Blocks World Presentation

Εισαγωγή

Ο κόσμος των κύβων είναι ένας τομέας σχεδιασμού (planning domain) στην τεχνητή νοημοσύνη. Ο αλγόριθμος είναι παρόμοιος με ένα σύνολο ξύλινων κύβων διαφόρων σχημάτων και χρωμάτων που βρίσκονται πάνω σε ένα τραπέζι. Ο στόχος είναι η κατασκευή μίας ή περισσότερων κάθετων στοίβων από κύβους. Μόνο ένας κύβος μπορεί να μετακινηθεί κάθε φορά: μπορεί είτε να τοποθετηθεί στο τραπέζι είτε πάνω σε έναν άλλο κύβο. Εξαιτίας αυτού, οποιοιδήποτε κύβοι βρίσκονται κάτω από άλλους κύβους σε μια δεδομένη χρονική στιγμή δεν μπορούν να μετακινηθούν. Επιπλέον, ορισμένοι τύποι κύβων δεν επιτρέπουν τη στοίβαξη άλλων κύβων πάνω τους.

Η απλότητα αυτού του κόσμου-παιχνιδιού τον καθιστά ιδανικό για τις κλασικές συμβολικές προσεγγίσεις της τεχνητής νοημοσύνης, στις οποίες ο κόσμος μοντελοποιείται ως ένα σύνολο αφηρημένων συμβόλων που μπορούν να υποβληθούν σε λογικό συλλογισμό.

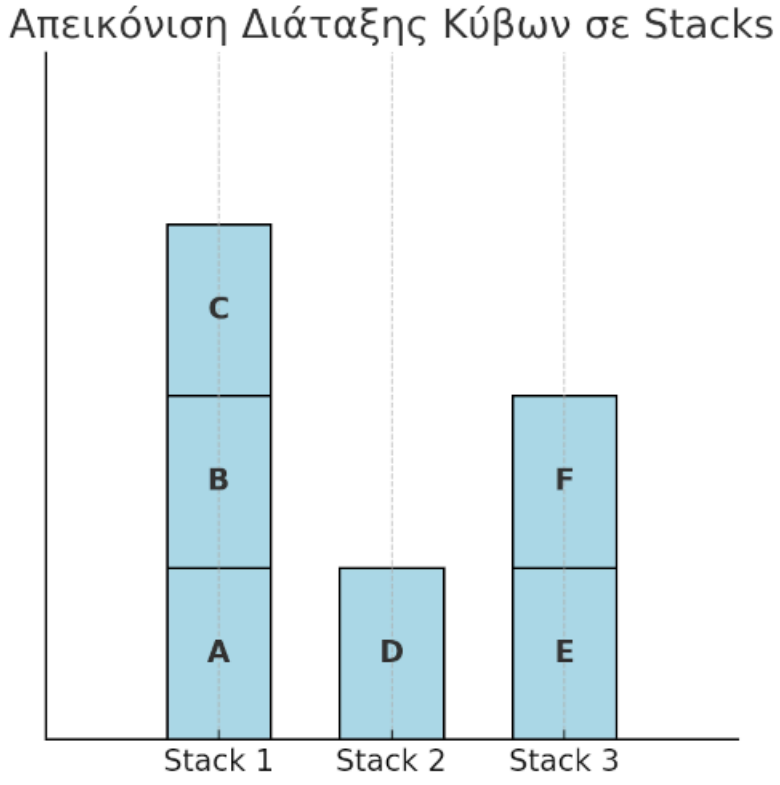
Πρόβλημα

Ξεκινάμε από μια αρχική διάταξη των κύβων και αναζητούμε τη σειρά των μετακινήσεων που θα οδηγήσει στη διάταξη-στόχο με τον ελάχιστο αριθμό κινήσεων.

Οι δυνατές, επιτρεπόμενες μετακινήσεις είναι:

α. Move(A,B,C) : Μετακίνηση του κύβου Α που βρίσκεται πάνω απο τον κύβο Β σε νέα θέση πάνω από τον κύβο C
β. Move(A,table,B) : Μετακίνηση του κύβου Α που βρίσκεται στο τραπέζι στη νέα θέση πανω από τον κύβο Β
γ. Move(A,B,table). : Μετακίνηση του κύβου Α που βρίσκεται πάνω απο τον κύβο Β σε νέα θέση πάνω στο τραπέζι

Παράδειγμα μια διάταξης κύβων



Το παραπάνω διάγραμμα απεικονίζει μια διάταξη κύβων σε **stacks**.

• Κάθε στοίβα (stack) περιέχει κύβους τοποθετημένους ο ένας πάνω στον άλλο.
• Ο κάτω κύβος λειτουργεί ως βάση (on table), ενώ ο πάνω κύβος δεν έχει άλλους πάνω του.
• Στο παράδειγμα υπάρχουν τρεις στοίβες:

• **Stack 1**: Κύβοι **A, B, C** (C στην κορυφή).
• **Stack 2**: Κύβος **D** μόνος του.
• **Stack 3**: Κύβοι **E, F** (F στην κορυφή).

Γλώσσα Προγραμματισμού

Επιλέχθηκε η R για την ευκολία διαχείρισης διανυσμάτων, λιστών και strings.

Οδηγίες για την εγκατάσταση της R : https://cran.r-project.org/

Εκτέλεση του Προγράμματος

Η εκτέλεση το προγράμματος γίνεται απο το τερματικό (shell) :

```
Rscript script.R depth|breadth input_file solution.txt
```

Είσοδος είναι ένα αρχείο pddl (Planning Domain Definition Language) οπου περιγράφονται

- οι κύβοι που συμμετέχουν στο πρόβλημα
- η αρχική διατάξη
- η διάταξη-στόχος

Παράδειγμα : probBLOCKS-4-0.pddl

```
(define (problem BLOCKS-4-0)
(domain BLOCKS)
(:objects D B A C )
(:INIT (CLEAR C) (CLEAR A) (CLEAR B) (CLEAR D) (ONTABLE C) (ONTABLE A)
 (ONTABLE B) (ONTABLE D) (HANDEMPTY))
(:goal (AND (ON D C) (ON C B) (ON B A)))
)
```

Είσοδος είναι επίσης ο αλγόριθμος που θα χρησιμοποιηθεί για την λύση, που μπορεί να είναι :

- **depth (Depth-First Search)** : είναι μια στρατηγική διερεύνησης γραφημάτων και δέντρων, όπου η αναζήτηση προχωρά όσο το δυνατόν πιο βαθιά πριν επιστρέψει σε προηγούμενους κόμβους.

  Βασική λειτουργία:

1. Ξεκινά από έναν αρχικό κόμβο (ρίζα).
2. Επιλέγει έναν γειτονικό κόμβο και μεταβαίνει σε αυτόν.
3. Συνεχίζει τη διαδικασία μέχρι να φτάσει σε έναν τερματικό κόμβο ή αδιέξοδο.
4. Όταν δεν υπάρχουν διαθέσιμες κινήσεις, επιστρέφει στον προηγούμενο κόμβο (backtracking) και αναζητά άλλη διαδρομή.
5. Συνεχίζει έως ότου βρεθεί ο στόχος ή εξερευνηθούν όλοι οι κόμβοι.

- **breadth (Breadth-First Search - BFS):** είναι μια στρατηγική διερεύνησης γραφημάτων και δέντρων, όπου οι κόμβοι εξερευνώνται επίπεδο προς επίπεδο πριν από τη μετάβαση σε βαθύτερα επίπεδα.

  Βασική λειτουργία:

1. Ξεκινά από έναν αρχικό κόμβο (ρίζα).
2. Προσθέτει όλους τους γειτονικούς κόμβους του στην ουρά.
3. Εξετάζει κάθε κόμβο από την ουρά με τη σειρά, προσθέτοντας τους γειτονικούς τους κόμβους αν δεν έχουν εξερευνηθεί.
4. Συνεχίζει τη διαδικασία μέχρι να βρεθεί ο κόμβος-στόχος ή να εξερευνηθούν όλοι οι κόμβοι.

Έξοδος είναι αρχείο (**solution.txt**) που περιλαμβάνει την λύση, δηλαδή το σύνολο των ελάχιστων σε αριθμό κινήσεων που οδηγούν στή διατάξη στόχο.

Παράδειγμα της λύσης με τον αλγόριθμο DFS για το παραπάνω pddl αρχείο

```
cat solution.txt

Move(B,table,A)
```

```
Move(C,table,B)
Move(D,table,C)
```

Περιγραφή του object κύβος και του object της διατάξης των κύβων

Α. ο κάθε κύβος παριστάνεται από μια λίστα δύο στοιχείων

• **Το πρώτο στοιχείο** δείχνει ποιος κύβος είναι από πάνω (ή -1 αν δεν υπάρχει).
• **Το δεύτερο στοιχείο** δείχνει ποιος κύβος είναι από κάτω (ή -1 αν βρίσκεται στο τραπέζι).

Για το παράδειγμα του διαγράμματος παραπάνω έχουμε

```
config <- list(
  "A" = list("B", -1),  # Ο Α έχει πάνω του τον Β, και είναι στη βάση
  "B" = list("C", "A"), # Ο Β έχει πάνω του τον C και στηρίζεται στον Α
  "C" = list(-1, "B"),  # Ο C δεν έχει κάτι πάνω του και στηρίζεται στον Β
  "D" = list(-1, -1),   # Ο D είναι μόνος του στο τραπέζι
  "E" = list("F", -1),  # Ο Ε έχει πάνω του τον F και είναι στη βάση
  "F" = list(-1, "E")   # Ο F είναι πάνω στον Ε και δεν έχει κάτι πάνω του
)
```

η μεταβλητή `config` ειναι η λίστα όλου του συνόλου των κύβων

Β. η κάθε διάταξη παριστάνεται σαν μια list των stacks των κύβων (R vectors) με σειρά από τη βάση προς την κορυφή

```
stacks <- list(
  "Stack 1" = c("A", "B", "C"),
  "Stack 2" = c("D"),
  "Stack 3" = c("E", "F")
)
```

η μεταβλητή `stacks` είναι η λίστα ολων των stack που παριστάνονται σαν R vectors (c(...))

Περιγραφή των συναρτήσεων (functions) του Προγράμματος

1. Συνάρτηση parse_file(file_path)

• **Λειτουργία:**
• Διαβάζει ένα αρχείο που περιέχει περιγραφές του προβλήματος στο **PDDL format**.
• **Είσοδος:**
• file_path: Διαδρομή του αρχείου.
• **Διαδικασία:**
• Βρίσκει τη λίστα αντικειμένων (:objects).
• Εντοπίζει την αρχική κατάσταση (:INIT) και το στόχο (:goal).
• Φιλτράρει τις σχετικές πληροφορίες (π.χ. ποιοι κύβοι είναι στο τραπέζι, ποιοι είναι πάνω σε άλλους).
• **Έξοδος:** Επιστρέφει τις αρχικές συνθήκες των κύβων.

2. Συνάρτηση parse_config(objects, config_strings)

Η συνάρτηση **parse_config(objects, config_strings)** χρησιμοποιείται για τη

• **Λειτουργία:**
• Δημιουργία της διάταξης των κύβων **(config)** από τα δεδομένα αρχικής ή τελικής κατάστασης.
• **Είσοδος:**
• objects: Λίστα με τα ονόματα των κύβων.
• config_strings: Λίστα με συμβολοσειρές που περιγράφουν τη διάταξη (π.χ. "ONTABLE A", "ON B A").
• **Διαδικασία:**

1. **Αρχικοποίηση config**
   • Δημιουργεί έναν πίνακα **config** όπου κάθε κύβος ξεκινά με (-1, -1), που σημαίνει ότι είναι **ελεύθερος** και **στο τραπέζι**.
2. **Επεξεργασία κάθε γραμμής** από το config_strings:
   • Αν η γραμμή περιέχει **ONTABLE X**, τότε ο κύβος X:

- Ορίζεται ως **στο τραπέζι** (configX2 = -1).
- Δεν έχει τίποτα από πάνω (configX1 = -1).
- Αν η γραμμή περιέχει **ON A B**, τότε:
- Ο A τοποθετείται πάνω στον B (configA2 = B).
- Ο B αποκτά τον A από πάνω (configB1 = A).
- **Έξοδος:**
- Επιστρέφει τη λίστα config με την τελική διάταξη των κύβων.

3. dfs_search(initial_state, goal_config, objects, max_depth, max_cost)

• **Υλοποιεί αναζήτηση σε βάθος (Depth-First Search - DFS).**
• Χρησιμοποιεί **στοίβα (stack)** για την αποθήκευση των καταστάσεων.
• **Διαδικασία:**
2. Ξεκινά με την αρχική κατάσταση (initial_state).
3. Παίρνει το τελευταίο στοιχείο από τη στοίβα.
4. Εξετάζει αν είναι η επιθυμητή κατάσταση (goal_config).
5. Αν όχι, επεκτείνει τους γειτονικούς κόμβους και τους προσθέτει στη στοίβα.
6. Επαναλαμβάνει μέχρι να βρεθεί λύση ή να εξαντληθούν οι επιλογές.

• **Περιορισμοί:**
• Χρησιμοποιεί όριο βάθους (max_depth) για αποφυγή άπειρης αναζήτησης.
• Δεν εγγυάται τη βέλτιστη λύση.

4. bfs_search(initial_state, goal_config, objects)

• **Υλοποιεί αναζήτηση κατά πλάτος (Breadth-First Search - BFS).**
• Χρησιμοποιεί **ουρά (queue)** για αποθήκευση καταστάσεων.
• **Διαδικασία:**

1. Ξεκινά με την αρχική κατάσταση (initial_state).
2. Παίρνει το πρώτο στοιχείο από την ουρά (frontier).
3. Εξετάζει αν είναι η επιθυμητή κατάσταση (goal_config).
4. Αν όχι, προσθέτει όλους τους γειτονικούς κόμβους στο τέλος της ουράς.
5. Συνεχίζει μέχρι να βρεθεί λύση ή να εξαντληθούν οι επιλογές.
   - **Πλεονεκτήματα:**
   - Εγγυάται τη **βέλτιστη λύση** (ελάχιστες κινήσεις).
   - Μπορεί να γίνει αργός σε μεγάλα προβλήματα λόγω της εκθετικής αύξησης των κόμβων.

5. trace_solution(goal_state)

Ανακατασκευάζει τη **σειρά των ενεργειών** που οδήγησαν από την αρχική κατάσταση στη λύση.

Λειτουργία:

1. Ξεκινά από την τελική κατάσταση (goal_state).
2. **Ακολουθεί τα "parent" states** (καταστάσεις γονείς) προς τα πίσω.
3. Για κάθε κατάσταση, **καταγράφει την ενέργεια (action)** που την δημιούργησε.
4. Συνεχίζει μέχρι να φτάσει στην αρχική κατάσταση.
5. **Επιστρέφει τη λίστα ενεργειών** που σχηματίζουν τη λύση.

6. simple_optimal_filter(goal_config, objects)

• **Λειτουργία:** Απλοποιεί τη λύση, **φιλτράροντας τις απαραίτητες μετακινήσεις**. Καλείται όταν ολοι οι κύβοι είναι πάνω στο τραπέζι.
• **Διαδικασία:**

1. Εντοπίζει τους **κύβους βάσης** (αυτούς που είναι στο τραπέζι στο goal_config).
2. **Ακολουθεί την αλυσίδα** των κύβων προς τα πάνω, διατηρώντας μόνο τις μετακινήσεις που συμβάλλουν στον τελικό στόχο.
3. Επιστρέφει τη λίστα των **απαραίτητων κινήσεων**.

7. dp_filter(actions, begin_config, objects)

• **Λειτουργία:** Εφαρμόζει **δυναμικό προγραμματισμό (Dynamic Programming - DP)** για **βελτιστοποίηση της λύσης**.
Καλείται όταν δεν είναι όλοι οι κύβοι στο τραπέζι.
• **Διαδικασία:**

1. Προσομοιώνει την ακολουθία των ενεργειών (actions).
2. Αποθηκεύει τις ενδιάμεσες καταστάσεις (states) μετά από κάθε κίνηση.
3. Ελέγχει αν μπορούν να αφαιρεθούν περιττές κινήσεις, ώστε η λύση να γίνει **πιο σύντομη και αποδοτική**.

Ροή του Προγράμματος

Διαβάζεται το αρχείο PDDL μέσω **parse_file**().

Η **parse_config()** καλείται δύο φορές:

1. Για την **αρχική κατάσταση** των κύβων: begin_config
2. Για τον **στόχο της αναζήτησης** : goal_config

Καταγράφεται ο αρχικός χρόνος **start_time**.
Καλείται ο αλγόριθμος που ζητήθηκε σαν όρισμα **dfs_search()** ή **bfs_search()**.
Η **dfs_search()** σε περίπτωση που δεν μπορεί να βρει λύση, διακόπτει το πρόγραμμα.

Καλείται η συνάρτηση **trace_solution()** που καταγράφει τις κινήσεις που οδήγησαν στην τελική διάταξη στόχο.
Όλο το σύνολο των κινήσεων γράφεται στο αρχείο all_actions.txt

Καλείται η **simple_optimal_filter()** ή η **dp_filter()** ανάλογα αν είναι όλοι οι κύβοι πάνω στο τραπέζι ή όχι,
με σκοπό να φιλτράρουν το σύνολο των κινήσεων και να κρατήσουν την βέλτιστη λύση.

Τυπώνεται ο χρόνος που χρειάσθηκε από το χρονικό σημείο **start_time**.
Η βέλτιστη λύση γράφεται στο αρχείο **solution.txt**

Παράδειγμα εκτέλεσης του Προγράμματος

```
Rscript blocks.R depth probBLOCKS-4-0.pddl solution.txt

Initial Configuration :
A
B
C
D
Goal Configuration :
A | B | C | D
running_time:  0.1243179
Optimized actions written to solution.txt
```

```
# print the optimal solution

cat solution.txt

Move(B, table, A)
Move(C, table, B)
Move(D, table, C)
```

Σύγκριση Αλγοριθμών

pddl	DFS time	BFS time	DFS moves	BFS moves
probBLOCKS-4-0.pddl	0.12696	0.126687	3	3
probBLOCKS-4-1.pddl	0.150434	0.1639769	5	5
probBLOCKS-4-2.pddl	0.159394	0.1578958	3	3
probBLOCKS-5-0.pddl	2.423042	0.7053611	7	6

pddl	DFS time	BFS time	DFS moves	BFS moves
probBLOCKS-5-1.pddl	0.9475911	0.987726	5	5
probBLOCKS-5-2.pddl	-	1.422243	-	8