Τυπικός ορισμός και επίλυση προβλημάτων με την χρήση του search.py (Ιεραπόστολοι & Κανίβαλοι)

Μαραβίτσας Νίκος
nmaravitsas@di.uoa.gr

23 Δεκεμβρίου 2013

Πως ορίζουμε τυπικά ένα πρόβλημα;

- Αρχική Κατάσταση: Από την οποία ξεκινά ο πράκτορας**(αναπαράσταση καταστάσεων ?)**.
- Συνάρτηση διαδόχων:$[(a_1, s_1),..,(a_n,s_n)]$=succ(state)

  (αναπαράσταση καταστάσεων ?)
- Χώρος καταστάσεων: Προκύπτει από την αρχική κατάσταση και την συνάρτηση διαδόχων.
- Έλεγχος στόχου
- Κόστος: cost = cost_path(state1, action, state2)

Κλάση ορισμού προβλήματος

class Problem(object):

"""The abstract class for a formal problem. You should **subclass** this and **implement** the methods **actions** and **result**, and **possibly __init__**, **goal_test**, and **path_cost**. Then you will create instances of your subclass and solve them with the various search functions."""

- Η κλάση αυτή ορίζεται στο αρχείο search.py. Το αρχείο αυτό κάνει import το αρχείο utils.py

Πρόβλημα Ιεραποστόλων & Κανιβάλων

Τρεις Ιεραπόστολοι και **τρεις κανίβαλοι** βρίσκονται στη μια όχθη ενός ποταμού με μια βάρκα που χωρά **δύο άτομα**. Βρείτε έναν τρόπο να περάσουν όλοι στην απέναντι όχθη χωρίς να μείνει σε ένα μέρος μια ομάδα ιεραποστόλων αριθμητικά **μικρότερη** από τους κανιβάλους

Αναπαράσταση Καταστάσεων

Δεν υπάρχει ένας σωστός τρόπος. Πρέπει να επιλέγουμε λύσεις που περιέχουν μόνο την επαρκή πληροφορία

- (<αριθμός κανιβάλων αριστερά>, <αριθμός ιεραποστόλων αριστερά>, <θέση βάρκας>, <αριθμός κανιβάλων δεξιά>, <αριθμός ιεραποστόλων δεξιά>)
- (<αριθμός κανιβάλων αριστερά>, <αριθμός ιεραποστόλων αριστερά>, <βάρκα αριστερά>)
- (<αριθμός κανιβάλων αριστερά>, <αριθμός ιεραποστόλων αριστερά>, <θέση βάρκας>)

Αρχική Κατάσταση και Κατάσταση Στόχου

- Αρχική κατάσταση: (3, 3, 'left')
- Κατάσταση στόχου: (0, 0, 'right')
- Constructor:

```
class MissionariesAndCannibals(Problem) :
    def __init__(self) :
        super(MissionariesAndCannibals,
        self).__init__((3, 3, 'left'), (0 , 0 ,
        'right'))
```

Συνάρτηση Διαδόχων

Αναπαράσταση Καταστάσεων

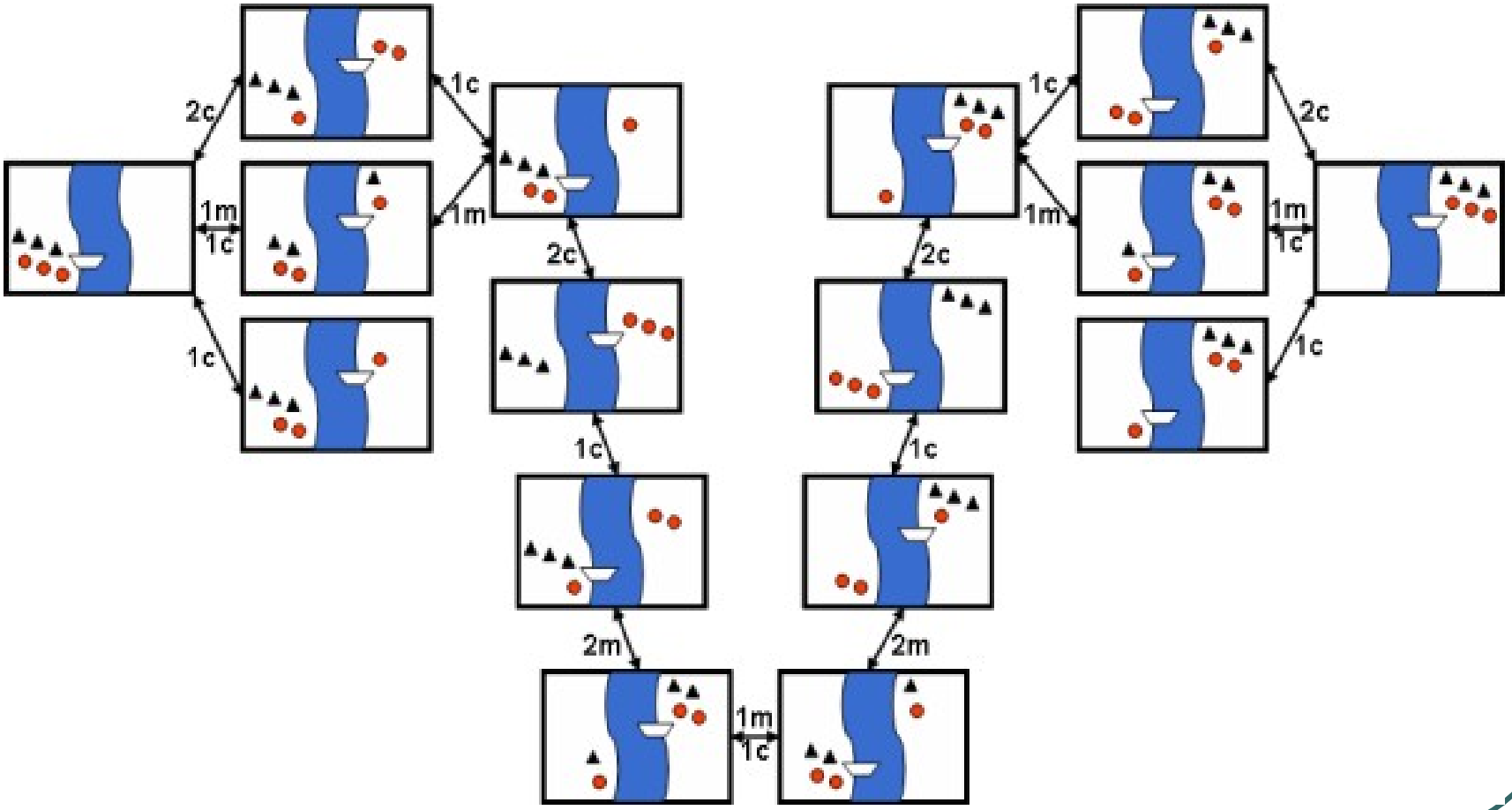
- (<αριθμός ιεραποστόλων που μεταφέρονται>, <αριθμός κανιβάλων που μεταφέρονται>)

  Η κατεύθυνση υπονοείται από την κατάσταση

  Έγκυρες καταστάσεις: Για κάθε κατάσταση (m, c, b) που προκύπτει πρέπει να ισχύει ένα από τα παρακάτω:

- m = c
- m = 3 και c < 3:
- m = 0 και c > 0

Χώρος Καταστάσεων



Συναρτήσεις actions και result

```
def actions(self, state) :
   possibleActions = [(1, 1), (1, 0), (2, 0), (0,1),
(0,2)]
   validActions = []
   if state[2] == 'left' :
      for possibleAction in possibleActions :
         m = state[0] - possibleAction[0]
         c = state[1] - possibleAction[1]
         if self.__isValidState(m, c) :
            validActions.append(possibleAction)
   else : # right
      # Αντίστοιχα (προσθέτω αντί να αφαιρώ)
   return validActions
```

```
def result(self, state, action) :
   m = state[0]
   c = state[1]
   b = state[2]
   if b == 'left' :
      m = state[0] - action[0]
      c = state[1] - action[1]
      b = 'right'
   else : # right
      # Αντίστοιχα
   return (m, c, b)
```

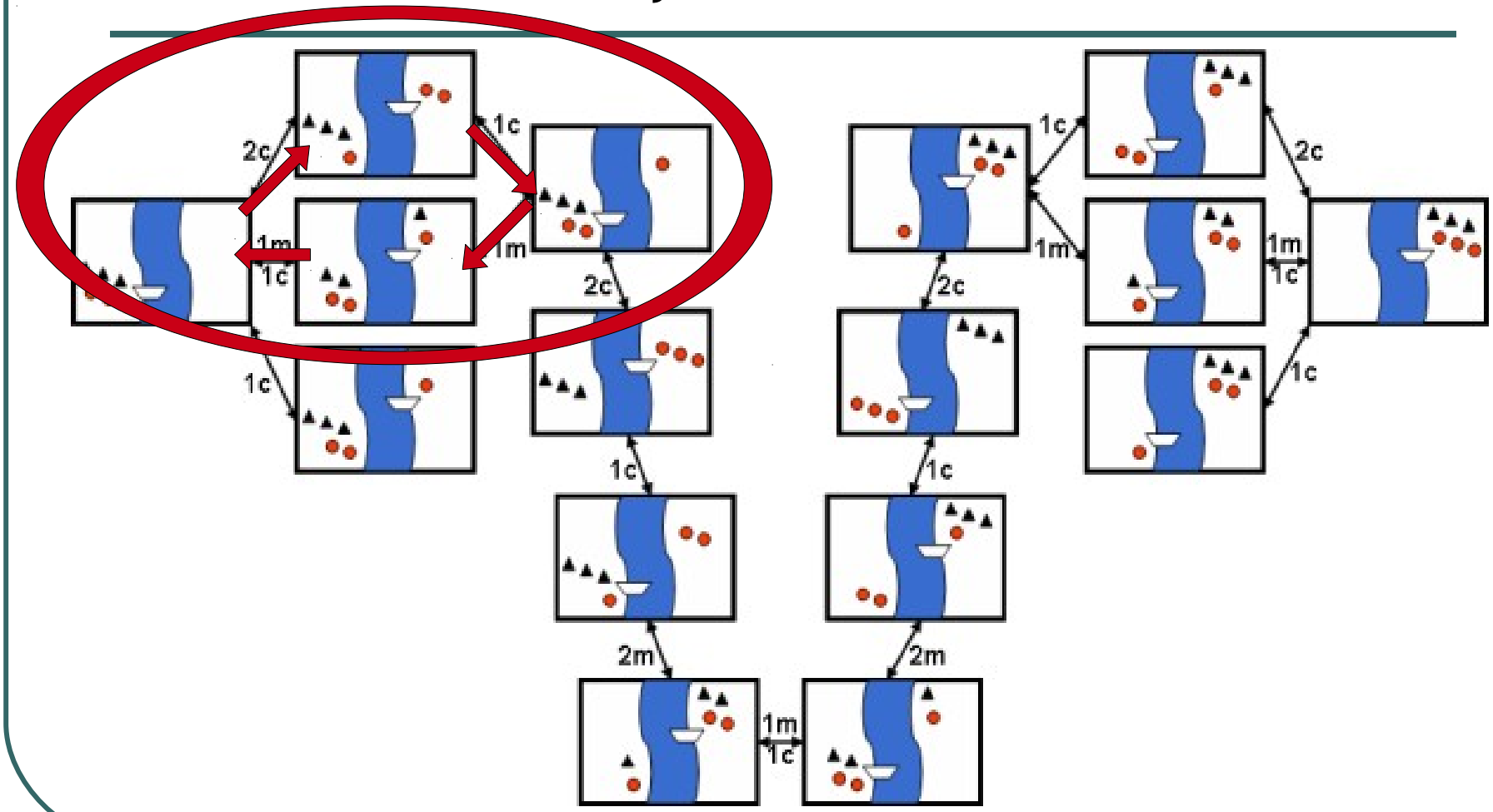
* Η συνάρτηση __isValidState(m,c) ελέγχει αν ισχύουν οι προηγούμενοι περιορισμοί ανάμεσα στα m και c

Κόστος

- Θεωρούμε ένα **σταθερό κόστος** για κάθε ενέργεια. Το συνολικό κόστος μίας λύσης είναι το σύνολο κόστους όλων των ενεργειών
- **Συνάρτηση path_cost:** (ορισμένη στην κλάση Problem)

```
def path_cost(self, c, state1, action, state2):
        return c + 1
```

Επαναλαμβανόμενες καταστάσεις



Κλάση κόμβος

- Αναπαριστά ένα κόμβο στο δέντρο αναζήτησης
- Δεν χρειάζεται να τη πειράξουμε
- Προσέξτε πως αν υπάρχουν δύο μονοπάτια προς μία κατάσταση τότε υπάρχουν δύο κόμβοι στο δέντρο αναζήτησης

class Node:

- def __init__(self, state, parent=None, action=None, path_cost=0):
- def expand(self, problem):
  "List the nodes reachable in one step from this node."
- def child_node(self, problem, action)
- def solution(self)
  "Return the sequence of actions to go from the root to this node."
- def path(self):
  "Return a list of nodes forming the path from the root to this node."

Επίλυση προβλήματος (1/2)

```
from MissionariesAndCannibals import *

p = MissionariesAndCannibals()

s = depth_first_graph_search(p)
sol = s.solution() # List of actions
path = s.path() # Solution path (nodes of search tree)
print "Solution: \n+{0}+\n|Action\t|State\t \t|Path Cost |\n+{0}+".format('-'*42)
for i in range(len(path)) :
   state = path[i].state
   cost = path[i].path_cost
   action = " "
   if i > 0 :
      action = sol[i-1]
   print "|{0}\t|{1} \t|{2} \t   |".format(action, state, cost)
print "+----------------------------------------+"
```

Επίλυση προβλήματος (2/2)

```
user@user-laptop:~/Dropbox/ArtificialIntelligence/myScripts$ python MissionariesAndCannibalsMain.py
Solution:
+-----------------------------------------+
|Action |State                  |Path Cost |
+-----------------------------------------+
|       |(3, 3, 'left')         |0         |
|(0, 2) |(3, 1, 'right')        |1         |
|(0, 1) |(3, 2, 'left')         |2         |
|(0, 2) |(3, 0, 'right')        |3         |
|(0, 1) |(3, 1, 'left')         |4         |
|(2, 0) |(1, 1, 'right')        |5         |
|(1, 1) |(2, 2, 'left')         |6         |
|(2, 0) |(0, 2, 'right')        |7         |
|(0, 1) |(0, 3, 'left')         |8         |
|(0, 2) |(0, 1, 'right')        |9         |
|(0, 1) |(0, 2, 'left')         |10        |
|(0, 2) |(0, 0, 'right')        |11        |
+-----------------------------------------+
user@user-laptop:~/Dropbox/ArtificialIntelligence/myScripts$
```

Αναφορές

- Ο κώδικας του βιβλίου AIMA σε python που χρησιμοποιήθηκε για αυτή την υλοποίηση μπορεί να βρεθεί στη παρακάτω διεύθυνση

  http://code.google.com/p/aima-python/source/checkout