RO dans les réseaux et transports : projet

MPRO

Dimitri Watel Alain Faye

www.ensiie.fr/∼dimitri.watel

www.ensiie.fr/∼faye

Problème d'arbre couvrant avec un nombre minimum de nœuds de branchement. (min-ACNB)

Instance

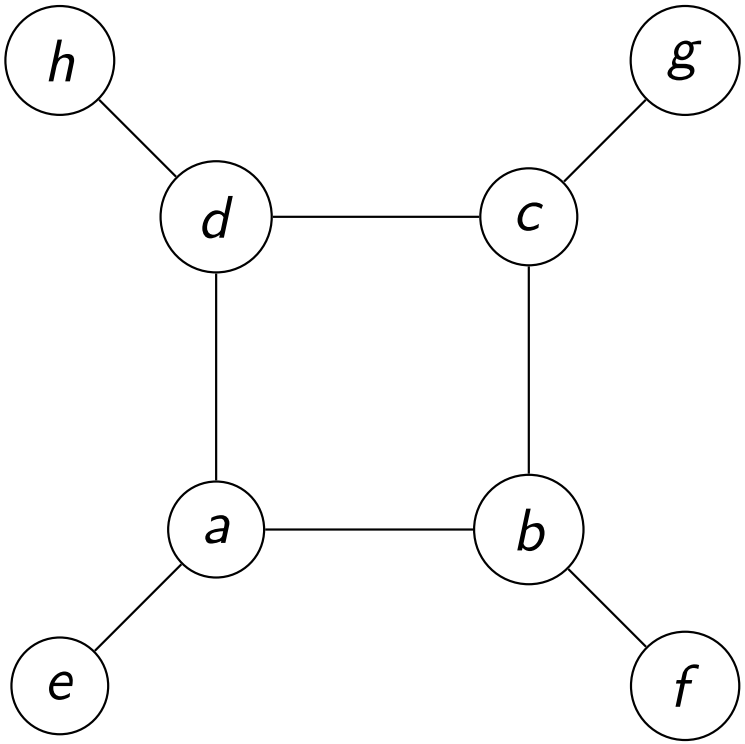
- Un graphe $G = (V, E)$ non orienté

Sortie

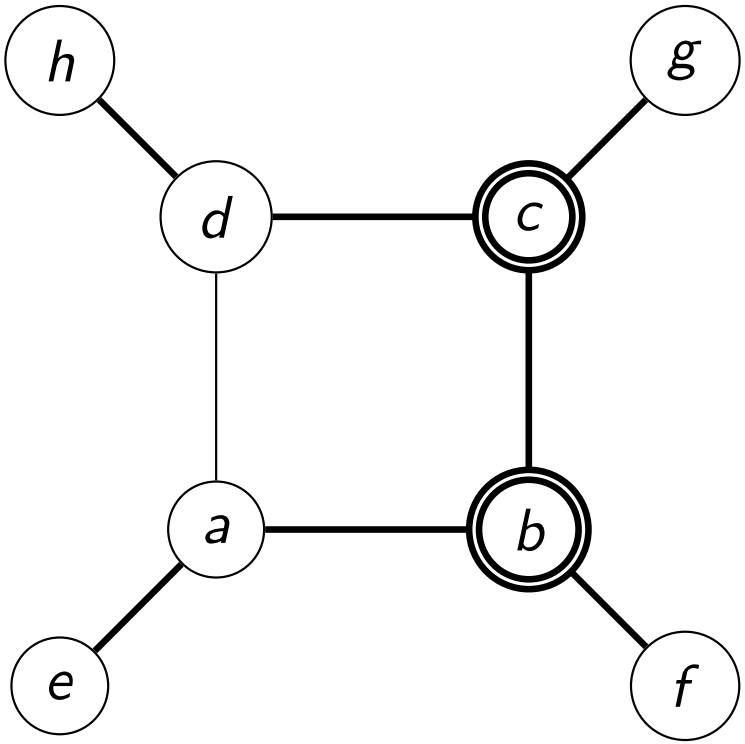
Un arbre couvrant de G minimisant le nombre de nœuds de branchement.

Un nœud de branchement est un nœud de degré 3 ou plus.

Exemple



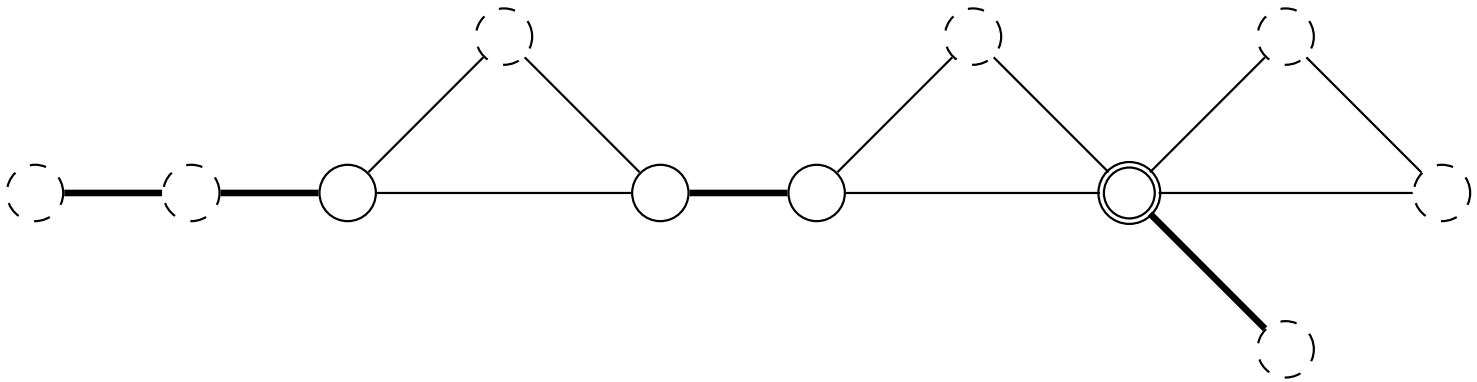
Exemple



Avant toute chose

Propriété

Certaines arêtes sont nécessairement dans un arbre couvrant : les isthmes. Selectionnez les.
Nécessairement, certains nœuds sont nœuds de branchement et, nécessairement, d'autres ne sont pas nœuds de branchement. Marquez les. Servez vous de ces informations dans vos programmes.



Définition du projet

Sujet

Le projet est en deux parties :

- Programmation linéaire
  - Modéliser le problème avec au moins deux programmes linéaires en variable mixte et les implanter
  - Étudier et implanter certaines inégalités valides
- Heuristique
  - Définir une heuristique, quelle qu'elle soit, et l'implanter.
  - Convraincre de l'utilité de l'heuristique.

Rendu

Le rendu est en 4 parties

- un rapport nommé `rapport.pdf`
- Vos codes de la partie 1 dans une archive `pl.zip` ou `pl.tar.gz`
- Vos codes de la partie 2 dans une archive `heuristic.zip` ou `heuristic.tar.gz`
- Tout le reste (annexe, fichiers de test, ...) dans une archive `other.zip` ou `other.tar.gz`

Rapport

Contenu

Le rapport doit relater synthétiquement

- le programme et les résultats de la partie 1
- l'algorithme de la partie 2, ainsi que ses avantages et inconvénients.

Rajouter une petite introduction et une conclusion.

Pensez à respecter les consignes suivantes

- Le rapport doit faire moins de 15 pages, images comprises. Restez **synthétiques**.
- Le rapport doit se suffire à lui même, cependant, si des documents trop volumineux peuvent aider à la compréhension (des tableaux de résultats, des images de graphes de grosse taille, ...) mettez les dans le dossier `other.zip`

Sujet

Travail attendu :

- Implanter au moins deux programmes linéaires différents en variables mixtes résolvant (min-ACMB)
- Définir des inégalités valides pour les deux programmes
- Tester

Consignes

Écrivez votre ou vos modèles avec au choix l'un des formats suivants

- GNU MathProg avec le solveur GLPK
- OPL ou AMPL avec le solveur CPLEX,
- Python avec le solveur SCIP.

Avec GLPK, vous devrez probablement prétraiter les entrées et vous devez tester les inégalités valides avec un langage classique : C, Python ou Java.
Avec OPL ou AMPL, vous pouvez le faire directement dans le même fichier que le modèle, mais vous pouvez aussi le faire avec un langage séparé.

Idées pour un programme pour (min-ACMB)

Propriété classique d'un arbre couvrant

Un arbre à n nœud vérifie les propriété suivantes :

- Un arbre couvrant est connexe
- Un arbre couvrant contient $n-1$ arêtes
- Un arbre couvrant n'a pas de cycle

Définir une variable binaire x_{ij} pour chaque arête déterminant son appartenance à l'arbre. Il existe 3 manières classiques de forcer les x_{ij} à respecter ces contraintes.

Méthode 1 pour un programme pour (min-ACMB)

Arbre couvrant : Propriété de coupes

Pour toute partition $U_1 \uplus U_2 = V$, il existe au moins une arête dans l'arbre reliant U_1 à U_2

Ecrire, pour chaque U_1, U_2 une contrainte concernant les x_{ij}.
Remarque : un nombre exponentiel de contraintes

Méthode 2 pour un programme pour (min-ACMB)

Arbre couvrant : Propriété de chemin

Soit r un nœud arbitrairement choisi dans V, il existe dans l'arbre un chemin de r vers tout autre nœud.

Dédoubler chaque arête en deux arcs opposés. Utiliser des contraintes de flot ou de multiflot.

Méthode 3 pour un programme pour (min-ACMB)

Arbre couvrant : Propriété de profondeur

Soit r un nœud arbitrairement choisi dans V, tout nœud a une profondeur si on enracine l'arbre en r sous forme d'une arborescence. Si u est de profondeur p_u et que l'arc (u, v) est dans l'arborescence, alors v est de profondeur $p_v = p_u + 1$.

Dédoubler chaque arête en deux arcs opposés. Utiliser une formulation type MTZ.

Propriété des nœuds de branchements

Un nœud u est un nœud de branchement s'il a 3 voisins ou plus dans l'arbre : on a sélectionné 3 arêtes ou plus incidentes à u.

Définir une variable discrète y_u pour chaque nœud u déterminant s'il est nœud de branchement ou non. Ajouter des contraintes pour relier la valeur des variables x_{ij} et des variables y_u.
Indice : jouer sur le fait que y_u est une variable discrètre pour créer des palliers.

Trouver des inégalités valides

Servez vous d'inégalités valides qu'on vous a déjà montrées (ou qui vous seront montrées) pour d'autres problèmes de réseau ou de transport.

Vous pouvez notamment vous intéresser à des inégalités valides permettant d'améliorer la recherche de nœuds de branchements.

- Décrivez la famille d'inégalités que vous utilisez.
- Proposez un algorithme de séparation.
- Testez la famille pour voir si elle améliore la relaxation continue et le temps de calcul

Sujet

Produire une heuristique pour (min-ACMB) et convraincre qu'elle est intéressante.
Vous êtes libre du langage de programmation ou de la technique algorithmique choisie (combinatoire pure, programmation mathématique, branch and bound, issu de la nature, ...).

Votre heuristique doit être assez rapide pour donner une réponse pour des instances d'une centaine de nœuds en moins de 10 minutes.

Consignes

Votre correcteur travaille sous UNIX, sur une distribution Debian stretch.
Vous devez rendre, au choix,

- un code codé dans un langage usuel, gratuit et simple à installer (C, C++, Python(2,3), Php, Java, Ruby, (O)CAML, Prolog ...) ou pour l'un des solveurs de programmation mathématique suivants GLPK, LPSOLVE ou SCIP (avec par exemple Mathprog, OPL ou les bibliothèques de langages de programmation pouvant utiliser ces solveurs)
- un fichier exécutable binaire pour la distribution debian pour tout logiciel utilisant un langage ou un outil non usuel ou payant (par exemple JuPyter, Matlab ou CPLEX).

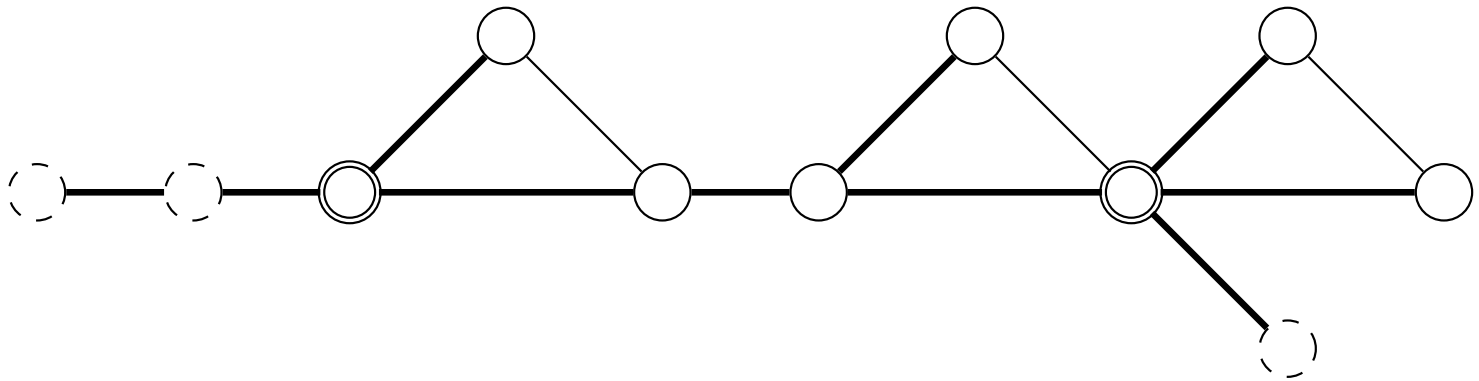
Dans tous les cas, un petit **README** placé dans le même dossier que le code peut être utile.

Quelques exemples d'heuristiques

Algorithme 1 : local

Trouver et renvoyer un arbre couvrant quelconque puis tenter une amélioration locale en échangeant des arêtes sélectionnées et non sélectionnées.

Suite possible : métaheuristiques

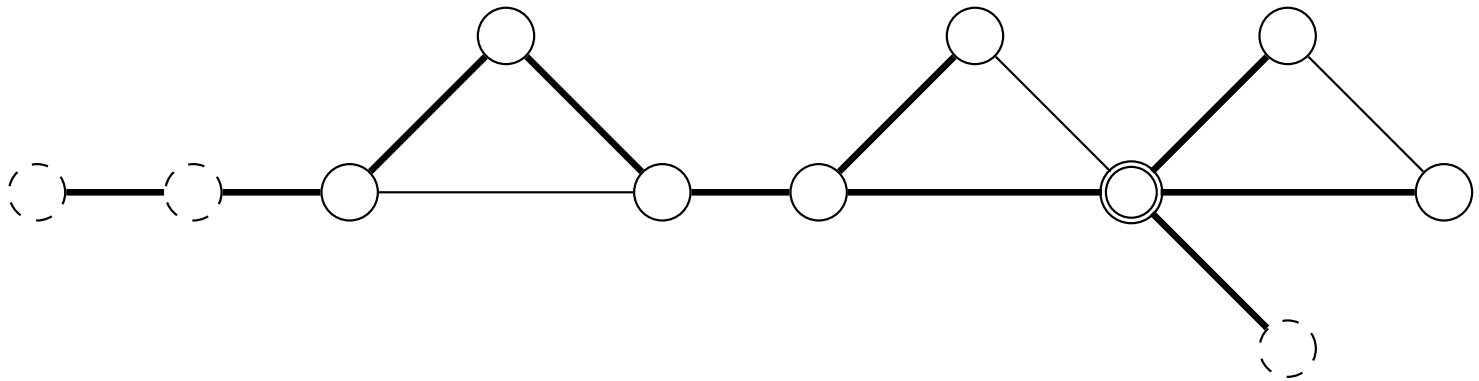


Quelques exemples d'heuristiques

Algorithme 1 : local

Trouver et renvoyer un arbre couvrant quelconque puis tenter une amélioration locale en échangeant des arêtes sélectionnées et non sélectionnées.

Suite possible : métaheuristiques



Quelques exemples d'heuristiques

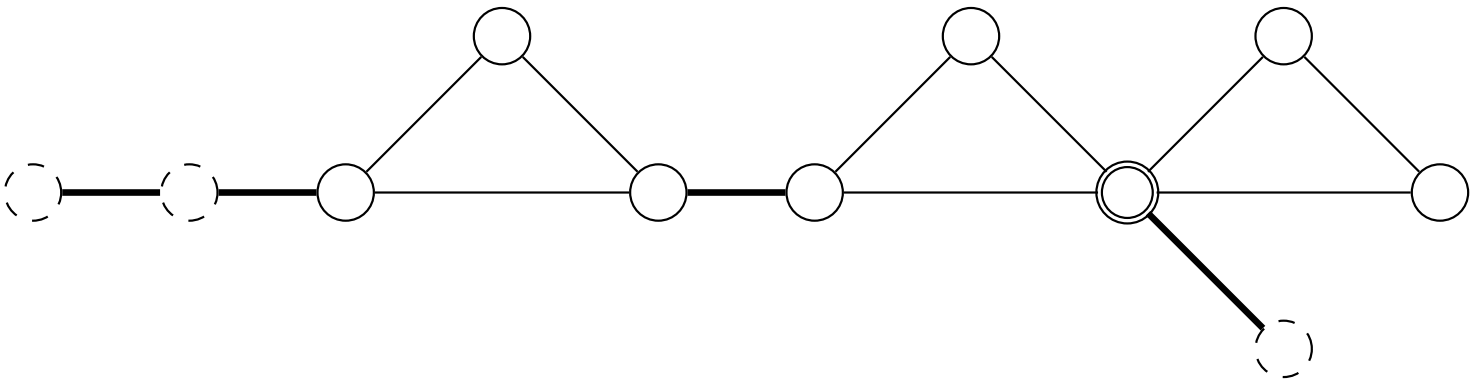
Algorithme 2 : glouton

Soit E' l'ensemble des arêtes non sélectionnées. Au départ, $E' = E$

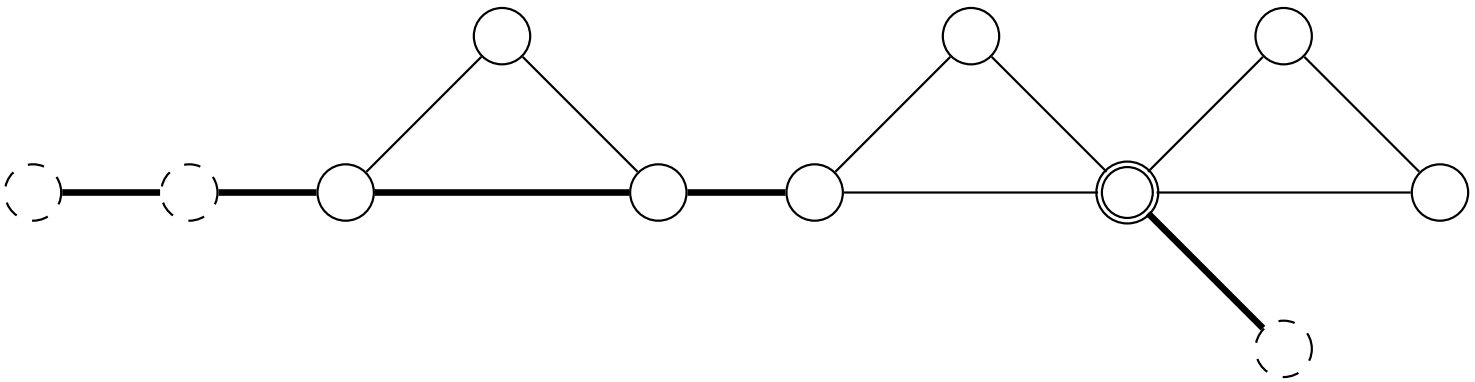
- Supprimer de E' toutes les arêtes qui créeraient un cycle si on les ajoutait à l'arbre.
- Si elle existe, choisir une arête de E/E' qui n'augmente pas le nombre de nœuds de branchement. Sinon, si possible, choisir une arête qui ne crée qu'un seul nœud de branchement. Sinon choisir une arête arbitrairement.
- Ajouter l'arête à l'ensemble choisi. Recommencer jusqu'à ce que tous les nœuds soient reliés.

Suite possible : pondérer intelligemment les nœuds, tenter de sélectionner plus d'une arête possible, branch and bound, ...

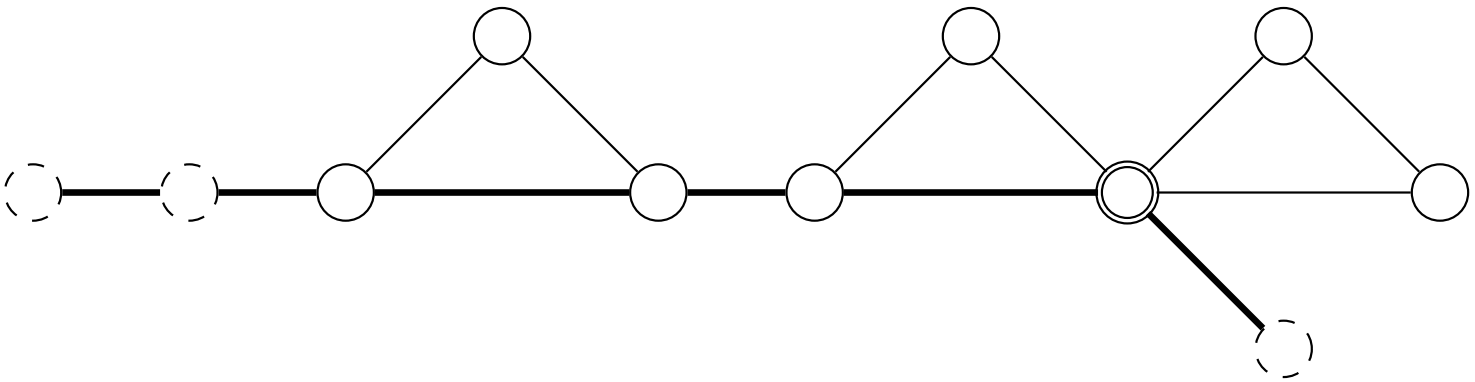
Quelques exemples d'heuristiques



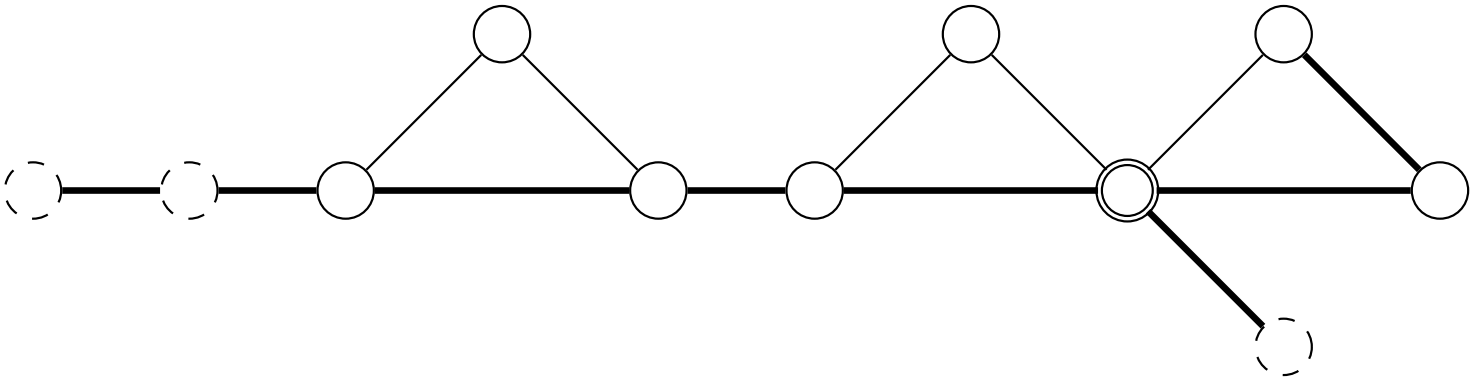
Quelques exemples d'heuristiques



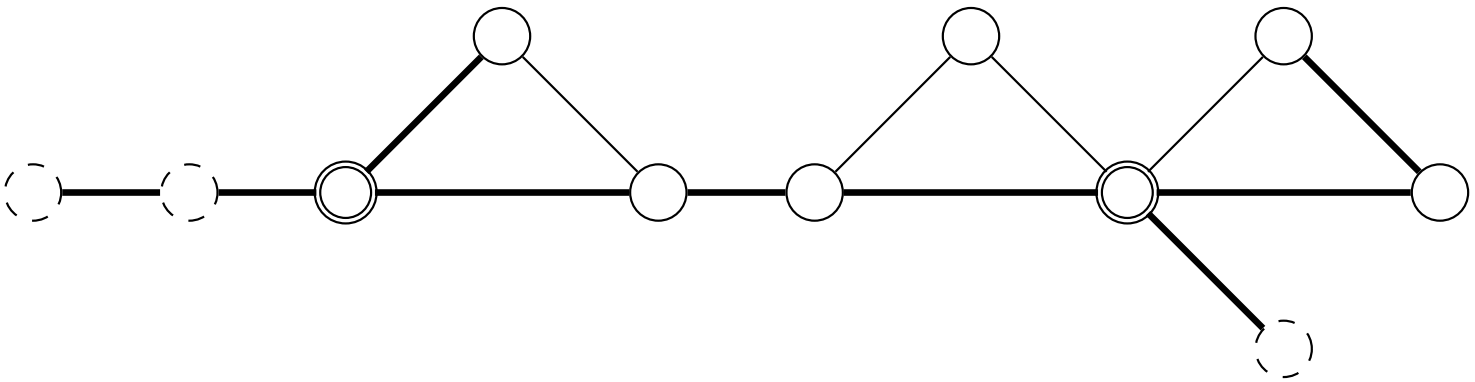
Quelques exemples d'heuristiques



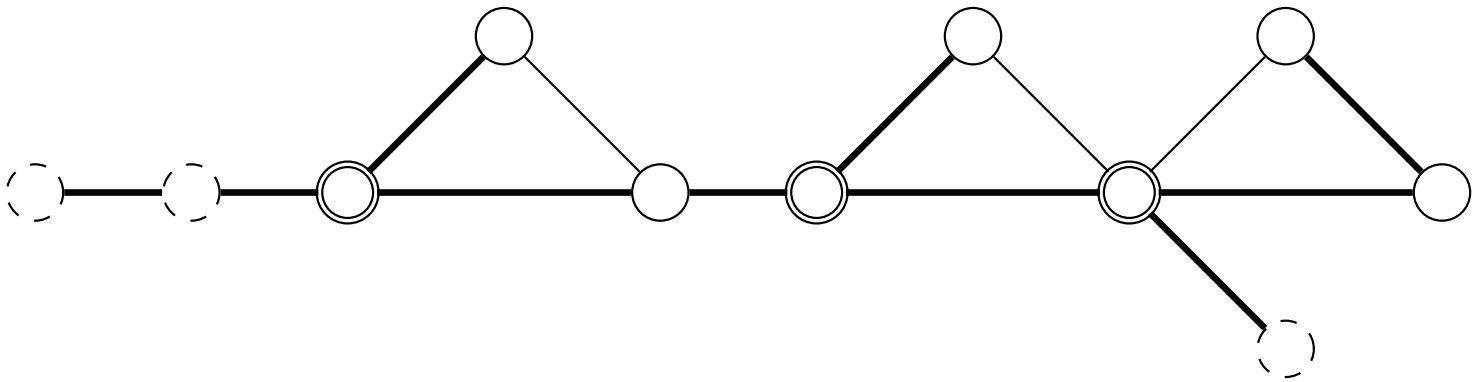
Quelques exemples d'heuristiques



Quelques exemples d'heuristiques



Quelques exemples d'heuristiques



Quelques exemples d'heuristiques

CAP ?

Soit V' les nœuds qui ne sont ni trivialement des nœuds de branchement ni trivialement des nœuds qui ne sont pas de branchement dans un arbre couvrant.
Il existe un algorithme XP en $|V'|$ et même un algorithme FPT en $|V'|$.

Tests unitaires : Description des entrées

10 entrées vous sont données sous un fichier texte pour faire des **tests unitaires**.

Pour un graphe $G = (V, E)$ de n sommets,

- les n première lignes vous donnent la matrice d'adjacence. Chacune contient n binaires séparés pas des espaces, le i^{e} entier de la j^{e} ligne vaut 1 si et seulement si le nœud i est relié au nœud j.
- la $n + 1^{\text{e}}$ ligne vous donne la valeur d'une solution optimale.

Tous les graphes sont simples. Leurs taille varie de 5 à 100 nœuds. Vous pouvez les trouver à l'adresse suivante :
`www.ensiie.fr/~faye/RORT/minacmb`

Évaluer les performances d'un algorithme

Il faut se poser les bonnes questions

- quelles sont les performances théoriques de mon algorithme ?
- sur quelles catégories d'instances vais-je tester mes algorithmes ?
- sur combien d'instances vais-je tester mes algorithmes ?
- quelles sont les mesures à faire ?
- sur quelle machine les tests ont-il été faits ? Seront-il reproductibles facilement ?

Performances théoriques

Mesure des performances théoriques

Généralement, on regarde

- la complexité en temps
- la complexité en espace
- le nombre de variables ou de contraintes pour un programme mathématique
- le ou les pires cas
- le ou les cas où l'algorithme est optimal
- le ou les cas où l'algorithme est "presque" optimal
- la complexité algorithmique, paramétrée ou l'approximabilité du problème
- ...

Catégories d'instances

Un algorithme ne fonctionne pas de la même manière sur toutes les instances. L'évaluation peut permettre de déterminer pour quelles instances il est efficace et pour lesquelles il vaut mieux se tourner vers un autre algorithme.

Exemples

Par exemple

- un graphe avec peu d'arêtes
- un graphe avec peu de cycles
- un graphe complet ou quasi-complet
- un graphe parfait
- ...

Combien d'instances

Etant donnée l'infinité des instances, des tests sur des instances fixées ne sont pas une preuve exacte de l'utilité de votre algorithme. Il s'agit au mieux d'une preuve statistique.

- pensez à la représentativité statistique de vos instances : combien de paramètres devez vous faire varier ? les tailles croissent-elles linéairement, logarithmiquement, exponentiellement ? combient d'instances d'une même taille ?
- pensez à faire plusieurs fois les tests pour des algorithmes probabilistes
- pensez aussi au temps de calcul de votre machine, avez-vous le temps de tout faire avant la date limite ?

Mesures

Mesurer les résultats

Mesures habituelles :

- Temps de calcul
- Valeur de la solution
- Comparaison à l'optimal si celui-ci est connu (si heuristique)
- Combien de fois l'optimal est atteint ? (si heuristique)
- Jusqu'à quelle taille l'algorithme donne une réponse en temps raisonnable ?
- Pour un programme mathématique : combien d'inégalités valides ont été ajoutées, quelle amélioration en temps ? sur la borne de la relaxation continue ?
- Pour un branch and bound et un programme mathématique : combien de branchements ont été effectués, à quelle vitesse la borne se raproche-t-elle de l'optimal ?
- ...

Présenter les résultats

Pensez à comment regrouper les instances pour faire des moyennes, des écart-types, des quartiles, ...

Faites des courbes, des tableaux synthétiques, des histogrammes, des boîtes à moustaches.

Reproductibilité

Donnez toutes les informations possibles de votre machine (entre autres le processeur, RAM, ...). Le code source existe-t-il ? est-il disponible ? Les instances testées sont-elles disponibles ?

Si aléatoire, quelles graines ont été utilisées ? Le résultat est-il bien meilleur si la graine n'est pas fixée ?