Universidade Federal do Rio Grande do Sul
Instituto de Informática
Departamento de Informática Teórica

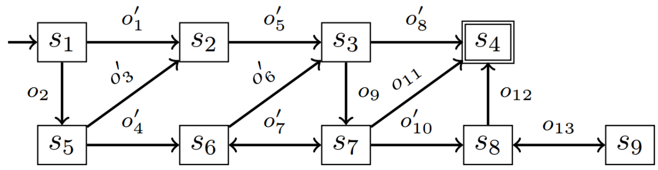
Inteligência Artificial Avançada
2025/1
André Grahl Pereira

List 4[1]

For the runs with Fast Downward, set a time limit of 1 minute and a memory limit of 2 GB. Using Linux, such limits can be set with `ulimit -t 60` and `ulimit -v 2000000`, respectively.

Exercise 1

G :



Consider the following graph G depicting a simple transition system. Assume that operators o_i have cost 1, while operators o'_i have cost 5. As usual, an incoming arrow indicates the initial state, and goal states are marked by a double rectangle. Provide the following graphs:

(a) a graph G_1 which is isomorphic to G but not the same.

(b) a graph G_2 which is graph equivalent to G but not isomorphic to it.

(c) a graph G_3 which is a strict homomorphism of G but not graph equivalent to it.

(d) a graph G_4 which is a non-strict homomorphism of G but not graph equivalent to it.

(e) a graph G_5 that is the transition system induced by the abstraction α that maps states that are in the column i in the image above to the abstract state s_i. For example, the two states in the first column are mapped to an abstract state t_1, the two states in the second column to an abstract state t_2, and so on.

(f) a graph G_6 that is the induced transition system of an abstraction β that is a non-trivial coarsening of α.

(g) a graph G_7 that is the induced transition system of an abstraction γ that is a non-trivial refinement of β but different from α.

In all graphs, highlight an optimal path and compute its cost. For graphs $G_1 - G_4$, justify (one sentence is enough) why they don't have the property they are not supposed to have, for example, why G_2 is not isomorphic to G. For graph G_5, justify why the graph is an abstraction of G. For graphs $G_6 - G_7$, justify why the graphs are a coarsening and a refinement.

[1]Exercício de Malte Helmert.

Exercise 2

(a) In the files `fast-downward/src/search/planopt_heuristics/projection.*` you can find an incomplete implementation of a class projecting a TNF task to a given pattern. Complete the implementation by projecting the initial state, the goal state and the operators.

The example task from the lecture and two of its projections are implemented in the method `test_projections`. *You can use them to test and debug your implementation by calling Fast Downward as* `./fast-downward.py --test-projections`.

(b) In the files `fast-downward/src/search/planopt_heuristics/pdb.*` you can find an incomplete implementation of a pattern database. Complete the implementation by computing the distances for all abstract states as described in the code comments.

You can use the built-in implementation of Fast Downward to debug your code as explained in exercise (c).

(c) Use the heuristic `pdb(pattern=greedy(1000))` to find a good pattern with at most 1000 abstract states for each instance in the directory `castle`. Then run your implementation from exercise (b) using the heuristic `planopt_pdb(pattern=`P`)`. For each instance use the same pattern P used by the built-in implementation.

Compare the two implementations and discuss the preprocessing time, the search time, and the number of expanded states excluding the last f-layer (printed as "Expanded until last jump"). Repeat the experiment for 100000 abstract states and compare the results.

(d) In the files `fast-downward/src/search/planopt_heuristics/canonical_pdbs.*` you can find an incomplete implementation of the canonical pattern database heuristic. Complete the implementation in the methods `build_compatibility_graph` and `compute_heuristic` to create the compatibility graph for a given pattern collection and for computing the heuristic value given the maximal cliques of that graph.

You can use the built-in implementation of Fast Downward to debug your code as explained in exercise (e).

(e) Use the pattern collections provided with the script with at most 1000 abstract states to solve each instance in the directory `nomystery-opt11-strips`. Run your implementation from exercise (d) using the heuristic `planopt_cpdbs(patterns=`C`)`. For each instance use the same pattern collection C used by the built-in implementation.

Compare the two implementations and discuss the total time, and the number of expanded states excluding the last f-layer (printed as "Expanded until last jump"). Also compare your results to using a single pattern database heuristic with up to 1000 abstract states as in exercise (c).

The bash scripts can be used to run your experiments.