

Partie 1

RÉSOLUTION DE PROBLÈMES

Partie 1 – Résolution de problèmes

1.1 ESPACE D'ETATS, ARBRE DE RECHERCHE, EXPLORATION

Problème

- Un problème est une collection d'informations que l'agent utilise pour décider quelle(s) action(s) accomplir.
- Définir un problème c'est choisir une abstraction de la réalité en termes :
 - Identification d'un **état initial** (donc choix d'un langage de description d'états du problème)
 - Identification des **actions possibles** par définition d'opérateurs de changement d'état (donc définition de l'ensemble des états possibles du problème)
 - » *Les opérateurs ont parfois des conditions d'applications*

Espace des états

- On appelle **espace des états** d'un problème l'ensemble des états atteignables depuis l'état initial par n'importe quelle séquence d'actions
- Un espace d'état peut être représenté par un **graphe orienté**
 - » Les **sommets** sont les états
 - » Les **arcs** sont les actions permettant de passer d'un état à un autre

Résoudre un problème

- Un problème est défini pour un objectif particulier. On doit donc également définir :
 - Une **fonction de test de but atteint** qui détermine si un état du problème correspond à un état but du problème
 - » Par liste d'états but
 - » Par la donnée d'une propriété pour un tel état
- Une **solution** est une séquence d'actions permettant de passer de l'état initial vers un état but
 - Donc un **chemin** dans l'espace des états de l'état initial vers un état but
- Résoudre un problème c'est trouver une/toutes les solutions
 - Pour certains problèmes, une **fonction de coût de chemin** permet de sélectionner une solution préférée parmi l'ensemble des solutions

Formalisation d'un problème

Etat
...

Action
resultat (e : Etat) : Etat
coût : Réel

Problème
étatInitial : Etat
actions (e : Etat) : Ensemble d'Action
but? (e : Etat) : Bool

- Hypothèses simplificatrices
 - Indépendance de l'environnement
 - Connaissance complète de l'effet des actions
 - Coût dépendant des coûts de chaque action
- Attention
 - Le nombre d'états peut être **infini** mais...
 - Le nombre d'actions possible à partir de tout état **doit être fini**

Algorithme de résolution

- Résoudre un problème consiste à trouver une séquence d'actions permettant de passer de l'état initial à un état but : **une solution**
- Il s'agit donc d'effectuer une **recherche à travers l'espace des états**
- L'idée est de maintenir et d'étendre un **ensemble de solutions partielles** : des séquences d'actions qui amènent à des états intermédiaires « plus proche » de l'état but.

Génération des solutions partielles

■ Un cycle en 3 phases

1. **Tester** si l'état actuel est un état but
2. **Générer** les états atteignables à partir de l'état actuel et des actions possibles
3. **Sélectionner** un des états générés (à cette étape ou précédemment) et recommencer...

EXPLORATION

SELECTION

*Le choix de l'état à considérer, la sélection, est déterminé par la **stratégie de recherche***

Processus de résolution

- Le processus de résolution de problème consiste donc à construire un **arbre de recherche** qui se « superpose » à l'**espace des états** du problème.
- Chaque **nœud** de l'arbre correspond soit à l'état initial du problème, soit à un état obtenu par développement du nœud parent (lors de l'**exploration**) en appliquant les actions du problème.

Arbres de recherche

- La **racine** de l'arbre correspond à l'état initial du problème.
- Les **feuilles** de l'arbre sont :
 - soit des nœuds associés à des états sans action permettant de poursuivre
 - soit des nœuds non encore **explorés**
- On peut associer un **coût** à tout nœud de l'arbre
 - Souvent le coût de la séquence d'actions sur le **chemin de la racine au nœud**

Nœuds d'un arbre de recherche

Nœud	
état : Etat	L'état dans l'espace des états associé à ce nœud
parent : Nœud	Le nœud parent de l'arbre (<i>null</i> pour la racine)
action : Action	L'action ayant permis d'atteindre cet état à partir de l'état associé à son nœud parent (<i>null</i> pour la racine)
coût : Réel	Le coût du chemin emprunté depuis la racine (noté $g(n)$ dans la suite)
Noeud (e : Etat, p : Nœud, a : Action, c : Réel) : Nœud	Constructeur de un nœud à partir d'un état, d'un nœud parent, d'une action (ayant permis d'atteindre cet état), du coût associé au chemin, d'une priorité de action

Par ailleurs, on appelle profondeur d'un nœud la longueur du chemin du nœud à la racine.

Gestion des nœuds à explorer

- On appelle **frontière** l'ensemble des nœuds non encore explorés de l'arbre de recherche
- Dans cette présentation des algos de recherche la frontière est gérée comme une **liste de nœuds**
 - On sélectionne toujours le nœud en tête de liste
 - La **stratégie** de la recherche est reportée sur la **procédure d'insertion** des nœuds dans la **liste frontière**

Liste<Nœud>
Liste () : Liste
vide? () : Bool
otterTête () : Nœud
insérer (n : Nœud) : void

Constructeur retournant une liste vide

Retourne vrai si la liste est vide

Extrait le nœud en tête et le retourne (la liste est modifiée)

Insère le nœuds **n** dans la liste selon une **stratégie particulière**

Fonction générale de Recherche

ExplorerGénérique (*p : Problème*) : Nœud ou null

// retourne une solution (chemin de la racine au nœud retourné) ou un échec (null)

Nœud racine = new Nœud(*p.étatInitial*, null, null, 0) ;

Liste frontière = new Liste() ;

frontière.insérer(racine) ;

tant que non frontière.vide?() faire

 Nœud *n* = frontière.ôterTête() ;

si *p.but?*(*n.état*) alors retourner *n*;

pour toute Action *a* dans *p.actions(n.état)* faire

 Nœud *sn* = new Nœud (*a.résultat(n.état)*, *n*, *a*, **cout**) ;

 frontière.insérer(*sn*);

fin pour

fin tant que

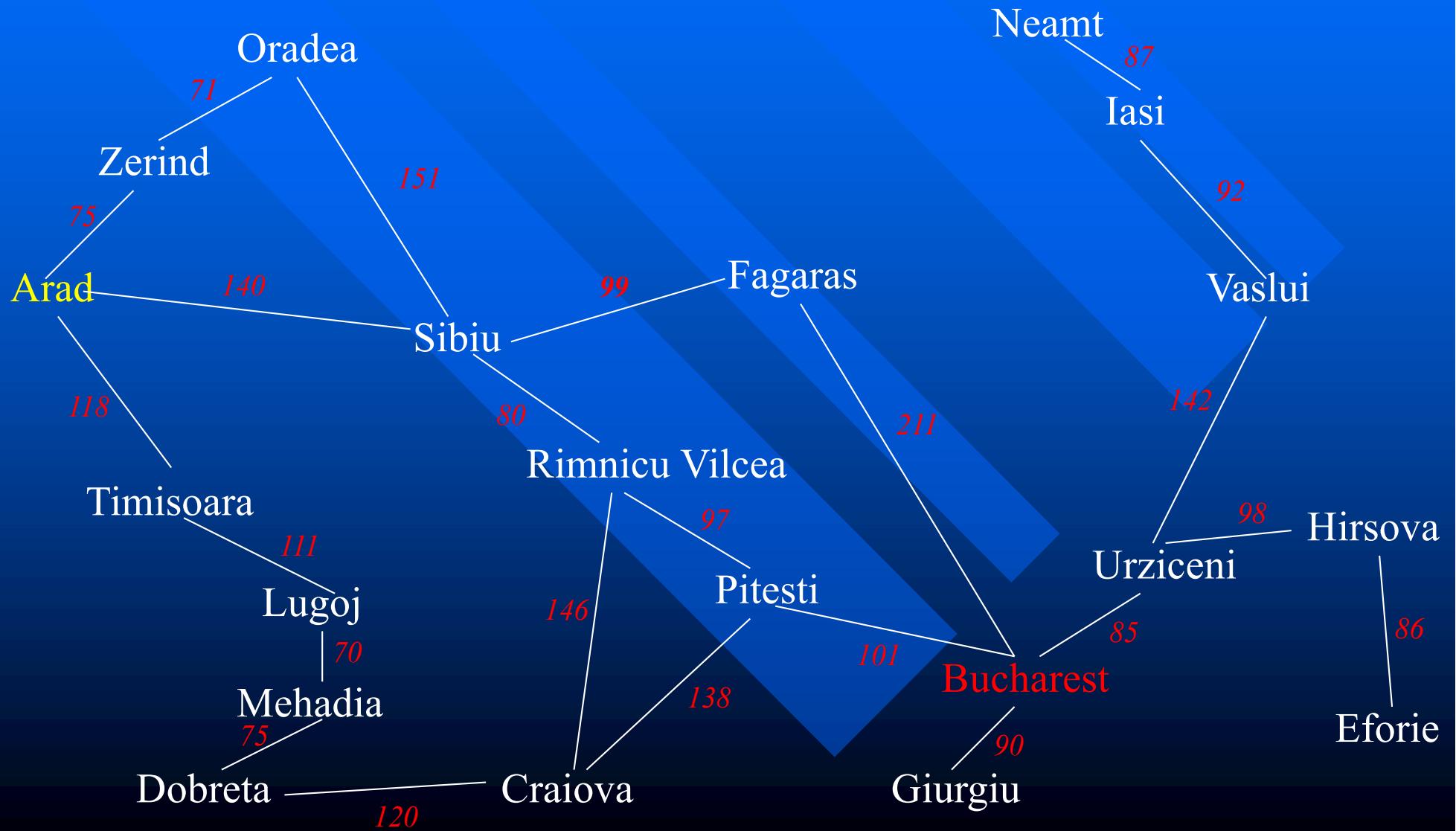
retourner null;

Différentes stratégies de recherche

- *largeur* \Rightarrow insertion en queue
- *profondeur* \Rightarrow insertion en tête
- *coûtMin*

...

Problème : recherche d'une route de Arad à Bucharest



Modélisation du problème des routes roumaines

- Les états et les actions de l'agent sont respectivement représentés par les sommets S et les arêtes A du graphe G=(S,A) précédent.
- L'état initial est le sommet *Arad* et l'état but le sommet *Bucharest*.

étatInitial = *Arad*

actions = A

but?(e : Etat) { return e=*Bucharest* }

a.cout() { return valeur_arête (a) }

la distance entre les deux villes liées par l'arête a

- Le coût d'un chemin est la somme des coûts des actions (somme des coûts des arêtes allant de l'état d'origine à l'état courant)
→ On fera cette hypothèse dans toute la suite du cours

Fonction générale de Recherche

ExplorerGénérique (p : Problème) : Nœud ou null

Nœud racine = new Nœud(p.étatInitial, null, null, 0) ;

Liste frontière = new Liste() ;

frontière.insérer(racine) ;

tant que non frontière.vide?() faire

 Nœud n = frontière.oterTête() ;

si p.but?(n.état) alors retourner n;

pour toute Action a dans p.actions(n.état) faire

 Nœud sn = new Nœud (a.résultat(n.état), n, a, n.cout+a.cout) ;

 frontière.insérer(sn);

fin pour

fin tant que

retourner null;

Les différentes stratégies

- Stratégies de recherche non informées
 - C'est-à-dire sans autre information que les données du problème : **étatInitial, actions, but?, coût**
- Stratégies de recherche informées
 - Disposant d'information supplémentaire sur le problème permettant de privilégier certaines branches dans l'arbre de recherche
 - Typiquement on utilise une **fonction heuristique** qui estime le coût d'une solution d'un état à l'état but

Performance d'une stratégie de résolution

- La performance d'une stratégie de résolution se mesure selon quatre points de vue :
 - **Complétude** : la technique de résolution marche-t-elle dans tous les cas ? C'est-à-dire, si le problème admet une solution, l'algorithme la trouve-t-il ?
 - **Optimalité** (si coût) : la technique de résolution trouve-t-elle une solution de coût minimal ?
 - **Complexité** : la technique est-elle coûteuse
 - » en **temps** ?
 - » en **mémoire** ?

Ne pas confondre performance de la résolution et celle de l'exécution de la solution