Online appendix for the paper

Planning as Tabled Logic Programming

published in Theory and Practice of Logic Programming

NENG-FA ZHOU
CUNY Brooklyn College and Graduate Center

ROMAN BARTÁK
Charles University

AGOSTINO DOVIER
Univ. di Udine

submitted 29 April 2015; revised 3 July 2015; accepted 14 July 2015

Appendix A Benchmarks used in the paper

In this section we summarize, for reader's convenience, the descriptions of all the domains used as benchmarks. Descriptions are drawn from `https://helios.hud.ac.uk/scommv/IPC-14/domains_sequential.html`; Picat's complete encodings for these benchmarks are available at `http://picat-lang.org/ipc14/`.

A.1 Barman

There is a robot *barman* that manipulates drink dispensers, glasses, and a shaker. The goal is to find a plan of robot's actions that serves a desired set of drinks. Robot hands can grasp at most one object at a time. Glasses need to be empty and clean to be filled. The benchmark was proposed by Sergio Jiménez Celorrio.

In Figure A 1 we represent the initial configuration and the corresponding input specifications.

Actions available are:

- `grasp(OBJ)` that executes the grasping either of a specific shot or shaker (`OBJ`)
- `leave(OBJ)` that allows us to leave the shot or shaker (`OBJ`)
- `fill_shot(SHOT,ING)` that allows us to fill the shot `SHOT` with the ingredient `ING`
- `empty_shot(SHOT)` (resp., `empty_shaker(SHAKER)`) that allows us to empty the shot `SHOT` (resp., the skaker `SHAKER`)
- `clean_shot(SHOT)` (resp., `clean_shaker(SHAKER)`) that allows us to clean the shot `SHOT` (resp., the skaker `SHAKER`)

```
%% INITIAL state
ontable(shaker1),                        ontable(shot1),
... ,                                    ontable(shot8),
clean(shaker1),                          clean(shot1),
... ,                                    clean(shot8),
empty(shaker1),                          empty(shot1),
...,                                     empty(shot8),
dispenses(dispenser1,ingredient1),       dispenses(dispenser2,ingredient2),
dispenses(dispenser3,ingredient3),       dispenses(dispenser4,ingredient4),
handempty(left),                         handempty(right),
%% Cocktail rules
cocktail_part1(cocktail1,ingredient1),   cocktail_part2(cocktail1,ingredient3),
...
cocktail_part1(cocktail6,ingredient2),   cocktail_part2(cocktail6,ingredient1),
%% GOAL
contains(shot1,cocktail1),               contains(shot2,cocktail1),
contains(shot3,cocktail2),               contains(shot4,cocktail6).
```

Fig. A 1. Example of Barman instance

- `pour_shot_to_shaker(SHOT,SHAKER)` (resp., `pour_shaker_to_shot(SHAKER,SHOT)`) that allows us to pour the content of the shot SHOT in the shaker `SHAKER` (resp., vice versa).
- `shake(SHAKER)` that executes that shaking of the shaker to mix the ingredients.
- `reduce` (remove a shot from the state once it contains a required cocktail

All actions have cost 1 but `reduce` that has cost 0.

A.2 Cave Diving

There is a set of divers, each of who can carry four tanks of air. These divers must be hired to go into an underwater cave and either take photos or prepare the way for other divers by dropping full tanks of air. The cave is too narrow for more than one diver to enter at a time. Divers have a single point of entry. Certain rooms of the cave branches are objectives that the divers must photograph. Swimming and photographing both consume air tanks. Divers must exit the cave and decompress at the end. They can therefore only make a single trip into the cave. Certain divers have no confidence in other divers and will refuse to work if someone they have no confidence in has already worked. Divers have hiring costs inversely proportional to how hard they are to work with. This domain was proposed by Nathan Robinson, Christian Muise, and Charles Gretton.

The cave system is represented by an undirected acyclic graph. Divers can carry an amount of tanks according to their capacity. Rooms that need to be reached are among the leaves of the graph. In Figure A 2 we represent an instance of the problem.

```
%% Divers information
available(d0)                available(d1)          available(d2)
capacity(d0,four)            capacity(d1,four)      capacity(d2,four)
=(hiring_cost(d0),60)        =(hiring_cost(d1),10)  =(hiring_cost(d2),10)
precludes(d1,d2)
%% Cave and tank information
in_storage(t1)
next_tank(t1,t2)             ...                    next_tank(t8,t9)
cave_entrance(l0)
connected(l0,l1),            ...                    connected(l5,l1)
%%GOAL
have_photo(l4)               have_photo(l5)
decompressing(d0)            decompressing(d1)
decompressing(d2)            decompressing(d3)
```

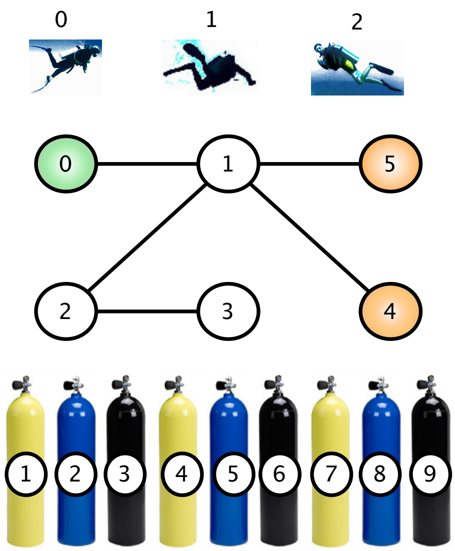

Fig. A 2. Example of Cave Diving instance

The actions available are:

- `hire_diver(Diver)` that requires the availability of hiring cost and should satisfy the compatibility constraints among divers,
- `prepare_tank(T)` that prepares the tank `T` for the current diver if his capacity allows it,
- `enter_water` that requires the diver to be in the cave entrance,
- `photograph(Loc)` that requires the diver to be in the target location `Loc`,
- `drop_tank(Loc)` that allows the diver to leave a tank in the location `Loc` (the tank can be either full or empty),
- `swim(Loc1,Loc2)` that allows the diver to swim between two locations that are adjacent in the graph,
- `pickup_tank(Loc)` that allows the diver to collect a tank stored in the location `Loc`,
- `decompress` should be made at the end of diving in the cave entrance.

Each action `swim` and `photograph` consumes (empties) one air tank. All actions but the first one have unitary cost.

A.3 Childsnack

This domain is to plan how to make and serve sandwiches for a group of children in which some are allergic to gluten. There are two actions for making sandwiches from their ingredients. The first one makes a sandwich and the second one makes a sandwich taking into account that all ingredients are gluten-free. There are also actions to put a sandwich on a tray and to serve sandwiches. Problems in this domain define the ingredients to make sandwiches at the initial state. Goals consist of having selected kids served with a sandwich to which they are not allergic. This domain was proposed by Raquel Fuentetaja, Tomàs de la Rosa Turbides.

Available actions are the following:

- `make_sandwich_no_gluten(Sw,B,Co)` and `make_sandwich(Sw,B,Co)` where `SW` is a sandwich, `B` is a (no-gluten) bread, and `Co` is a (no-gluten) content allows us to make the sandwiches.
- `put_on_tray(Sw,T)` puts the sandwich `Sw` on the tray `T`
- `serve_sandwich_no_gluten(Sw,Ch,T,Loc)` and `serve_sandwich(Sw,Ch,T,Loc)` serves the (no-gluten) sandwich `Sw` which is on tray `T` to the children `Ch` at the location `Loc`
- `move_tray(T,Loc1,Loc2)`, where `T` is a tray, `Loc1` and `Loc2` is a location (i.e., a table, the kitchen)

Each action has cost 1. `make_sandwich` (no-gluten) consumes ingredients.

A.4 Citycar

This model aims to simulate the impact of road building/demolition on traffic flows. A city is represented as an acyclic graph, in which each node is a junction and edges are “potential” roads. Some cars start from different positions and have to reach their final destination as soon as possible. The agent has a finite number of roads

```
%% Positions
at(tray1,kitchen)              at(tray2,kitchen)
at_kitchen_bread(bread1)       at_kitchen_bread(bread2)
at_kitchen_bread(bread3)
at_kitchen_content(content1)   at_kitchen_content(content2)
at_kitchen_content(content3)   at_kitchen_content(content4)
at_kitchen_content(content5)
no_gluten_bread(bread3)
no_gluten_content(content1)    no_gluten_content(content4)
waiting(child1,table1)         waiting(child2,table2)
waiting(child3,table3)         waiting(child4,table4)
%% Info on allergies
allergic_gluten(child2)        allergic_gluten(child4)
%% Not yet ready sandwhiches
notexist(sandw1)               notexist(sandw2)
notexist(sandw3)               notexist(sandw4)
notexist(sandw5)               notexist(sandw6)
%% Goal
served(child2)                 served(child3)
```

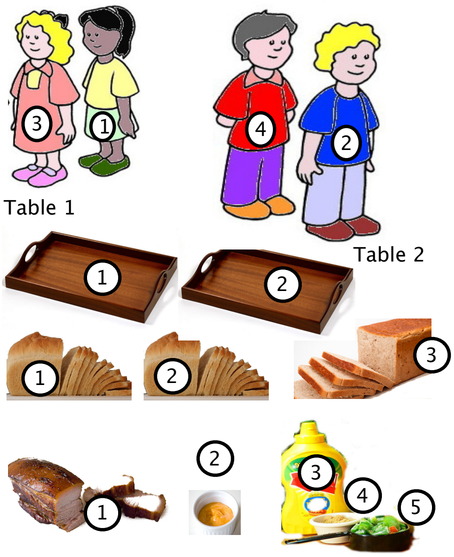

Fig. A 3. Example of Childsnack instance

available, which can be built for connecting two junctions and allowing a car to move between them. Roads can also be removed, and placed somewhere else, if needed. In order to place roads or to move cars, the destination junction must be clear, i.e., no cars should be in there. The domain was proposed by Mauro Vallati.

Allowed actions are the following:

- `car_arrived(Dest)`, which has cost 0. It allows to remove a car from the network and to remove the occurrence of the destination `Dest` (a junction) from the list of all final destinations.

```
%% Initial State
same_line(junction0_0,junction0_1)    same_line(junction0_1,junction0_0)
same_line(junction1_0,junction1_1)    same_line(junction1_1,junction1_0)
same_line(junction0_0,junction1_0)    same_line(junction1_0,junction0_0)
same_line(junction0_1,junction1_1)    same_line(junction1_1,junction0_1)
diagonal(junction0_0,junction1_1)     diagonal(junction1_1,junction0_0)
diagonal(junction0_1,junction1_0)     diagonal(junction1_0,junction0_1)
clear(junction0_0)                    clear(junction0_1)
clear(junction1_0)                    clear(junction1_1)
at_garage(garage0,junction0_1)
starting(car0,garage0)                starting(car1,garage0)
%% GOAL
arrived(car0,junction1_1)             arrived(car1,junction1_0)
```

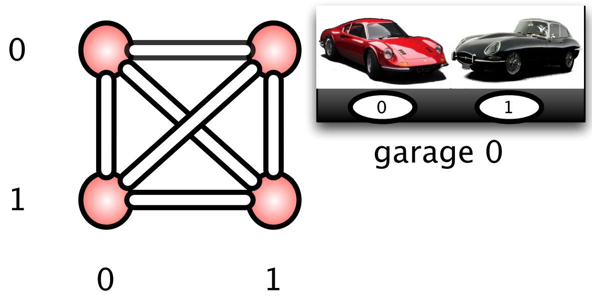

Fig. A 4. Example of Citycars instance

- `car_start(Loc)`: A car is put in the road from the garage of location `Loc`: it has cost 1.
- `move_car_in_road(FromLoc)` allows us to move a car in a road from the junction `FromLoc` (cost 1—the road is a straight line or a diagonal road starting in `FromLoc`).
- `move_car_out_road(ToLoc)` allows us to move a out of a road as soon as the junction `ToLoc` is reached by the car (cost 1—the road is a straight line or a diagonal road ending in `ToLoc`).
- These actions allow us to build diagonal, straight roads or of deleting one road:
  - `build_diagonal_oneway(FromLoc,ToLoc)` (cost 30),
  - `build_straight_oneway(FromLoc,ToLoc)` (cost 20),
  - `destroy_road(FromLoc,ToLoc)` (cost 10).

Let us observe that search symmetries are eliminated by considering the cars equivalent during the search. It is trivial to label them a-posteriori given a correct plan.

A.5 GED

The GED problem is to find a min-cost sequence of operations that transforms one genome (signed permutation of genes) into another. The purpose of this is to use this cost as a measure of the distance between the two genomes, which is used to construct hypotheses about the evolutionary relationship between the organisms. The domains was proposed by Patrik Haslum.

This problem can be stated at several abstraction levels. A general version could include gene insertions and deletions. Let us focus on the abstraction level and on the three rules required by the competition benchmarks.

A gene is identified by a symbolic name. The connection between genes is stated by a binary predicate `cw` that encodes a linear graph. Each gene can occur in a regular direction (normal) or in reverse direction (inverted).

The three rules allowed are cut (of a substring) from the main genome, and then a splice of the cut substring directed or reversed in a selected point of the main genome. The reverse of a single gene is also allowed. Just to fix the ideas, let us consider the example in figure A 5. Reversed genes are overlined.

`%% INITIAL STATE`
 	
`normal(a),`
`normal(b),`
`normal(c),`
`normal(d),`
`cw(a,b),`
`cw(b,c),`
`cw(c,d)`
	a · b · c · d
	⇓ (cut 2–4, temp situation)
	a b · c d
	⇓ (cut 2-4, final situation)
	a · d b · c
	⇓ (reverse of the 2nd string)
	a · d $\overline{\mathtt{c}} \cdot \overline{\mathtt{b}}$
	⇓ (and splice in the 1st)
	a · $\overline{\mathtt{c}} \cdot \overline{\mathtt{b}}$ · d
		`%% GOAL`
		
		`normal(a),`
		`inverted(b),`
		`inverted(c),`
		`normal(d),`
		`cw(a,c),`
		`cw(c,b),`
		`cw(b,d)`

Fig. A 5. An instance of the GED problem and a possible solution

Each complex action (cut and splice) is split in some sub-actions as done by Patrik Haslum in his PDDL encoding (`http://picat-lang.org/ipc14/ged.pddl`).

A.6 Floortile, Parking, and Tetris

For the three domains discussed extensively in the core of paper we only show here an instance both in concrete form and as a picture (see Figures A 6–A 8). The Transport domain is discussed in detail in the next section.

```
%% Floor description
clear(01)                          ...   clear(64),
up(11,01)                          ...   up(64,54),
down(01,11)                        ...   down(54,64)
right(02,01)                       ...   right(64,63)
left(01,02)                        ...   left(63,64)
%% Robots positions and states
robot_at(robot1,11)                robot_has(robot1,white)
robot_at(robot2,52)                robot_has(robot2,black)
available_color(white)             available_color(black)
%% GOAL
painted(11,white)                  painted(12,black)
painted(13,white)                  painted(14,black)
⋮                                  ⋮
painted(61,black)                  painted(62,white)
painted(63,black)                  painted(64,white)
```

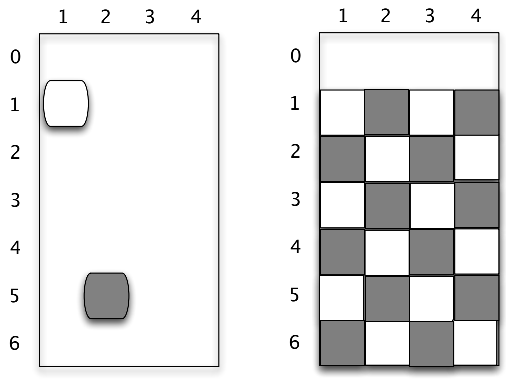

Fig. A 6. Example of Floortile instance. A solution with plancost 104 exists (benchmark instance p01642).

```
%% INITIAL STATE

at_curb(car3),              at_curb_num(car3,curb0),
behind_car(car2,car3),      car_clear(car2),
at_curb(car4),              at_curb_num(car4,curb1),
behind_car(car10,car4),     car_clear(car10),
at_curb(car0),              at_curb_num(car0,curb2),
behind_car(car5,car0),      car_clear(car5),
at_curb(car1),              at_curb_num(car1,curb3),
behind_car(car9,car1),      car_clear(car9),
at_curb(car7),              at_curb_num(car7,curb4),
behind_car(car8,car7),      car_clear(car8),
at_curb(car11),             at_curb_num(car11,curb5),
behind_car(car6,car11),     car_clear(car6),
curb_clear(curb6)

%% GOAL

at_curb_num(car0,curb0),    behind_car(car7,car0),
at_curb_num(car1,curb1),    behind_car(car8,car1),
at_curb_num(car2,curb2),    behind_car(car9,car2),
at_curb_num(car3,curb3),    behind_car(car10,car3),
at_curb_num(car4,curb4),    behind_car(car11,car4),
at_curb_num(car5,curb5),    at_curb_num(car6,curb6)
```

Fig. A 7. An instance of parking (left: initial state, right: goal). A solution with 18 moves exists (benchmark instance p_12_7_01).

```
%% Board description
connected(f0_0f,f0_1f),          ...  connected(f0_2f,f0_3f)
connected(f1_1f,f1_0f),          ...  connected(f1_2f,f1_3f),
                                 ...
connected(f6_0f,f7_0f),          ...  connected(f6_3f,f7_3f)
clear(f0_3f),                    ...  clear(f7_3f),
%% Pieces
at_right_l(rightl0,f0_0f,f1_0f,f1_1f),   at_right_l(rightl1,f2_1f,f3_1f,f3_2f),
at_two(straight0,f0_2f,f1_2f),           at_square(square0,f0_1f)
%% Goal
clear(f0_0f),                    ...  clear(f0_3f)
clear(f1_0f)                     ...  clear(f1_3f)
clear(f2_0f)                     ...  clear(f2_3f)
clear(f3_0f)                     ...  clear(f3_3f)
```

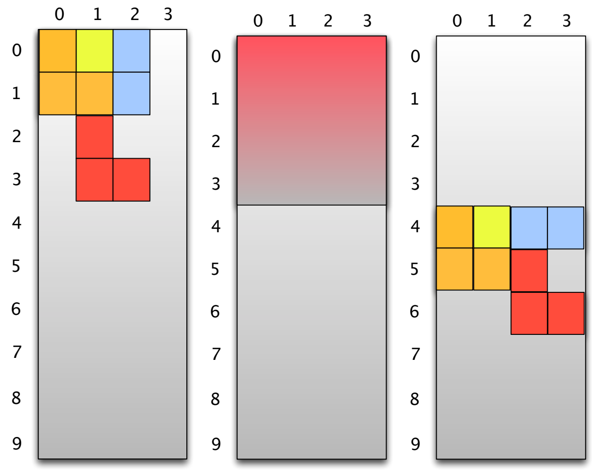

Fig. A 8. Example of Tetris instance: initial state (left), goal (center). A plan of length 36 exists (instance 01_8 of the benchmarks) leading to the final situation to the right.

Appendix B The Transport Domain

B.1 PDDL Encoding of the Transport Domain

```
(define (domain transport)
  (:requirements :typing :action-costs)
  (:types
        location target locatable - object
        vehicle package - locatable
        capacity-number - object
  )

  (:predicates
     (road ?l1 ?l2 - location)
     (at ?x - locatable ?v - location)
     (in ?x - package ?v - vehicle)
     (capacity ?v - vehicle ?s1 - capacity-number)
     (capacity-predecessor ?s1 ?s2 - capacity-number)
  )

  (:functions
     (road-length ?l1 ?l2 - location) - number
     (total-cost) - number
  )

  (:action drive
    :parameters (?v - vehicle ?l1 ?l2 - location)
    :precondition (and
        (at ?v ?l1)
        (road ?l1 ?l2)
      )
    :effect (and
        (not (at ?v ?l1))
        (at ?v ?l2)
        (increase (total-cost) (road-length ?l1 ?l2))
      )
  )

 (:action pick-up
    :parameters (?v - vehicle ?l - location ?p - package ?s1 ?s2 - capacity-number)
    :precondition (and
        (at ?v ?l)
        (at ?p ?l)
        (capacity-predecessor ?s1 ?s2)
        (capacity ?v ?s2)
      )
    :effect (and
        (not (at ?p ?l))
        (in ?p ?v)
        (capacity ?v ?s1)
        (not (capacity ?v ?s2))
        (increase (total-cost) 1)
      )
  )

  (:action drop
    :parameters (?v - vehicle ?l - location ?p - package ?s1 ?s2 - capacity-number)
    :precondition (and
        (at ?v ?l)
        (in ?p ?v)
        (capacity-predecessor ?s1 ?s2)
        (capacity ?v ?s1)
      )
    :effect (and
        (not (in ?p ?v))
        (at ?p ?l)
        (capacity ?v ?s2)
        (not (capacity ?v ?s1))
        (increase (total-cost) 1)
      )
  )
```

```
)
```

B.2 Picat Encoding of the Transport Domain

```
final({Trucks,[]}) =>   % no waiting packages and no loaded packages
    foreach([_Loc,Dests|_] in Trucks)
        Dests == []
    end.

% unload a package
action({Trucks,Packages},NextState,Action,ActionCost),
    select([Loc,Dests,Cap],Trucks,TrucksR),
    select(Loc,Dests,DestsR)   % unload it deterministically
=>
    Action = $unload(Loc),
    ActionCost = 1,
    NewTrucks = insert_ordered(TrucksR,[Loc,DestsR,Cap]),
    NextState = {NewTrucks,Packages}.
action({Trucks,Packages},NextState,Action,ActionCost) ?=>
    Action = $unload(Loc),
    ActionCost = 1,
    select([Loc,Dests,Cap],Trucks,TrucksR),
    select(Dest,Dests,DestsR),
    NewTrucks = insert_ordered(TrucksR,[Loc,DestsR,Cap]),
    NewPackages = insert_ordered(Packages,(Loc,Dest)),
    NextState = {NewTrucks,NewPackages}.

% load a package onto a truck if the truck and the package are at the same location
action({Trucks,Packages},NextState,Action,ActionCost) ?=>
    Action = $load(Loc),
    ActionCost = 1,
    select([Loc,Dests,Cap],Trucks,TrucksR),
    length(Dests) < Cap,
    select((Loc,Dest),Packages,PackagesR),   % the package is at the same location as the truck
    NewTrucks = insert_ordered(TrucksR,[Loc,insert_ordered(Dests,Dest),Cap]),
    NextState = {NewTrucks,PackagesR}.

% drive a truck from Loc to NextLoc
action({Trucks,Packages},NextState,Action,ActionCost) =>
    Action = $move(Loc,NextLoc),
    select([Loc|Tail],Trucks,TrucksR),
    road(Loc,NextLoc,ActionCost),
    NewTrucks = insert_ordered(TrucksR,[NextLoc|Tail]),
    NextState = {NewTrucks,Packages},
    estimate_cost(NextState) =< current_resource()-ActionCost.

table
estimate_cost({Trucks,Packages}) = Cost =>
    LoadedPackages = [(Loc,Dest) : [Loc,Dests,_] in Trucks, Dest in Dests],
    NumLoadedPackages = length(LoadedPackages),
    TruckLocs = [Loc : [Loc|_] in Trucks],
    travel_cost(TruckLocs,LoadedPackages,Packages,0,TCost),
    Cost = TCost+NumLoadedPackages+length(Packages)*2.  % includes load and unload costs

% the maximum of the minimum cost of transporting each single package
travel_cost(_Trucks,[],[],Cost0,Cost) => Cost=Cost0.
travel_cost(Trucks,[(PLoc,PDest)|Packages],Packages2,Cost0,Cost) =>
    Cost1 = min([D1+D2 : TLoc in Trucks,
                         shortest_dist(TLoc,PLoc,D1),
                         shortest_dist(PLoc,PDest,D2)]),
    travel_cost(Trucks,Packages,Packages2,max(Cost0,Cost1),Cost).
travel_cost(Trucks,[],Packages2,Cost0,Cost) =>
    travel_cost(Trucks,Packages2,[],Cost0,Cost).
```

B.3 An Instance of the Transport Domain

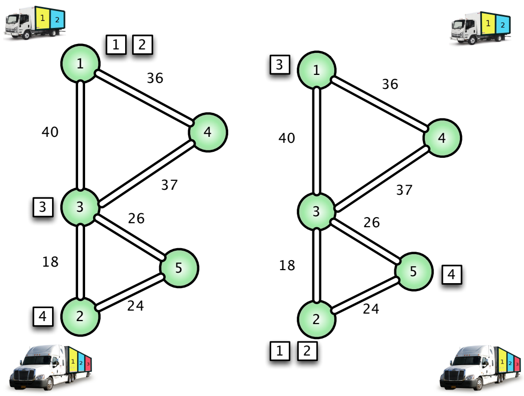

Fig. B 1. An Instance of the Transport Domain (p01).

B.3.1 Solving the instance with Picat

```
main =>
    Facts =
      $[road(c3,c1,40),road(c1,c3,40),road(c3,c2,18),
        road(c2,c3,18),road(c4,c1,36),road(c1,c4,36),
        road(c4,c3,37),road(c3,c4,37),road(c5,c2,24),
        road(c2,c5,24),road(c5,c3,26),road(c3,c5,26)],
    cl_facts(Facts,[$road(+,-,-)]),
    Trucks = [[c2,[],3],[c1,[],2]],
    Packages = [(c1,c2),(c1,c2),(c3,c1),(c2,c5)],
    best_plan({sort(Trucks),sort(Packages)},Plan,PlanCost),
    foreach ({I,Action} in zip(1..len(Plan),Plan))
        printf("%3d. %w\n",I,Action)
    end,
    println(plan_cost=PlanCost).
```

B.3.2 An Optimal Plan for the Instance

```
  1. load(c1)
  2. load(c1)
  3. load(c2)
  4. move(c1,c3)
  5. move(c2,c5)
  6. unload(c5)
  7. move(c3,c2)
  8. unload(c2)
  9. unload(c2)
 10. move(c2,c3)
 11. load(c3)
 12. move(c3,c1)
 13. unload(c1)

 plan_cost = 148
```