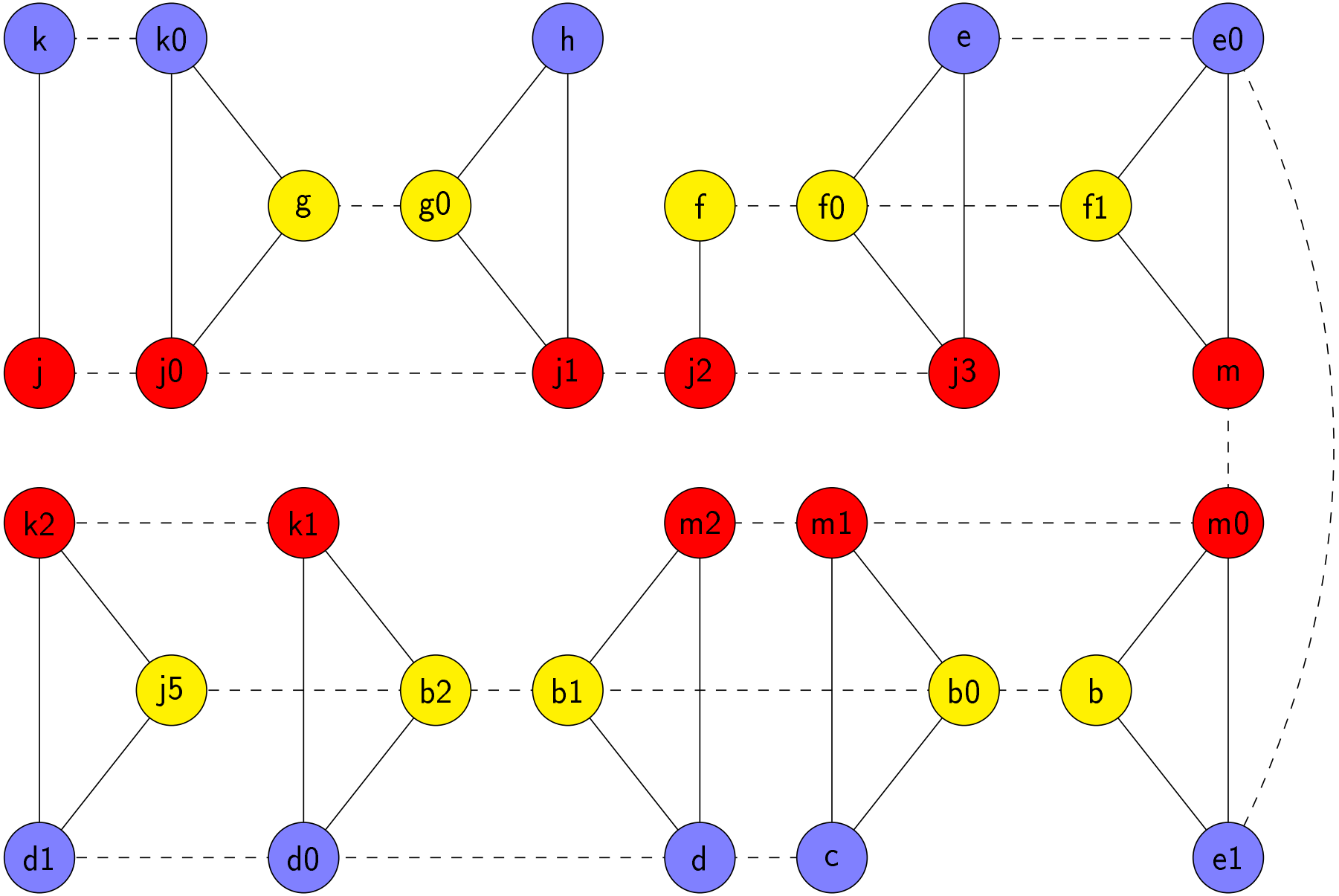


Algorithme appliqué à l'exemple

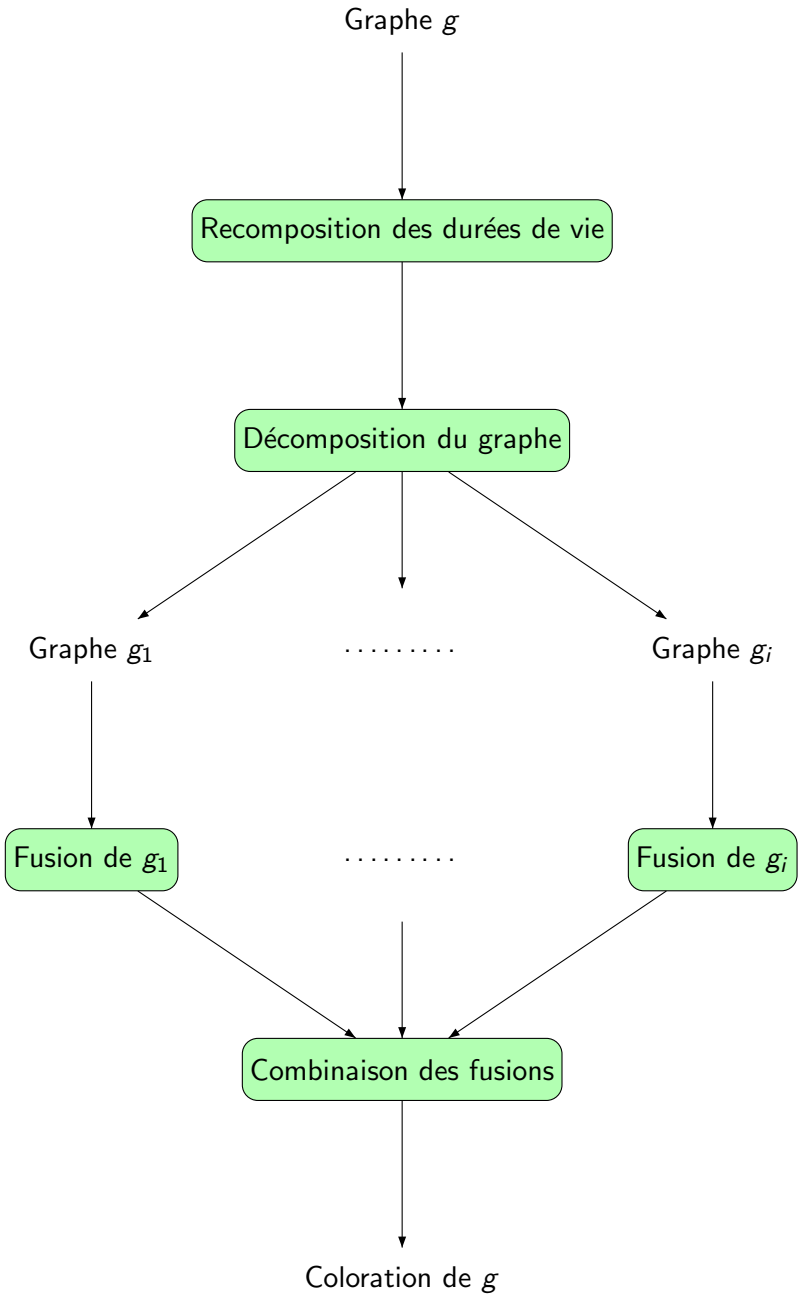


Algorithme appliqué à l'exemple

Algorithme appliqué à l'exemple



Algorithme de réduction complet



Réduction moyenne

OCC : 474 instances de fusion

Sommets	Instances	Réduction moyenne
0 à 499	292	81,6%
500 à 999	97	86,2%
1000 à 2999	63	87,3%
3000 et plus	22	87,4%

Toutes les instances de l'OCC résolues (300 fois plus vite)

Bilan et perspectives

Bilan

Fusion avec découpage extrême des durées de vie

- Problème d'une grande combinatoire
- Règles de réduction efficaces (85%)
- Bons résultats sur les instances de l'OCC

Perspectives

Stratégie de découpage idéale

Conclusion

Nécessité de l'utilisation conjointe de l'optimisation combinatoire et de la preuve de programme pour les systèmes embarqués critiques

Conclusion

Nécessité de l'utilisation conjointe de l'optimisation combinatoire et de la preuve de programme pour les systèmes embarqués critiques

Allocation de registres :

Conclusion

Nécessité de l'utilisation conjointe de l'optimisation combinatoire et de la preuve de programme pour les systèmes embarqués critiques

Allocation de registres :

- Un exemple réaliste

Conclusion

Nécessité de l'utilisation conjointe de l'optimisation combinatoire et de la preuve de programme pour les systèmes embarqués critiques

Allocation de registres :

- Un exemple réaliste
- La preuve que c'est possible

Conclusion

Nécessité de l'utilisation conjointe de l'optimisation combinatoire et de la preuve de programme pour les systèmes embarqués critiques

Allocation de registres :

- Un exemple réaliste
- La preuve que c'est possible
- La démonstration de l'adéquation du système Coq

Conclusion

Nécessité de l'utilisation conjointe de l'optimisation combinatoire et de la preuve de programme pour les systèmes embarqués critiques

Allocation de registres :

- Un exemple réaliste
- La preuve que c'est possible
- La démonstration de l'adéquation du système Coq
- Le début d'une aventure passionnante...

Questions ?

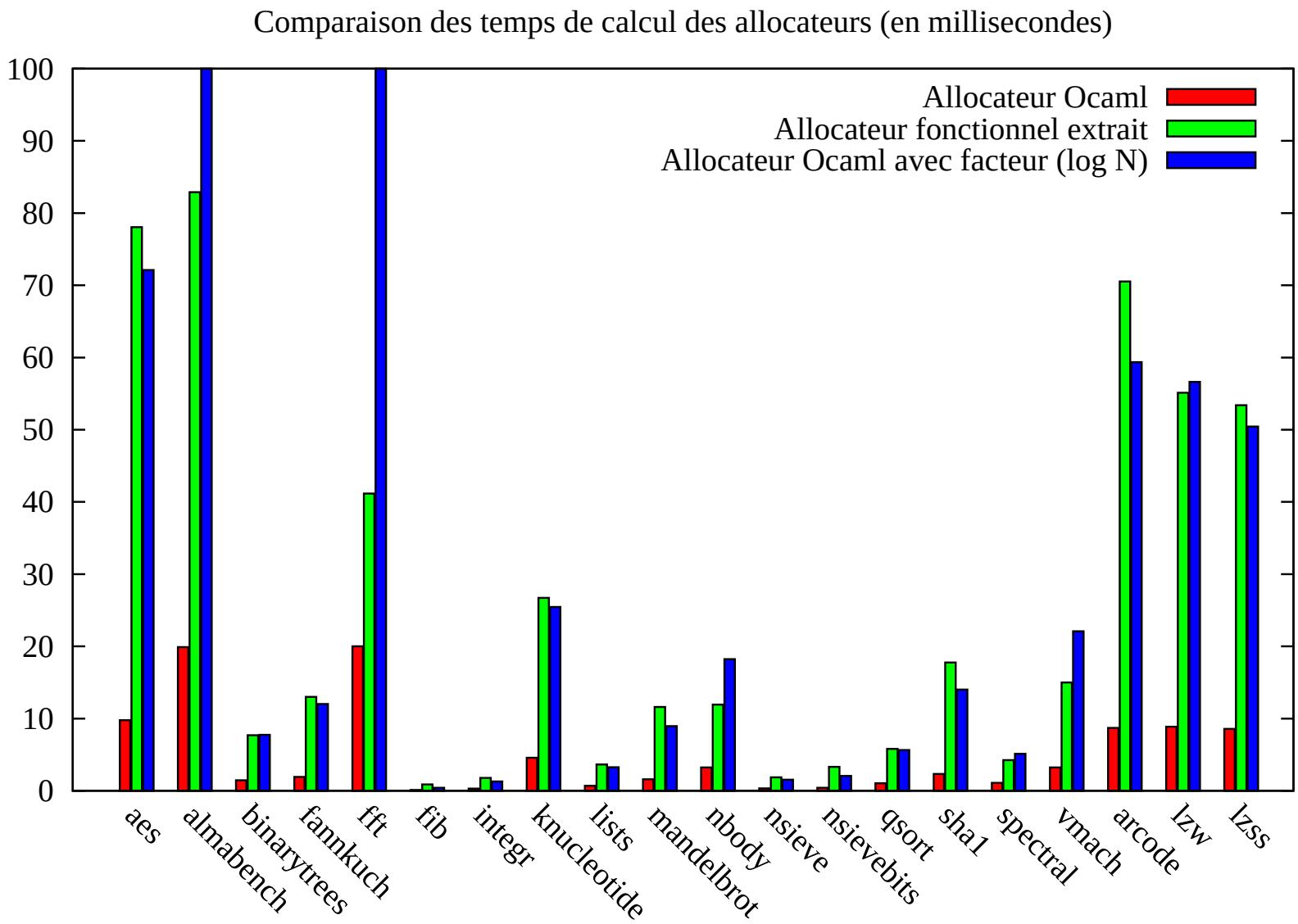
Détails : http://www.ensiie.fr/~robillard/

Sandrine Blazy et Benoit Robillard.
Live-Range Unsplitting for Faster Optimal Coalescing.
LCTES'09 : Proc. of the 12th ACM Conference on Languages, Compilers and Tools for Embedded Systems, Dublin, Ireland.

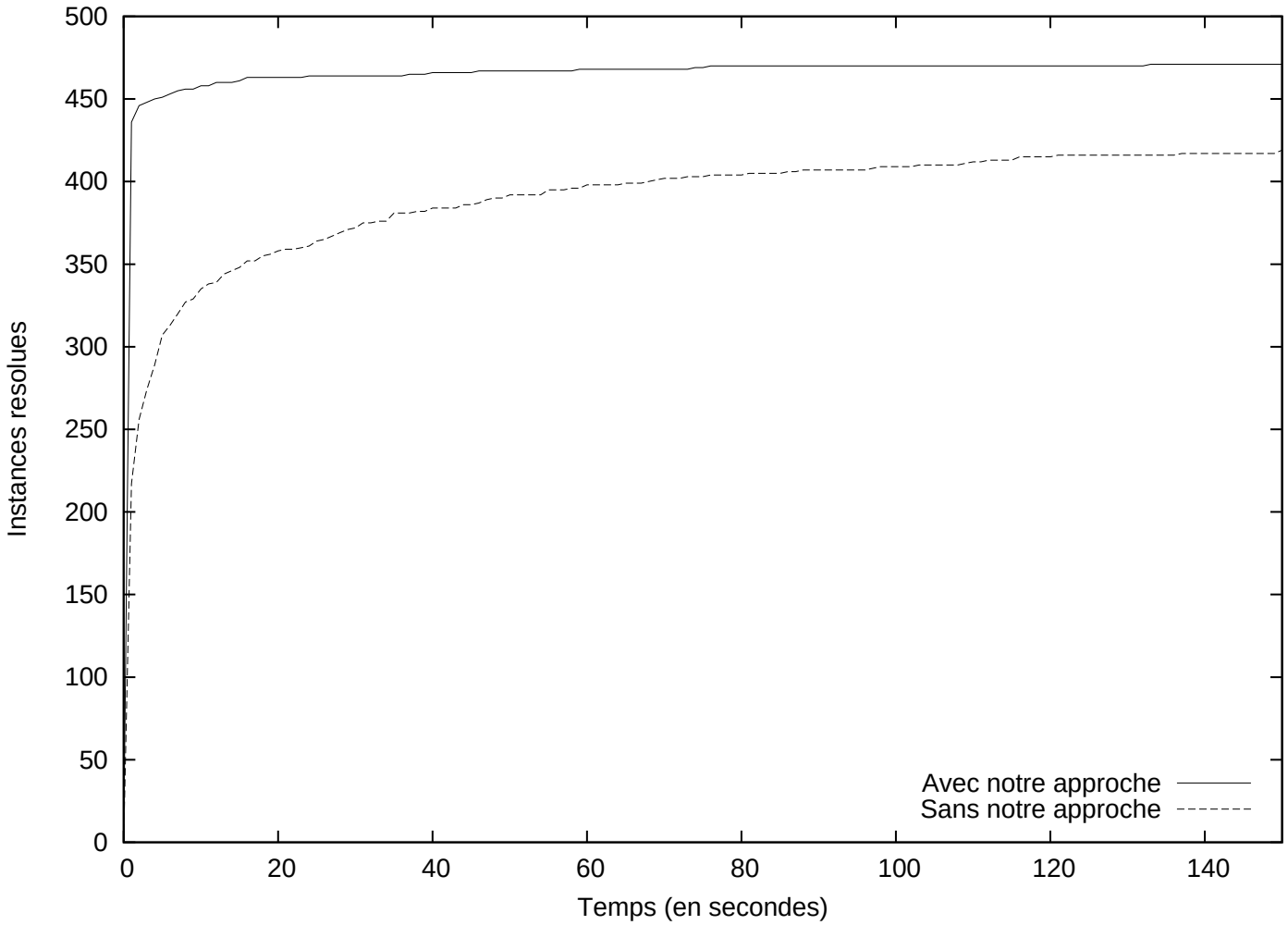
Sandrine Blazy, Benoit Robillard et Andrew W. Appel.
Formal Verification of Coalescing Graph-Coloring Register Allocation.
ESOP'10 : Proc. of the 19th European Symposium On Programming, Paphos, Cyprus.

Benoit Robillard et Eric Soutif.
Maximum Affinity Coloring : Complexity in Bipartite Conflict Graphs and Links with Multiway Cut.
Soumis à une revue, Novembre 2010.

Efficacité du code extrait



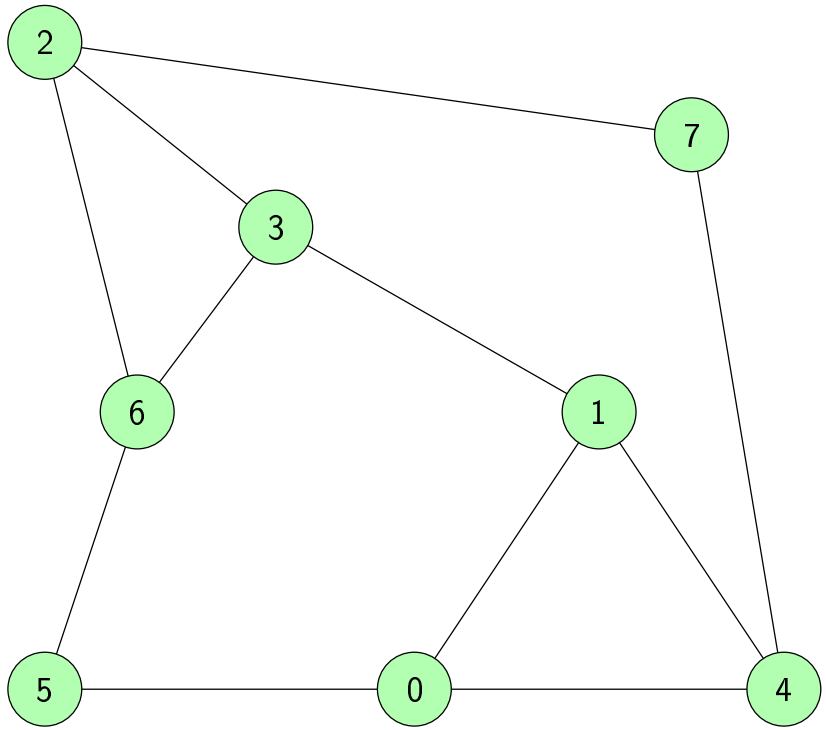
Comparaison des temps de calcul (1)



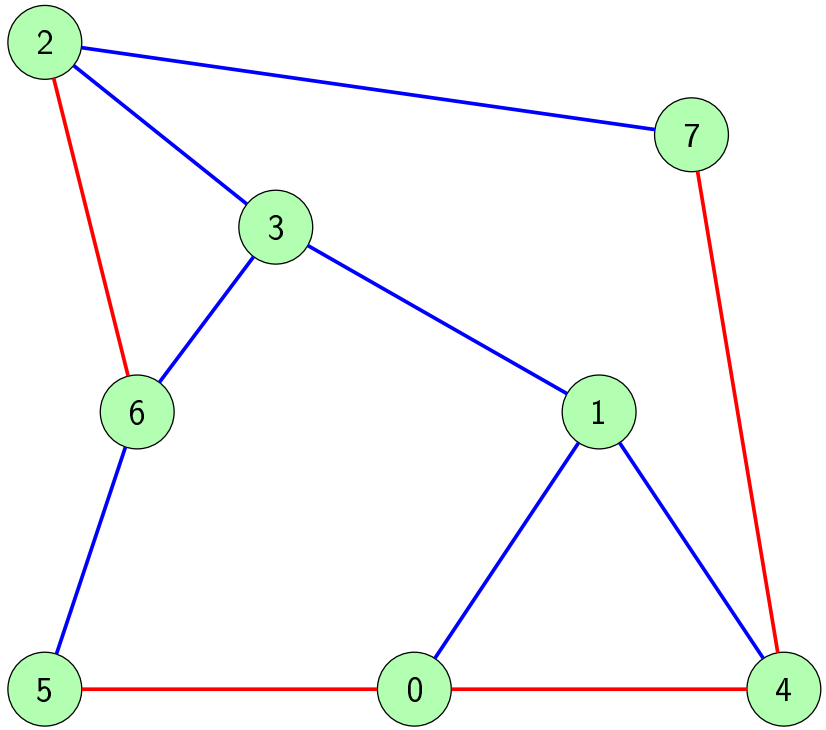
Comparaison des temps de calcul (2)

Nb d'instances résolues	Temps avec notre stratégie (s)	Temps sans notre stratégie (s)	Ratio temps sans/avec
436	1	298	298
446	2	636	318
448	3	732	244
450	4	1198	300
451	5	1228	245
453	6	3153	525
455	7	3321	474
456	8	3574	447
458	10	4022	402
460	12	5630	469
461	15	7214	480
463	16	8709	544
464	24	10397	433
465	37	11757	317
466	40	14941	373
467	59	24916	422
468	74	25490	344
469	76	25663	337
470	133	69956	525
471	11026	71208	7

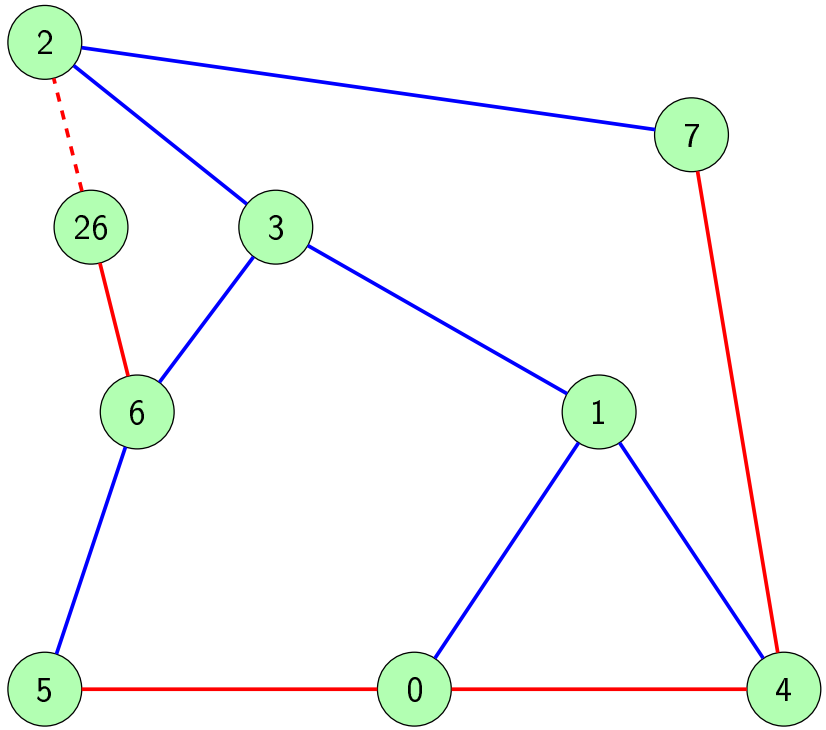
Réduction de K-coloration (1)



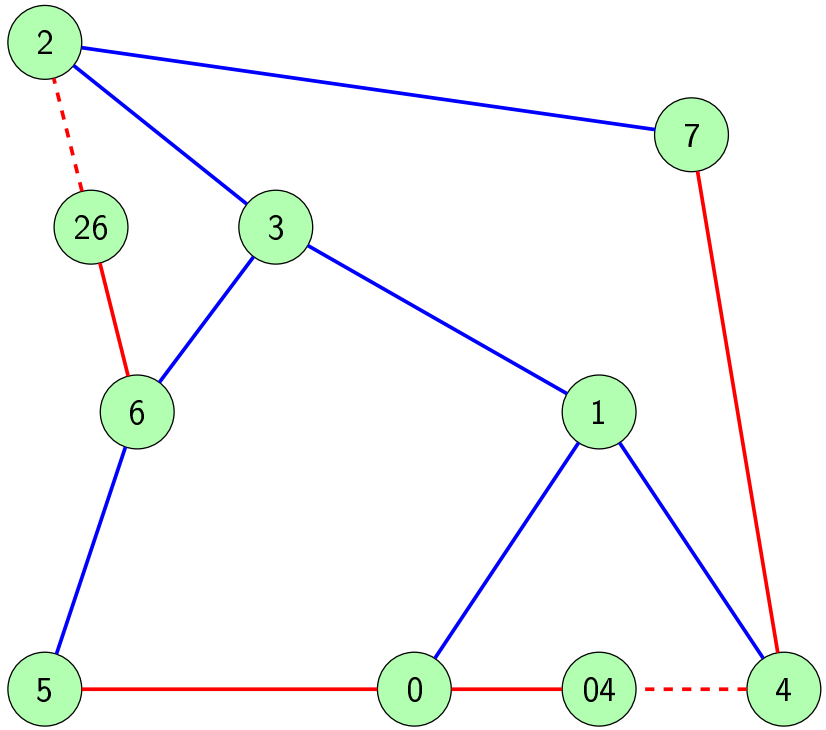
Réduction de K-coloration (2)



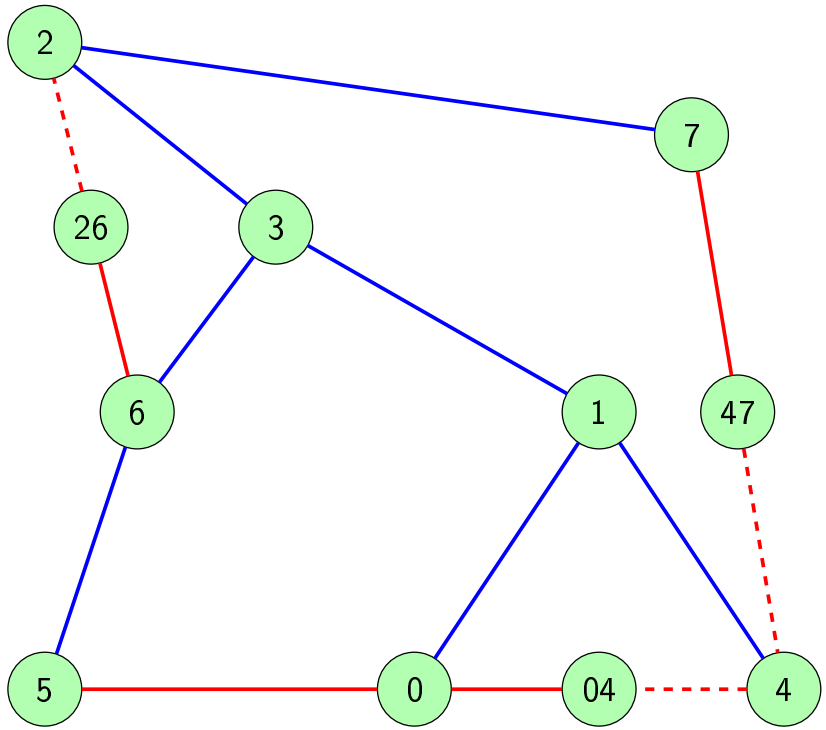
Réduction de K-coloration (3)



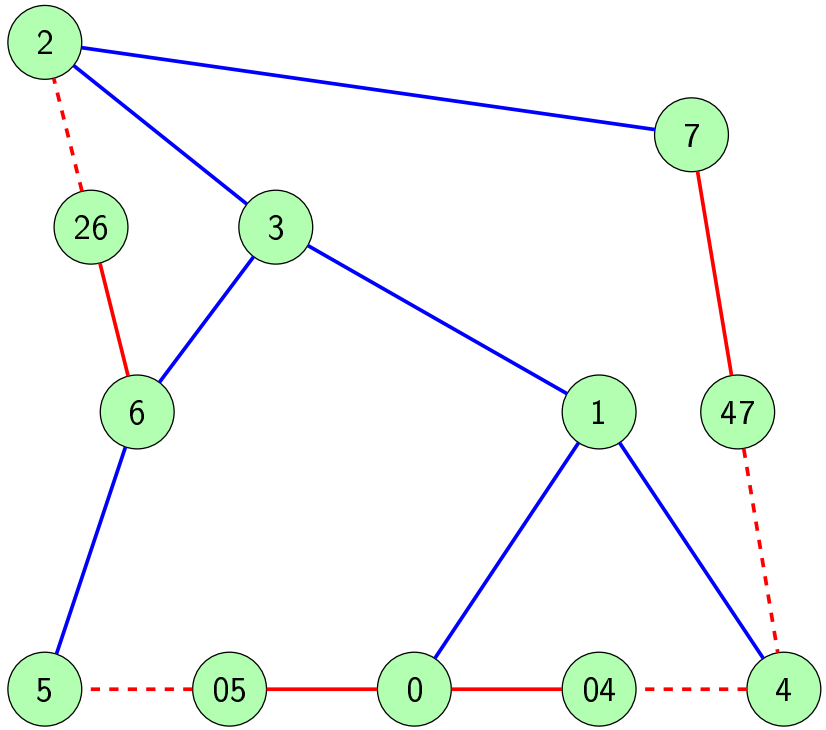
Réduction de K-coloration (4)



Réduction de K-coloration (5)



Réduction de K-coloration (6)



Preuve de la réduction

Correction de la réduction

Le graphe G est K-colorable ssi $G_{\nu(G)}$ admet une K-coloration avec préférences de valeur $\nu(G)$

Preuve

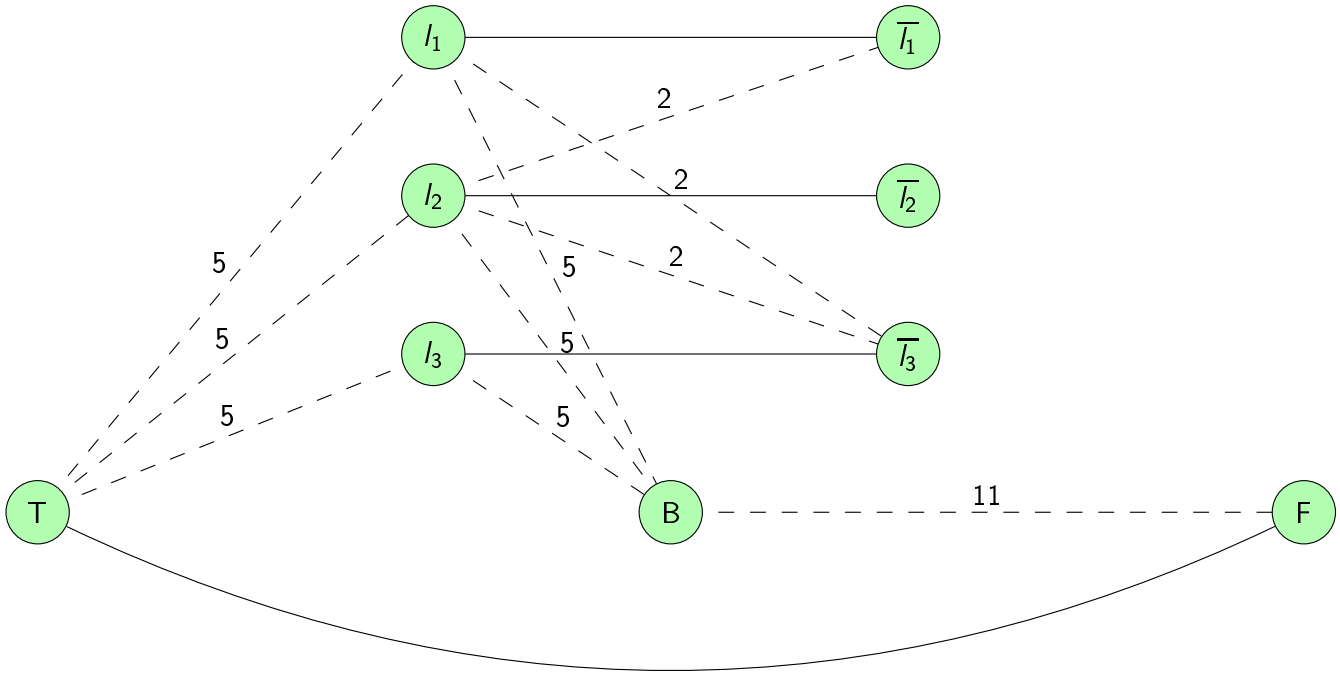
G est K-colorable $\Leftrightarrow \exists$ une K-coloration de G_i de valeur i

Preuve par récurrence sur i

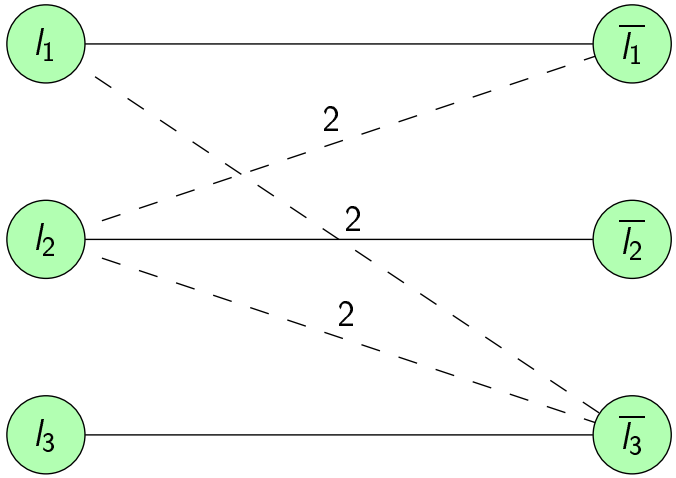
Cas de base : trivial car $G = G_0$

Cas de récurrence : $\exists$ une K-coloration de G_{i+1} de valeur $i+1$
$\Leftrightarrow \exists$ une K-coloration de G_{i+1} satisfaisant toutes les arêtes de préférence de G_{i+1}
$\Leftrightarrow \exists$ une K-coloration de G_i de valeur i
$\Leftrightarrow \exists$ une K-coloration de G

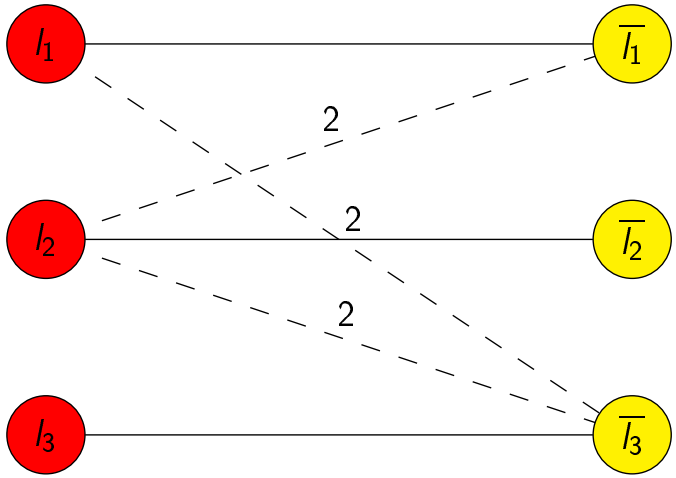
Preuve de complexité de la fusion/affectation (K=2)



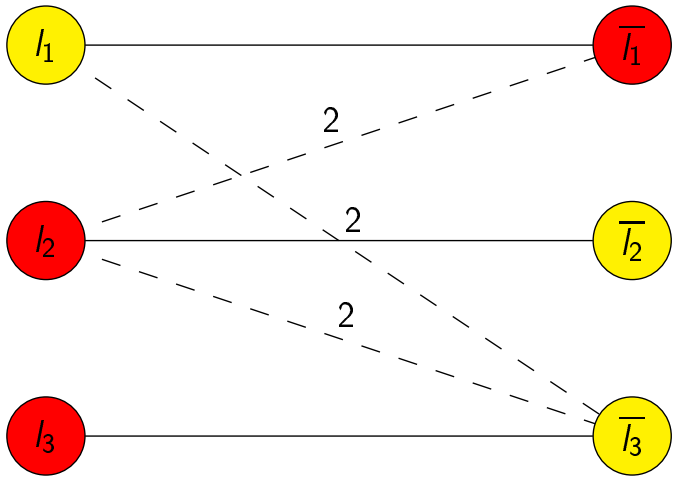
Transformation d'une clause en graphe (1)



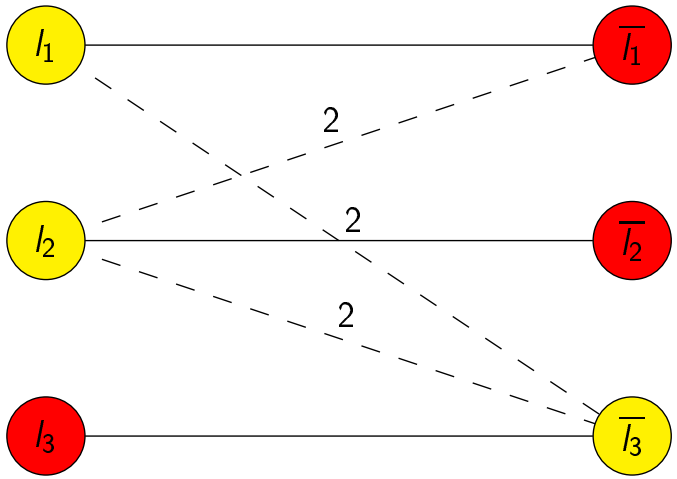
Transformation d'une clause en graphe (1)



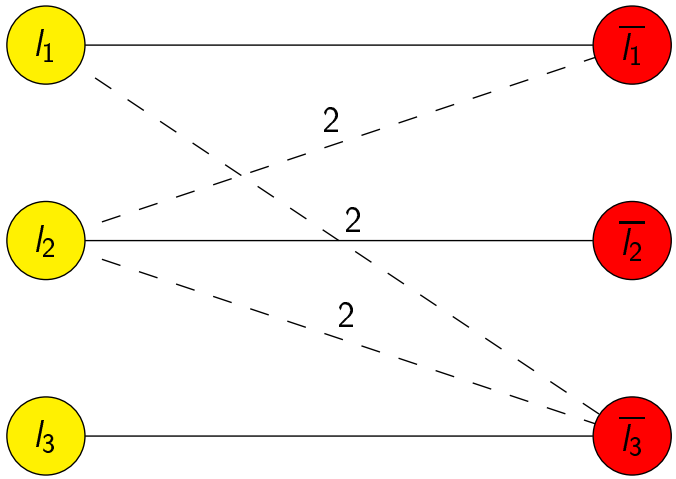
Transformation d'une clause en graphe (1)



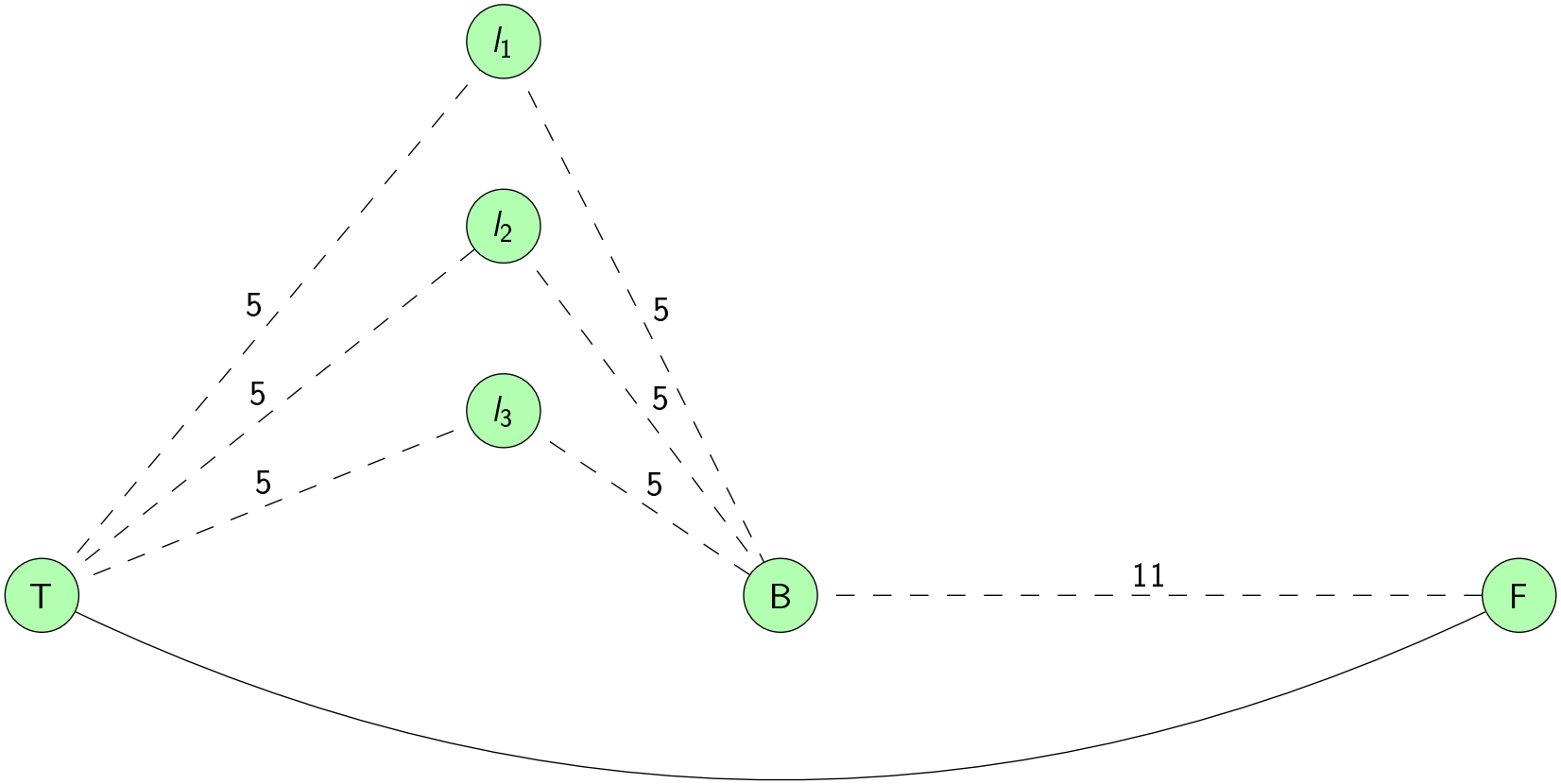
Transformation d'une clause en graphe (1)



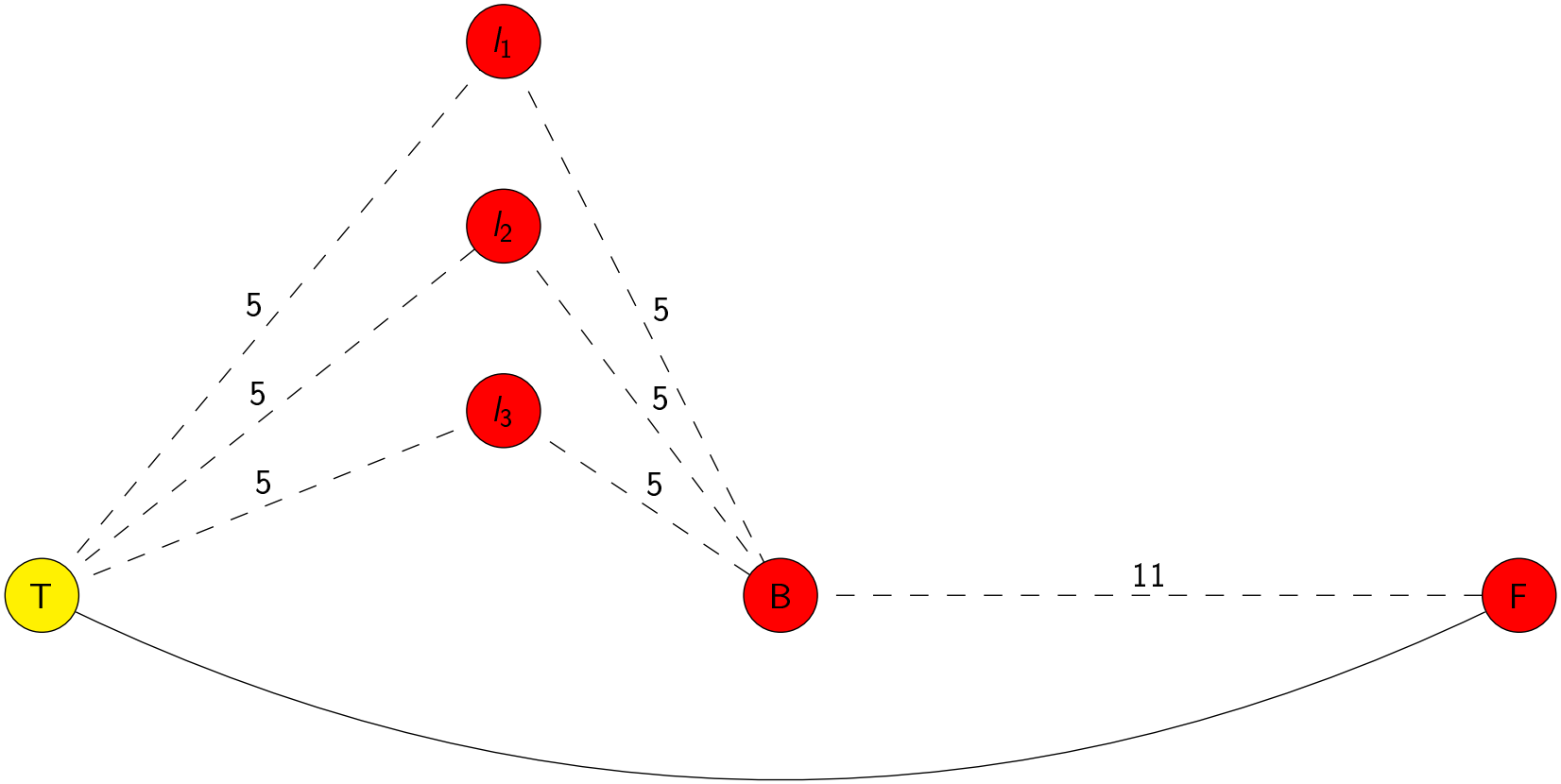
Transformation d'une clause en graphe (1)



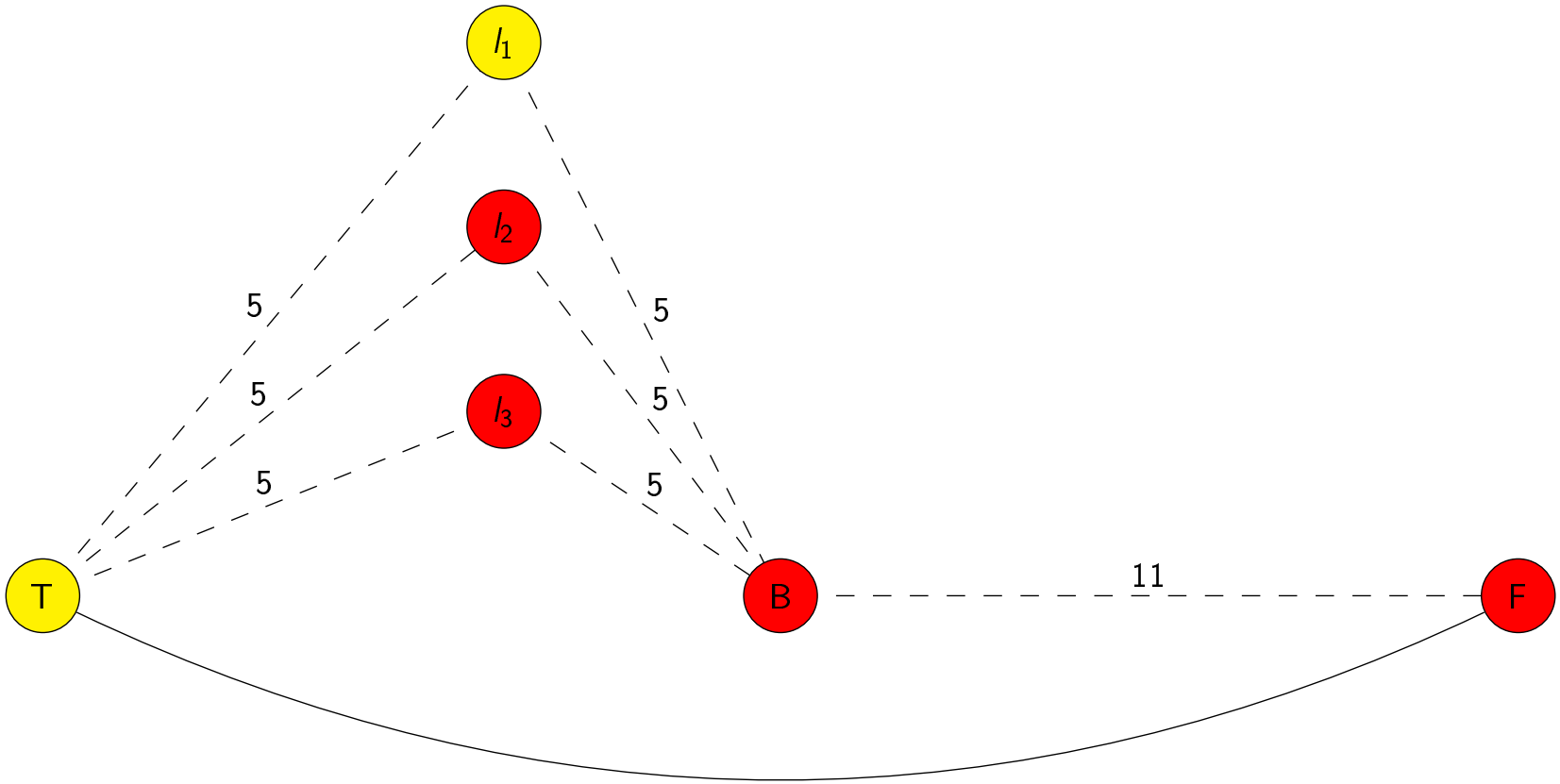
Transformation d'une clause en graphe (2)



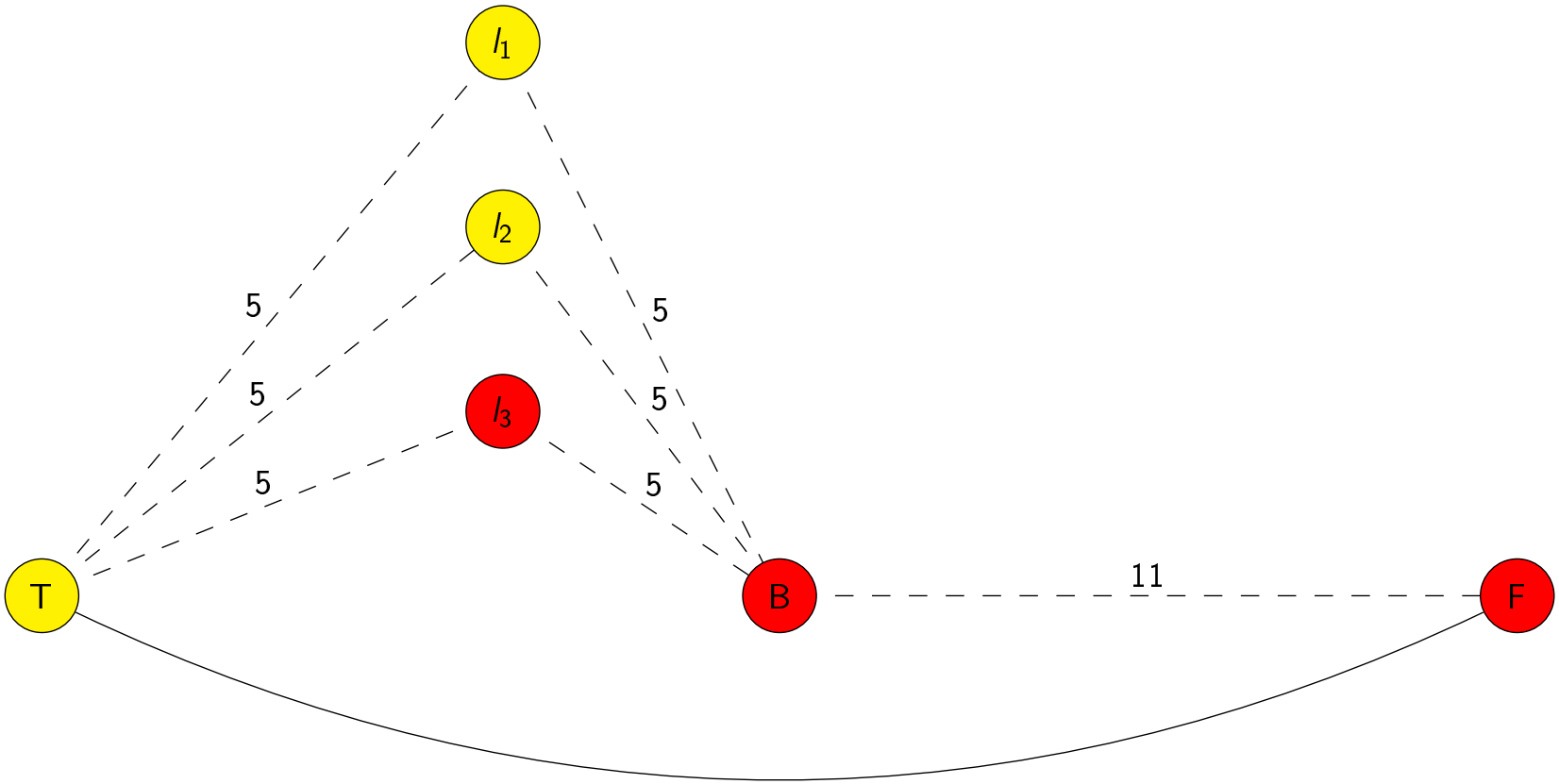
Transformation d'une clause en graphe (2)



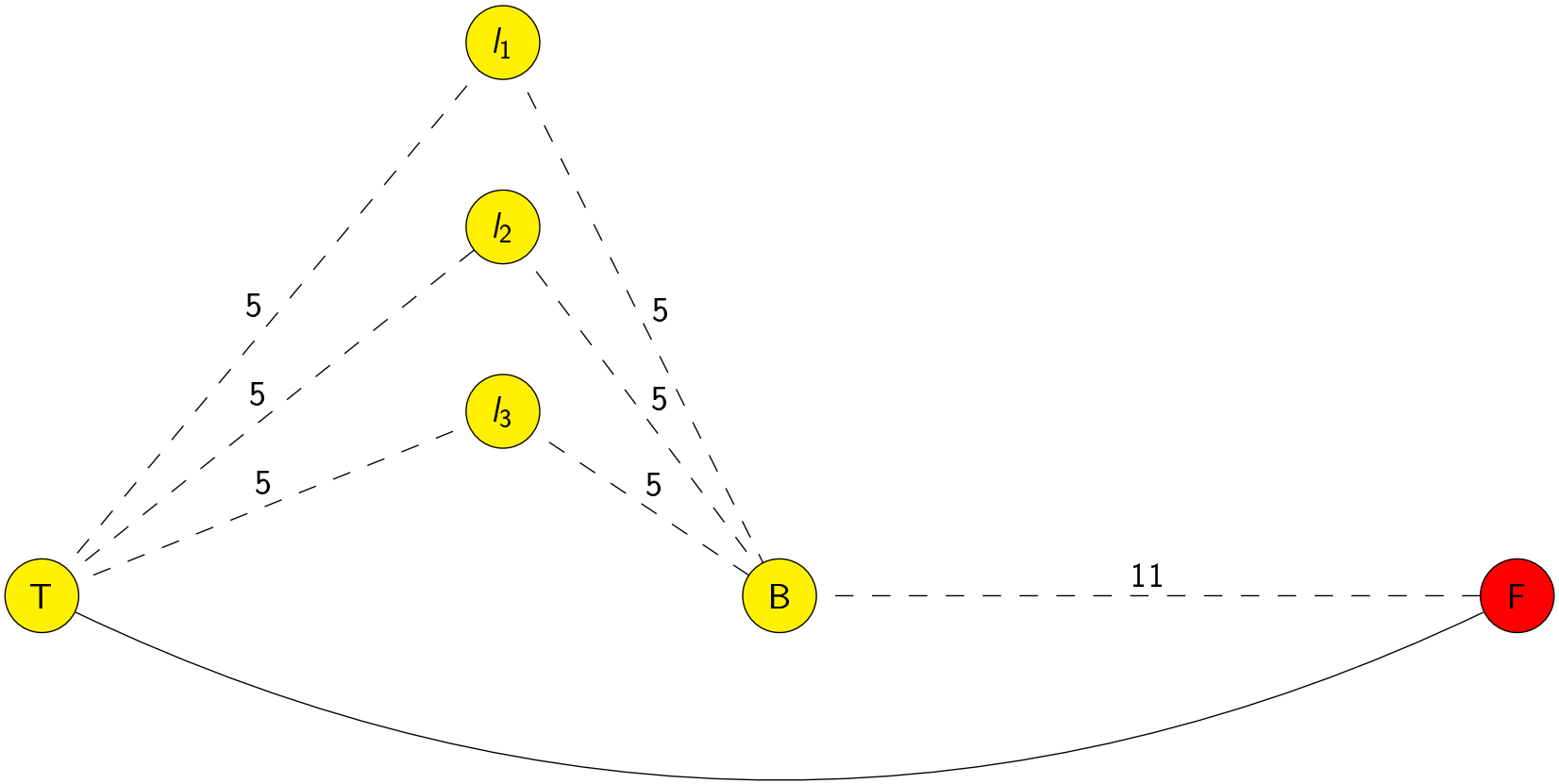
Transformation d'une clause en graphe (2)



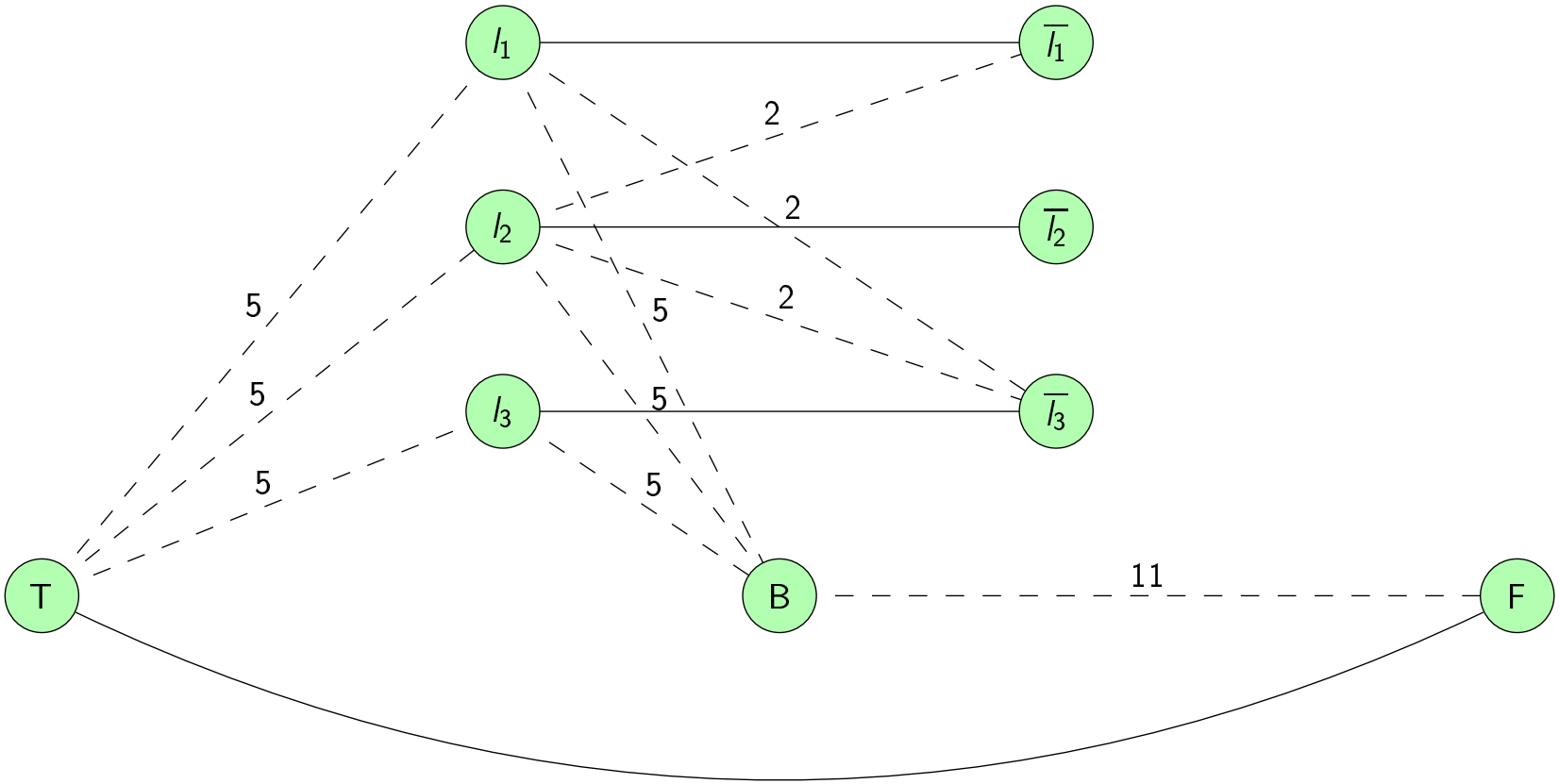
Transformation d'une clause en graphe (2)



Transformation d'une clause en graphe (2)



Transformation d'une clause en graphe (3)



Transformation d'une formule en graphe

Pour que les valeurs des littéraux soient cohérentes on ajoute p préférences dites de cohérence entre toutes les occurences d'un même littéral, de poids $w = (30n + 1)$

Cohérence des valeurs des littéraux

Toute 2-coloration avec préférence optimale respecte les préférences de cohérence

Preuve

Si elles sont toutes respectées alors

$$v \geq p(30n + 1)$$

Si elles ne le sont pas alors

$$v \leq 30n + (p - 1)(30n + 1) = p(30n + 1) - 1$$

Théorème de réduction

Réduction de 3-SAT vers la coloration avec préférences

La formule F est satisfiable si et seulement s'il existe une coloration avec préférence du graphe de valeur $v_{opt} = p(30n + 1) + 30n$

Preuve

1. Les cohérences rapportent $p(30n + 1)$ pour toute solution optimale
2. Le gain correspondant aux clauses est égal à $30x + 26(n - x)$, x étant le nombre de clauses satisfaites

Bibliographie


Florent Bouchez, Alain Darte, Christophe Guillon, and Fabrice Rastello.
Register allocation and spill complexity under ssa.
Technical report, 2005.


Keith D. Cooper and L. Taylor Simpson.
Live range splitting in a graph coloring register allocator.
In *CC '98 : Proc. of the 7th Int. Conference on Compiler Construction*, pages 174–187, London, UK, 1998. Springer-Verlag.


Lal George and Andrew W. Appel.
Iterated register coalescing.
ACM Transactions On Programming Languages and Systems (TOPLAS), 18(3) :300–324, 1996.


Daniel Grund and Sebastian Hack.
A fast cutting-plane algorithm for optimal coalescing.
In *Compiler Construction, CC'07, Braga, Portugal*, volume 4420 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2007.


Sebastian Hack.
Interference Graphs of Programs in SSA Form.
Technical Report 2005-15, Universität Karlsruhe, June 2005.