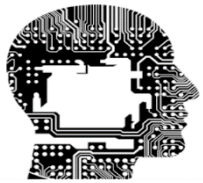
Université Constantine 2
جامعة قسنطينة 2



Foundation of Artificial Intelligence

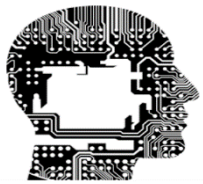
Chapitre 02 (Partie 03)

Dr. NECIBI Khaled

Faculté des nouvelles technologies

Khaled.necibi@univ-constantine2.dz

Université Constantine 2
جامعة قسنطينة 2



Foundation of Artificial Intelligence

- Résolution de Problèmes via des Stratégies de Recherche -

Dr. NECIBI Khaled

Faculté des nouvelles technologies

Khaled.necibi@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	IFA	Master 01	SDIA

Objectif du cours

- Comprendre les stratégies de base de la recherche informée
- Maîtriser les stratégies de recherche informée
- Maîtriser les algorithmes de la recherche informée
- La recherche basée sur les heuristiques
- Évaluer les performances d'un algorithme de recherche informée

Stratégies de recherche informée

- Recherche informée
  - Rappel : une stratégie est définie en choisissant un ordre dans lequel les états (les nœuds) sont développés
  - La recherche informée utilise des connaissances spécifiques au problèmes (en plus de sa définition) afin de guider le processus de recherche
  - Les stratégies de recherche informée peuvent trouver des solutions plus efficaces (contrairement à celles proposées par les stratégies de recherche non informée)

Pourquoi la recherche informée ?

- Recherche informée
  - La recherche sans utilisation de connaissances est aveugle
    - Éventuellement une recherche complète dans tout l'espace d'état pour trouver une solution
  - Elle peut conduire à une complexité ingérable
  - L'utilisation des connaissances permet de guider la recherche
    - Exemple : l'exploration d'un labyrinthe où on a des indications pour se déplacer
  - Ces connaissances représentent une information heuristique
    - L'information heuristique aide à trouver la solution d'une façon meilleure
    - L'information heuristique améliore les performances des algorithmes de recherche

Pourquoi la recherche informée ?

- Trois classes de recherches informées à étudier :
  - Recherche du meilleur d'abord
    - Recherche du meilleur d'abord Best-First Search
    - Recherche gourmande du meilleur d'abord Greedy Best First Search
    - La Recherche A*
  - Recherche locale
  - Recherche Stochastique

Recherche du Meilleur d'Abord

- Idée de base : Best-First Search
  - Utiliser une fonction d'évaluation *f(n)* pour chaque nœud n
  - La fonction *f(n)* permet d'estimer la distance par rapport au but
  - Où estimer le coût du chemin le plus cours afin de se rendre à l'état but
  - Le successeur du nœud avec la fonction d'évaluation la moins coûteuse est généré

Recherche du Meilleur d'Abord

- Implémentation
- Les nœuds sont ordonnés dans la frontière selon les valeurs de la fonction d'évaluation *f*
- Cas spéciaux de la stratégie BFS
  - Recherche gloutonne (Greedy Search)
  - A*

BFS : Description Formelle

Meilleur premier d'abord

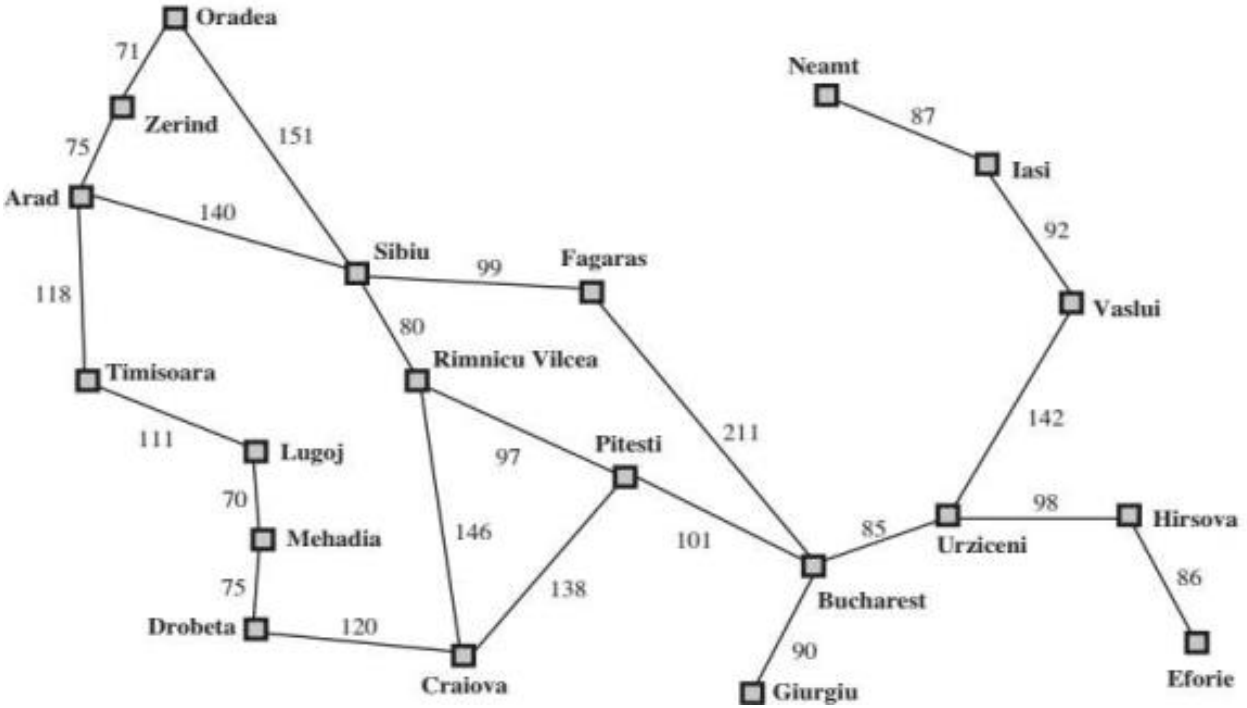
```
Fonction best-first (problème, frontière, f) returns solution ou échec {

        // f : fonction d'évaluation

        frontière ← Insert(Make-Node(initial-state[problème], Null, Null,
                                     profondeur, coût_de_chemin), frontière);
        répéter
                if Empty?(frontière) then returns échec;

                node ← Remove-First(frontière);

                if Goal-Test[problème] appliqué à State[node] réussit then

                returns solution(node);

                frontière ← Insert-all(Expand(node, problème), frontière);

                //Insert-all : le tri de la frontière se fait en ordre
                //croissant des valeur de la fonction f
        Fin
}
```

Meilleur Premier D'abord : Recherche Gloutonne

- Dans le cas de la stratégie de recherche gloutonne, la fonction d'évaluation est une fonction heuristique
  - $f(n) = h(n)$
  - Exemple : Problème de touriste dans la Romanie voulant se rendre à Bucharest
  - On peut utiliser la distance euclidienne, entre le nœud *n* et Bucharest
  - $H_{SLD}(n)$: la distance la plus courte entre *n* et Bucharest (Straight Line Distance)
  - La recherche gloutonne développe le nœud qui paraît *le plus proche* de l'état final
  - Différente par rapport à la recherche non informée UCS qui est plutôt guidée par le coût du chemin au nœud *n*
  - Fonctions Heuristiques : les connaissances aux problèmes sont utilisées pour guider la recherche

Meilleur Premier D'abord : Recherche Gloutonne

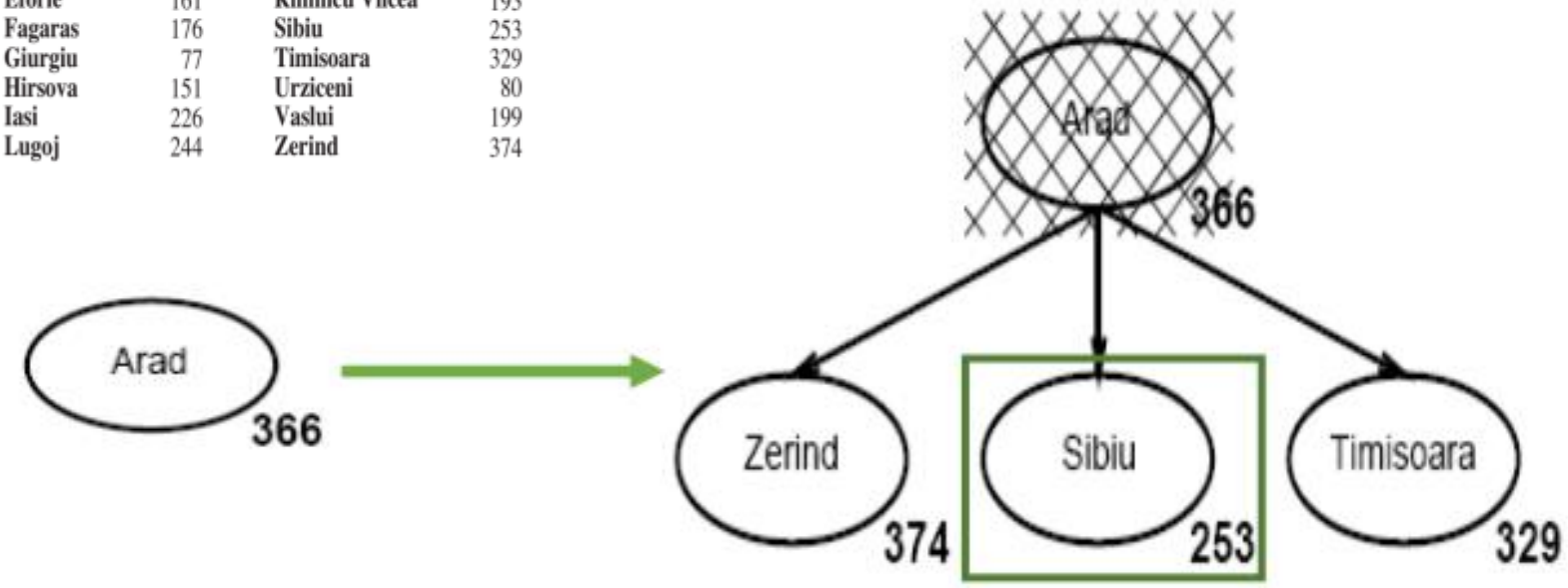
- Exemple : H_{SLD}(Dans(Arad)) = 366
- Distance de cités par rapport à Bucharest



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

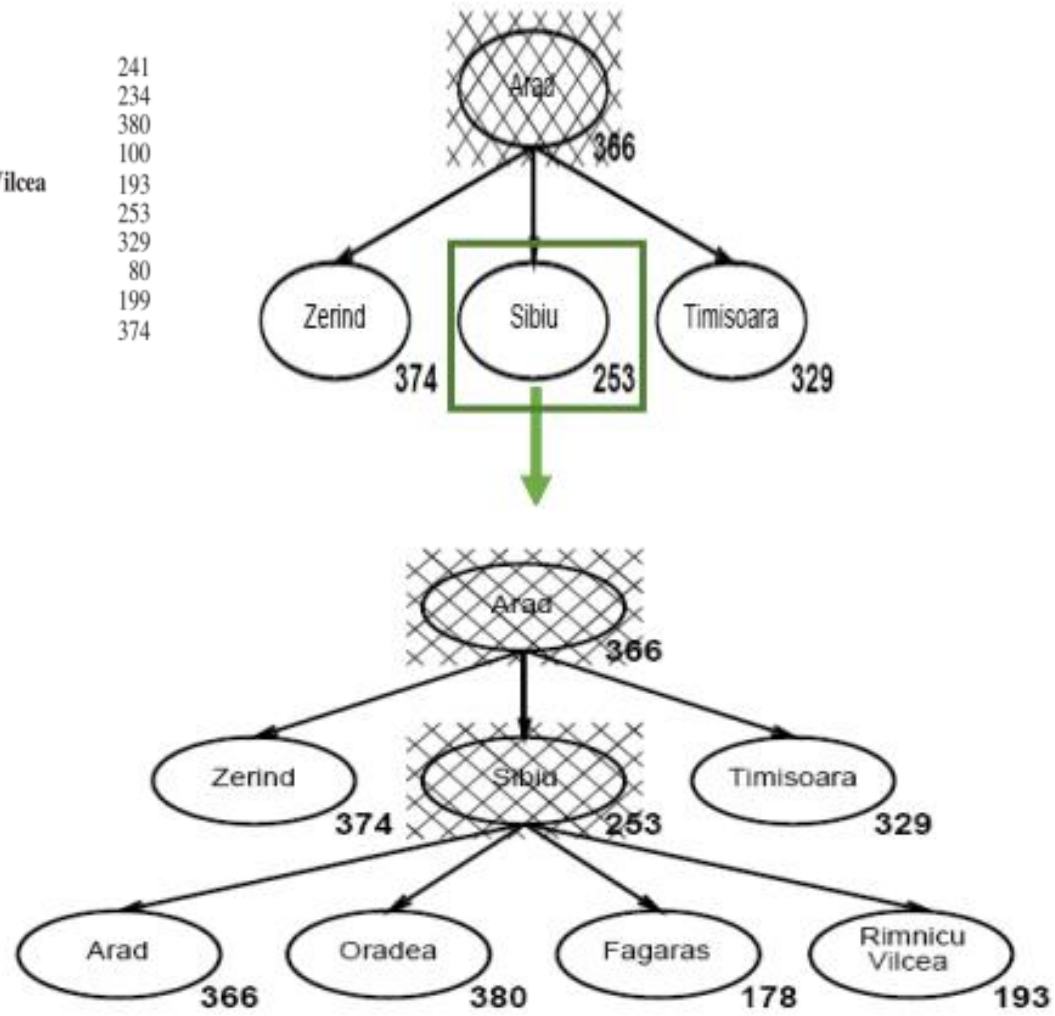
Meilleur Premier D'abord : Recherche Gloutonne

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



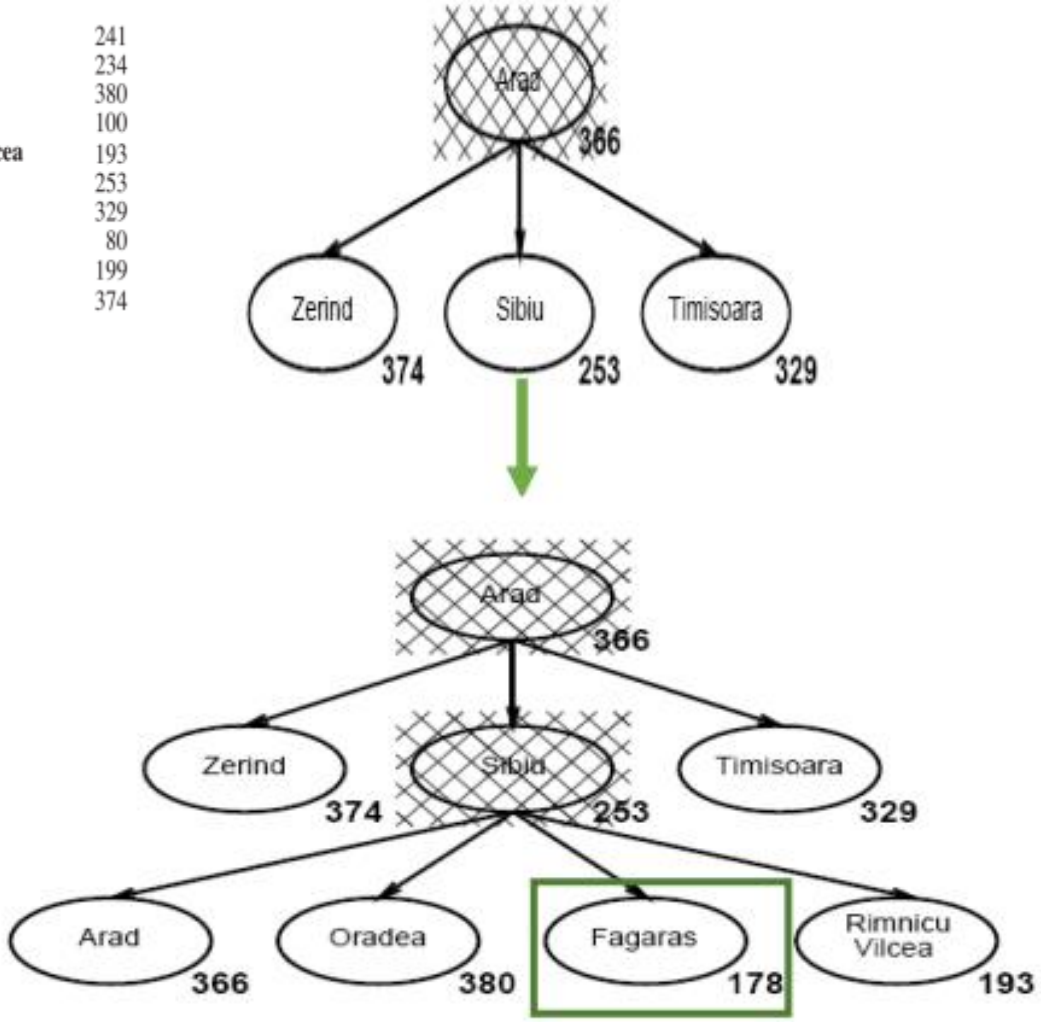
Meilleur Premier D'abord : Recherche Gloutonne

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



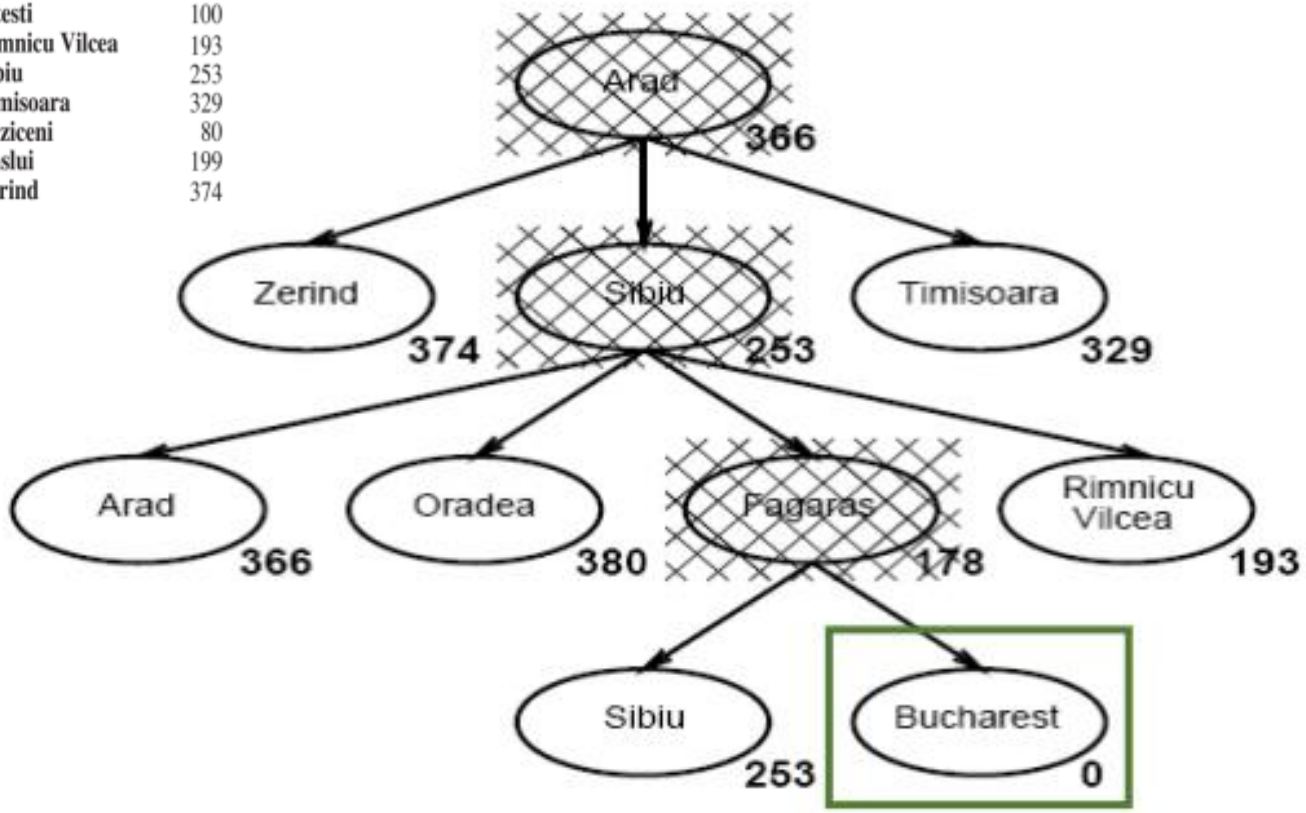
Meilleur Premier D'abord : Recherche Gloutonne

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Meilleur Premier D'abord : Recherche Gloutonne

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Recherche du Meilleur d'Abord

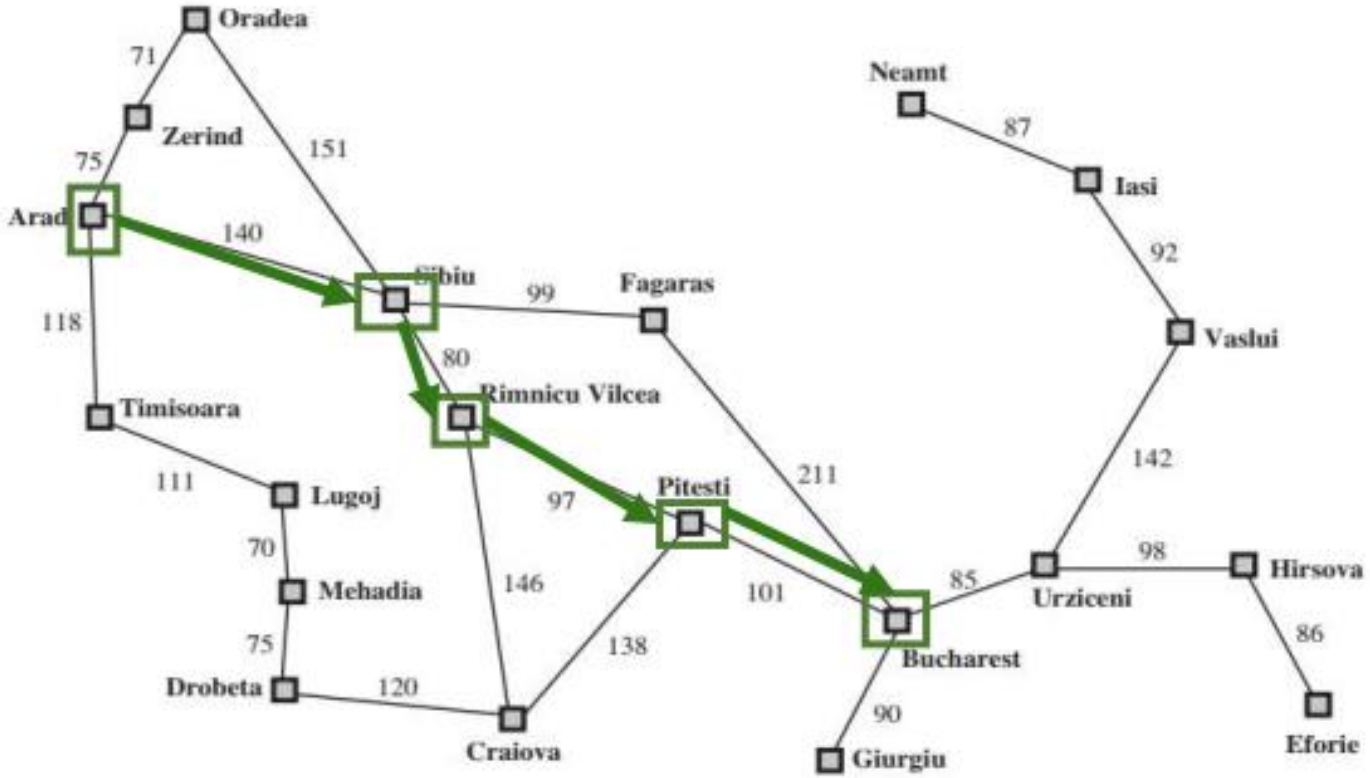
- Évaluation des performances de BFS
  - Complétude : incomplet (peut resté bloqué dans des boucles)
    - Exemple : Arad → Zerind → Arad
    - Complet si on rajoute un test pour éviter les état répétés
  - Optimalité : cet algorithme n'est pas optimal
  - Complexité temporelle : dans le pire des cas, l'algorithme génère $O(b^m)$
  - Complexité spatiale : garde tout les nœuds en mémoire
  - Minimiser h(n) est sensible à l'initialisation
    - Exemple : aller de Lasi à Fagaras; selon la fonction heuristique h(n) Neamet est choisie, chose qui va conduire vers un point mort

Meilleur Premier D'abord : A*

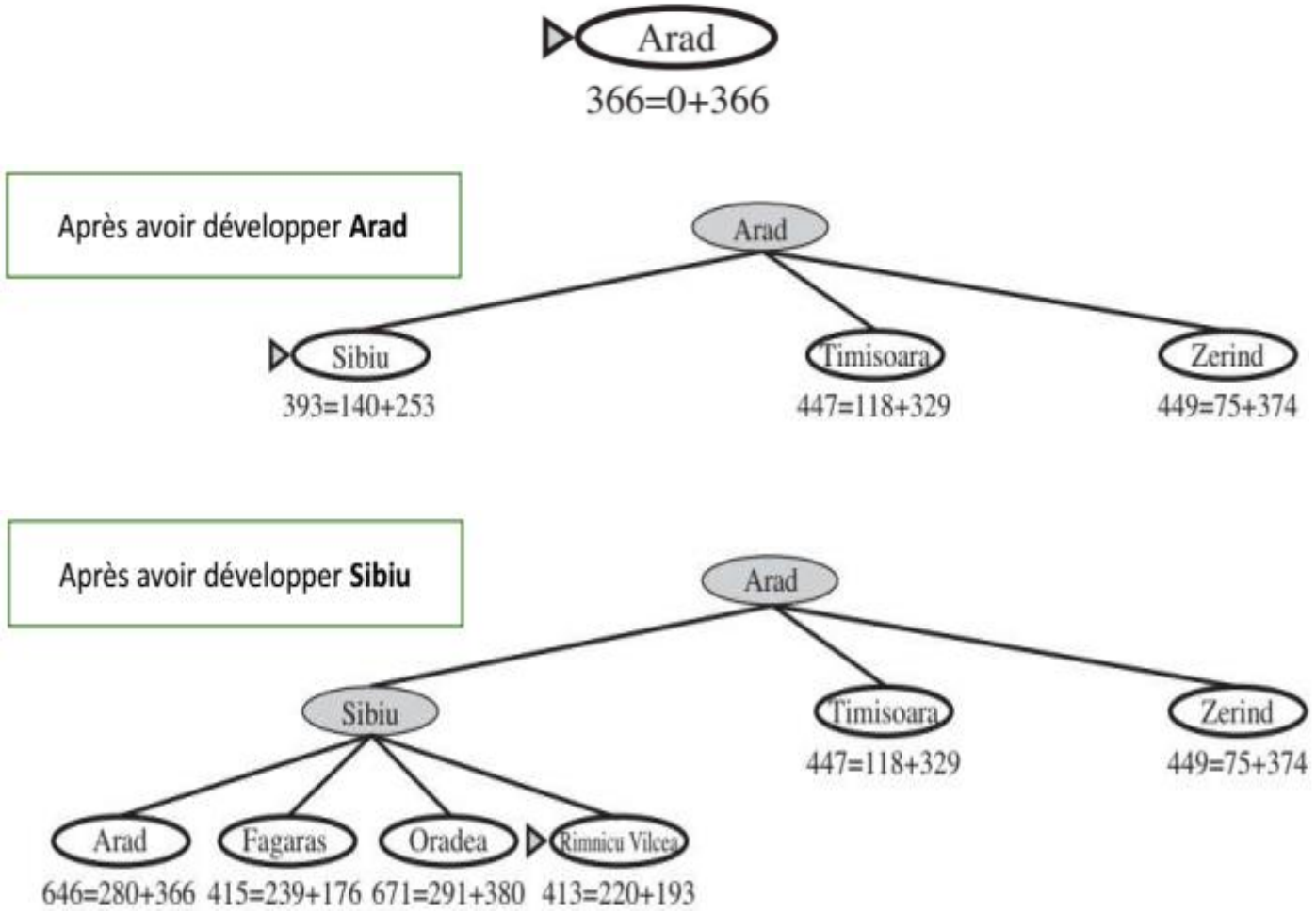
- A* est l'algorithme le plus connu de la classe meilleur premier d'abord
  - L'idée de base : éviter de développer les chemins qui sont coûteux
  - Fonction d'évaluation :
  - $f(n) = g(n) + h(n)$
  - $g(n)$: coût du chemin de la racine jusqu'au nœud n (comme utilisé dans USC)
  - $h(n)$: estimation du coût du chemin, allant du nœud n jusqu'au nœud but (qui représente l'état final)
  - $f(n)$: estimation du coût total du chemin, allant du nœud racine, jusqu'au nœud but, tout en passant par le nœud n

Meilleur Premier D'abord : A*

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

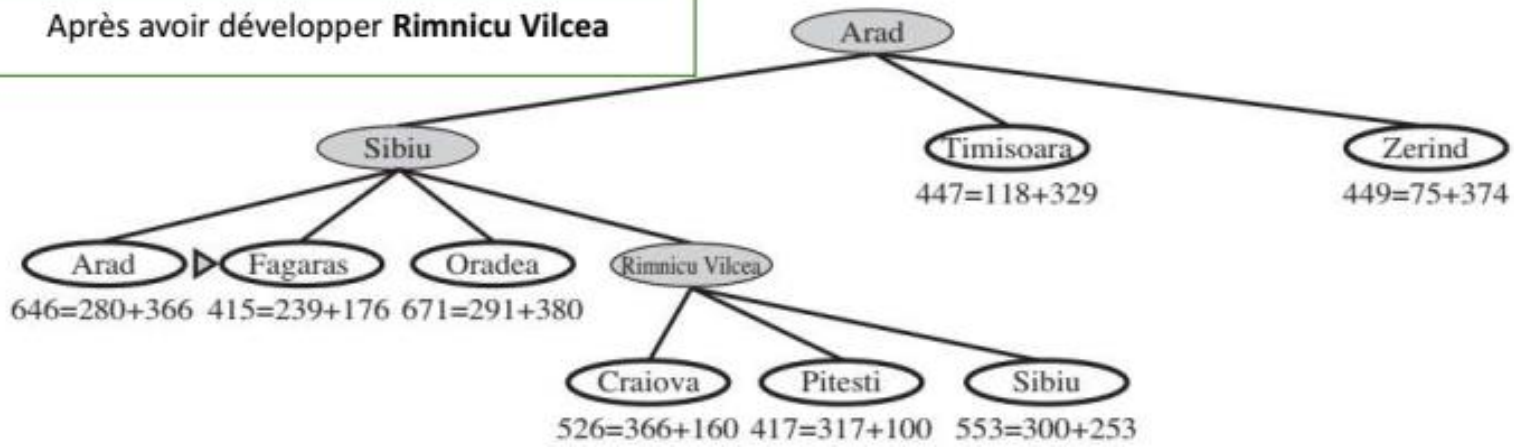


Meilleur Premier D'abord : A*

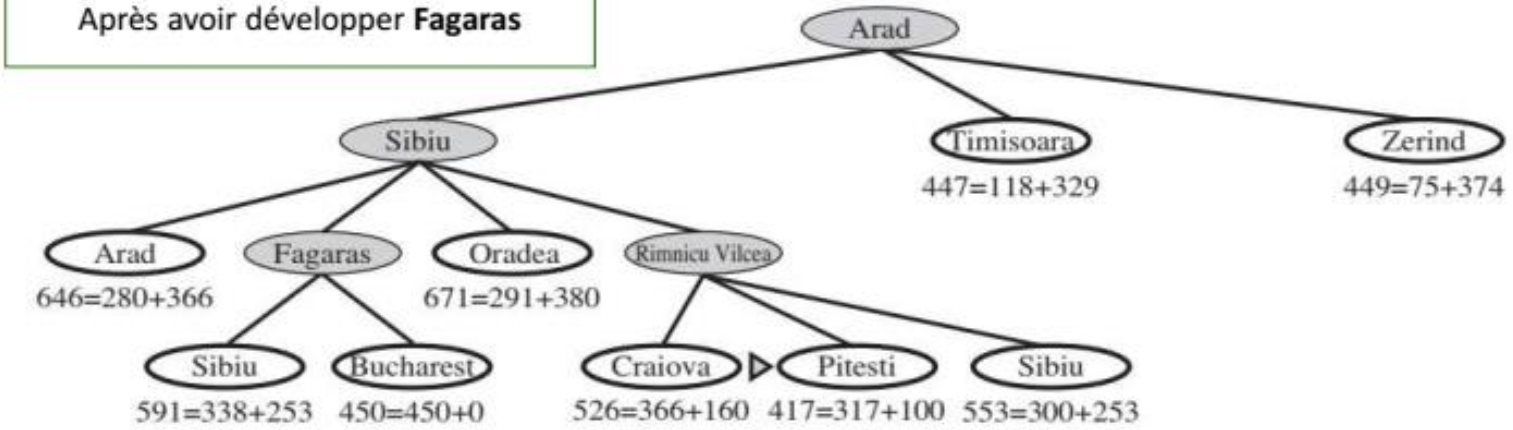


Meilleur Premier D'abord : A*

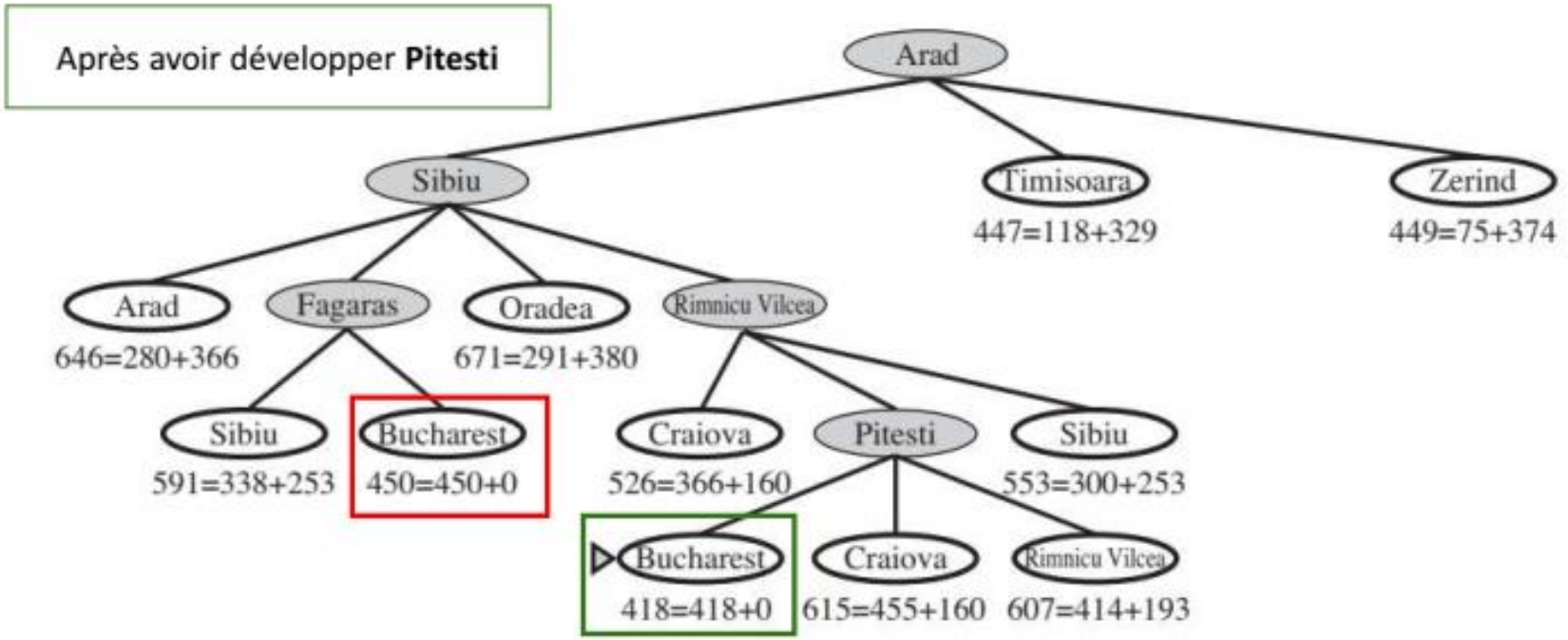
Après avoir développer **Rimnicu Vilcea**



Après avoir développer **Fagaras**



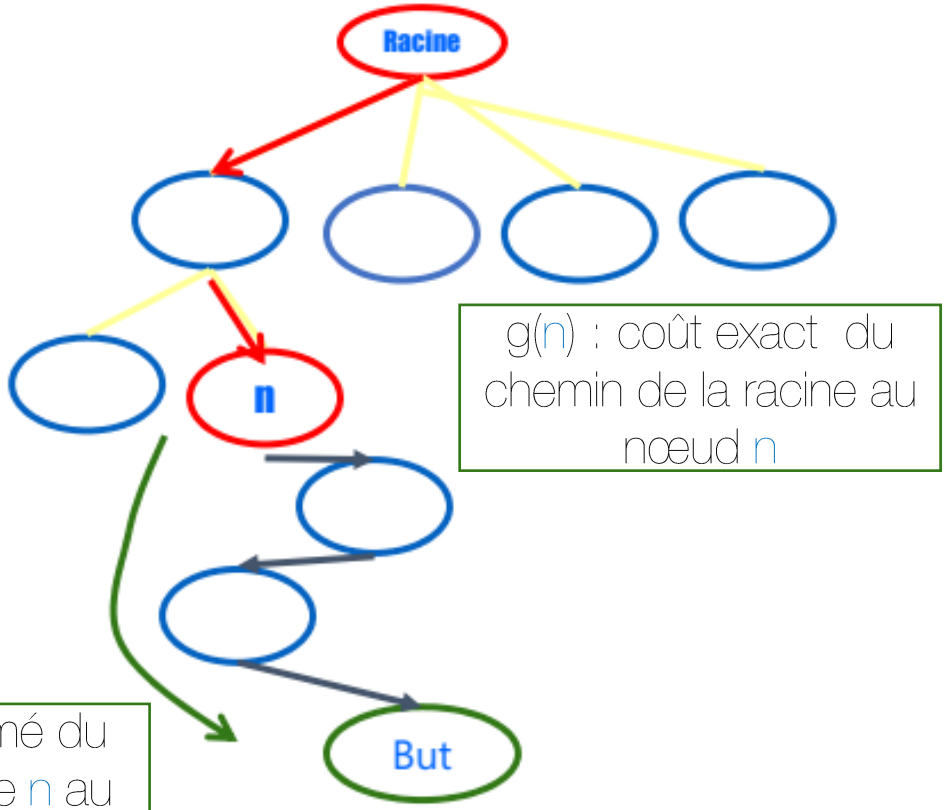
Meilleur Premier D'abord : A*



Meilleur Premier D'abord : A*

- Définition : une heuristique h(n) est admissible si elle ne surestime pas le coût réel nécessaire pour atteindre le nœud but
  - h(n) < = h*(n) ou h*(n) est le coût réel pour atteindre le but à partir de n
  - Exemple : h_{sld} la distance euclidienne ne fournit jamais de surestimation (c'est la plus petite distance entre deux points de l'espace)
  - Conséquence : si h(n) est une heuristique admissible, alors f(n) ne surestime jamais le coût allant au nœud but à travers n nœuds : Pourquoi ?
  - C'est vrai car g(n) donne le coût exact allant de la racine à n

Meilleur Premier D'abord : A*



g(n) : coût exact du chemin de la racine au nœud n

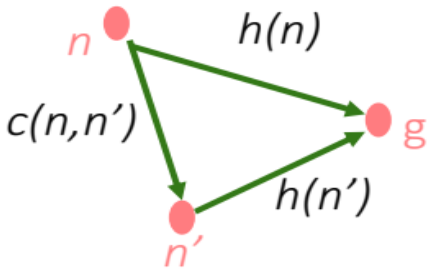
h(n) : coût estimé du chemin allant de n au But

Meilleur Premier D'abord : A*

- Évaluation des performances de A*
  - Complétude : oui
  - Optimalité : oui
  - Complexité temporelle : exponentielle; selon la largeur de la solution
  - Complexité spatiale : exponentielle; garde tous les nœuds en mémoire

Meilleur premier D'abord : A*

- Consistance (Monotonie) : une heuristique est dite consistante si pour tout nœud n et n' tel que n' est le successeur de n, on :
  - $H(n) \leq c(n, n') + h(n')$ où $c(n, n')$ est le coût minimum entre n et n'
  - La consistance est une forme d'inégalité triangulaire :



  - Toute heuristique consistante est admissible (l'inverse n'est pas vrai)
  - Quand une heuristique est admissible, les valeurs de la fonction d'évaluation f(n) le long d'un chemin sont croissantes

Les algorithmes de recherche locale

- Les algorithmes précédents doivent maintenir l'information à propos de l'état courant, la frontière et le chemin qui mène à l'état final
  - Dans certains problèmes, on ne s'intéresse pas au chemin qui mène vers l'état final, mais à l'état final lui-même et qui n'est pas connu
  - La recherche dans ce cas est dite locale
  - Remarque : l'état lui-même est la solution et non pas le chemin
  - Idée : modifier l'état en l'améliorant au fur et à mesure
  - Espace d'état : ensemble de configuration possible des états
  - Besoin de définir une fonction d'utilité d'état

Les algorithmes de recherche locale

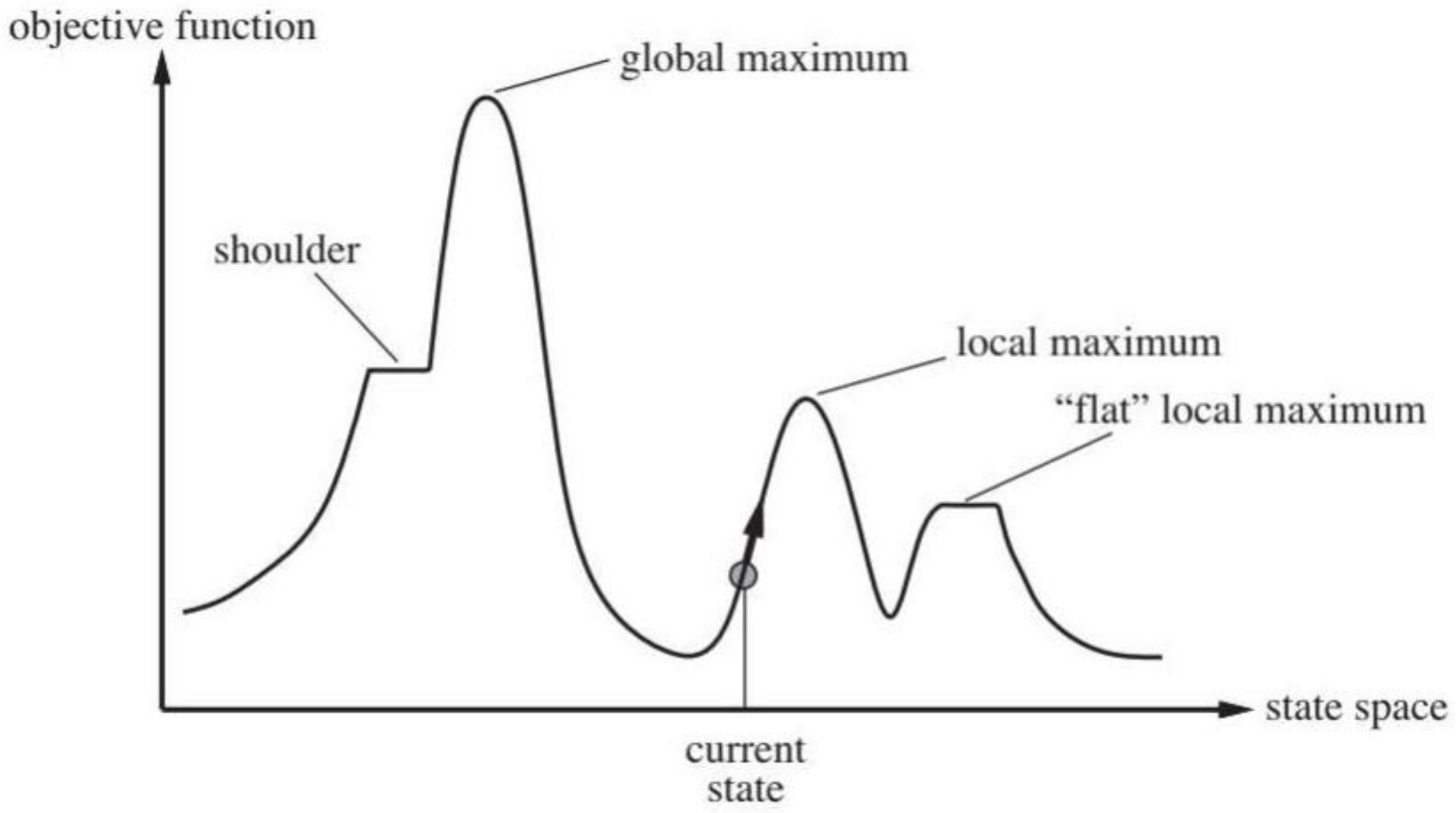
- Idée de base
  - Les algorithmes de recherche local opèrent sur un seul état (état courant)
  - Essayent par la suite de passer à un nœud voisin
  - Conséquence : le chemin à la solution n'est pas maintenu et la recherche est dite locale
- Avantage
  - Ces algorithmes utilisent peu de mémoire
  - Applicable à l'exploration d'espaces de recherche plus large
  - Permettent de trouver des solutions raisonnables

Les algorithmes de recherche locale

- L'algorithme Ascension du gradient (Greedy Local Search où Steepest Ascent)
  - Utilise une boucle qui permet de se déplacer à une solution voisine meilleur dans le sens de la montée
  - Il termine la recherche quand un pic ou un optimum est atteint
  - Ou bien il termine quand aucun voisin n'est meilleur

Les algorithmes de recherche locale

- L'algorithme Ascension du gradient (Greedy Local Search où Steepest Ascent)



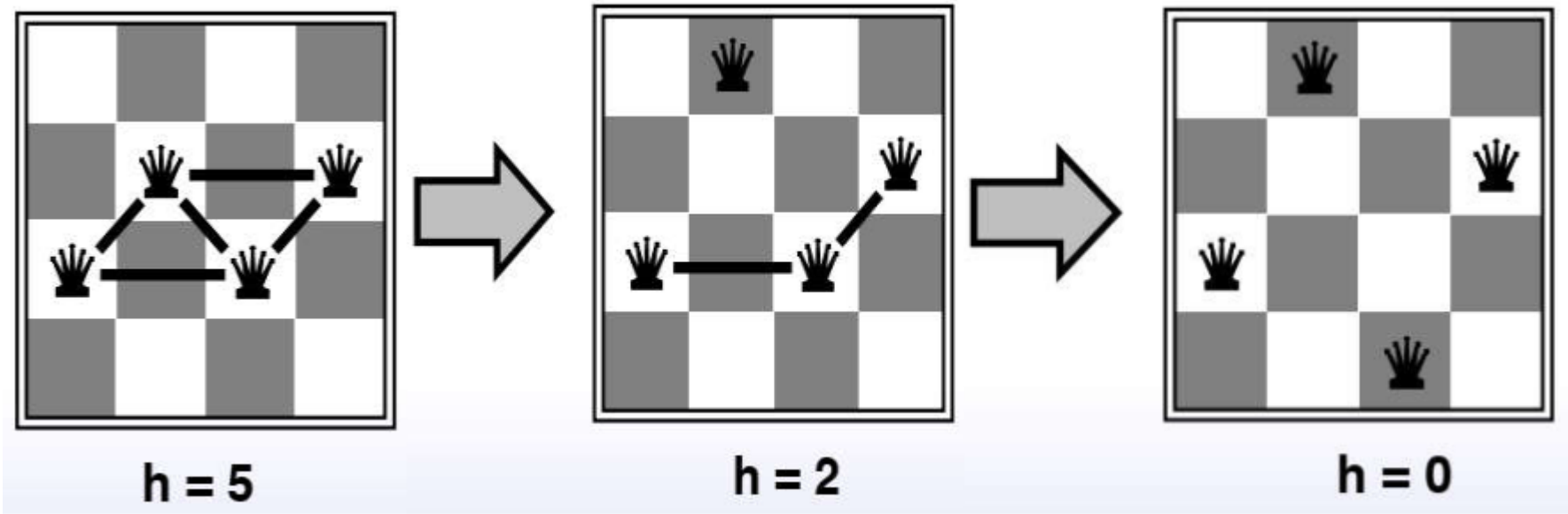
Les algorithmes de recherche locale

- L'algorithme Ascension du gradient (Greedy Local Search où Steepest Ascent)

Steepest ascent (ascension du gradient)

```
Fonction Hill-Climbing (problème) returns état local maximum {

        inputs : problem;
        local variables : current (node);
                          neighbor (node);

        current ← Make-Node(initial-state[problème]);

        répéter
                neighbor ← highest-valued successor of current;

                if Value[neighbor] ≤ Value[current] then returns State[current]

                        current ← neighbor ;

        Fin
}
```

- L'algorithme Ascension du gradient (Greedy Local Search ou Steepest Ascent)
  - Exemple : placer n reines sur un plateau de taille n x n sans que deux reines se trouvent sur la même ligne, colonne ou diagonal
  - À chaque fois, il faut déplacer une reine pour réduire le nombre de conflits



Les algorithmes de recherche locale

- Local Beam Search (LBS)
  - Dans cet algorithme, on fait évoluer K solutions au lieu d'une seule
  - Il commence par générer aléatoirement K solutions (ou états initiaux)
  - À chaque étape, tous les successeurs de tous les nœuds sont générés
  - Si l'état final est atteint l'algorithme s'arrête, sinon il sélectionne les K meilleurs successeurs parmi tous les successeurs
  - Ce processus sera répété jusqu'à ce que l'état final soit atteint

Recherche Stochastique : Recuit simulé

- Qu'est-ce que le recuit ?
  - Processus physique utilisé pour faire chauffer les métaux ou les verres à de très hautes températures
  - Ensuite, les refroidir graduellement permettant aux matériaux d'atteindre un état à basse énergie
- Idée de base
  - Le recuit simulé combine la recherche locale avec une marche aléatoire de manière à rendre la recherche complète et plus efficace
  - Algorithme utilisé au début des années 80 dans le domaine de la fabrication des composants électroniques

Recherche Stochastique : Recuit simulé

Simulated annealing

```
Fonction Simulated-annealing (problème, schedule) returns solution état {

        inputs : problème;
                 schedule; //mapping from time to temperature
        local variables : current (nœud);
                          next (nœud)
                          neighbor (nœud);
                          T; temperature controling downward steps probability

        current ← Make-Node(initial-state[problème]);

        for t ← 1 to ∞ do
        T ← schedule[t]
         if T = 0 then return current

        répéter
                 next ← sélection aléatoire de successor de current;
                 ΔE ← Value[next] - Value[current]
                 if ΔE < 0 then current ← next ; //problème de minimisation
                 else current ← next seulement avec une probabilité e^-ΔE/kT
        Fin
}
```

Recherche Stochastique : les algorithmes génétiques

- Les Algorithmes génétiques
  - Introduits pour la première fois aux USA dans les années 70 grâce aux travaux de Jhon Holland
  - Les AGs s'inspirent des idées de la théorie de l'évolution naturelle et la génétique
- Conception d'un algorithme génétique
  - Comment un individu est-il représenté ?
  - Quelle fonction objective (fonction de fitness) faut-il utiliser?
  - Comment se fait la sélection des individus ?
  - Comment se fait la reproduction et la mutation des individus ?

Recherche Stochastique : les algorithmes génétiques

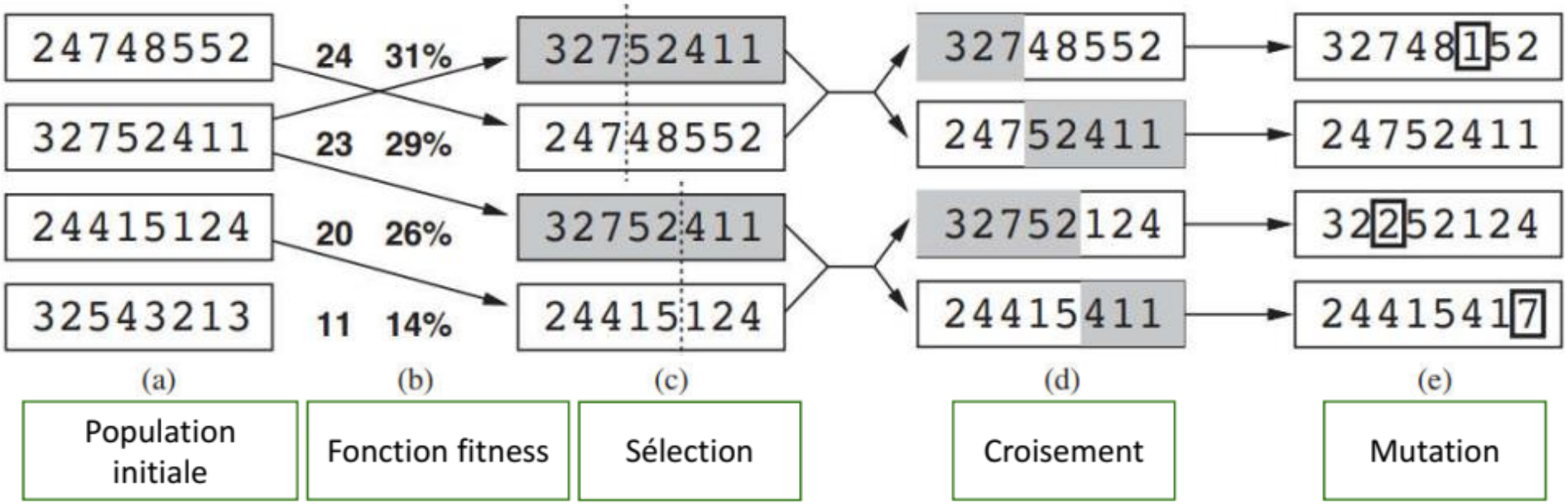
- Les algorithmes génétiques peuvent être considérés comme une variante de l'algorithme LBS
- Les successeurs sont obtenus en croisant deux parents ou en mutant un individu
- Tout comme LBS, un algorithme génétique commence avec un ensemble de solutions appelé population
- Chaque individu est représenté sous forme chromosomique
  - Une chaine est définie sur un certain alphabet
- Le codage des chromosomes se fait selon le problème traité

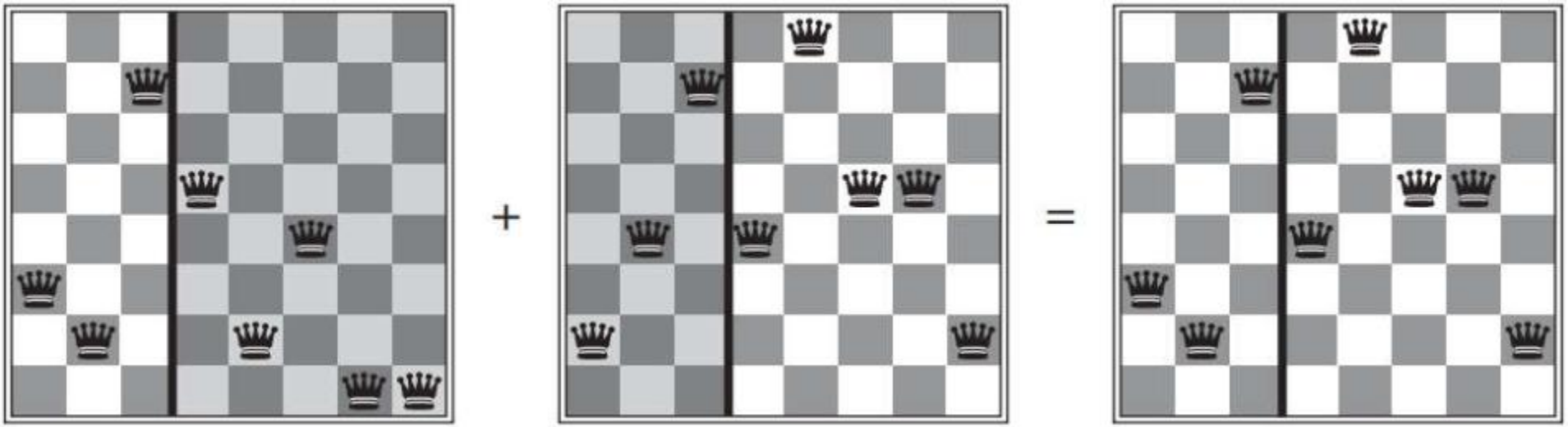
Recherche Stochastique : les algorithmes génétiques

- Chaque état correspond à une solution potentielle et admet une fonction de fitness
- La fonction de fitness donne les meilleures valeurs aux meilleurs individus
- Pour se reproduire, les individus sont sélectionnés selon :
  - La valeur de leur fitness (la fonction fitness attribuée)
  - Une stratégie de sélection
  - Un point de croisement est choisi et les parties correspondantes sont échangées
- La mutation est effectuée d'une manière aléatoire avec une certaine probabilité

- Exemple d'application des AGs
  - Le jeu de 8 reines
  - Un état de 8 reines doit spécifier la position des 8 reines dans chaque colonne des 8 carrés
  - L'état peut être représenté par 8 numéros entre 1 et 8

Recherche Stochastique : les algorithmes génétiques





Genitic Algorithm

```
Fonction Genetic-Algorithm (population, fitness-fn) returns un individu {

        inputs : population; //ensemble d'individus
                 Fitness-fn; //fonction qui mesure la fitness d'un individu

        répéter
                 new-population ← ensemble vide ;

                 for i=1 to Size(population) do
                 x ← Random-Selection(population, fitness-fn);
                 y ← Random-Selection(population, fitness-fn);
                 Childe ← Reproduce(x, y);

                 if (Small Random probability) then Childe ← Mutate(Childe)
                 add Childe to new-population;
                 population ← new-population;

        jusqu'à ce qu'un individu ait la valeur de fitness la plus optimale, ou
        assez de temps a été écoulé
Return le meilleur individu dans la population selon fitness-fn;
}
```