uc3m

Heuristics and Optimization

Lab assignment #2: *Logical Satisfiability and Heuristic Search*

Computer Science Engineering, 2017–2018

1. Goal

The goal of this assignment is to learn to model satisfiability (SAT) and heuristic search problems and to solve them using standard tools of each kind.

2. Problem statement

Due to the traffic congestion, the *shared parking lot* is becoming more and more common in many places. A driver leaves and picks up his car in the door of a business, but in contraposition to the traditional parking service, different nearby businesses share the same parking lot, which is managed in an innovative way using techniques of discrete optimization.

The first innovation is how space is handled. The parking lot is split into M parallel lanes with a unique entry and a unique leaving point, where it is possible to park up to N cars in each lane. The whole parking lot is surrounded by other streets that facilitate the free movement of cars either entering or leaving the parking lot.

Basically, the idea consists of parking cars in a line in each lane, and to relocate them, if needed, to allow another car to leave the parking lot. For instance, to allow car B, in Figure 1, to get out, it is necessary first to move car A which can be placed in any of the locations shown in dashed blue: either at the beginning or the end in lane 2; or, alternatively, in the second position in the third lane. There are some common sense rules to take into account:

- All cars point to the right
- If a lane is empty, a car can be placed in any of the N available locations.
- If a lane contains, at least, one car then any car entering the lane (either by the beginning or the end) should be parked in an adjacent location to it, i.e., it is not allowed to leave empty locations between cars.

The second innovation is that cars are arranged according to the type of business, which determines the estimated time the driver will take to pick up the car, as shown in Figure 2. To better estimate the pickup order, the parking lot attendants record the arrival time of each car:

- If two cars belong to different categories, then it should be expected that the car with the minimum waiting time will leave sooner.

  For example, if two cars C_1 and C_2 belong to the categories A and B respectively, then it should be assumed that car C_1 will leave before C_2.

- If two cars belong to the same category, then the car that entered before should be expected to leave before.

  Consider, for example, two cars C_1 and C_2 in category B. If C_1 was parked before C_2, then the former is assumed to leave before the latter.

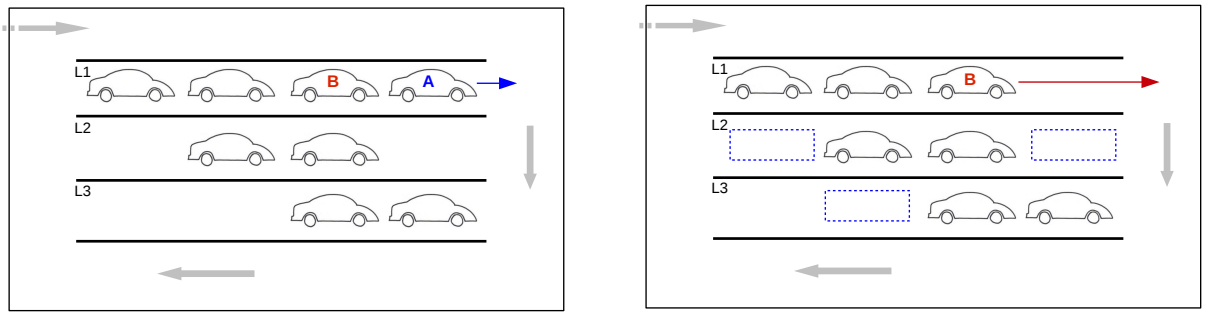

Figura 1: Feasible relocations of car A to allow car B to leave

Category (Duration)	**Comercios**	**Tiempo Estimado**
A (short)	Pharmacies and Pastries	< 0.5h
B (medium)	Restaurants and Bars	2h
C (large)	Theatre and Discos	> 3h

Figura 2: Categories and estimated waiting time of each category

2.1. Part 1: SAT verification of the parking configuration

The goal of the first part of this lab assignment is to determine if there is, or not, a blocked car in a specific configuration of the parking lot. A car is said to be blocked when there are cars parked with a longer estimated waiting time before and after it. In this regard:

- A car is blocked by another car, if the latter belongs to a category with a larger waiting time.
- If both cars belong to the same category, then a car is blocked by another, if the former entered the lane later.

For instance, the car B shown in Figure 1 is not blocked in spite of the category of the cars behind it, if car A can leave before it, thus freeing car B.

It is requested:

1. To model the problem of verifying the configuration of the parking lot as a SAT problem, in Conjunctive Normal Form.

   The formula shall be satisfiable if and only, no car is blocked, and unsatisfiable otherwise.

2. Code the previous formula using JaCoP to determine if a specific configuration is satisfiable or not. The program should be run from the command line as follows:

   ```
   java SATParking <parking-state>
   ```

   where *parking-state* is the name of a file with extension *.input* (e.g., `parking1_parte1.input`) which details the current state of the parking. The contents of the file should adhere to the format shown in Figure 3.

   If the problem is satisfiable, the program should generate a text file, with the same map given in the input file, telling for each occupied position whether the car can escape by the front or the back with "$>$" or "$<$", respectively. The name of the output file should follow the name of the input file replacing the extension with *.output* —e.g., `parking1_parte1.output`.

   If the problem is unsatisfiable, a message should be shown on the terminal.

3 4 A2 A1 B1 A3 __ C1 C2 __ __ __ C3 A4	▪ The first line shows the number of lanes and locations in each lane ▪ Each lane is shown in a separate line ▪ Each car is represented by its category followed by its arrival order ▪ Unoccupied locations are shown with 2 underlines

Figura 3: Example of a file detailing the configuration shown in Figure 1. The description of the input and output format is shown to the right.

3. It is requested to create different test cases (in the form of different input files) with different configurations of different size. It is mandatory to supply, at least, 5 different parking states.

2.2. Part 2: Heuristic Search

The goal of this part is to determine the necessary sequence of movements required to reconfigure the current state in a different one explicitly given. To do this, consider that each manoeuvre has a different cost as detailed below:

- To move a car forward in the same lane has cost 1, in spite of the number of locations the car is pushed forward.
- To move a car backward in the same lane has cost 2, in spite of the number of locations the car is pushed backward.
- To park a car at the beginning of a different lane, or in the same lane but first getting out and then re-entering, has cost 3.
- To park a car at the end of a different lane, or in the same lane re-entering the lane, has cost 4.

It is requested:

1. To model the problem as a heuristic search problem.
2. Implement A* to optimally solve the problem using an informed admissible heuristic function that estimates the cost of the necessary manoeuvres. The program shall be executable from the command line with the following command:

```
java AstarParking <init-parking> <goal-parking>
```

where:

- *init-parking* is the name of a file (with extension *.init*) that contains the start state of the parking lot according to the format given in Figure 3.
- *goal-parking* is the name of a file (with extension *.goal*) that details the goal configuration of the parking.

The program shall generate two output files:

- Plan file: A file with extension *.plan* shall be generated detailing the sequence of steps necessary to reconfigure the starting configuration. This file should detail each step in a different line according to the following format:

  ```
  STEP, CAR, CURRENT-LOCATION, NEW-LOCATION, COST
  ```

  For example, the following command relocates car $A3$ in Figure 3 at the beginning of the second lane:

  ```
  1,A3,L1 P4,L2 P1,3
  ```

- Information file: A file with extension *.info* shall be generated with a summary of various statistics of the search algorithm. It should include, at least, information about the solution length, total cost, number of expansions and running time. For example:

  ```
  Step length: 6
  Running time (seconds): 14.5
  Total cost: 11
  Expansions: 172
  ```

3. Generate different test cases with different configurations for the init and goal states. These tests cases should be generated in accordance to the efficiency obtained with the implementation given.

4. Make a comparative analysis of the test cases: number of nodes expanded, running time, etc. with regard to the number of locations and cars.

3. Requirements for the report

The report must be delivered in PDF format and it should consist of a maximum of 15 pages, including the cover, the table of contents and the back cover. It should contain, at least:

1. Brief introduction explaining the contents of the document.

2. Description of the models, discussing the decisions carried out.

3. Analysis of results.

4. Conclusions.

Important: Reports delivered in a format other than pdf will be penalized with 1 point.

The report **should not include source code** in any case.

4. Grading

This lab assignment will be graded over 10 points. Nevertheless, it is possible to get up one additional point if the extra part is submitted —hence, scoring for a maximum of 11 points.

To assure that the assignment is graded you must do at least the first part and the report.

The distribution of points will be as follows:

1. Part 1 (4 points)

   - SAT model (2 points)
   - Implementation of the model in JaCoP (1 point)
   - Resolution and analysis of different test cases (1 point)

2. Part 2 (6 points)

    - Problem Modeling (2 points)
    - Model Implementation (2 points)
    - Resolution and analysis of different test cases (1 point)
    - Comparative analysis (1 point)

When grading the model proposed, a correct model will result in half of the points. To obtain the whole score, the model must:

- Be correctly formalized in the report.
- Be simple and concise.
- Be properly explained (it should remain clear what is the reason for each component of the model).
- Justify in the report all the design decisions taken.

When grading the model implementation, a correct implementation will result in half of the points. To obtain the whole score, the implementation must:

- Faithfully implement the model proposed above.
- Deliver source code correctly organized and commented. Names shall be descriptive.
- Contain a good deal of different test cases that prove the vaidity of your implementation.

When grading the results analysis, it will be positively valued to include in the report personal conclusions about the assignment difficulty and about what you learnt while carrying it out.

5. Submission

The deadline for submitting the assignment is December, 18 at 23:55. Each pair of students must submit a unique `.zip` file to the assignment section of 'Aula Global'. This file must be named `p2-NIA1-NIA2.zip`, where NIA1 and NIA2 are the last 6 digits of each student's NIA, e.g., `p1-154000-154001.zip`.

Uncompressing the `.zip` file must generate automatically a directory called `NIA1-NIA2/`. For example, `154000-154001/` and it must contain the report in PDF and the source code, inside independent files, for each model.

> **Important:** If the requirements for the submission are not carefully observed, the practice will be penalized with 1 point.