TÉCNICO LISBOA

Artificial Intelligence and Decision Systems (IASD)
Mini-projects, 2016/2017

Assignment #1

Version 1.1 - October 6th, 2016

ITER Logistics

1 Introduction

The ITER[1] is a joint international research project, aiming at the demonstration of the scientific and technological feasibility of nuclear fusion power as an alternative and safe power source. Due to the levels of radiation inside the reactor building and annex facilities, all maintenance operations have to be performed remotely by robotic technology.

This project is focused on the logistics problem of transportation and storage of activated payloads between the Tokomak Building (TB), where the reactor is located, and the Hot Cell Building (HCB), where these payloads are stored (Figure 1 left).

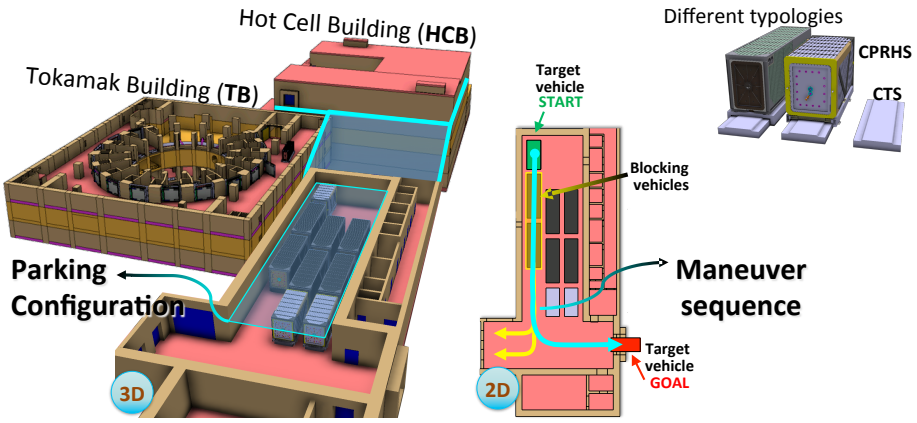

Figure 1: From left to right: a perspective view of the Tokomak and the Hot Cell Buildings, a top view of the Hot Cell Building illustrating the problem of cask storage, and the structure of the CPRHS comprising the CTS vehicle and its load on a pallet.

[1]International Thermonuclear Experimental Reactor, `http://www.iter.org`.

The transportation is assured by the Cask and Plug Remote Handling System (CPRHS), comprising three sub-systems: (1) a cask containing the payload, (2) a pallet that supports the cask, and (3) the Cask Transfer System (CTS). The CTS acts as a mobile robot, provides the mobility for the CPRHS and can be decoupled from the entire system (Figure 1 right).

Due to space constraints, the area available to store these casks is limited. These are stored in lines, called stacks, where casks block other casks (Figure 1 center). Thus, to retrieve one blocked cask, the blocking casks have to be moved to other stacks before it can be retrieved.

The goal of this project is to develop a solver that determines the best sequence of operations that a CTS vehicle has to perform in order to retrieve an arbitrary cask from the HCB. In the next section the problem will be formally specified.

2 Problem statement

For the simplicity sake, we will only consider the casks and a single CTS vehicle, thus ignoring the pallets. The CTS vehicle is capable of transporting a single cask from one point to another.

The problem is formulated as follows. The HCB is modeled as a graph $\mathcal{K} = \langle \mathcal{X}, \mathcal{E} \rangle$, where $\mathcal{X}$ is a set of nodes, representing physical locations in the environment, and $\mathcal{E}$ is a set of undirected edges, each one denoting a feasible direct navigation path between the corresponding pair of nodes. That is, the CTS can navigate among two arbitrary adjacent nodes in the graph. Furthermore, each edge $e \in \mathcal{E}$ has an associated length $L_e > 0$.

Three types of nodes are considered: *stacks* are nodes that allow the storage of one or more casks in line along their length, *crossings* are transit nodes where a CTS can pass through, and a special node denoted *exit point*, representing the exit from the HCB. Thus, the set of nodes is partitioned as a union of three disjoint subsets, $\mathcal{X} = \mathcal{S} \cup \mathcal{C} \cup \{\epsilon\}$, where $\mathcal{S}$ and $\mathcal{C}$ are the sets of stacks and crossings, while ϵ is the exit point. Figure 2 provides an example of such a graph, where $\mathcal{S} = \{S1, S2, S3, S11, S12\}$, $\mathcal{C} = \{A, B, C, D\}$, and $\epsilon = EXIT$.

Each stack $s \in \mathcal{S}$ has an integer size $S_s \geq 1$, and has one and only one edge connected to it. Casks are stored in a Last-In-First-Out (LIFO) manner (*i.e.,* a stack) accessed through an entrance represented by its single connected edge.

Let $\mathcal{A}$ denote a set of casks. Each cask $a \in \mathcal{A}$ has an integer length $L_a \geq 1$, occupying that amount of space in a stack, and a weight $W_a \geq 0$. Casks are stored in a stack one after the other from the stack "bottom", that is, without any gaps, except between the "top" (last inserted) cask and the stack entrance. At any time instant, a stack state comprises an ordered list of casks that is stored in it. A stack can only store casks such that the

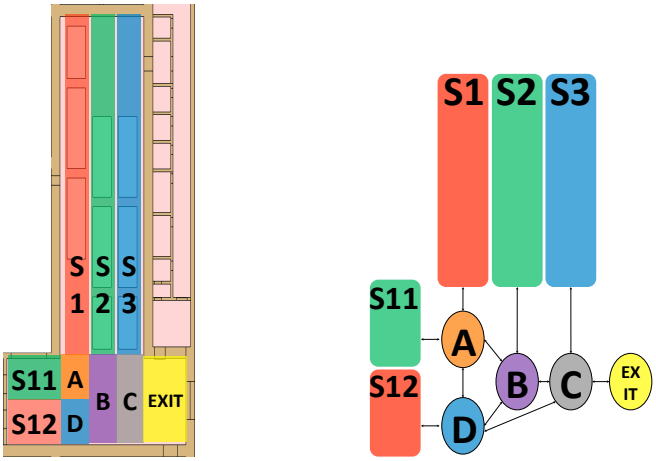

Figure 2: An example HCB configuration (left) together with a graph representing it (right).

sum of their lengths does not exceed the stack size.

Consider a single CTS in the HCB. At each time instant the CTS is positioned in a certain node in only one of two possible states: loaded with a single cask, or unloaded. The actions available to the CTS, positioned in node $p \in \mathcal{X}$, are: (1) *move* to an adjacent node of the graph, that is, a node connected with an edge to p, (2) *load* a cask from a stack (the "top" one), and (3) *unload* a cask to a stack (at the "top"). Actions (2) and (3) are only available if p is a stack. Action (3) is only available if the stack has enough free space to store the loaded cask. Only an unloaded CTS can be loaded and only a loaded CTS can be unloaded.

Each one of these actions has a cost to execute:

- moving along an edge $e \in \mathcal{E}$ in the graph has a cost of L_e if the CTS is unloaded and $(1 + W_c)L_e$ if loaded with a cask $c \in \mathcal{A}$;
- loading or unloading a cask $c \in \mathcal{A}$ has a cost of $1 + W_c$.

The rationale behind this cost structure is the following: cost is measured in terms of a weight-distance product, where casks weights are normalized to a unitary CTS weight and lengths to a unitary load/unload operation cost.

Given a graph $\mathcal{K}$, an initial state of all the stacks, and a given cask $a \in \mathcal{A}$ as input, and assuming that the CTS initial position is at node ϵ, the goal of this project is to develop a program to generate an optimal sequence of actions, satisfying the above constraints, that transports that cask from its stack to the exit point. That is, the initial state is the CTS unloaded at ϵ, and the final state is the CTS positioned at ϵ loaded with the cask a. Optimality is here understood as the minimal total cost, that is, the sum of the costs in the sequence of actions.

3 Implementation requirements

The problem is specified in a data file where each line may specify a cask, a stack, or an edge:

- a line starting with a 'C' specifies a cask $a \in \mathcal{A}$ with the syntax:

  `C`*id length weight*

  where `C`*id* is its unique name, *length* its (integer) size L_a, and *weight* its (float) weight W_a;

- a line starting with a 'S' specifies a stack $s \in \mathcal{S}$ with the syntax:

  `S`*id size cask1 cask2 cask3 ...*

  where `S`*id* is its unique name, *size* its (integer) size S_s, followed by zero or more stored cask names by order of unload, that is, the last one is the "top" most cask in the stack;

- a line starting with an 'E' specifies an edge $e \in \mathcal{E}$ with the syntax:

  `E` *node1 node2 length*

  connecting nodes *node1* and *node2* where *length* is its (float) length L_e.

All other lines should be ignored. Lines may appear in any order.

Crossing nodes are implicitly defined by the specification of edges. The exit node ϵ has the reserved name `EXIT`.

The graph shown in the right side of Figure 2, together with three casks in stack S1, would be specified using the following file contents:

```
Ca 3 2
Cb 3 1.5
Cc 2 0.9

S1  8 Cc Cb Ca
S2  8
S3  8
S11 3
S12 3

E S1  A    1
E S2  B    2
E S3  C    2
E S11 A    1.5
E A   B    1.5
E B   C    1
E C   EXIT 2
E A   D    1.5
```

```
E B    D     0.6
E C    D     0.4
E S12  D     1
```

The project is to be implemented in the Python programming language (version 3) as a program that accepts two arguments: the filename of the problem specification, as described above, and the name of the cask to be retrieved. For instance, to solve the problem of retrieving cask `Cb` from the above problem one may run

```
python3 s1.dat Cb
```

assuming that the above file is stored in the current directory with filename `s1.dat`. Recall that the CTS initial position is at node $\epsilon =$ `EXIT`.

The output of the program should be a sequence of actions, one per line, with the following syntax:

operation arg1 arg2 cost

where *operation* is one of the keywords '`move`', '`load`', or '`unload`', for the operations (1), (2), and (3) defined in section 2, *arg1* and *arg2* are the arguments of the operation, defined below, and *cost* is the step cost of the operation, as defined in Section 2. If the operation is a *move*, *arg1* and *arg2* are the origin and the destination nodes of the edge traveled by the CTS. Otherwise, if the operation is either a *load* or an *unload*, *arg1* is the cask to be loaded/unloaded and *arg2* is the stack where the operation is performed. The line following this sequence should contain the total cost of the solution.

An example of a solution to the problem above is:

```
move EXIT C 2
move C D 0.4
move D A 1.5
move A S1 1
load Ca S1 3
move S1 A 3
move A S11 4.5
unload Ca S11 3
move S11 A 1.5
move A S1 1
load Cb S1 2.5
move S1 A 2.5
move A D 3.75
move D C 1
move C EXIT 5.0
35.65
```

4 Assignment goals

1. Develop a Python (version 3) program for solving the problem described above using Search methods. The solver must include at least one uninformed search method, and one informed search method with an heuristic function.

   In your program the domain-independent part *should be* explicitly isolated from the domain-dependent one. In particular this should be done using two different Python files, one for the domain-independent part and another for the domain-dependent part (with self-evident filenames). Moreover, use the General Search to develop a generic search algorithm and then particularize it to each one of the two search methods.

   Note: All code should be adequately commented.

2. The answers to the following points should be included in a short **technical** report, delivered together with the source code.

   (a) Describe how do you represent the problem state, the operator(s), the initial state, the goal state, and the path cost;

   (b) Identify and justify the search algorithm implemented;

   (c) Describe and justify the heuristic functions developed;

   (d) Compare quantitatively and comment the performance of at least two different heuristic functions implemented.

3. The deliverable of this Lab Assignment consists of two components:

   - the Python source code
   - the short report (**pdf** format) with no more than 2 pages.

Submission: by email to rodrigo.ventura@isr.tecnico.ulisboa.pt
Deadline: 24h00 of 28-Out-2016
(Projects submitted after the deadline will be penalized.)