PROGETTO INTELLIGENZA ARTIFICIALE

Giulia Perri 242645 , Antonio Paonessa 252318,
Desirè Chiappetta 257192

1 Modellazione

Nel file di dominio abbiamo delineato gli elementi universali del problema in esame, ossia tutti quei fattori che restano costanti a prescindere dal contesto particolare che intendiamo affrontare. In particolare, abbiamo definito: requisiti, tipi, funzioni, predicati e azioni.

1.1 Dominio senza il carrier

Il primo dominio che abbiamo implementato prevede le seguenti condizioni iniziali sono:

- Ogni robot può portare solo una box alla volta;
- Ogni workstation ha una specifica locazione;
- Ogni box è ad una specifica locazione e può essere riempito da uno specifico contenuto;
- Ogni workstation può avere oppure non avere una box con uno specifico contenuto;
- Ci può essere più di una workstation in una location.

Requisiti

I **requisiti**, che indicano che un dato planner deve supportare un determinato aspetto del linguaggio, abbiamo usato i seguenti requisiti:

- Strips, consente l'uso di effetti di aggiunta e cancellazione di base;
- Typing, consente l'uso della tipizzazione per gli oggetti, simile alle classi e sotto-classi nella programmazione orientata agli oggetti;

```
(:requirements :strips :typing)
```

Tipi

I **tipi**, consentono di creare tipi di base e sottotipi ai quali possiamo applicare predicati.

- Location: Rappresenta un luogo generico.
- Box: Rappresenta una scatola.
- Agent: Rappresenta un agente, ad esempio un robot.
- Workstation: Rappresenta una postazione di lavoro.
- Content: Rappresenta un contenuto generico e astratto.
- Type: è utilizzato per specificare i vari tipi di contenuto che possono esistere nel dominio.

```
(:types
   location box content agent workstation type - object )
```

Predicati

I predicati rappresentano informazioni di stato e relazioni tra gli oggetti nel dominio di pianificazione.

- `(connected ?loc1 ?loc2 - location)`: Indica che due posizioni (`?loc1` e `?loc2`) sono collegate tra loro.
- ` (content_at_loc ?content - content ?loc - location) `: Indica che un contenuto (`?content`) si trova in una determinata posizione (`?loc`).
- ` (box_at_loc ?box - box ?loc - location) `: Indica che una scatola (`?box`) si trova in un determinato luogo (`?loc`).
- ` (agent_at_loc ?agent - agent ?loc - location) `: Indica che un agente (`?agent`) si trova in un determinato luogo(`?loc`).
- ` (ws_at_loc ?ws - workstation ?loc - location) `: Indica che una postazione di lavoro (`?ws`) si trova in un determinato luogo (`?loc`).
- `(is_type ?content - content ?t - type)`: Questo predicato indica che un determinato contenuto (`?content`) è di un tipo specifico (`?t`), rappresentando una relazione di tipo tra il contenuto e il suo tipo.
- `(empty_box ?box - box)`: Indica l'assenza di contenuto in una scatola (`?box`).
- `(filled_box ?box - box ?content - content)`: Indica che una scatola (`?box`) è riempita con un determinato contenuto (`?content`).
- `(box_at_ws ?box - box ?ws - workstation)`: Indica che una scatola (`?box`) si trova presso una postazione di lavoro (`?ws`).
- `(content_type_at_workstation ?ws - workstation ?t - type)`: Indica che un contenuto di un certo tipo (`?t`) si trova presso una postazione di lavoro (`?ws`).
- `(box_at_agent ?box - box ?agent - agent)`: Indica che una scatola (`?box`) è posseduta da un certo agente (`?agent`).
- `(empty_agent ?agent - agent)`: Indica l'agente (`?agent`) non possiede nessuna scatola.

```
(:predicates
  (connected ?loc1 ?loc2 - location)
  (content_at_loc ?content - content ?loc - location)
  (box_at_loc ?box - box ?loc - location)
  (agent_at_loc ?agent - agent ?loc - location)
  (ws_at_loc ?ws - workstation ?loc - location)
  (is_type ?content - content ?t - type)
  (empty_box ?box - box)
  (filled_box ?box - box ?content - content)
  (box_at_ws ?box - box ?ws - workstation)
  (content_type_at_ws ?t - type ?ws - workstation)
  (box_at_agent ?box - box ?agent - agent)
  (empty_agent ?agent - agent)
)
```

Azioni

Un'azione, generalmente, definisce una trasformazione dello stato del mondo.

a) Pick_up:

Si procede a definire la action relativa al ritiro di una box da una location. I parametri utili alla action in oggetto sono: la location, la scatola da ritirare e l'agente robotico.

Le precondizioni che consentono l'esecuzione prevedono che:

- l'agente robotico e la box si trovino nella stessa locazione;
- L'agente robotico è libero.

Gli effetti dell'esecuzione portano ad uno stato in cui:

- L'agente robotico è occupato;
- la box non risulta più alla location.

```
(:action pick_up
  :parameters (
    ?loc - location
    ?box - box
    ?agent - agent
  )
  :precondition (and
    (empty_agent ?agent)
    (agent_at_loc ?agent ?loc)
    (box_at_loc ?box ?loc)
   )
  :effect (and
    (not (empty_agent ?agent))
    (box_at_agent ?box ?agent)
    (not (box_at_loc ?box ?loc))
  )
)
```

b) Move:

Si procede a definire la action relativa allo spostamento di un agente da una location all'altra. I parametri utili alla action in oggetto sono: le location (from e to) e l'agente robotico coinvolto.

Le precondizioni che consentono l'esecuzione prevedono che:

- l'agente robotico si trovi nella location di partenza;
- le locations siano connesse.

Gli effetti dell'esecuzione portano ad uno stato in cui:

- l'agente robotico non si trovi nella location di partenza;
- l'agente robotico si trovi nella location di arrivo.

```
(:action move
        :parameters (?from ?to - location ?agent - agent)
        :precondition (and
            (at-agent ?agent ?from)
            (connected ?from ?to)
        )
```

```
        :effect (and
            (not (at-agent ?agent ?from))
            (at-agent ?agent ?to)
        )
    )
```

c) Deliver:

Si procede a definire la action relativa alla consegna di una box ad una workstation. I parametri utili alla action in oggetto sono: la box da consegnare, la workstation a cui consegnare la box, la location in cui si trova la workstation, l'agente robotico coinvolto.

Le precondizioni che consentono l'esecuzione prevedono che:

- l'agente robotico si trovi nella stessa location della workstation a cui consegnare la box;
- l'agente robotico stia portando la box da consegnare.

Gli effetti dell'esecuzione portano ad uno stato in cui:

- l'agente robotico non porta più la box;
- la box si trova nella workstation a cui è stata consegnata.

```
(:action deliver
  :parameters (
    ?box - box
    ?ws - workstation
    ?location - location
    ?agent - agent
  )
  :precondition (and
    (ws_at_loc ?ws ?location)
    (agent_at_loc ?agent ?location)
    (box_at_agent ?box ?agent
)
  )
  :effect (and
    (not (box_at_agent ?box ?agent))
    (empty_agent ?agent)
    (box_at_ws ?box ?ws)
  )
)
```

d) Fill:

Si procede a definire la action relativa al riempimento di una box con un contenuto. I parametri utili alla action in oggetto sono: la box da riempire, il contenuto della box, la location in cui si trova il contenuto, l'agente robotico coinvolto. Le precondizioni che consentono l'esecuzione prevedono che:

- la box sia vuota;
- l'agente robotico stia portando la box da riempire;
- il contenuto e l'agente robotico si trovino nella stessa location.

Gli effetti dell'esecuzione portano ad uno stato in cui:

- la box non è più vuota;
- la box è riempita con il contenuto;
- il contenuto non si trova più nella location.

```
(:action fill
  :parameters (
    ?box - box
    ?content - content
    ?loc - location
    ?agent - agent
  )
  :precondition (and
    (content_at_loc ?content ?loc)
    (agent_at_loc ?agent ?loc)
    (box_at_agent ?box ?agent)
    (empty_box ?box)
  )
  :effect (and
    (not (empty_box ?box))
    (filled_box ?box ?content)
    (not (content_at_loc ?content ?loc))
  )
)
```

e) Empty:

Si procede a definire la action relativa allo svuotamento di una box. I parametri utili alla action in oggetto sono: la box da svuotare, il contenuto della box, la workstation che riceve il contenuto, l'agente robotico coinvolto e la location. Le precondizioni che consentono l'esecuzione prevedono che:

- la box sia riempita con il contenuto da consegnare;
- la box si trovi nella workstation che deve ricevere il contenuto;
- l'agente e la workstation si trovino nella stessa locazione.

Gli effetti dell'esecuzione portano ad uno stato in cui:

- la box non ha più contenuto;
- la box viene rilasciata nella locazione;
- la workstation ha il contenuto della box.

```
(:action empty
  :parameters (
    ?box - box
    ?content - content
    ?t - type
    ?ws - workstation
    ?agent - agent
```

```
    ?loc - location
  )
  :precondition (and
    (is_type ?content ?t)
    (box_at_ws ?box ?ws)
    (ws_at_loc ?ws ?loc)
    (filled_box ?box ?content)
    (agent_at_loc ?agent ?loc)
  )
  :effect (and
    (not (filled_box ?box ?content))
    (empty_box ?box)
    (box_at_loc ?box ?loc)
    (not (box_at_ws ?box ?ws))
    (content_type_at_ws ?t ?ws)
  )
)
```

1.2 Dominio con il carrier

Abbiamo sviluppato un nuovo modello per la gestione del carrier, che estende il primo dominio. In questa versione, abbiamo aggiunto una nuova caratteristica: ogni robot ora ha una capacità massima di carico.

La capacità di carico di un carrier è definita dal numero massimo di place assegnati.

Nuovi tipi

Per questa nuova versione, abbiamo aggiunto due nuovi tipi:

- Carrier: Rappresenta un mezzo di trasporto.
- Place: Rappresenta una locazione del Carrier.

```
(:types
   ... carrier place - object )
```

Nuovi predicati

- `(box-at-place ?box - box ?place - place)` : Indica che una scatola (`?box`) si trova presso una locazione del carrier (`?place`).
- `(empty-place  ?place - place)` : Indica l'assenza di scatola in una locazione del carrier (`?place`).
- `(place-at-carrier  ?place- place ?carrier - carrier)` : Indica che la locazione del carrier (`?place`) è relativa ad un determinato mezzo di trasporto (`? carrier `).
- `(carrier-at-agent ?carrier - carrier ?agent - agent)` : Indica che il mezzo di trasporto (`?carrier`) è relativa ad un agente (`? agent`).

```
(:predicates
```

```
  ...
  (box_at_place ?box - box ?place - place)
  (empty-place ?place - place)
  (place-at-carrier ?place - place ?carrier - carrier)
  (carrier-at-agent ?carrier - carrier ?agent - agent)
)
```

Azioni modificate

a) Pick up con il carrier

In questo caso, l'azione di pick up permette all'agente robotico di caricare in un specifico place del proprio carrier una box, solo se quella locazione del carrier risulta non occupata.

```
(:action pick_up
  :parameters (
    ?loc - location
    ?box - box
    ?agent - agent
    ?carrier - carrier
    ?place - place)
  :precondition (and
    (agent_at_loc ?agent ?loc)
    (box_at_loc ?box ?loc)
    (carrier_at_agent ?carrier ?agent)
    (place_at_carrier ?place ?carrier)
    (empty_place ?place)
  )
  :effect (and
    (box_at_place ?box ?place)
    (not (empty_place ?place))
    (not (box_at_loc ?box ?loc))
  )
)
```

b) Move

Il funzionamento di questa azione non prevede l'utilizzo dei place e quindi rimane invariato.

c) Deliver con il carrier

L'azione di deliver permette all'agente robotico di consegnare una box, posizionata in un specifico place del proprio carrier, ad una certa workstation.

```
(:action deliver
  :parameters (
    ?box - box
    ?ws - workstation
    ?location - location
    ?agent - agent
    ?carrier - carrier
    ?place - place
```

```
  )
  :precondition (and
    (ws_at_loc ?ws ?location)
    (agent_at_loc ?agent ?location)
    (carrier_at_agent ?carrier ?agent)
    (place_at_carrier ?place ?carrier)
    (box_at_place ?box ?place)
  )
  :effect (and
    (not (box_at_place ?box ?place))
    (empty_place ?place)
    (box_at_ws ?box ?ws)
  )
)
```

d) Fill con il carrier

L'azione di fill permette ad un agente robotico di riempire una box vuota posizionata in una specifica locazione del carrier con un certo contenuto.

```
(:action fill
  :parameters (
    ?box - box
    ?content - content
    ?loc - location
    ?agent - agent
    ?carrier - carrier
    ?place - place
  )
  :precondition (and
    (content_at_loc ?content ?loc)
    (agent_at_loc ?agent ?loc)
    (carrier_at_agent ?carrier ?agent)
    (place_at_carrier ?place ?carrier)
    (box_at_place ?box ?place)
    (empty_box ?box)
  )
  :effect (and
    (not (empty_box ?box))
    (filled_box ?box ?content)
    (not (content_at_loc ?content ?loc))
  )
)
```

e) Empty

Il funzionamento di questa azione non prevede l'utilizzo dei place e quindi rimane invariato.

2 Classical Planning

Sono state realizzate due istanze del problema, una che rispetta il primo dominio (punto1/domain0.pddl) realizzato al punto 1 e l'altra che include la capacità massima dei robot (punto1/domain.pddl).

Inoltre, è stato sviluppato un planner che utilizza l'algoritmo A* e un'euristica specifica per il dominio con il carrier, poiché il primo dominio può essere interpretato come un sistema con un unico place e un solo carrier.

2.1 Istanza 1

Nella prima istanza tutte le box e tutti i content sono collocati inizialmente in una singola location, la *central_warehouse*, non dotata di workstation; un singolo agente robotico, anch'esso collocato inizialmente alla *central_warehouse*, si occupa di consegnare le box alle workstation. Il goal richiede che alcune workstation ricevano un content, altre ne ricevano più di uno, altre ancora non ricevano nulla.
È stato scelto di definire un problema con due location aggiuntive, oltre alla *central_warehouse*, cinque workstation e tre diversi tipi di content. Il goal prevede che la ws1 e la ws3 ricevano due content, la ws2 e la ws4 ne ricevano solo uno e la ws5 non riceva niente.

```
(define (problem instance1) (:domain industrial_manufacturing_no_carrier)
  (:objects
    ; define the world
    cw loc1 loc2 - location
    ws1 ws2 ws3 ws4 ws5 - workstation
    b1 b2 - box
    a1 - agent
    valve1 valve2 bolt1 bolt2 hammer1 hammer2 - content
    valve bolt hammer - type
  )
  (:init
    ; define the type of content
    (is_type valve1 valve)
    (is_type valve2 valve)
    (is_type bolt1 bolt)
    (is_type bolt2 bolt)
    (is_type hammer1 hammer)
    (is_type hammer2 hammer)
    ; define the empty box located at central_warehouse
    (box_at_loc b1 cw)
    (box_at_loc b2 cw)
    (empty_box b1)
    (empty_box b2)
    ; define content located at central_warehouse
    (content_at_loc valve1 cw)
    (content_at_loc valve2 cw)
    (content_at_loc bolt1 cw)
    (content_at_loc bolt2 cw)
    (content_at_loc hammer1 cw)
    (content_at_loc hammer2 cw)
    ; define workstation located at central_warehouse
    (ws_at_loc ws1 loc1)
    (ws_at_loc ws2 loc1)
    (ws_at_loc ws3 loc2)
```

```
    (ws_at_loc ws4 loc2)
    (ws_at_loc ws5 loc2)
    ; define workstation located at central_warehouse
    (connected cw loc1)
    (connected cw loc2)
    (connected loc1 cw)
    (connected loc2 cw)
    ; define agent located initially located at central_warehouse
    (agent_at_loc a1 cw)
    (empty_agent a1)
  )
  (:goal
    ; define goals to satisfied
    (and
      (content_type_at_ws bolt ws1)
      (content_type_at_ws valve ws1)
      (content_type_at_ws bolt ws2)
      (content_type_at_ws hammer ws3)
      (content_type_at_ws valve ws3)
      (content_type_at_ws hammer ws4)
   )
  )
)
```

Abbiamo eseguito l'istanza con Fast Forward (FF), che esplora lo spazio degli stati partendo dallo stato iniziale e cercando di raggiungere uno stato che soddisfi l'obiettivo, utilizzando il seguente comando:

```
(planutils) root@bf885b8679bc:/ProgettoAI/punto1$ ff domain0.pddl
./problems/instance1.pddl
```

Questo comando ha utilizzato il dominio definito nel file domain0.pddl e l'istanza del problema specificata nel file ./problems/instance1.pddl.

L'output è il seguente:

```
step    0: PICK_UP CW B2 A1
        1: FILL B2 BOLT1 CW A1
        2: MOVE CW LOC1 A1
        3: DELIVER B2 WS2 LOC1 A1
        4: EMPTY B2 BOLT1 BOLT WS2 A1 LOC1
        5: PICK_UP LOC1 B2 A1
        6: MOVE LOC1 CW A1
        7: FILL B2 BOLT2 CW A1
        8: MOVE CW LOC1 A1
        9: DELIVER B2 WS1 LOC1 A1
       10: EMPTY B2 BOLT2 BOLT WS1 A1 LOC1
       11: PICK_UP LOC1 B2 A1
       12: MOVE LOC1 CW A1
```

```
       13: FILL B2 VALVE1 CW A1
       14: MOVE CW LOC1 A1
       15: DELIVER B2 WS1 LOC1 A1
       16: EMPTY B2 VALVE1 VALVE WS1 A1 LOC1
       17: MOVE LOC1 CW A1
       18: PICK_UP CW B1 A1
       19: FILL B1 VALVE2 CW A1
       20: MOVE CW LOC2 A1
       21: DELIVER B1 WS3 LOC2 A1
       22: EMPTY B1 VALVE2 VALVE WS3 A1 LOC2
       23: PICK_UP LOC2 B1 A1
       24: MOVE LOC2 CW A1
       25: FILL B1 HAMMER1 CW A1
       26: MOVE CW LOC2 A1
       27: DELIVER B1 WS3 LOC2 A1
       28: EMPTY B1 HAMMER1 HAMMER WS3 A1 LOC2
       29: PICK_UP LOC2 B1 A1
       30: MOVE LOC2 CW A1
       31: FILL B1 HAMMER2 CW A1
       32: MOVE CW LOC2 A1
       33: DELIVER B1 WS4 LOC2 A1
       34: EMPTY B1 HAMMER2 HAMMER WS4 A1 LOC2
```

2.2 Istanza 2

La seconda istanza, rispetto a quella discussa in precedenza, considera più agenti robotici che possano trasportare più di una scatola alla volta per mezzo di un carrier, la cui capacità è stata modellata tramite il tipo *place*.

I due agenti robotici a1 e a2 sono caratterizzati rispettivamente da:

- c1 carrier: con place p11 e p12;
- c2 carrier: con place p21 e p22.

```
(define (problem instance2) (:domain industrial_manufacturing)
(:objects
  ; define the world
  cw loc1 loc2 - location
  ws1 ws2 ws3 ws4 ws5 - workstation
  b1 b2 b3 b4 - box
  a1 a2 - agent
  c1 c2 - carrier
  p11 p12 p21 p22 - place
  valve1 valve2 bolt1 bolt2 hammer1 hammer2 - content
  valve bolt hammer - type
)
```

```
; define the empty box located at central_warehouse
(box_at_loc b1 cw)
```

```
(box_at_loc b2 cw)
(box_at_loc b3 cw)
(box_at_loc b4 cw)
(empty_box b1)
(empty_box b2)
(empty_box b3)
(empty_box b4)
```

```
; define agent located initially located at central_warehouse
(agent_at_loc a1 cw)
(carrier_at_agent c1 a1)
(place_at_carrier p11 c1)
(place_at_carrier p12 c1)
(empty_place p11)
(empty_place p12)
(agent_at_loc a2 cw)
(carrier_at_agent c2 a2)
(place_at_carrier p21 c2)
(place_at_carrier p22 c2)
(empty_place p21)
(empty_place p22)
```

Abbiamo eseguito l'istanza con Fast Forward (FF) utilizzando il seguente comando:

```
(planutils) root@bf885b8679bc:/workspace/ProgettoAI/punto1$ ff domain.pddl
./problems/instance2.pddl
```

Questo comando ha utilizzato il dominio definito nel file domain.pddl e l'istanza del problema specificata nel file ./problems/instance2.pddl. Il risultato è il seguente:

```
step    0: MOVE CW LOC1 A1
        1: PICK_UP CW B4 A2 C2 P21
        2: FILL B4 BOLT1 CW A2 C2 P21
        3: MOVE CW LOC1 A2
        4: DELIVER B4 WS2 LOC1 A2 C2 P21
        5: MOVE LOC1 CW A2
        6: PICK_UP CW B3 A2 C2 P21
        7: EMPTY B4 BOLT1 BOLT WS2 A1 LOC1
        8: FILL B3 BOLT2 CW A2 C2 P21
        9: MOVE CW LOC1 A2
       10: PICK_UP LOC1 B4 A1 C1 P11
       11: DELIVER B4 WS1 LOC1 A1 C1 P11
       12: DELIVER B3 WS1 LOC1 A2 C2 P21
       13: MOVE LOC1 CW A2
       14: PICK_UP CW B2 A2 C2 P21
       15: EMPTY B3 BOLT2 BOLT WS1 A1 LOC1
       16: FILL B2 VALVE1 CW A2 C2 P21
       17: PICK_UP CW B1 A2 C2 P22
       18: FILL B1 HAMMER1 CW A2 C2 P22
       19: MOVE CW LOC2 A2
```

```
        20: DELIVER B2 WS3 LOC2 A2 C2 P21
        21: PICK_UP LOC1 B3 A1 C1 P11
        22: EMPTY B2 VALVE1 VALVE WS3 A2 LOC2
        23: MOVE LOC1 CW A1
        24: FILL B3 VALVE2 CW A1 C1 P11
        25: MOVE CW LOC1 A1
        26: DELIVER B3 WS1 LOC1 A1 C1 P11
        27: EMPTY B3 VALVE2 VALVE WS1 A1 LOC1
        28: DELIVER B1 WS3 LOC2 A2 C2 P22
        29: PICK_UP LOC2 B2 A2 C2 P21
        30: MOVE LOC2 CW A2
        31: FILL B2 HAMMER2 CW A2 C2 P21
        32: MOVE CW LOC2 A2
        33: DELIVER B2 WS4 LOC2 A2 C2 P21
        34: EMPTY B1 HAMMER1 HAMMER WS3 A2 LOC2
        35: EMPTY B2 HAMMER2 HAMMER WS4 A2 LOC2

time spent:
     0.00 seconds instantiating 664 easy, 0 hard action templates
     0.00 seconds reachability analysis, yielding 107 facts and 472 actions
     0.00 seconds creating final representation with 107 relevant facts
     0.00 seconds building connectivity graph
     0.00 seconds searching, evaluating 364 states, to a max depth of 5
     0.00 seconds total time
```

2.3 Planner

In questa sezione ci si concentrerà sulla parte di sviluppo di un planner utilizzando la libreria PDDL4J. Questa libreria è un potente strumento che offre una vasta gamma di funzionalità per la creazione e l'utilizzo di solver, partendo da descrizioni di domini e problemi scritti in PDDL (Planning Domain Definition Language).

Abbiamo sfruttato PDDL4J per definire un solver specifico, progettato su misura per affrontare il problema discusso in precedenza, descritto nel file domain.pddl. L'obiettivo principale era creare un planner capace di risolvere efficacemente le varie istanze del problema, rispettando le specifiche del dominio.

Per questo scopo, abbiamo sviluppato la classe IMPlanner, che include il metodo *solve*, utilizzato per risolvere un problema specifico.

```
public class IMPlanner extends AbstractPlanner {
  public static void main(String[] args) {
    // The path to the benchmarks directory
   final String benchmark = "src/main/java/fr/uga/pddl4j/myproject/benchmark/";
   final String domainName = "domain.pddl";
   final String problemName = "problems/instance2.pddl";
   final String outputName = "plans/instance2.pddl.txt";

    // Creates the planner
    final IMPlanner planner = new IMPlanner();
    // Sets the domain of the problem to solve
    planner.setDomain(benchmark + domainName);
    // Set the problem to solve
```

```
    planner.setProblem(benchmark + problemName);
    // Set the timeout of the search
    planner.setTimeout(1000);
    // Sets log level
    planner.setLogLevel(LogLevel.INFO);
    // Set split size and heuristic weight
    planner.setSplitSize(3);
    planner.setHeuristicWeight(1.5);

    // Solve and print the result on terminal
    try {
        planner.solve();
    } catch (InvalidConfigurationException e) {
        e.printStackTrace();
    }
    // Store the result
    ....
}
```

Nel *main*, oltre a specificare il dominio e il problema su cui lavorare, è possibile impostare due parametri cruciali:

1. La ***splitSize***: questo parametro determina la dimensione massima dei sottoproblemi in cui il problema principale viene suddiviso, consentendo una gestione più efficiente e mirata delle istanze complesse.
2. Il **hWeight**: un parametro che influenza il comportamento dell'algoritmo di ricerca utilizzato dal planner, modificando il peso della euristica nella ricerca dell'ottimo.

Un aspetto centrale del nostro approccio è stato lo *splitting* (suddivisione) del problema. L'idea è stata quella di ridurre la complessità delle istanze, suddividendo il goal complessivo in sotto-goal più gestibili.

Questa suddivisione è stata effettuata secondo due criteri:

1. In base alla *location* delle richieste da soddisfare.
2. In base alle *workstations* da servire in una determinata location.

Il parametro *splitSize* definisce quante richieste si desidera soddisfare contemporaneamente in ogni location. Questo approccio è giustificato dal fatto che suddividere il problema per location consente di concentrare gli agenti sul completamento dei task in una specifica area, prima di passare a quella successiva. A sua volta, la suddivisione per workstation permette di semplificare ulteriormente il problema all'interno di ogni location.

```
   private Problem[] buildSubproblems(Problem problem) {
        DefaultParsedProblem pp = problem.getParsedProblem();

        // No split (Full problem)
        if (this.splitSize==0) return new Problem[]{problem};

        // Extraction of locations and workstation from the problem
       // code...
```

```
        // Splitting of goals
        // code ...

        // Sub problems building and instantiation
        System.out.println("\n=========================================...");
        System.out.println("SubProblems instantiated with:");
        System.out.println("-----------------------------------------\n");
        Problem[] ret = new DefaultProblem[splitIndex];
        for (int i = 0; i < ret.length; i++) {
             Expression<String> tmp = new Expression<>();
             tmp.setChildren(splitGoals.get(i));
             pp.setGoal(tmp);
             ret[i] = new DefaultProblem(pp);
             ret[i].instantiate();
         }
         System.out.println("\n=========================================...");
         return ret;
  }
```

Il metodo solve si basa, se richiesto, sulla suddivisione del problema in sottoproblemi utilizzando il metodo buildSubproblems. Per risolvere ciascun sottoproblema, viene impiegata una versione modificata dell'algoritmo A* (IMAStar) già presente nella libreria pddl4j, alla quale è stata aggiunta la capacità di avviare una ricerca a partire da uno stato diverso da quello iniziale. In questo modo, una volta completato un piano parziale (sub-plan), lo stato di uscita diventa lo stato iniziale per il sottoproblema successivo.

Questo approccio permette di gestire in modo più efficiente istanze con obiettivi complessi, risolvendo i sotto-goal in sequenza e indirizzando gli agenti verso una location alla volta.

```
@Override
public Plan solve(Problem problem) throws ProblemNotSupportedException {
    // Strategy & Heuristic
    SearchStrategy.Name strategyName = SearchStrategy.Name.ASTAR;
    StateHeuristic.Name heuristic = StateHeuristic.Name.IM_HEURISTIC;
    int timeout = 1000000;

    System.out.println("\n************************************************");
    System.out.println("Start solving using "+strategyName+" with "+heuristic+"
heuristic");
    System.out.println("
      + h weight="+heuristicWeight+"\n
      + split factor="+splitSize+"\n
      + debug mode="+debug);
    System.out.println("********************************************** \n");

    // Utility structs for sub problem splitting
    Problem[] subProblems = buildSubproblems(problem);
    Node[] subSolutions = new Node[subProblems.length];
    Plan[] subplans = new Plan[subProblems.length];

    // Solving with A* and IMHeuristic
    long searchTime=0, memoryUsed=0;
```

```
    for (int i = 0; i<subProblems.length; i++){
        if (i==0) {
            // solve sub problem 0
            StateSpaceSearch alg = StateSpaceSearch.getInstance(strategyName,
heuristic, heuristicWeight);
            subSolutions[i] = alg.searchSolutionNode(subProblems[i]);
            subplans[i] = alg.extractPlan(subSolutions[i],subProblems[i]);
            searchTime += alg.getSearchingTime();
            memoryUsed += alg.getMemoryUsed();
        }
        else {
            // solve sub problem from prev sub solution founded
            subSolutions[i - 1].setParent(null);
            StateSpaceSearch alg = new IMAStar(timeout, heuristic,
heuristicWeight, subSolutions[i - 1]);
            subSolutions[i] = alg.searchSolutionNode(subProblems[i]);
            subplans[i] = alg.extractPlan(subSolutions[i], subProblems[i]);
            searchTime += alg.getSearchingTime();
            memoryUsed += alg.getMemoryUsed();
        }
        System.out.println("\n=======================================");
        System.out.printf("Plan %d: \n",i);
        System.out.println("-----------------------------------------");
        if(subplans[i] != null)
            System.out.print(problem.toString(subplans[i]));
        else
            System.out.println("Plan empty");
        System.out.println("=========================================");
    }

     // Print the stats
    double searchTimeInSeconds = searchTime / 1000.0;
    double memoryUsedInMB = memoryUsed / (1024.0 * 1024.0);
    System.out.println("\n=======================================");
    System.out.println("Search stats:");
    System.out.println("-----------------------------------------------");
    System.out.println("
      + Search time: " + String.format("%.2f", searchTimeInSeconds) +
      " seconds" + "\n + Memory used: " + String.format("%.2f",memoryUsedInMB)
      + " MB");
    System.out.println("=========================================");

    // Return the full plan
    List<Action> sol = new LinkedList<>();
    for(Plan p: subplans)
        if(p != null)
            sol.addAll(p.actions());
    Plan solution = new SequentialPlan();
    Iterator<Action> it = sol.iterator();
    for(int i = 0; i<sol.size(); i++)
        solution.add(i,it.next());
    return solution;
}
```

2.5 Euristica

Un'euristica specifica per il dominio è stata definita nella classe ***IMHeuristic***.

La classe IMHeuristic è un'estensione di RelaxedGraphHeuristic e rappresenta un'euristica sviluppata per stimare il costo di raggiungere un obiettivo a partire da un dato stato in un problema di pianificazione. L'euristica sfrutta informazioni estratte dai fluents (espressioni logiche che descrivono lo stato del mondo) per valutare lo stato corrente e stimare il costo delle azioni necessarie per soddisfare l'obiettivo.

Lo sviluppo dell'euristica è stato il frutto di una serie di tentativi iterativi, che hanno permesso non solo di definirla, ma anche di calibrare attentamente i parametri impiegati nella valutazione degli stati. Questo processo di affinamento ha avuto l'obiettivo di ottenere una stima sempre più precisa della distanza dal goal, migliorando così l'efficacia complessiva del planner. Attraverso questi sforzi, è stato possibile raggiungere un compromesso tra complessità computazionale e accuratezza, elementi chiave per il successo del solver.

2.5.1 Inizializzazione e costruzione delle strutture di supporto

La costruzione dell'euristica inizia con il metodo costruttore IMHeuristic(Problem problem). In questo metodo, vengono inizializzate diverse strutture dati fondamentali:

- ***fluents***: Mappa che associa a ciascun indice una lista di stringhe rappresentanti i fluents estratti dal problema.
- ***placeOnCarrier***, ***carrierOnAgent***: Mappe che tracciano rispettivamente le associazioni tra luoghi e carrier, e tra carrier e agenti.
- ***typeOf***: Mappa che associa un contenuto al suo tipo.
- ***wsRequested***, ***typeRequested***: Strutture per gestire i tipi richiesti dalle workstations.

```
public IMHeuristic(Problem problem) {
    super(problem);
    // Extraction the fluents and initializing of support struct
    int index = 0;
    String predicate, content, type, place, carrier, agent;
    for (Fluent f : problem.getFluents()) {
        StringTokenizer st = new StringTokenizer(problem.toString(f), "( )", false);
        this.fluents.computeIfAbsent(index, k -> new ArrayList<>());
        while (st.hasMoreTokens())
            this.fluents.get(index).add(st.nextToken());
        // Collect default predicate
        Predicate = this.fluents.get(index).get(0);
        switch (predicate) {
            case "is_type":
                // is_type ?content - content ?t - type
                content=this.fluents.get(index).get(1);
                type=this.fluents.get(index).get(2);
                this.typeOf.put(content, type);
                break;
            case "place_at_carrier":
                // place_at_carrier ?place - place ?carrier - carrier
                place=this.fluents.get(index).get(1);
                carrier=this.fluents.get(index).get(2);
                this.placeOnCarrier.computeIfAbsent(carrier, k -> new ArrayList<>());
                this.placeOnCarrier.get(carrier).add(place);
                break;
            case "carrier_at_agent":
                // carrier_at_agent ?carrier - carrier ?agent - agent
                carrier=this.fluents.get(index).get(1);
                agent=this.fluents.get(index).get(2);
                this.carrierOnAgent.put(carrier, agent);
```

```
                break;
            }
            index++;
        }
```

I fluents vengono estratti dal problema e scomposti in token per individuare i predicati rilevanti come *is_type*, *place_at_carrier*, e *carrier_at_agent*. Ogni predicato viene processato e memorizzato nelle strutture dati corrispondenti.

```
  // Structs that contains the type requested by ws
    String ws;
    BitVector gpk = new BitVector(problem.getGoal());
    for (int p = gpk.nextSetBit(0); p >= 0; p = gpk.nextSetBit(p + 1)) {
        // content_type_at_ws ?t - type ?ws - workstation
        type = this.fluents.get(p).get(1);
        ws = this.fluents.get(p).get(2);
        this.typeRequested.add(type);
        this.wsRequested.computeIfAbsent(ws, k->new LinkedList<>());
        this.wsRequested.get(ws).add(type);
    }
}
```

Questa porzione di codice è responsabile della costruzione di due strutture dati cruciali per l'euristica: typeRequested e wsRequested. Entrambe queste strutture vengono utilizzate per monitorare le workstation (ws) e i tipi di contenuto (type) che ciascuna workstation richiede nel problema da risolvere.

- **typeRequested**: Viene utilizzata per sapere quali tipi di contenuti sono necessari nel goal complessivo, indipendentemente da quale workstation li richieda. Questo permette di verificare rapidamente se un contenuto è richiesto.
- **wsRequested**: Permette di monitorare le richieste specifiche per ogni workstation. Questo è utile per verificare se una workstation ha già ricevuto tutto ciò di cui ha bisogno o se deve ancora ricevere alcuni contenuti per soddisfare il goal.

Queste strutture forniscono un modo efficiente per valutare quanto lo stato corrente si avvicina al soddisfacimento del goal e per decidere quali azioni potrebbero essere più promettenti per progredire verso di esso.

2.5.2 Calcolo della stima per raggiungere il goal

Il metodo estimate è un componente fondamentale nel contesto del problema di pianificazione in esame, poiché si occupa di valutare quanto lo stato attuale del sistema sia vicino al raggiungimento dell'obiettivo finale (goal). Questo metodo, che sovrascrive il comportamento della funzione estimate ereditata dalla classe RelaxedGraphHeuristic, è cruciale per guidare il processo decisionale nel piano, orientando la selezione delle azioni successive sulla base di una valutazione quantitativa.

```
@Override
public int estimate(State state, Condition goal) {
    super.setGoal(goal);
```

```
    /**
    *       1. Estimated cost from current state
    * */
    ...

    /**
    *        2. For each available action for the current state, assign a value
    * */
    ...

   int estimated = evalState
        + (moves[0]<Integer.MAX_VALUE ? 2*actions[0] : 0)    // pick_up
        + (moves[1]<Integer.MAX_VALUE ? 3*actions[1] : 0)    // move
        + (moves[2]<Integer.MAX_VALUE ? 6*actions[2] : 0)    // deliver
        + (moves[3]<Integer.MAX_VALUE ? 4*actions[3] : 0)    // fill
        + (moves[4]<Integer.MAX_VALUE ? 9*actions[4] : 0);  // empty

    return estimated;
}
```

La logica di estimate si sviluppa in due fasi principali: **la valutazione dello stato corrente** e **la stima dei costi associati alle azioni disponibili**. La prima fase riguarda l'analisi del grado di soddisfazione degli obiettivi già raggiunti o ancora mancanti nello stato attuale. La seconda fase, invece, si concentra sulla previsione dell'impatto che le azioni possibili, se eseguite, avrebbero sulla capacità del sistema di avvicinarsi ulteriormente all'obiettivo.

L'output finale del metodo è una somma ponderata dei vari contributi determinati sia dalla valutazione dello stato corrente sia dai costi stimati per le azioni possibili. Questa somma fornisce una misura complessiva dell'efficacia dello stato attuale, influenzata dalle potenziali transizioni di stato che le azioni disponibili potrebbero innescare.

In altre parole, il valore restituito da estimate rappresenta una stima della "distanza" dal goal, tenendo conto sia della situazione corrente sia delle opportunità offerte dalle azioni future.

2.5.3 Valutazione dello stato corrente

Nella fase di valutazione dello stato corrente, il metodo prende in esame i *fluents* che descrivono lo stato attuale del sistema. Questi *fluents* rappresentano condizioni o proprietà che possono essere vere o false in un dato stato e sono utilizzati per determinare quanto lo stato corrente si avvicina al raggiungimento dell'obiettivo finale. La logica di valutazione si basa su un sistema di bonus e penalità, assegnati in base a specifiche condizioni verificate nello stato corrente.

Il codice seguente definisce i valori iniziali per i bonus e le penalità utilizzati durante la valutazione:

```
  /**
    * 1. Estimated cost from current state
    */
    int goalSatisfied = -10;
    int penaltyBoxAtLoc = +5;
    int penaltyContentAtCw = +1;
    int bonusTypeNecessary = -5;
    int malusNoTypeNecessary = +10;
```

```
    int bonusBoxAtPlace = -1;
    int penaltyBoxAtWs = +2;
    int penaltyAgentAtLoc = +10;
    int evalState = 0;
```

Questi valori determinano il costo stimato dello stato attuale:

- **goalSatisfied**: Rappresenta un bonus significativo (-10) quando un obiettivo specifico è già soddisfatto nello stato corrente.
- **penaltyBoxAtLoc**: Penalità (+5) assegnata se una scatola si trova in una posizione diversa dal magazzino centrale.
- **penaltyContentAtCw**: Penalità minore (+1) per contenuti necessari che sono ancora localizzati nella *central warehouse*, indicando che potrebbero non essere necessari nella esecuzione dei task (con l'idea di favorire il riutilizzo di box).
- **bonusTypeNecessary**: Bonus (-5) applicato quando una scatola contiene un tipo di contenuto richiesto da una workstation.
- **malusNoTypeNecessary**: Penalità significativa (+10) per una scatola che contiene un contenuto non richiesto da nessuna workstation.
- **bonusBoxAtPlace**: Un piccolo bonus (-1) per scatole posizionate correttamente su un trasportatore.
- **penaltyBoxAtWs**: Penalità (+2) per una scatola collocata in una workstation senza che il suo contenuto sia richiesto.
- **penaltyAgentAtLoc**: Penalità (+10) per agenti che si trovano in una posizione diversa dal magazzino centrale (favorendo gli spostamenti di agenti solo se strettamente necessari per il completamento dei task).

Il seguente blocco di codice itera attraverso i *fluents* dello stato corrente, applicando la logica di bonus e penalità a seconda dei predicati rilevati:

```
    BitVector ppk = new BitVector(state);
    for (int p = ppk.nextSetBit(0); p >= 0; p = ppk.nextSetBit(p + 1)) {
        predicate = this.fluents.get(p).get(0);
        switch (predicate) {

            case "content_type_at_ws":
                type = this.fluents.get(p).get(1);
                ws = this.fluents.get(p).get(2);
                evalState += goalSatisfied;
                break;

            case "content_at_loc":
                content = this.fluents.get(p).get(1);
                loc = this.fluents.get(p).get(2);
                evalState +=  loc.startsWith(CENTRAL_WAREHOUSE)
                  ? (this.typeRequested.contains(this.typeOf.get(content))
                        ? penaltyContentAtCw   :  -1)
                  : 0;
                break;

           case "filled_box":
              box = this.fluents.get(p).get(1);
              content = this.fluents.get(p).get(2);
              type = this.typeOf.getOrDefault(content, NO_TYPE);
```

```
                evalState += this.typeRequested.contains(type)
                    ? bonusTypeNecessary : malusNoTypeNecessary;
                break;

            case "box_at_loc":
               loc = this.fluents.get(p).get(2);
               evalState += loc.startsWith(LOCATION) ? penaltyBoxAtLoc : 0;
               break;

            case "box_at_place":
               box = this.fluents.get(p).get(1);
               place = this.fluents.get(p).get(2);
               evalState += bonusBoxAtPlace;
               break;

            case "agent_at_loc":
               loc = this.fluents.get(p).get(2);
               evalState += loc.startsWith(LOCATION) ? penaltyAgentAtLoc : 0;
               break;

            case "box_at_ws":
              ws = this.fluents.get(p).get(2);
              evalState += this.wsRequested.containsKey(ws) ? 0 : penaltyBoxAtWs;
              break;

            default:
               evalState += 11;  // Penalità generica per predicati non gestiti
(MAX penalty + 1)
               break;
          }
      }
```

In questo ciclo, il sistema esamina ogni predicato presente nei fluents dello stato corrente:

- **content_type_at_ws**: Se un tipo di contenuto è già presente nella *workstation* corretta, viene assegnato un bonus significativo, poiché ciò rappresenta un obiettivo già soddisfatto.
- **content_at_loc**: Penalità per i contenuti necessari ancora localizzati nella *central warehouse,* poiché devono ancora essere distribuiti.
- **filled_box**: Penalità o bonus in base al tipo di contenuto delle scatole, se sono richiesti o meno dalle *workstation*.
- **box_at_loc**: Penalità per scatole collocate fuori dalla *central warehouse* (per favorire il recupero delle box che si trovano lontano dalla central warehouse).
- **box_at_place**: Bonus per scatole sotto il controllo di un agente.
- **agent_at_loc**: Penalità per agenti situati lontani dalla *central warehouse*.
- **box_at_ws**: Penalità per scatole nelle *workstation* sbagliate o non necessarie.

Questo ciclo continua fino a quando tutti i predicati rilevanti nello stato corrente sono stati esaminati, permettendo una valutazione completa e precisa dello stato in termini di vicinanza al raggiungimento dell'obiettivo finale.

2.5.4 Assegnazione del Costo alle Azioni Disponibili

In questa sezione del codice viene implementato il processo di valutazione delle possibili azioni che possono essere eseguite a partire dallo stato corrente. L'obiettivo principale è di assegnare un costo a ciascuna azione disponibile, tenendo conto degli effetti positivi e negativi che essa produce. Questa valutazione è cruciale per determinare quale azione, a partire dallo stato corrente, è più conveniente eseguire per avvicinarsi all'obiettivo finale, influenzando positivamente o negativamente il valore assegnato allo stato (discusso nel paragrafo precedente).

```
    /**
    * 2. For each available action from the current state, assign a value
    * */
    int[] actions = new int [ ]  {
        Integer.MAX_VALUE, // 0 - pick_up
        Integer.MAX_VALUE, // 1 - move
        Integer.MAX_VALUE, // 2 - deliver
       Integer.MAX_VALUE, // 3 - fill
       Integer.MAX_VALUE // 4 - empty
    };
```

All'inizio, il codice definisce un array *actions*, in cui ogni posizione rappresenta un'azione specifica, e il costo iniziale di ciascuna azione viene impostato al massimo valore possibile (Integer.MAX_VALUE), indicando che, di default, tutte le azioni sono considerate estremamente costose fino a che non si dimostra il contrario.

Due variabili chiave, *moveBonus* e *movePenalty*, sono definite per stabilire il costo di un'azione in base agli effetti positivi o negativi che produce:

```
    int moveBonus = -1, movePenalty = +1;
```

Il codice successivo valuta le azioni disponibili (*pick_up*, *move*, *deliver*, *fill*, *empty*) utilizzando un ciclo for, esaminando quali azioni sono applicabili nello stato corrente basandosi sulle condizioni che risulteranno vere o false una volta che l'azione sarà stata eseguita.

```
    for (Action op : this.getActions()) {
        if (op.isApplicable(state)) {
              effects = op.getConditionalEffects().get(0).getEffect();
              positiveFluents = effects.getPositiveFluents();
              negativeFluents = effects.getNegativeFluents();
              moveCost = Integer.MAX_VALUE;
              index = -1;

       switch (op.getName()) {
              case "pick_up":
                     // code...
                     index=0;
                     break;

              case "move":
                     // code...
                     index=1;
                     break;
```

```
                case "deliver":
                        // code...
                        index=2;
                        break;

                case "fill":
                        // code...
                        index=3;
                        break;

                case "empty":
                        // code...
                        index=4;
                        break;

                default:
                        break;
                }
                if (index!=-1 && moveCost < moves[index]) moves[index] = moveCost;
        }
}
```

Dopo aver calcolato il costo di un'azione, il codice verifica se tale costo è inferiore al costo precedentemente registrato per lo stesso tipo di azione. Se il nuovo costo è inferiore, viene aggiornato nell'array *actions*. Questo garantisce che, alla fine del ciclo, actions contenga i costi minimi per ogni tipo di azione applicabile nello stato corrente.

pick_up:

```
case "pick_up":
    /**
     *      Positive fluents:
     *          [box_at_place, box, place]
     *      -------------------------------
     *      Negative fluents:
     *          [empty_place, place]
     *          [box_at_loc, box, loc]
     */
    moveCost = tmpTypeRequested.size()*moveBonus;
    index=0;
    break;
```

Questa azione rappresenta il sollevamento di una scatola da un determinato luogo.
Gli effetti positivi includono il fatto che la scatola sarà ora associata a un luogo (*box_at_place)*.
Il costo viene calcolato in base al numero di tipi richiesti (tmpTypeRequested) moltiplicato per un bonus, poiché raccogliere una scatola utile è generalmente vantaggioso se i tipi richiesti sono numerosi.

move:

```
case "move":
    /**
```

```
     *      Positive fluents:
     *          [agent_at_loc, agent, loc]
     *      ------------------------------
     *      Negative fluents:
     *          [agent_at_loc, agent, loc]
     * */
    agent = this.fluents.get(positiveFluents.nextSetBit(0)).get(1);
    toLoc = this.fluents.get(positiveFluents.nextSetBit(0)).get(2);
    // Get the carrier on agent
    carrier = "";
    for (Map.Entry<String, String> entry : this.carrierOnAgent.entrySet()) {
        carrier = entry.getKey();
        if (entry.getValue().equals(agent)) {
            // carrier founded
            break;
        }
    }
    // Get the number of places on carrier
    int totalCapacity = this.placeOnCarrier.get(carrier).size();
    // Check all boxes on carrier (boxAtAgent)
    List<String> boxes = boxAtAgent.getOrDefault(agent, new LinkedList<>());
    if (boxes.isEmpty()) {
        moveCost = totalCapacity*movePenalty;
    } else {
        int necessary = 0, unnecessary = 0;
        for (String theBox : boxes) {
            type = this.typeOf.getOrDefault(theBox, NO_TYPE);
            if (!type.equals(NO_TYPE)){
                if (tmpTypeRequested.contains(type))
                    necessary++;
                else
                    unnecessary++;
            }
        }
        moveCost = toLoc.startsWith(LOCATION) ?
             (necessary>0 ? (necessary-unnecessary)*moveBonus :
             totalCapacity*movePenalty) : (necessary>0 ?
             totalCapacity*moveBonus : (totalCapacity-
             unnecessary)*movePenalty);
    }
    index=1;
    break;
```

Questa azione sposta un agente da un luogo a un altro.
Gli effetti positivi e negativi sono simili, rappresentando la transizione dell'agente tra due luoghi. Il costo viene calcolato considerando la capacità totale del trasportatore e il numero di scatole utili o non necessarie che l'agente sta trasportando. Se l'agente si sposta verso una posizione utile per il goal (toLoc), viene applicato un bonus o una penalità a seconda delle scatole che trasporta.

deliver:

```
case "deliver":
    /**
     *      Positive fluents:
     *          [empty_place, place]
```

```
     *          [box_at_ws, box, ws]
     *      -------------------------------
     *      Negative fluents:
     *          [box_at_place, box, place]
     */
    for ( int p = positiveFluents.nextSetBit(0); p >= 0;
          p = positiveFluents.nextSetBit(p + 1))
        {
        predicate = this.fluents.get(p).get(0);
        if (predicate.equals("box_at_ws")) {
            box = this.fluents.get(p).get(1);
            ws = this.fluents.get(p).get(2);
            type = boxContentType.getOrDefault(box, NO_TYPE);
            if (type.equals(NO_TYPE))
                moveCost = movePenalty;
            else {
                tmp = tmpWsRequested.getOrDefault(ws, new LinkedList<>());
                if (tmp.isEmpty())
                    moveCost = movePenalty;
                else
                    moveCost = tmp.contains(type) ?
                        (lastOne ? 10*moveBonus : moveBonus) : movePenalty;
            }
            break;
        }
    }
    index=2;
    break;
```

Questa azione consegna una scatola a una *workstation*.
Gli effetti positivi includono la presenza della scatola nella *workstation* (*box_at_ws*).
Il costo viene determinato in base al tipo di contenuto della scatola e se tale contenuto è richiesto dalla *workstation*. Se la scatola contiene un tipo richiesto, viene applicato un bonus; altrimenti, una penalità. Il bonus è significativo se il contenuto è dell'ultimo tipo richiesto.

fill:

```
case "fill":
    /**
     *      Positive fluents:
     *          [filled_box, box, content]
     *      -------------------------------
     *      Negative fluents:
     *          [content_at_loc, content, loc]
     *          [empty_box, box]
     * */
    content = this.fluents.get(positiveFluents.nextSetBit(0)).get(2);
    type = this.typeOf.getOrDefault(content, NO_TYPE);
    if (type.equals(NO_TYPE))
        moveCost = movePenalty;
    else
        moveCost = tmpTypeRequested.contains(type) ?
                    (lastOne ? 10*moveBonus : moveBonus) : movePenalty;
    index=3;
    break;
```

Questa azione riempie una scatola con un contenuto specifico.
Gli effetti positivi includono che la scatola ora è piena (filled_box).
Il costo dipende dal fatto che il contenuto inserito nella scatola sia richiesto o meno. Se il contenuto è richiesto, viene applicato un bonus, significativo se il contenuto è dell'ultimo tipo necessario per soddisfare il goal.

empty:

```
case "empty":
    /**
     *      Positive fluents:
     *          [box_at_loc, box, loc]
     *          [empty_box, box]
     *          [content_type_at_ws, type, ws]
     *      ------------------------------
     *      Negative fluents:
     *          [box_at_ws, box, ws]
     *          [filled_box, box, content]
     */
    for (int p = positiveFluents.nextSetBit(0); p >= 0; p =
             positiveFluents.nextSetBit(p + 1)) {
        predicate = this.fluents.get(p).get(0);
        if (predicate.equals("content_type_at_ws")) {
            type = this.fluents.get(p).get(1);
            ws = this.fluents.get(p).get(2);
            tmp = tmpWsRequested.getOrDefault(ws, new LinkedList<>());
            if (tmp.isEmpty())
                moveCost = movePenalty;
            else
                moveCost = tmp.contains(type) ?
                        (lastOne ? 10 * moveBonus : moveBonus) : movePenalty;
            break;
        }
    }
    index=4;
    break;
```

Questa azione svuota una scatola e posiziona il contenuto in una *workstation*.
Gli effetti positivi includono che la scatola è ora vuota (empty_box) e il tipo di contenuto è disponibile alla workstation (content_type_at_ws).
Il costo viene calcolato considerando se il tipo di contenuto era richiesto o meno dalla *workstation*.

2.6.1 Istanza 1

L'istanza senza carrier, discussa nel capitolo 2.1, è stata costruita basandosi su domain.pddl ed utilizzando un solo *carrier* con un singolo *place*, in questo modo è stato possibile eseguire istance1.pddl con il planner costruito ad hoc (*IMPlanner*).

```
problem instantiation done successfully (92 actions, 74 fluents)

**********************************************
Start solving using ASTAR with IM_HEURISTIC heuristic
```

```
    + h weight=1.5
    + split factor=0
    + debug mode=false
************************************************
==========================================
Search stats:
------------------------------------------
+ Search time: 2,21 seconds
+ Memory used: 38,75 MB
==========================================

found plan as follows:

00: (             pick_up cw b1 a1 c1 p1) [0]
01: (        fill b1 hammer2 cw a1 c1 p1) [0]
02: (                   move cw loc2 a1) [0]
03: (       deliver b1 ws4 loc2 a1 c1 p1) [0]
04: (                   move loc2 cw a1) [0]
05: (             pick_up cw b2 a1 c1 p1) [0]
06: (        fill b2 hammer1 cw a1 c1 p1) [0]
07: (                   move cw loc2 a1) [0]
08: (       deliver b2 ws3 loc2 a1 c1 p1) [0]
09: (empty b2 hammer1 hammer ws3 a1 loc2) [0]
10: (empty b1 hammer2 hammer ws4 a1 loc2) [0]
11: (           pick_up loc2 b1 a1 c1 p1) [0]
12: (                   move loc2 cw a1) [0]
13: (          fill b1 bolt1 cw a1 c1 p1) [0]
14: (                   move cw loc1 a1) [0]
15: (       deliver b1 ws1 loc1 a1 c1 p1) [0]
16: (    empty b1 bolt1 bolt ws1 a1 loc1) [0]
17: (           pick_up loc1 b1 a1 c1 p1) [0]
18: (                   move loc1 cw a1) [0]
19: (          fill b1 bolt2 cw a1 c1 p1) [0]
20: (                   move cw loc1 a1) [0]
21: (       deliver b1 ws2 loc1 a1 c1 p1) [0]
22: (    empty b1 bolt2 bolt ws2 a1 loc1) [0]
23: (           pick_up loc1 b1 a1 c1 p1) [0]
24: (                   move loc1 cw a1) [0]
25: (         fill b1 valve1 cw a1 c1 p1) [0]
26: (                   move cw loc2 a1) [0]
27: (       deliver b1 ws3 loc2 a1 c1 p1) [0]
28: (           pick_up loc2 b2 a1 c1 p1) [0]
29: (  empty b1 valve1 valve ws3 a1 loc2) [0]
30: (                   move loc2 cw a1) [0]
31: (         fill b2 valve2 cw a1 c1 p1) [0]
32: (                   move cw loc1 a1) [0]
33: (       deliver b2 ws1 loc1 a1 c1 p1) [0]
34: (  empty b2 valve2 valve ws1 a1 loc1) [0]

time spent:       0,06 seconds parsing
                  0,10 seconds encoding
                  2,21 seconds searching
                  2,27 seconds total time

memory used:      0,48 MBytes for problem representation
```

```
                    38,75 MBytes for searching
                    39,23 MBytes total
```

A differenza del plan generato da FF, (si guardi il paragrafo 2.1) il plan di IMPlanner segue una sequenza di azioni più diretta e ottimizzata. Rispetto a FF per lo stesso problema, IMPlanner mantiene il numero di azioni pari a 35, ma più rilevante è come queste azioni siano distribuite in modo più efficiente, soprattutto all'inizio del plan, in cui l'agente non si preoccupa di svuotare la box b1 appena consegnata alla workstation ws4, ma preferisce recuperare con b2 il contenuto necessario per soddisfare la richiesta di ws3, per, alla fine, svuotare le due box con due mosse consecutive.

2.6.2 Istanza 2

Per quanto riguarda istance2.pddl, sono state eseguite diverse istanze del problema variando i parametri di "split size" e "h weight", al fine di analizzare l'efficacia del metodo sviluppato.

Nella tabella sono stati riportati per ogni istanza lanciata: il numero di azioni, tempi e memoria allocata per la ricerca.

	No split	Split size=1	Split size=2	Split size=3
h weight=0.5	Timeout	# Action: 39 Time: 0,38 s Memory: 0,53 MB	# Action: 33 Time: 0,10 s Memory: 0,14 MB	# Action: 30 Time: 0,18 s Memory: 0,73 MB
h weight=1.0	Timeout	# Action: 35 Time: 0,41 s Memory: 0,44 MB	# Action: 33 Time: 0,09 s Memory: 0,13 MB	# Action: 30 Time: 0,20 s Memory: 0,73 MB
h weight=1.5	Timeout	# Action: 35 Time: 0,28 s Memory: 0,37 MB	# Action: 33 Time: 0,09 s Memory: 0,13 MB	# Action: 30 Time: 0,19 s Memory: 0,73 MB
h weight=3.0	Timeout	# Action: 35 Time: 0,35 s Memory: 0,37 MB	# Action: 33 Time: 0,11 s Memory: 0,13 MB	# Action: 30 Time: 0,18 s Memory: 0,73 MB

Dai risultati riportati in tabella, si osserva che tutte le esecuzioni, ad eccezione di quella con h weight=0.5 e split size=1, hanno restituito un piano che permette di raggiungere il goal con un numero di azioni inferiore o pari a 35. Questo indica un miglioramento rispetto al piano generato con l'algoritmo FAST FORWARD presentato nel capitolo 2.2, il quale richiedeva esattamente 36 mosse per il completamento dei task.

Inoltre, il meccanismo di "split" applicato al problema sembra migliorare sia i tempi di esecuzione che la precisione della stima. In particolare, si nota che aumentando il valore di "split size" da 1 a 3, il numero di azioni necessarie per raggiungere il goal diminuisce in modo

significativo: da 39 azioni con split size=1 e h weight=0.5 a 30 azioni con split size=3 per vari pesi di h.

In tutti i casi, la memoria utilizzata per la ricerca resta al di sotto di 1MB; nello specifico si ha minor impatto per “split size” pari a 2, con un'occupazione non superiore al 15%, ma allo stesso modo determina una minore precisione nella ricerca, con un plan di 33 azioni.

Di seguito è riportato il piano migliore ottenuto per l’istanza 2, in cui si evidenziano i parametri fissati per la ricerca e i sotto-problemi generati e risolti dalla strategia impostata:

```
problem instantiation done successfully (472 actions, 128 fluents)

************************************************
Start solving using ASTAR with IM_HEURISTIC heuristic
    + h weight=1.5
    + split factor=3
    + debug mode=false
************************************************

==========================================
SubProblems instantiated with:
------------------------------------------

Goal0: (and (content_type_at_ws bolt ws1)
  (content_type_at_ws valve ws1)
  (content_type_at_ws bolt ws2))
Goal1: (and (content_type_at_ws hammer ws3)
  (content_type_at_ws valve ws3)
  (content_type_at_ws hammer ws4))

==========================================

==========================================
Plan 0:
------------------------------------------
00: (          pick_up cw b1 a1 c1 p11) [0]
01: (      fill b1 valve1 cw a1 c1 p11) [0]
02: (          pick_up cw b2 a1 c1 p12) [0]
03: (      fill b2 valve2 cw a1 c1 p12) [0]
04: (          pick_up cw b3 a2 c2 p21) [0]
05: (       fill b3 bolt1 cw a2 c2 p21) [0]
06: (          pick_up cw b4 a2 c2 p22) [0]
07: (       fill b4 bolt2 cw a2 c2 p22) [0]
08: (                  move cw loc1 a2) [0]
09: (    deliver b3 ws1 loc1 a2 c2 p21) [0]
10: (  empty b3 bolt1 bolt ws1 a2 loc1) [0]
11: (    deliver b4 ws2 loc1 a2 c2 p22) [0]
12: (  empty b4 bolt2 bolt ws2 a2 loc1) [0]
13: (                  move cw loc1 a1) [0]
14: (    deliver b2 ws1 loc1 a1 c1 p12) [0]
15: (empty b2 valve2 valve ws1 a1 loc1) [0]
==========================================

==========================================
```

```
Plan 1:
-----------------------------------------
00: (          pick_up loc1 b2 a1 c1 p12) [0]
01: (                  move loc1 cw a1) [0]
02: (       fill b2 hammer1 cw a1 c1 p12) [0]
03: (          pick_up loc1 b3 a2 c2 p21) [0]
04: (                  move loc1 cw a2) [0]
05: (       fill b3 hammer2 cw a2 c2 p21) [0]
06: (                  move cw loc2 a1) [0]
07: (      deliver b2 ws4 loc2 a1 c1 p12) [0]
08: (empty b2 hammer1 hammer ws4 a1 loc2) [0]
09: (                  move cw loc2 a2) [0]
10: (      deliver b1 ws3 loc2 a1 c1 p11) [0]
11: (      deliver b3 ws3 loc2 a2 c2 p21) [0]
12: (empty b3 hammer2 hammer ws3 a1 loc2) [0]
13: (  empty b1 valve1 valve ws3 a1 loc2) [0]
=========================================

=========================================
Search stats:
-----------------------------------------
 + Search time: 0,19 seconds
 + Memory used: 0,73 MB
=========================================

found plan as follows:

00: (            pick_up cw b1 a1 c1 p11) [0]
01: (        fill b1 valve1 cw a1 c1 p11) [0]
02: (            pick_up cw b2 a1 c1 p12) [0]
03: (        fill b2 valve2 cw a1 c1 p12) [0]
04: (            pick_up cw b3 a2 c2 p21) [0]
05: (         fill b3 bolt1 cw a2 c2 p21) [0]
06: (            pick_up cw b4 a2 c2 p22) [0]
07: (         fill b4 bolt2 cw a2 c2 p22) [0]
08: (                  move cw loc1 a2) [0]
09: (      deliver b3 ws1 loc1 a2 c2 p21) [0]
10: (    empty b3 bolt1 bolt ws1 a2 loc1) [0]
11: (      deliver b4 ws2 loc1 a2 c2 p22) [0]
12: (    empty b4 bolt2 bolt ws2 a2 loc1) [0]
13: (                  move cw loc1 a1) [0]
14: (      deliver b2 ws1 loc1 a1 c1 p12) [0]
15: (  empty b2 valve2 valve ws1 a1 loc1) [0]
16: (          pick_up loc1 b2 a1 c1 p12) [0]
17: (                  move loc1 cw a1) [0]
18: (       fill b2 hammer1 cw a1 c1 p12) [0]
19: (          pick_up loc1 b3 a2 c2 p21) [0]
20: (                  move loc1 cw a2) [0]
21: (       fill b3 hammer2 cw a2 c2 p21) [0]
22: (                  move cw loc2 a1) [0]
23: (      deliver b2 ws4 loc2 a1 c1 p12) [0]
24: (empty b2 hammer1 hammer ws4 a1 loc2) [0]
25: (                  move cw loc2 a2) [0]
26: (      deliver b1 ws3 loc2 a1 c1 p11) [0]
27: (      deliver b3 ws3 loc2 a2 c2 p21) [0]
```

```
28: (empty b3 hammer2 hammer ws3 a1 loc2) [0]
29: (  empty b1 valve1 valve ws3 a1 loc2) [0]

time spent:       0,03 seconds parsing
                  0,13 seconds encoding
                  0,19 seconds searching
                  0,34 seconds total time

memory used:      1,80 MBytes for problem representation
                  0,73 MBytes for searching
                  2,53 MBytes total
```

2.6.3 Confronto delle Performance della IMHeuristic con Altre Euristiche

Il confronto tra la IMHeuristic e le altre euristiche della libreria pddl4j è stato effettuato su diversi problemi (p01, p02, p03, p04) di complessità crescente, utilizzando parametri di split size = 3 e h weight = 1.5.

La complessità dei problemi è stata espressa in termini di workstation da soddisfare, contenuti presenti nell'ambiente, tipi di contenuti definiti e numero di agenti coinvolti.

Di seguito sono riportate le caratteristiche dei problemi istanziati, che riflettono la loro complessità nella ricerca del plan:

- **p01**: 11 azioni, 25 fluents, 1 problema, 3 goal
- **p02**: 370 azioni, 113 fluents, 2 sotto-problemi, 6 goal
- **p03**: 1540 azioni, 223 fluents, 5 sotto-problemi, 15 goal
- **p04**: 6060 azioni, 516 fluents, 8 sotto-problemi, 23 goal

	p01	p02	p03	p04
IMHeuristic	# Action: 17 Time: 0,01 s Memory: 0,01 MB	# Action: 28 Time: 0,12 s Memory: 0,29 MB	# Action: 67 Time: 8,99 s Memory: 77,16 MB	# Action: 125 Time: 156,15 s Memory: 428,77 MB
FAST FORWARD	# Action: 17 Time: 0,02 s Memory: 0,01 MB	# Action: 27 Time: 71,37 s Memory: 414,01 MB	Timeout	Timeout
MAX	# Action: 17 Time: 0,01 s Memory: 0,01 MB	Timeout	Timeout	Timeout

SET LEVEL	# Action: 17 Time: 1,05 s Memory: 0,01 MB	Error Out of Memory	Error Out of Memory	Timeout

*Timeout: il piano non è stato generato entro i 5 min.

*Error Out of Memory (Java Heap Space): eccezione sollevata durante l'esecuzione.

Dalla tabella emerge chiaramente che la IMHeuristic ha ottenuto risultati notevolmente superiori rispetto alle altre euristiche, completando con successo tutti i problemi, inclusi i più complessi, come p04. Ecco un'analisi più dettagliata:

- **p01**: Tutte le euristiche riescono a risolvere il problema in tempi simili, con IMHeuristic che impiega solo 0,01 secondi, dimostrando un'ottima efficienza.
- **p02**: La IMHeuristic risolve il problema in soli 0,12 secondi, mentre FAST FORWARD impiega molto più tempo (71,37 secondi) e un notevole consumo di memoria (414,01 MB). Le altre euristiche non riescono a completare l'elaborazione, mostrando un chiaro fallimento di fronte a una complessità maggiore.
- **p03 e p04**: Solo IMHeuristic riesce a completare con successo questi problemi. In particolare, per p04, la IMHeuristic impiega 156,15 secondi e utilizza 428,77 MB di memoria, dimostrando la sua capacità di gestire problemi di elevata complessità. Le altre euristiche falliscono per timeout o errori di memoria, evidenziando una scarsa scalabilità e una gestione meno efficiente delle risorse.

Questi risultati sottolineano l'efficacia della IMHeuristic, che non solo gestisce con successo problemi complessi, ma lo fa anche in tempi ragionevoli e con un utilizzo ottimizzato della memoria.

3 Temporal Planning & Robotics

3.1 Temporal Planning

Il dominio precedentemente definito è stato esteso così da supportare le *durative actions*, ossia è stata associata una durata temporale ad ogni azione.
Di seguito vengono riportate le modifiche apportate rispetto alla versione commentata nel capitolo 1.

In primo luogo, ai *requirements* è stato aggiunto il riferimento alle *durative-actions*:

```
 (:requirements :strips :typing :fluents :durative-actions)
```

Tra i predicati è stato aggiunto *free-agent*, per evitare che si verifichi l'esecuzione contemporanea di due azioni diverse da parte dello stesso agente:

```
(:predicates
    [...]
    (free-agent ?agent - agent)
)
```

Sono state, inoltre, definite cinque funzioni che indicano la durata delle diverse azioni:

```
(:functions
    (pick-duration)
    (fill-duration)
    (move-duration)
    (deliver-duration)
    (empty-duration)
)
```

Le azioni precedentemente definite sono state modificate; in particolare, oltre all'aggiunta della specifica della durata, ne sono state modificate le *condition* e gli *effect*, utilizzando i costrutti *at start*, *over all* e *at end*:

- Il costrutto *at start* indica una condizione o un effetto vero all'inizio dell'azione.
- Il costrutto *over all* indica una condizione valida per tutta la durata dell'azione.
- Il costrutto *at end* indica un effetto vero al completamento dell'azione.

L'utilizzo di tali costrutti ha permesso, ad esempio, di evitare che la stessa box possa essere presa da due agenti robotici contemporaneamente, cambiandone la location non appena inizia un'azione di *pick up*, invece che al termine dell'azione.

```
    (:durative-action pick_up
        :parameters (?loc - location ?box - box ?agent - agent ?carrier -
                     carrier ?place - place)
        :duration (= ?duration (pick-duration))
        :condition (and
            (at start (free_agent ?agent))
            (over all (agent_at_loc ?agent ?loc))
            (at start (box_at_loc ?box ?loc))
            (over all (carrier_at_agent ?carrier ?agent))
            (over all (place_at_carrier ?place ?carrier))
```

```
            (over all (empty_place ?place))
        )
        :effect (and
            (at start (not(free_agent ?agent)))
            (at end (box_at_place ?box ?place))
            (at end (not (empty_place ?place)))
            (at start (not (box_at_loc ?box ?loc)))
            (at end (free_agent ?agent))
        )
    )

    (:durative-action move
        :parameters (?from ?to - location ?agent - agent)
        :duration (= ?duration (move-duration))
        :condition (and
            (at start (free_agent ?agent))
            (over all (connected ?from ?to))
            (at start (agent_at_loc ?agent ?from))
        )
        :effect (and
            (at start (not(free_agent ?agent)))
            (at start (not (agent_at_loc ?agent ?from)))
            (at end (agent_at_loc ?agent ?to))
            (at end (free_agent ?agent))
        )
    )

    (:durative-action deliver
        :parameters (?box - box ?ws - workstation ?location - location ?agent -
                     agent ?carrier - carrier ?place - place)
        :duration (= ?duration (deliver-duration))
        :condition (and
            (at start (free_agent ?agent))
            (over all (ws_at_loc ?ws ?location))
            (over all (agent_at_loc ?agent ?location))
            (over all (carrier_at_agent ?carrier ?agent))
            (over all (place_at_carrier ?place ?carrier))
            (over all (box_at_place ?box ?place))
        )
        :effect (and
            (at start (not(free_agent ?agent)))
            (at end (not (box_at_place ?box ?place)))
            (at end (empty_place ?place))
            (at end (box_at_ws ?box ?ws))
            (at end (free_agent ?agent))
        )
    )

    (:durative-action fill
        :parameters (?box - box ?content - content ?loc - location ?agent -
                     agent ?carrier - carrier ?place - place)
        :duration (= ?duration (fill-duration))
        :condition (and
            (at start (free_agent ?agent))
            (at start (content_at_loc ?content ?loc))
            (over all (agent_at_loc ?agent ?loc))
            (over all (carrier_at_agent ?carrier ?agent))
            (over all (place_at_carrier ?place ?carrier))
            (over all (box_at_place ?box ?place))
```

```
            (over all (empty_box ?box))
        )
        :effect (and
            (at start (not(free_agent ?agent)))
            (at end (not (empty_box ?box)))
            (at end (filled_box ?box ?content))
            (at start (not (content_at_loc ?content ?loc)))
            (at end (free_agent ?agent))
        )
    )

    (:durative-action empty
        :parameters (?box - box ?content - content ?t - type ?ws - workstation
                     ?agent - agent ?loc - location)
        :duration (= ?duration (empty-duration))
        :condition (and
            (at start (free_agent ?agent))
            (over all (is_type ?content ?t))
            (at start (box_at_ws ?box ?ws))
            (over all (ws_at_loc ?ws ?loc))
            (over all (filled_box ?box ?content))
            (over all (agent_at_loc ?agent ?loc))
        )
        :effect (and
            (at start (not(free_agent ?agent)))
            (at end (not (filled_box ?box ?content)))
            (at end (empty_box ?box))
            (at end (box_at_loc ?box ?loc))
            (at start (not (box_at_ws ?box ?ws)))
            (at end (content_type_at_ws ?t ?ws))
            (at end (free_agent ?agent))
        )
    )
```

A questo punto una delle istanze definite per il punto precedente è stata modificata in modo da rispettare il nuovo dominio: nello specifico, nella sezione *init* sono state indicate le durate relative ad ogni azione.

```
(:init
    […]
    (free-agent a1)
    (free-agent a2)
    (= (pick-duration) 2)
    (= (fill-duration) 3)
    (= (move-duration) 5)
    (= (deliver-duration) 2)
    (= (empty-duration) 3)
)
```

Con l'ausilio di planutils è stato generato un piano per l'istanza. La scelta del planner è ricaduta su LPG-TD, un'estensione di LPG (Local search for Planning Graphs), un planner basato sulla ricerca locale e i planning graph compatibile con le *durative actions*. Di seguito, viene riportato l'output ottenuto:

```
0.0003: (MOVE CW LOC1 A1) [D:5.0000; C:1.0000]
0.0005: (PICK_UP CW B1 A2 C2 P21) [D:2.0000; C:1.0000]
2.0008: (FILL B1 BOLT1 CW A2 C2 P21) [D:3.0000; C:1.0000]
5.0010: (MOVE CW LOC1 A2) [D:5.0000; C:1.0000]
10.0013: (DELIVER B1 WS2 LOC1 A2 C2 P21) [D:2.0000; C:1.0000]
12.0015: (EMPTY B1 BOLT1 BOLT WS2 A1 LOC1) [D:3.0000; C:1.0000]
15.0017: (PICK_UP LOC1 B1 A1 C1 P12) [D:2.0000; C:1.0000]
17.0020: (MOVE LOC1 CW A1) [D:5.0000; C:1.0000]
22.0023: (FILL B1 VALVE1 CW A1 C1 P12) [D:3.0000; C:1.0000]
25.0025: (MOVE CW LOC1 A1) [D:5.0000; C:1.0000]
30.0028: (DELIVER B1 WS1 LOC1 A1 C1 P12) [D:2.0000; C:1.0000]
32.0030: (EMPTY B1 VALVE1 VALVE WS1 A2 LOC1) [D:3.0000; C:1.0000]
35.0033: (PICK_UP LOC1 B1 A1 C1 P12) [D:2.0000; C:1.0000]
37.0035: (MOVE LOC1 CW A1) [D:5.0000; C:1.0000]
42.0037: (FILL B1 BOLT2 CW A1 C1 P12) [D:3.0000; C:1.0000]
45.0040: (MOVE CW LOC1 A1) [D:5.0000; C:1.0000]
50.0042: (DELIVER B1 WS1 LOC1 A1 C1 P12) [D:2.0000; C:1.0000]
52.0045: (EMPTY B1 BOLT2 BOLT WS1 A2 LOC1) [D:3.0000; C:1.0000]
52.0047: (MOVE LOC1 CW A1) [D:5.0000; C:1.0000]
57.0050: (MOVE CW LOC2 A1) [D:5.0000; C:1.0000]
55.0052: (MOVE LOC1 CW A2) [D:5.0000; C:1.0000]
60.0055: (PICK_UP CW B3 A2 C2 P21) [D:2.0000; C:1.0000]
62.0057: (FILL B3 VALVE2 CW A2 C2 P21) [D:3.0000; C:1.0000]
65.0060: (PICK_UP CW B2 A2 C2 P22) [D:2.0000; C:1.0000]
67.0062: (FILL B2 HAMMER2 CW A2 C2 P22) [D:3.0000; C:1.0000]
70.0065: (MOVE CW LOC2 A2) [D:5.0000; C:1.0000]
75.0068: (DELIVER B3 WS3 LOC2 A2 C2 P21) [D:2.0000; C:1.0000]
77.0070: (EMPTY B3 VALVE2 VALVE WS3 A1 LOC2) [D:3.0000; C:1.0000]
77.0072: (DELIVER B2 WS4 LOC2 A2 C2 P22) [D:2.0000; C:1.0000]
79.0075: (EMPTY B2 HAMMER2 HAMMER WS4 A2 LOC2) [D:3.0000; C:1.0000]
82.0078: (MOVE LOC2 CW A2) [D:5.0000; C:1.0000]
87.0080: (PICK_UP CW B4 A2 C2 P21) [D:2.0000; C:1.0000]
89.0082: (FILL B4 HAMMER1 CW A2 C2 P21) [D:3.0000; C:1.0000]
92.0085: (MOVE CW LOC2 A2) [D:5.0000; C:1.0000]
97.0088: (DELIVER B4 WS3 LOC2 A2 C2 P21) [D:2.0000; C:1.0000]
99.0090: (EMPTY B4 HAMMER1 HAMMER WS3 A1 LOC2) [D:3.0000; C:1.0000]
```

3.2 Robotics Planning

Per eseguire il piano mediante l'utilizzo di PlanSys2 è stato creato il workspace *project_ai e* sono stati definiti:

- Il file di launch in Python, necessario per avviare i nodi ROS.

```
def generate_launch_description():
    # Get the launch directory
    example_dir = get_package_share_directory('project_ai')
    namespace = LaunchConfiguration('namespace')

    declare_namespace_cmd = DeclareLaunchArgument(
        'namespace',
        default_value='',
        description='Namespace')

    plansys2_cmd = IncludeLaunchDescription(
```

```
        PythonLaunchDescriptionSource(os.path.join(
            get_package_share_directory('plansys2_bringup'),
            'launch',
            'plansys2_bringup_launch_monolithic.py')),
        launch_arguments={
          'model_file': example_dir + '/pddl/domain_durative.pddl',
          'namespace': namespace
          }.items())

    # Specify the actions
    pickup_cmd = Node(
        package='project_ai',
        executable='pickup_action',
        name='pickup_action',
        namespace=namespace,
        output='screen',
        parameters=[])

    fill_cmd = Node(
        package='project_ai',
        executable='fill_action',
        name='fill_action',
        namespace=namespace,
        output='screen',
        parameters=[])

    move_cmd = Node(
        package='project_ai',
        executable='move_action',
        name='move_action',
        namespace=namespace,
        output='screen',
        parameters=[])

    deliver_cmd = Node(
        package='project_ai',
        executable='deliver_action',
        name='deliver_action',
        namespace=namespace,
        output='screen',
        parameters=[])

    empty_cmd = Node(
        package='project_ai',
        executable='empty_action',
        name='empty_action',
        namespace=namespace,
        output='screen',
        parameters=[])
    ld = LaunchDescription()

    ld.add_action(declare_namespace_cmd)

    # Declare the launch options
    ld.add_action(plansys2_cmd)
```

```
    ld.add_action(pickup_cmd)
    ld.add_action(fill_cmd)
    ld.add_action(move_cmd)
    ld.add_action(deliver_cmd)
    ld.add_action(empty_cmd)

    return ld
```

- Le *fake actions* in C++; di seguito è stata riportata quella relativa alla *move*.

```
#include <memory>
#include <algorithm>
#include "plansys2_executor/ActionExecutorClient.hpp"
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_action/rclcpp_action.hpp"

using namespace std::chrono_literals;

class MoveAction : public plansys2::ActionExecutorClient
{
public:
  MoveAction()
  : plansys2::ActionExecutorClient("move", 200ms)
  {
    progress_ = 0.0;
  }

private:
  void do_work()
  {
    if (progress_ < 1.0) {
      progress_ += 0.02;
      send_feedback(progress_, "Move running");
    } else {
      finish(true, 1.0, "Move completed");

      progress_ = 0.0;
      std::cout << std::endl;
    }

    std::cout << "\r\e[K" << std::flush;
    std::cout << "Moving ... [" << std::min(100.0, progress_ * 100.0) << "%]
" <<
      std::flush;
  }

  float progress_;
};

int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<MoveAction>();

  node->set_parameter(rclcpp::Parameter("action_name", "move"));
```

```
  node-
>trigger_transition(lifecycle_msgs::msg::Transition::TRANSITION_CONFIGURE);

  rclcpp::spin(node->get_node_base_interface());

  rclcpp::shutdown();

  return 0;
}
```

- I file *CMakeLists.txt* e *package.xml* per effettuare la build del progetto e settare le dipendenze.

Il workspace contiene le seguenti cartelle:

- *src*: contiene i file C++ delle fake actions.
- *pddl*: contiene il file di dominio.
- *launch*: contiene il file *launch.py*, il file di testo con i comandi necessari per istanziare il problema ed il plan generato con planutils, opportunamente modificato per essere letto da PlanSys2.

In seguito alla build del progetto sono state generate anche le cartelle *build*, *install* e *log*.
Il progetto è stato lanciato tramite i comandi seguenti:

```
colcon build --symlink-install
source install/setup.bash
ros2 launch project_ai launch.py
```

Da un secondo terminale è stato avviato il terminale di PlanSys2 con il comando:

```
ros2 run plansys2_terminal plansys2_terminal
```

Da qui è stato istanziato il problema:

```
source path/commands.txt
```

Questo comando esegue il file commands.txt, permettendo che tutti i comandi al suo interno vengano eseguiti come se fossero stati digitati direttamente nella linea di comando. In pratica, serve a caricare e applicare le configurazioni o le istruzioni contenute nel file all'ambiente di lavoro corrente, in questo caso definendo gli oggetti, le funzioni, i predicati ed il goal dell'istanza.
Di seguito è riportato il contenuto del file:

```
set instance cw location
set instance loc1 location
set instance loc2 location
set instance ws1 workstation
set instance ws2 workstation
set instance ws3 workstation
set instance ws4 workstation
set instance ws5 workstation
set instance b1 box
set instance b2 box
set instance b3 box
set instance b4 box
set instance a1 agent
```

```
set instance a2 agent
set instance c1 carrier
set instance c2 carrier
set instance p11 place
set instance p12 place
set instance p21 place
set instance p22 place
set instance valve1 content
set instance valve2 content
set instance bolt1 content
set instance bolt2 content
set instance hammer1 content
set instance hammer2 content
set instance valve type
set instance bolt type
set instance hammer type
set function (= (pick_duration) 2)
set function (= (fill_duration) 3)
set function (= (move_duration) 5)
set function (= (deliver_duration) 2)
set function (= (empty_duration) 3)
set predicate (is_type valve1 valve)
set predicate (is_type valve2 valve)
set predicate (is_type bolt1 bolt)
set predicate (is_type bolt2 bolt)
set predicate (is_type hammer1 hammer)
set predicate (is_type hammer2 hammer)
set predicate (box_at_loc b1 cw)
set predicate (box_at_loc b2 cw)
set predicate (box_at_loc b3 cw)
set predicate (box_at_loc b4 cw)
set predicate (empty_box b1)
set predicate (empty_box b2)
set predicate (empty_box b3)
set predicate (empty_box b4)
set predicate (content_at_loc valve1 cw)
set predicate (content_at_loc valve2 cw)
set predicate (content_at_loc bolt1 cw)
set predicate (content_at_loc bolt2 cw)
set predicate (content_at_loc hammer1 cw)
set predicate (content_at_loc hammer2 cw)
```

Infine, è stato eseguito il plan precedentemente generato:

```
run plan-file path/plan.txt
```

Questo comando esegue il file di piano plan.txt, che contiene la sequenza di azioni da eseguire per raggiungere il goal.

```
0.0003: (move cw loc1 a1) [5.0000]
 0.0005: (pick_up cw b1 a2 c2 p21) [2.0000]
 2.0008: (fill b1 bolt1 cw a2 c2 p21) [3.0000]
 5.0010: (move cw loc1 a2) [5.0000]
 10.0013: (deliver b1 ws2 loc1 a2 c2 p21) [2.0000]
 12.0015: (empty b1 bolt1 bolt ws2 a1 loc1) [3.0000]
 15.0017: (pick_up loc1 b1 a1 c1 p12) [2.0000]
 17.0020: (move loc1 cw a1) [5.0000]
```

```
 22.0023: (fill b1 valve1 cw a1 c1 p12) [3.0000]
 25.0025: (move cw loc1 a1) [5.0000]
 30.0028: (deliver b1 ws1 loc1 a1 c1 p12) [2.0000]
 32.0030: (empty b1 valve1 valve ws1 a2 loc1) [3.0000]
 35.0033: (pick_up loc1 b1 a1 c1 p12) [2.0000]
 37.0035: (move loc1 cw a1) [5.0000]
 42.0037: (fill b1 bolt2 cw a1 c1 p12) [3.0000]
 45.0040: (move cw loc1 a1) [5.0000]
 50.0042: (deliver b1 ws1 loc1 a1 c1 p12) [2.0000]
 52.0045: (empty b1 bolt2 bolt ws1 a2 loc1) [3.0000]
 52.0047: (move loc1 cw a1) [5.0000]
 57.0050: (move cw loc2 a1) [5.0000]
 55.0052: (move loc1 cw a2) [5.0000]
 60.0055: (pick_up cw b3 a2 c2 p21) [2.0000]
 62.0057: (fill b3 valve2 cw a2 c2 p21) [3.0000]
 65.0060: (pick_up cw b2 a2 c2 p22) [2.0000]
 67.0062: (fill b2 hammer2 cw a2 c2 p22) [3.0000]
 70.0065: (move cw loc2 a2) [5.0000]
 75.0068: (deliver b3 ws3 loc2 a2 c2 p21) [2.0000]
 77.0070: (empty b3 valve2 valve ws3 a1 loc2) [3.0000]
 77.0072: (deliver b2 ws4 loc2 a2 c2 p22) [2.0000]
 79.0075: (empty b2 hammer2 hammer ws4 a2 loc2) [3.0000]
 82.0078: (move loc2 cw a2) [5.0000]
 87.0080: (pick_up cw b4 a2 c2 p21) [2.0000]
 89.0082: (fill b4 hammer1 cw a2 c2 p21) [3.0000]
 92.0085: (move cw loc2 a2) [5.0000]
 97.0088: (deliver b4 ws3 loc2 a2 c2 p21) [2.0000]
 99.0090: (empty b4 hammer1 hammer ws3 a1 loc2) [3.0000]
```

Lo screenshot riportato evidenzia come il piano sia stato eseguito correttamente da PlanSys2.