The Fast Downward Planning System

Ethan Coots

An example

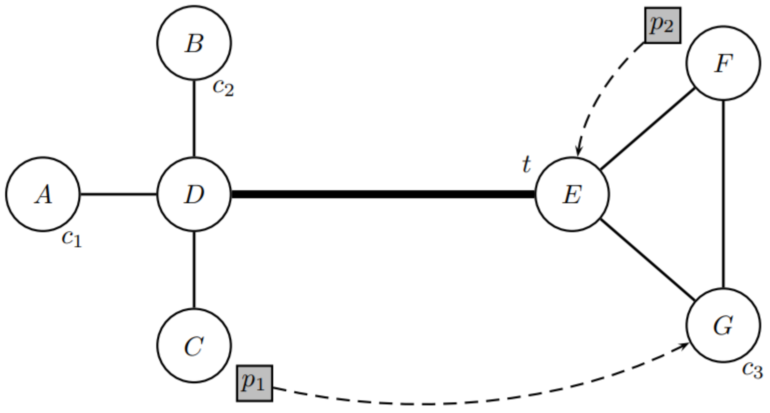

Figure 1: A transportation planning task. Deliver parcel p_1 from C to G and parcel p_2 from F to E, using the cars c_1, c_2, c_3 and truck t. The cars may only use inner-city roads (thin edges), the truck may only use the highway (thick edge).

Propositional Encoding

Variables:
```
  at-p1-a, at-p1-b, at-p1-c, at-p1-d, at-p1-e, at-p1-f, at-p1-g,
  at-p2-a, at-p2-b, at-p2-c, at-p2-d, at-p2-e, at-p2-f, at-p2-g,
  at-c1-a, at-c1-b, at-c1-c, at-c1-d,
  at-c2-a, at-c2-b, at-c2-c, at-c2-d,
  at-c3-e, at-c3-f, at-c3-g,
  at-t-d, at-t-e,
  in-p1-c1, in-p1-c2, in-p1-c3, in-p1-t,
  in-p2-c1, in-p2-c2, in-p2-c3, in-p2-t
```
Init:
```
  at-p1-c, at-p2-f, at-c1-a, at-c2-b, at-c3-g, at-t-e
```
Goal:
```
  at-p1-g, at-p2-e
```
Operator `drive-c1-a-d`:
  PRE: `at-c1-a` ADD: `at-c1-d` DEL: `at-c1-a`
Operator `drive-c1-b-d`:
  PRE: `at-c1-b` ADD: `at-c1-d` DEL: `at-c1-b`
Operator `drive-c1-c-d`:
  PRE: `at-c1-c` ADD: `at-c1-d` DEL: `at-c1-c`
...
Operator `load-c1-p1-a`:
  PRE: `at-c1-a, at-p1-a` ADD: `in-p1-c1` DEL: `at-p1-a`
Operator `load-c1-p1-b`:
  PRE: `at-c1-b, at-p1-b` ADD: `in-p1-c1` DEL: `at-p1-b`
Operator `load-c1-p1-c`:
  PRE: `at-c1-c, at-p1-c` ADD: `in-p1-c1` DEL: `at-p1-c`
...
Operator `unload-c1-p1-a`:
  PRE: `at-c1-a, in-p1-c1` ADD: `at-p1-a` DEL: `in-p1-c1`
Operator `unload-c1-p1-b`:
  PRE: `at-c1-b, in-p1-c1` ADD: `at-p1-b` DEL: `in-p1-c1`
Operator `unload-c1-p1-c`:
  PRE: `at-c1-c, in-p1-c1` ADD: `at-p1-c` DEL: `in-p1-c1`

Domain Transition Graphs

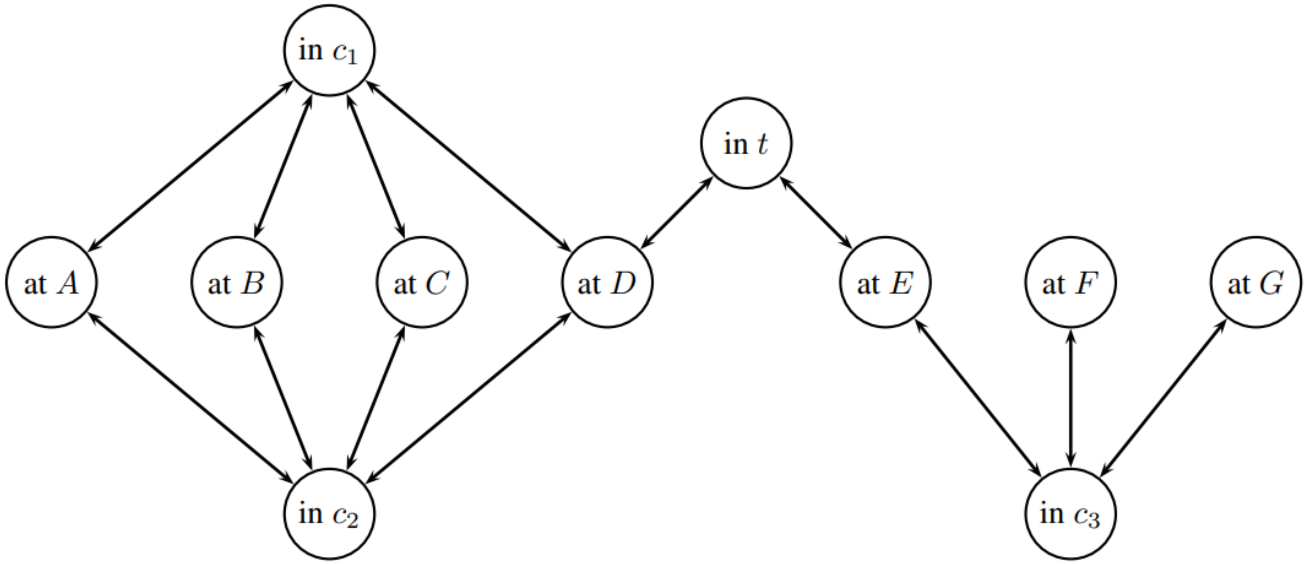

Figure 3: Domain transition graph for the parcels p_1 and p_2. Indicates how a parcel can change its state. For example, the arcs between "at D" and "in t" correspond to the actions of loading/unloading the parcel at location D with the truck t.

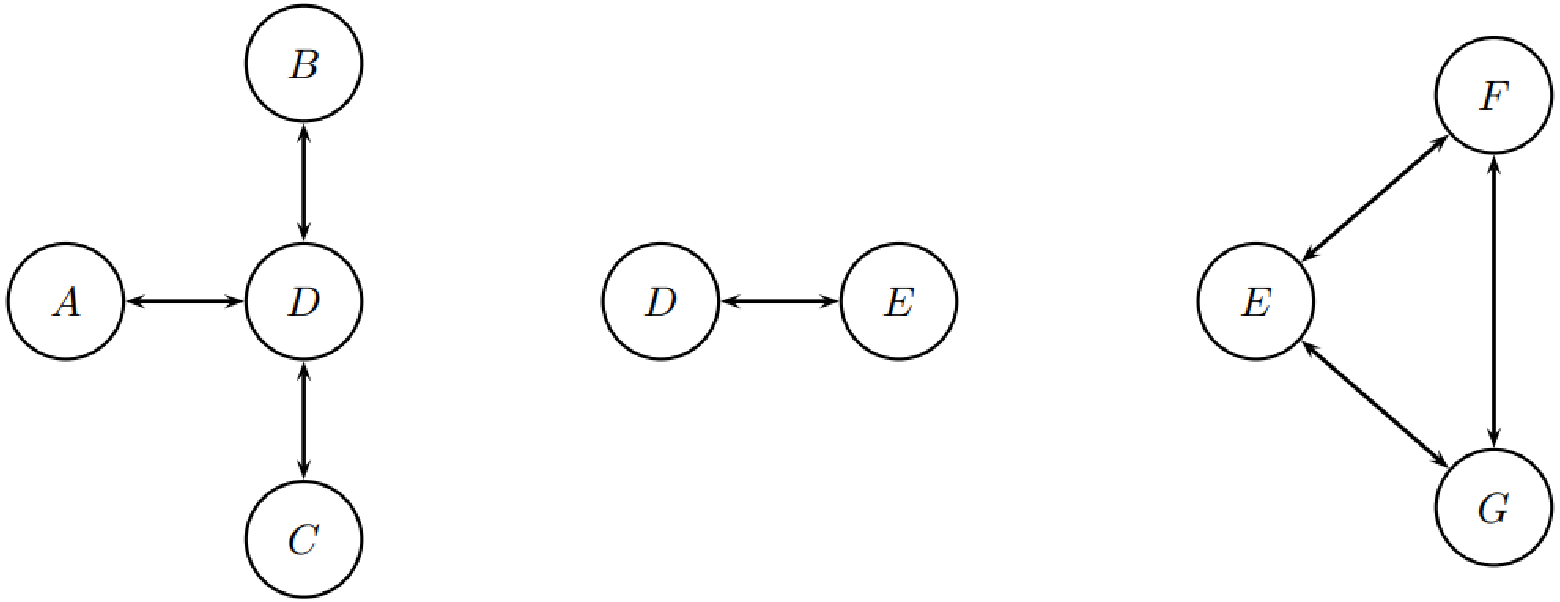

Figure 4: Domain transition graphs for the cars c_1 and c_2 (left), truck t (centre), and car c_3 (right). Note how each graph corresponds to the part of the roadmap that can be traversed by the respective vehicle.

Causal Graph

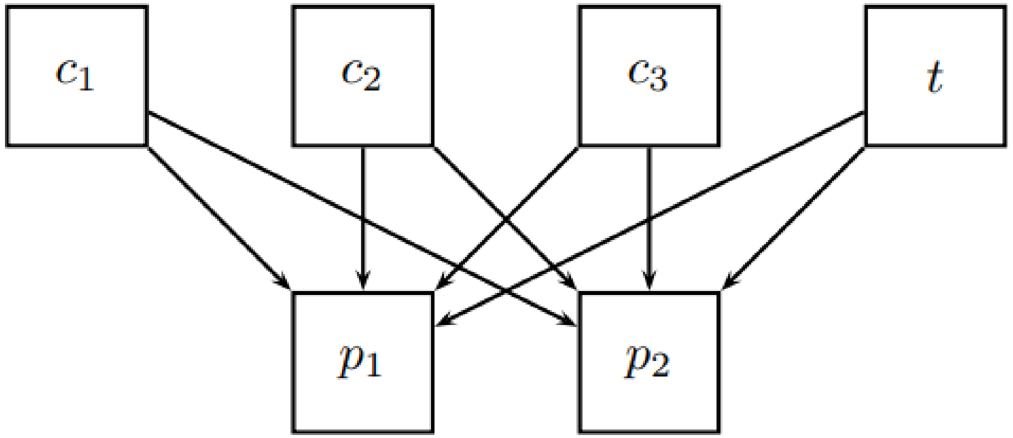

Figure 6: Causal dependencies in the transportation planning task.

Fast Downward Planning System

- Goal: *develop an algorithm that efficiently solves general propositional planning tasks by exploiting the hierarchical structure inherent in causal graphs*
- Obstacles:
  - Encoding PDDL representations to FDR
  - Cycles (Some causal graphs are not hierarchical in nature)
  - Finding a solution is difficult (still PSPACE-complete)

Three stages

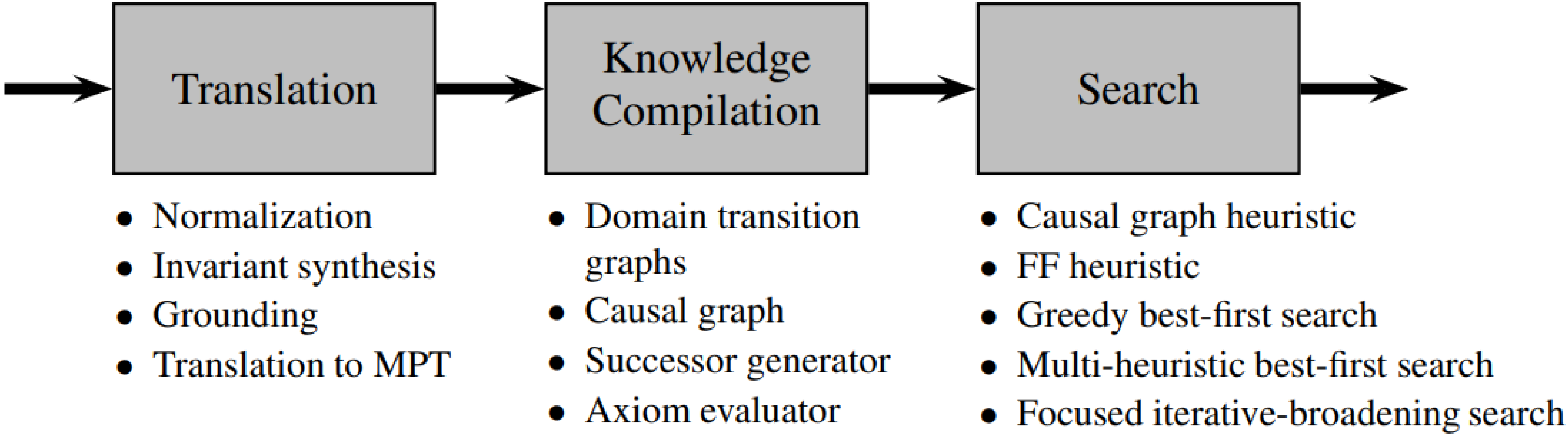

Figure 9: The three phases of Fast Downward's execution.

Translation

- Turns PDDL input into *multi-valued planning task*
- As previously discussed, four stages:
  - Normalization
  - Invariant synthesis
  - Grounding
  - Translation to multi-valued planning task

Knowledge Compilation

- Generates four data structures:
  - Domain transition graphs: encode how state variables can change their values
  - Causal graphs: shows hierarchical dependencies between state variables
  - Successor generator: determines the set of applicable operators for a given state
  - Axiom evaluator: computes values of derived variables

Robot Domain Graph

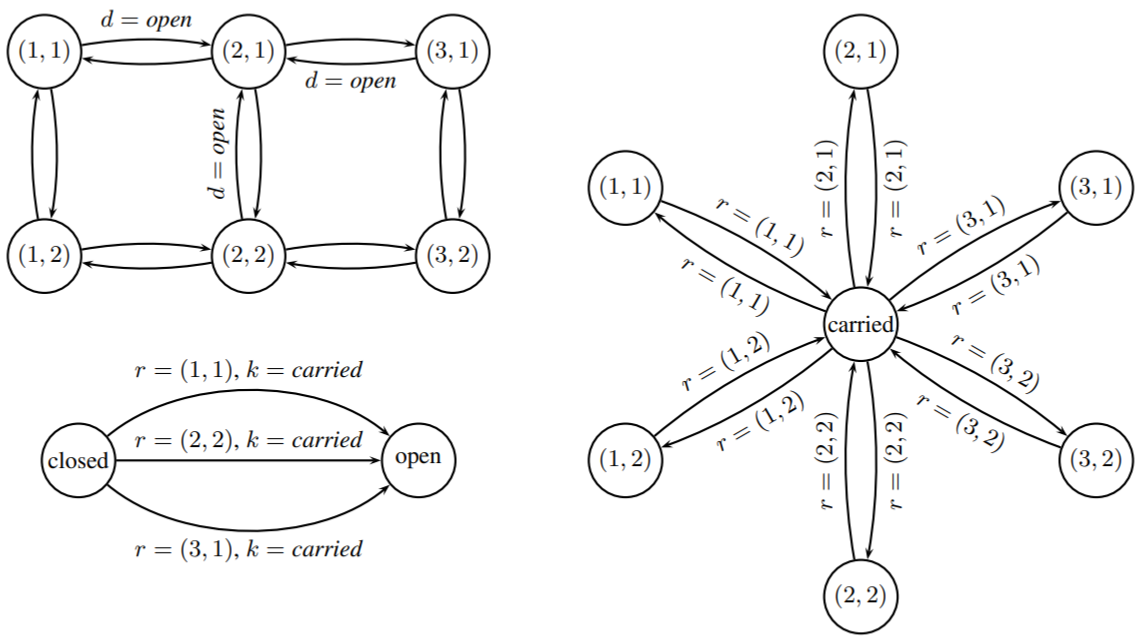

Figure 10: Domain transition graphs of a GRID task. Top left: $DTG(r)$ (robot); right: $DTG(k)$ (key); bottom left: $DTG(d)$ (door).

Relaxing/Eliminating cycles

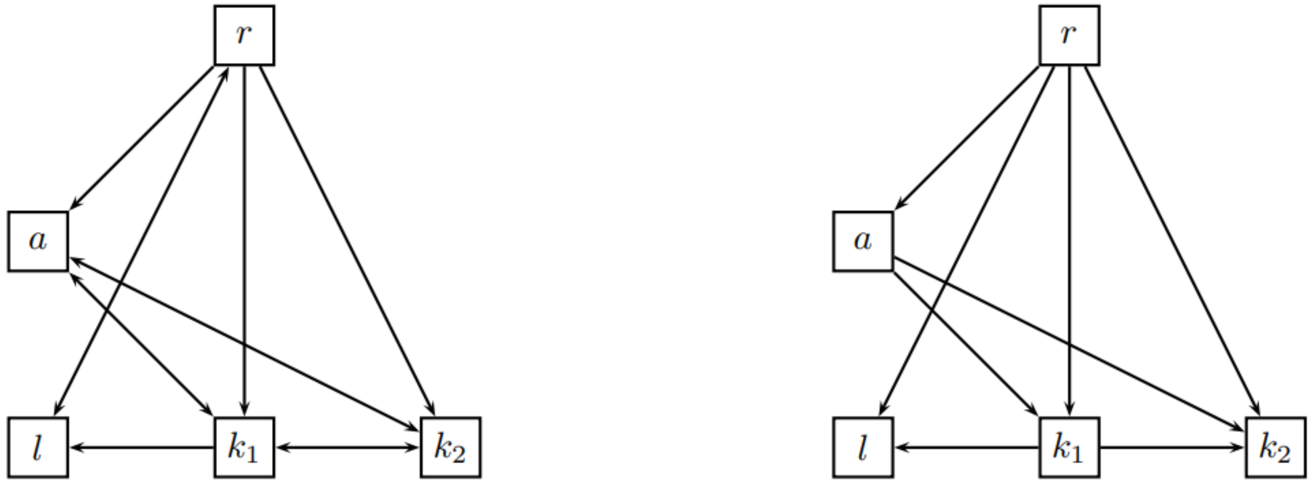

Figure 15: Causal graph of a GRID task (left) and of a relaxed version of the task (right). State variable r encodes the location of the robot, a encodes the status of the robot arm (empty or carrying a key), l encodes the status of the locked location (locked or open), and k_1 and k_2 encode the locations of the two keys.

Search

- Uses three different search functions:
  - Best-first search using causal graph heuristic
  - Multi-heuristic best-first search (combines causal graph and FF heuristics)
  - Focused iterative-broadening search: estimates "usefulness" of operators

The Causal Graph Heuristic

- Centerpiece of Fast Downward’s heuristic search
- Estimates cost of reaching goal by solving a sample of subproblems within small “windows” of the causal graph
- Computing costs for a variable-value pairing is similar to Dijkstra’s algorithm within the causal graph
  - Caching this data improves performance
- Definitions:
  - Derived predicates: occur in the head of an axiom
  - Fluent predicates: occur in the initial state or in the effects of operators

```
algorithm compute-costs(Π, s, v, d):
    Let V' be the set of immediate predecessors of v in the pruned causal graph of Π.
    Let DTG be the pruned domain transition graph of v.
    cost_v(d, d) := 0
    cost_v(d, d') := ∞ for all d' ∈ D_v \ {d}
    local-state_d := s restricted to V'
    unreached := D_v
    while unreached contains a value d' ∈ D_v with cost_v(d, d') < ∞:
        Choose such a value d' ∈ unreached minimizing cost_v(d, d').
        unreached := unreached \ {d'}
        for each transition t in DTG leading from d' to some d'' ∈ unreached:
            transition-cost := 0 if v is a derived variable; 1 if v is a fluent
            for each pair v' = e' in the condition of t:
                e := local-state_d'(v')
                call compute-costs(Π, s, v', e).
                transition-cost := transition-cost + cost_v'(e, e')
            if cost_v(d, d') + transition-cost < cost_v(d, d''):
                cost_v(d, d'') := cost_v(d, d') + transition-cost
                local-state_d'' := local-state_d'
                for each pair v' = e' in the condition of t:
                    local-state_d''(v') := e'
```

Figure 18: Fast Downward's implementation of the causal graph heuristic: the *compute-costs* algorithm for computing the estimates $cost_v(d, d')$ for all values $d' \in \mathcal{D}_v$ in a state s of an MPT Π.