

Modélisation et résolution de trois énigmes en se basant sur les algorithmes de recherche

Youssef Lamzaouak `youssef.lamzaouak@etu.emse.fr`

Adil El Hakouni `adil.elhakouni@etu.emse.fr`

Anas Rachyd `anas.rachyd@etu.emse.fr`

Abstract: Nous essayerons dans ce document de représenter notre modélisation des 3 problèmes qui sont respectivement « Wolf, Farmer, Cabbage, Goat Problem », « Cannibals and Missionaries Problem » et finalement « Water Jug Problem ». Nous exposerons tout de suite les grandes lignes du code qui seront détaillés dans les fichiers attachés.

1 Problem One

1.1 Modeling

- States: C'est l'ensemble des états du farmer, wolf ,goat ou cabbage qui peuvent etre soit du coté droite soit du coté gauche du lac, donc en somme on a 16.

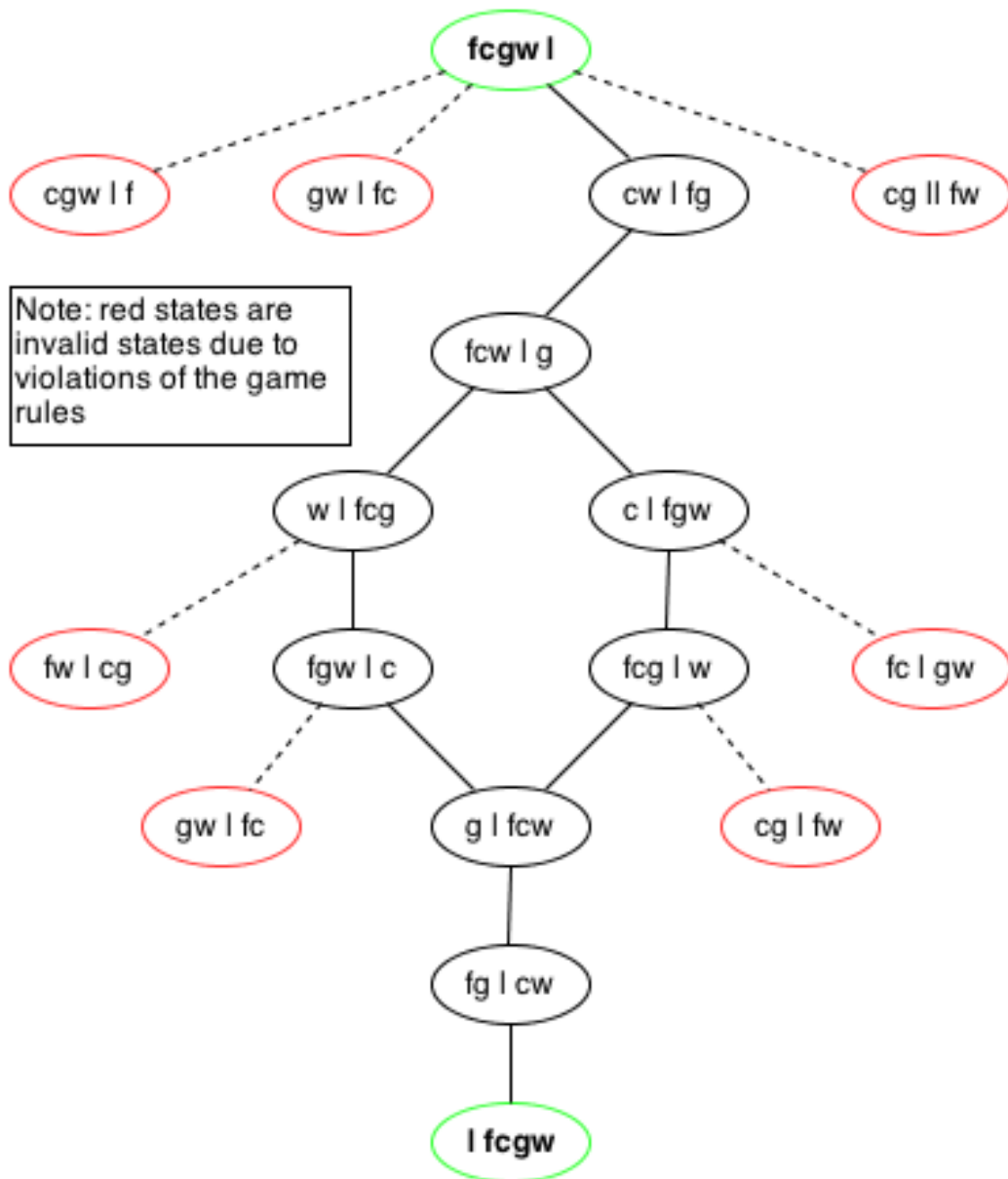


Figure 1: Algorithm evolution from initial state to goal state

- Initial State: l'état initial est celui où chacun d'eux est au côté gauche de la rivière.
- Actions: chacun des quatre agents de notre énigme a deux possibil-

ités soit d'aller au coté droite de la rivière soit au coté gauche. Plus explicitement on a soit le fermier traverse seul, soit avec la chèvre a sa compagnie ; soit avec les choux soit finalement avec le loup.

- Transition model: bien illustré dans la figure 1 qui décrit l'espace des états.
- Goal Test: l'état final est celui ou chacun d'eux est au coté droite de la rivière.
- Path Cost: chaque action coute 1, donc le cout total est la somme des actions.
- *Constraints*: 2 agents au maximum peuvent traverser la rivière en une seule fois. La chèvre et les choux ne peuvent pas rester ensemble sans que le fermier soit avec eux. La chèvre et le loup ne peuvent eux aussi rester ensemble sans la présence du fermier.
- *Solution*: Prendre le loup de l'autre côté laissera la chèvre et le chou ensemble. En emportant également le chou, le loup et la chèvre seront seuls. Par conséquent, l'agriculteur prendra d'abord la chèvre de l'autre côté et reviendra seul. Nous avons fermier, loup et chou d'un côté et chèvre de l'autre côté. Maintenant, il emmènera le loup, déposera le loup de l'autre côté et reviendra avec la chèvre. Alors maintenant, d'un côté, nous avons fermier, chou et chèvre et de l'autre côté, nous avons un loup. Maintenant, il emmène le chou et revient seul. Alors maintenant, le scénario est: fermier, chèvre d'un côté et loup, chou de l'autre côté. Maintenant, enfin, il traverse la rivière avec la chèvre et réussit donc à emporter toutes ses affaires avec lui.

1.2 Code headlines

1. Breadth First Search

```
FarmerTest x
Inspecting node [parent=[parent=[parent=[pa
Goal node reached => SUCCESS
Action : null
State : (FWGC, ____ )
Action : [F_G_]>
State : (_W_C, F_G_)
Action : [F____]<
State : (FW_C, __G_)
Action : [FW__]>
State : (____C, FWG_)
Action : [F_G_]<
State : (F_GC, _W__)
Action : [F__C]>
State : (__G_, FW_C)
Action : [F____]<
State : (F_G_, _W_C)
Action : [F_G_]>
State : (____, FWGC)

Process finished with exit code 0
```

Figure 2: BFS Result

2. Depth First Search

```
MAndCTest x
Expanding node, frontier is empty-Error
Goal node reached => SUCCESS
Action : null
State : (1,3,3) (0,0,0)
Action : [_c]>
State : (0,3,2) (1,0,1)
Action : [_c]<
State : (1,3,3) (0,0,0)
Action : [mc]>
State : (0,2,2) (1,1,1)
Action : [m_]<
State : (1,3,2) (0,0,1)
Action : [cc]>
State : (0,3,0) (1,0,3)
Action : [_c]<
State : (1,3,1) (0,0,2)
Action : [mm]>
State : (0,1,1) (1,2,2)
Action : [mc]<
State : (1,2,2) (0,1,1)
Action : [mm]>
State : (0,0,2) (1,3,1)
Action : [_c]<
State : (1,0,3) (0,3,0)
Action : [cc]>
State : (0,0,1) (1,3,2)
Action : [m_]<
State : (1,1,1) (0,2,2)
Action : [mc]>
State : (0,0,0) (1,3,3)

Process finished with exit code 0
```

Figure 3: DFS Result

2 Problem two

2.1 Modeling

- States: c'est l'ensemble des états des 3 cannibals et des 3 missionnaires qui peuvent être soit du côté droit soit du côté gauche du lac combiné avec la position du bateau donc en somme $2^3 \times 2 = 16$ en compte des positions interdites. Plus explicitement soit on transporte 2 missionnaires ou 2 cannibals ou 1 missionnaire seul ou un cannibal seul ou finalement un cannibal et un missionnaire.

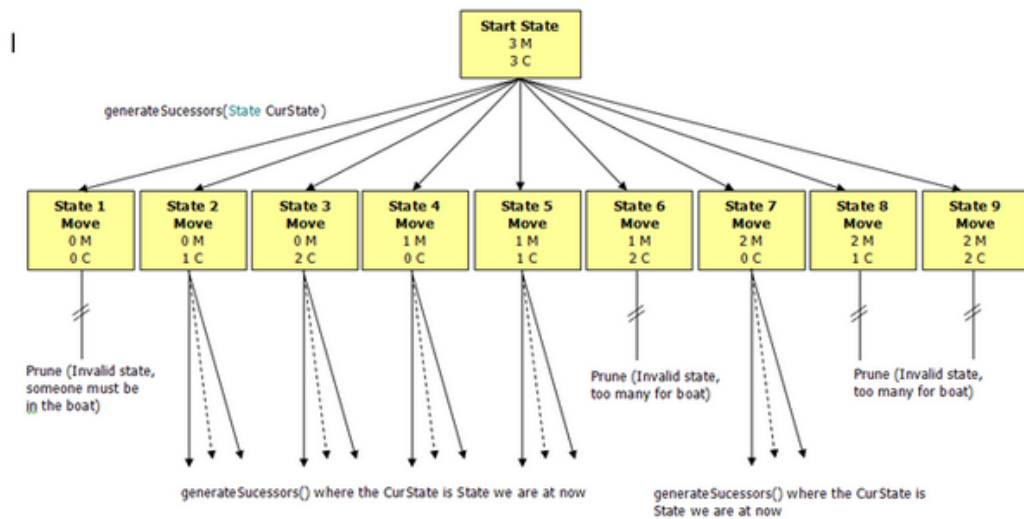


Figure 4: Algorithm evolution from initial state to goal state

- Initial State: l'état initial est celui où chacun d'eux est au côté gauche de la rivière.
- Actions: chacun des six agents de notre énigme ainsi que le bateau a deux possibilités soit d'aller au côté droit de la rivière soit au côté gauche.
- Transition model: bien illustré dans la figure1 qui décrit l'espace des états.

- Goal Test: l'état final est celui ou chacun d'eux est au coté droite de la rivière.
- Path Cost: chaque action coute 1, donc le cout total est la somme des actions.
- *Constraints*: 2 agents au maximum peuvent traverser la rivière en une seule fois. Sur chacune des deux cotés de la rivière et a chaque fois, le nombre des cannibales ne doit pas dépasser ceux des missionnaires.
- *Solution*:

After application of rule	persons in the river bank-1	persons in the river bank-2	boat position
Start state	M, M, M, C, C, C	0	bank-1
5	M, M, C, C	M, C	bank-2
2	M, M, C, C, M	C	bank-1
7	M, M, M	C, C, C	bank-2
10	M, M, M, C	C, C	bank-1
3	M, C	C, C, M, M	bank-2
6	M, C, C, M	C, M	bank-1
3	C, C	C, M, M, M	bank-2
10	C, C, C	M, M, M	bank-1
7	C	M, M, M, C, C	bank-2
10	C, C	M, M, M, C	bank-1
7	0	M, M, M, C, C, C	bank-2

Figure 5: Solution step by step

2.2 Code headlines

1. Breadth First Search

```
MAndCTest x
Expanding node, frontier is [[parent
Inspecting node [parent=[parent=[par
Goal node reached => SUCCESS
Action : null
State : (1,3,3) (0,0,0)
Action : [cc]>
State : (0,3,1) (1,0,2)
Action : [_c]<
State : (1,3,2) (0,0,1)
Action : [cc]>
State : (0,3,0) (1,0,3)
Action : [_c]<
State : (1,3,1) (0,0,2)
Action : [mm]>
State : (0,1,1) (1,2,2)
Action : [mc]<
State : (1,2,2) (0,1,1)
Action : [mm]>
State : (0,0,2) (1,3,1)
Action : [_c]<
State : (1,0,3) (0,3,0)
Action : [cc]>
State : (0,0,1) (1,3,2)
Action : [_c]<
State : (1,0,2) (0,3,1)
Action : [cc]>
State : (0,0,0) (1,3,3)
Process finished with exit code 0
```

Figure 6: BFS Result

2. Depth First Search:

```
MAndCTest x
Expanding node, frontier is empty-Error
Goal node reached => SUCCESS
Action : null
State : (1,3,3) (0,0,0)
Action : [_c]>
State : (0,3,2) (1,0,1)
Action : [_c]<
State : (1,3,3) (0,0,0)
Action : [mc]>
State : (0,2,2) (1,1,1)
Action : [m_]<
State : (1,3,2) (0,0,1)
Action : [cc]>
State : (0,3,0) (1,0,3)
Action : [_c]<
State : (1,3,1) (0,0,2)
Action : [mm]>
State : (0,1,1) (1,2,2)
Action : [mc]<
State : (1,2,2) (0,1,1)
Action : [mm]>
State : (0,0,2) (1,3,1)
Action : [_c]<
State : (1,0,3) (0,3,0)
Action : [cc]>
State : (0,0,1) (1,3,2)
Action : [m_]<
State : (1,1,1) (0,2,2)
Action : [mc]>
State : (0,0,0) (1,3,3)

Process finished with exit code 0
```

Figure 7: DFS Result

3 Problem three

3.1 Modeling

- States: c'est le volume de chacune des bouteilles de capacité respective L1 et L2 litres a chaque versement de l'eau d'une bouteille A vers B ou a chaque versement de l'eau dans le sol.

1.	$(X, Y) \text{ if } X < 4 \rightarrow (4, Y)$	Fill the 4-gallon jug
2.	$(X, Y) \text{ if } Y < 3 \rightarrow (X, 3)$	Fill the 3-gallon jug
3.	$(X, Y) \text{ if } X = d \ \& \ d > 0 \rightarrow (X-d, Y)$	Pour some water out of the 4-gallon jug
4.	$(X, Y) \text{ if } Y = d \ \& \ d > 0 \rightarrow (X, Y-d)$	Pour some water out of 3-gallon jug
5.	$(X, Y) \text{ if } X > 0 \rightarrow (0, Y)$	Empty the 4-gallon jug on the ground
6.	$(X, Y) \text{ if } Y > 0 \rightarrow (X, 0)$	Empty the 3-gallon jug on the ground
7.	$(X, Y) \text{ if } X + Y \leq 4 \text{ and } Y > 0 \rightarrow 4, (Y - (4 - X))$	Pour water from the 3-gallon jug into the 4-gallon jug until the gallon jug is full.
8.	$(X, Y) \text{ if } X + Y \geq 3 \text{ and } X > 0 \rightarrow (X - (3 - Y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9.	$(X, Y) \text{ if } X + Y \leq 4 \text{ and } Y > 0 \rightarrow (X + Y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10.	$(X, Y) \text{ if } X + Y \leq 3 \text{ and } X > 0 \rightarrow (0, X + Y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11.	$(0, 2) \rightarrow (2, 0)$	Pour the 2-gallons water from 3-gallon jug into the 4-gallon jug
12.	$(2, Y) \rightarrow (0, Y)$	Empty the 2-gallons in the 4-gallon jug on the ground.

Fig. 2.3. Production rules (operators) for the water jug problem.

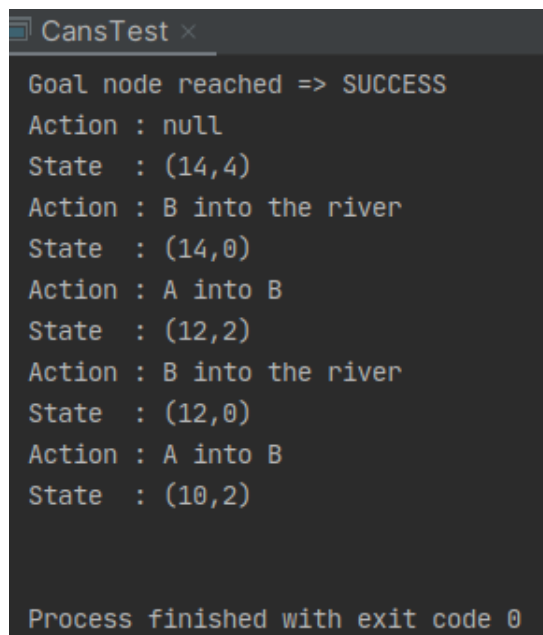
Figure 8: différentes possibilités de l'espace des états

- Initial State: l'état initial est le volume de départ de chacune des 2 bouteilles.Par exemple A a 12 litres et B a 3 litres.
- Actions: A chaque fois ,on peut verser de l'eau dans le sol ou dans l'autre bouteille.
- Transition model: bien illustré dans la figure8 qui décrit l'espace des états.
- Goal Test: l'état final est décrit par le volume des 2 bouteilles.

- Path Cost: chaque action coute 1, donc le cout total est la somme des actions.
- *Constraints*: Tous les versements sont possibles entre les 2 bouteilles ,il faut juste ne pas dépasser la capacité des 2 bouteilles .

3.2 Code headlines

1. Breadth First Search



```
CansTest x
Goal node reached => SUCCESS
Action : null
State : (14,4)
Action : B into the river
State : (14,0)
Action : A into B
State : (12,2)
Action : B into the river
State : (12,0)
Action : A into B
State : (10,2)

Process finished with exit code 0
```

Figure 9: BFS Result

2. Depth First Search

```
CansTest x
Expanding node, frontier is [[par
Inspecting node [parent=[parent=[
Expanding node, frontier is [[par
Goal node reached => SUCCESS
Action : null
State : (14,4)
Action : B into the river
State : (14,0)
Action : A into B
State : (12,2)
Action : B into the river
State : (12,0)
Action : A into B
State : (10,2)
```

Figure 10: DFS Result

4 A* Algorithm

4.1 Principle

Considérons une grille carrée comportant de nombreux obstacles et on nous donne une cellule de départ et une cellule cible. Nous voulons atteindre la cellule cible (si possible) depuis la cellule de départ le plus rapidement possible. Ici, un algorithme de recherche A* vient à la rescousse.

Ce que fait A* Search Algorithm, c'est qu'à chaque étape, il sélectionne le nœud selon une valeur f qui est un paramètre égal à la somme de deux autres paramètres - g et h . À chaque étape, il sélectionne le nœud, cellule ayant le f le plus bas et traite ce nœud comme cellule.

Nous définissons g et h aussi simplement que possible ci-dessous

g = le coût de déplacement pour se déplacer du point de départ à un carré donné sur la grille, en suivant le chemin généré pour y arriver. h = le coût de mouvement estimé pour passer de ce carré donné sur la grille à la destination finale. Ceci est souvent appelé l'heuristique, qui n'est rien

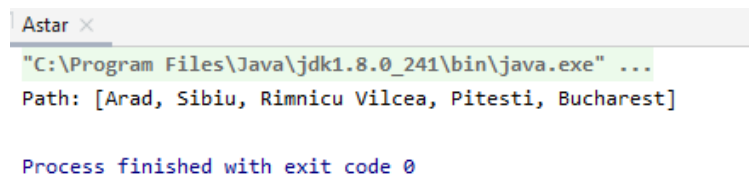
d'autre qu'une sorte de supposition intelligente. Nous ne savons vraiment pas la distance réelle jusqu'à ce que nous trouvions le chemin, car toutes sortes de choses peuvent être sur le chemin (murs, eau, etc.).

4.2 Problem Statement

Example: Romania On holiday in Romania; currently in Arad. Flight leaves tomorrow from Bucharest

- Goal: be in Bucharest
- States: various cities
- Actions: drive between cities Find solution: sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

4.3 A* Solution



```
Astar x
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Path: [Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]

Process finished with exit code 0
```

Figure 11: Résultat donné par l'Algorithme A star