Search and Planning Exercises

Instituto Superior Técnico

2022-23

Contents

Chapter 1

Constraint Programming

Exercise 0

A Latin square of order n is defined as an $n \times n$ matrix made out of the integers in $[1..n]$ with the property that each of these n integers occurs exactly once in each row and exactly once in each column of the array. The following matrix is an example of a Latin square of order 5.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \\ 3 & 4 & 5 & 1 & 2 \\ 4 & 5 & 1 & 2 & 3 \\ 5 & 1 & 2 & 3 & 4 \end{bmatrix}$$

a) Propose a formulation of the problem as a constraint network. Identify variables, domains, and constraints.

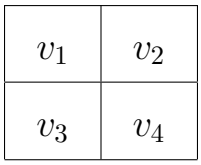
b) Now consider the problem of orthogonal Latin squares. Let A and B be Latin squares of order n and let the entry on the i^{th} row and the j^{th} column of A and B be denoted as a_{ij} and b_{ij}, respectively, with $i, j = 1, ..., n$. A and B are orthogonal if the n^2 order pairs (a_{ij}, b_{ij}) are all distinct. For example, the following juxtaposed Latin squares are orthogonal.

$$\begin{bmatrix} (3,2) & (2,3) & (1,1) \\ (2,1) & (1,2) & (3,3) \\ (1,3) & (3,1) & (2,2) \end{bmatrix}$$

Propose a formulation of the problem as a constraint network. Identify variables, domains, and constraints.

Exercise 1

Consider the following 2×2 matrix. Suppose we want to fill the matrix with numbers ranging from 0 to 9 such that for each entry in the matrix the numbers in the entries below and to the right are exactly one unit smaller.



a) Identify the variables involved, domains, and constraints between the identified variables and draw the corresponding constraint graph.

b) Perform a backtracking search without constraint propagation. The variables and values must be chosen using an alphabetical/numerical ascending order.

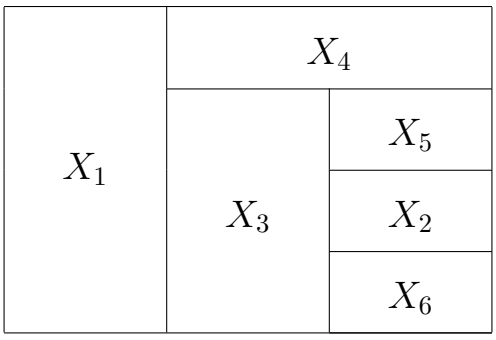
c) Solve the problem as described in b) combined with forward checking.

d) Apply the AC3 algorithm to the constraint graph to obtain arc consistency.

e) After applying the AC-3, use the backtracking algorithm to solve the problem as described in b).

Exercise 2

Consider the following diagram, in which different areas are labelled with letters. Suppose we want to paint each area with a shade of red (light, medium or dark red), such that no adjacent areas are painted with the same shade. Moreover, the shade of X_1 must be lighter than the shade of X_4, and the shade of X_4 must be lighter than the shade of X_3.



a) Model this problem as a CSP, showing the respective variables, domains and constraints.

b) Perform a backtracking search without constraint propagation. The variables and values must be chosen using an alphabetical/numerical ascending order.

c) Solve the problem as described in b) combined with forward checking.

d) Apply the AC3 algorithm to the constraint graph to obtain arc consistency. Apply the algorithm to create the first 10 lines of the representation used in the course

e) After applying AC-3, use the backtracking algorithm to solve the problem as described in b).

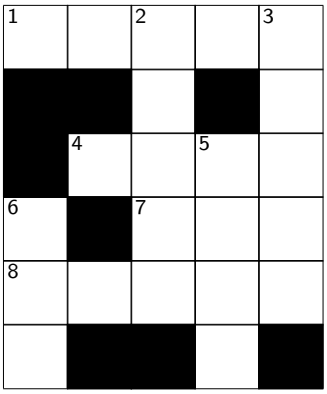
f) Perform a backtracking using the *Gashnig backjumping algorithm*. The variables and values must be chosen using an alphabetical/numerical ascending order.

g) Perform a backtracking using *graph-based backjumping*. The graph order is $(X_1, X_2, X_3, X_4, X_5, X_6)$ and values must be chosen using an alphabetical/numerical ascending order.

h) Perform backtracking using *conflict-based backjumping*. The variables and values must be chosen using an alphabetical/numerical ascending order.

Exercise 3

Suppose we want to use constraint satisfaction techniques to solve the crossword puzzle presented below.



AFT
ALE
LEE
EEL
TIE
LINE
HEEL
HIKE
KEEL
KNOT
LASER
SAILS
SHEET
HOSES
STEER

a) Identify the variables involved and their respective domains. Identify the constraints between the identified variables and draw the corresponding constraint graph.

b) After applying the AC-3, use the *backtracking* algorithm with *forward checking* to solve the problem as described in b).

Exercise 4

Given the following crypto-arithmetic problem.

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

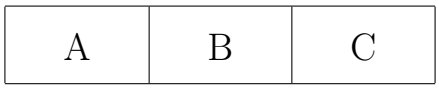
a) Identify the variables involved and their respective domains. Identify the constraints between the identified variables and draw the corresponding constraint graph.

b) Perform the 34 first assignments using backtracking search without using constraint propagation. The variables and values must be chosen using an alphabetical/numerical ascending order.

c) Perform the first 31 assignments using backtracking search combined with forward checking.

Exercise 5

Consider the following figure with squares labelled with letters. Suppose we want to assign an integer from 1 to 8 to each square such that the number in square B must be half the number of square A (using integer division), and the number of square C must be smaller than the number of square B.



a) Identify the variables involved and their respective domains. Identify the constraints between the identified variables and draw the corresponding constraint graph.

b) Perform a backtracking search without using constraint propagation. The variables and values must be chosen using an alphabetical/numerical ascending order.

c) Solve the problem as described in b) combined with forward checking.

d) Apply the AC-3 algorithm to the constraint graph to obtain arc consistency.

e) After applying the AC-3, use the backtracking algorithm to solve the problem as described in b).

f) Perform a backtracking using the *Gashnig backjumping algorithm*. The variables and values must be chosen using an alphabetical/numerical ascending order.

g) Perform a backtracking using *graph-based backjumping*. The variables and values must be chosen using an alphabetical/numerical ascending order.

h) Perform a backtracking using *conflict-based backjumping*. The variables and values must be chosen using an alphabetical/numerical ascending order.

i) **Briefly** discuss the advantages of backjumping when compared with backtracking.

Exercise 6

You are in charge of scheduling for computer science classes that meet Mondays, Wednesdays and Fridays. There are 5 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

The classes are:

- Class 1 - Intro to Programming: meets from 8:00-9:00am
- Class 2 - Intro to Artificial Intelligence: meets from 8:30-9:30am
- Class 3 - Natural Language Processing: meets from 9:00-10:00am
- Class 4 - Computer Vision: meets from 9:00-10:00am
- Class 5 - Machine Learning: meets from 9:30-10:30am

The professors are:

- Professor A, who is available to teach Classes 3 and 4.

- Professor B, who is available to teach Classes 2, 3, 4, and 5.
- Professor C, who is available to teach Classes 1, 2, 3, 4, 5.

a) Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely but may be implicit rather than explicit.

b) Draw the constraint graph associated with your CSP.

c) Show the domains of the variables after running arc-consistency on this initial graph (after having already enforced any unary constraints).

d) Give one solution to this CSP.

Exercise 7

You are designing a menu for a special event. There are several choices, each represented as a variable: (A)ppetizer, (B)everage, main (C)ourse, and (D)essert. The domains of the variables are as follows:

- A: (v)eggies, (e)scargot
- B: (w)ater, (s)oda, (m)ilk
- C: (f)ish, (b)eef, (p)asta
- D: (a)pple pie, (i)ce cream, (ch)eese

Because all of your guests get the same menu, it must obey the following dietary constraints:

(i) Vegetarian options: The appetizer must be veggies or the main course must be pasta or fish (or both).

(ii) Total budget: If you serve the escargot, you cannot afford any beverage other than water.

(iii) Calcium requirement: You must serve at least one of milk, ice cream, or cheese.

a) Draw the constraint graph over the variables A, B, C, and D.

b) Imagine we first assign $A = e$. Cross out eliminated values to show the domains of the variables after forward checking.

c) Again imagine we first assign $A = e$. Cross out eliminated values to show the domains of the variables after arc consistency has been enforced.

d) Give a solution for this CSP or state that none exists.

e) For general CSPs, will enforcing arc consistency after an assignment always prune at least as many domain values as forward checking? Briefly explain why or why not.

Exercise 8

Arthur is looking for a group of friends for his start-up, which develops and provides some web-based p2p downloading solutions to college students (this is before the lawsuits). Arthur has determined that he needs 2 C# Programmers, 2 Flash Designers, 1 Photoshop Guru, 1 Database Admin, and 1 Systems Engineer.

Assume that if a person knows two languages/softwares, he or she can take on two roles in the company.

So Arthurs narrowed down his selections to the following people:

Name	Abilities
Peter	C# and Flash
John	Photoshop and Flash
Jim	Flash and Systems
Jane	C# and Database
Mary	Photoshop and Flash
Bruce	Systems and C#
Chuck	Photoshop and Flash

a) Suppose Arthur knows C#, and only has funds to hire three more people. Model this scenario as a CSP (using variables, value domains, and constraints).

b) Suppose Arthur decides to make Jim a co-founder. Arthur and Jim discover that all the developers absolutely refuse to abandon their favourite platforms, and that they can only afford two single-booted workstations.

Name	Abilities	OS
Arthur	C#	Windows
Peter	C# and Flash	Windows
John	Photoshop and Flash	Windows
Jim	Flash and Systems	FreeBSD
Jane	C# and Database	FreeBSD
Mary	Photoshop and Flash	Linux
Bruce	Systems and C#	Linux
Chuck	Photoshop and Flash	Windows

What are the domains for the two remaining positions after constraint propagation?

c) Assume Arthur and Jim hired Bruce and Mary and they have awarded a contract for their first project, a rush job due Friday at 5 PM. It's Monday at 9 AM and they have to put in 50 total hours of work on it by the due date. There are a number of constraints associated with their work schedules:

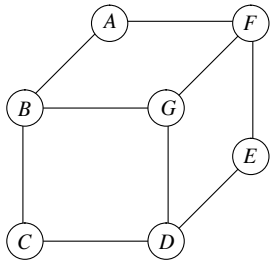
- They only have access to two machines of the requisite platform, and only have access to those two machines between 9 AM and 5 PM every day.
- Each person can work a maximum of 20 hours over the course of the week.
- A work session by a single person on a particular machine can last no fewer than two hours.
- Arthur cannot work from 12-4 PM Tu/Th.
- Jim can't work MWF 9-12.
- Bruce can only work between noon and 2 PM every day.

- Mary can only work Thursday and Friday.

Model this scheduling problem as a CSP. Indicate how the indicated constraints impact the domains for all variables.

Exercise 9

Consider the following constraint graph.

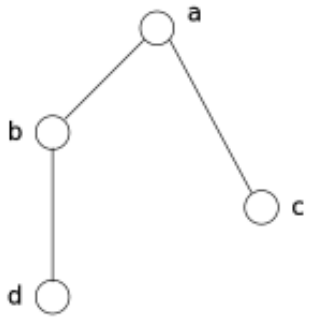


a) What is the width of the nodes given the order (A, B, C, D, E, F, G)? What is the width of the graph for this ordering?

b) What is the width of the nodes given the order (G, F, E, D, C, B, A)? What is the width of the graph for this ordering?

Exercise 10

Consider the following constraint graph.

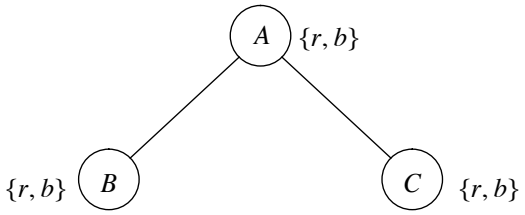


a) What is the (induced) width of the nodes given the order (D, C, B, A)? What is the (induced) width of the graph for this ordering?

b) What is the (induced) width of the nodes given the order (D, B, C, A)? What is the (induced) width of the graph for this ordering?

Exercise 11

Consider the following constraint graph concerning a graph coloring problem.



a) What is the width of the nodes given the order (B, C, A)? What is the width of the graph for this ordering?

b) What is the width of the nodes given the order (A, B, C)? What is the width of the graph for this ordering?

c) What is the width of the nodes given the order (B, A, C)? What is the width of the graph for this ordering?

d) Apply directional arc-consistency to the different orderings.

e) What can you conclude about search using the three different orderings? Show the resulting search tree for each ordering.

f) What can you say about the induced width of the orderings (b) and (c)?

g) Apply path-consistency to ordering (a). What can you conclude about search in the resulting graph?

Exercise 12

Consider the following map colouring problem with five regions A, B, C, D and E, having the following pairs of regions adjacent to each other (A,B), (B,C), (B,D), (C,D), (D,E). The colours that can be used for each region are as follows D_A = {blue, green}, D_B = {red, blue, green}, D_C = {blue}, D_D = {red, blue, green} and D_E = {red}. Assume that the ordering on the graph is A, B, C, D, E. Draw the constraint graph before answering the following questions. Draw the matching diagram as well, with abbreviations r, b, g for red, blue and green, respectively.

a) What are the colours in the domains of the five regions / variables after directional arc consistency (DAC) has been achieved on the corresponding CSP?

b) Which additional edges are present in the constraint graph after executing the Directional Path Consistency (DPC) algorithm?

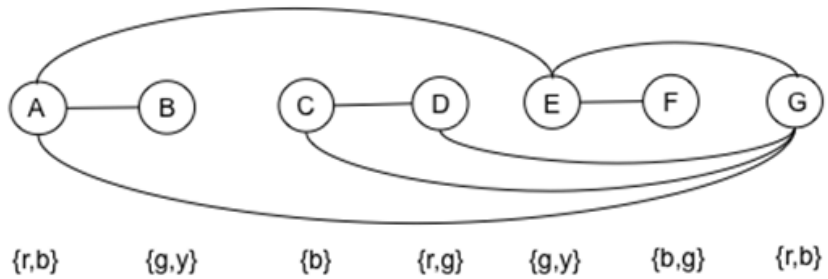
c) Which edges are present in the matching diagram for each relation after executing the DPC algorithm?

d) What are the colours in the domains of the five regions / variables after directional path consistency (DPC) has been achieved on the corresponding CSP?

e) What can you conclude after executing the DPC algorithm?

Exercise 13

The graph bellow represents a graph colouring problem. The regions or nodes are {A,B,C,-D,E,F,G} have ordering (A,B,C,D,E,F,G) which is the lexical ordering. The domains are D_A = {red,blue}, D_B = {green,yellow}, D_C = {blue}, D_D = {red,green}, D_E = {green,yellow}, D_F = {blue,green}, D_G = {red,blue}. The search algorithms try the colours in the order given in the domain. The regions which are adjacent are (A,B), (A,E), (A,G), (C,D), (C,G), (D,G), (E,F), (E,G). Each pair of regions that is adjacent must have different colours.



Hand simulate the algorithms backtracking, Gashnig's backjumping, graph-based backjumping, and conflict-directed backjumping, till a solution is found, and answer the following questions.

a) The first dead-end encountered by algorithm backtracking is...

b) The next node the algorithm backtracking tries to assign a value is...

c) After attempting the above node, the next node that backtracking tries to assign a value is...

d) Backtracking is able to assign a value to the node in the preceding question. (T/F)

e) The first dead-end encountered by Gashnig's backjumping is...

f) The next node the algorithm Gashnig's backjumping tries to assign a value is...

g) After attempting the above node, the next node that Gashnig's backjumping tries to assign a value is...

h) Gashning's backjumping is able to assign a value to the node in the preceding question. (T/F)

i) The first dead-end encountered by graph-based backjumping is...

j) The next node the algorithm graph-based backjumping tries to assign a value is...

k) After attempting the above node, the next node that graph-based backjumping tries to assign a value is...

l) Graph-based backjumping is able to assign a value to the node in the preceding question. (T/F)

m) After attempting the above node, the next node that graph-based backjumping tries to assign a value is...

n) Graph-based backjumping is able to assign a value to the node in the preceding question. (T/F)

o) After attempting the above node, the next node that graph-based backjumping tries to assign a value is...

p) Graph-based backjumping is able to assign a value to the node in the preceding question. (T/F)

q) After attempting the above node, the next node that graph-based backjumping tries to assign a value is...

r) Graph-based backjumping is able to assign a value to the node in the preceding question. (T/F)

s) The first dead-end encountered by conflict-directed backjumping is...

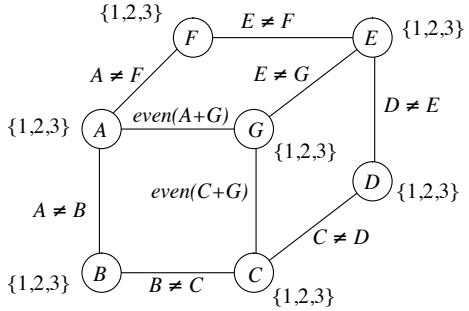
t) The next node the algorithm conflict-directed backjumping tries to assign a value is...

u) After attempting the above node, the next node that conflict-directed backjumping tries to assign a value is...

v) Conflict-directed backjumping is able to assign a value to the node in the preceding question. (T/F)

Exercise 14

Consider the following constraint graph, with variables A, B, C, D, E, F, G and domain 1,2,3 for all the variables. Constraints are shown on the edges.



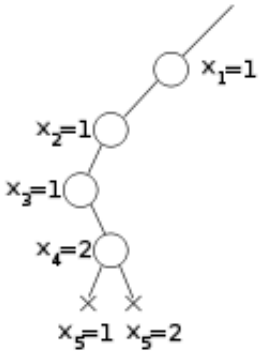
a) Draw the constraint graph assuming that the ordering on the graph is alphabetic.

b) Suppose that a value has been assigned to each of the variables A to F, and G is currently looked at. Suppose further that no label which is compatible with the labels committed to so far can be found for G. Identify the culprit variable in the context of graph-based backjumping.

c) Continuing with the above example, assume that the assignment committed to before G is labelled is: (<A,1><B,3><C,2><D,1><E,2><F,1>). Describe the behavior of a graph-based algorithm after backjumping to the culprit variable.

Exercise 15

Consider the following snapshot of a search tree that has reached a dead end.

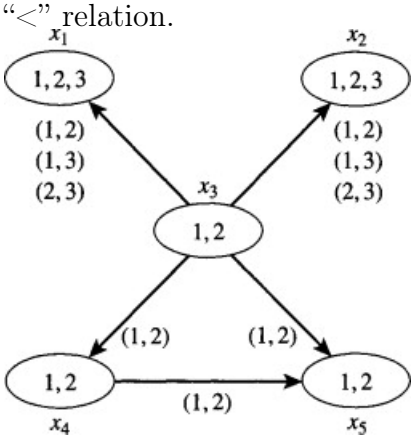


a) Assume that inconsistency is caused by the values of x_1 and x_4 only. How would you store this fact in a new constraint?

b) What happens if the algorithm reaches the same values of x_1 and x_4 again?

Exercise 16

Consider the following CSP. The constraints are x3 < x1, x3 < x2, x3 < x5, x3 < x4, x4 < x5. The allowed pairs are shown on each arc. The arrows only denote the direction of "<" relation.



a) Show the trace and no-goods recorded by graph-based learning.

Exercise 17

Consider a three-variable network z,x,y with $D_z = D_x = \{2, 5\}$ and $D_y = \{2, 4\}$. There are two constraints: R_{zx} specifying that z evenly divides x, and R_{zy} specifying that z evenly divides y. Apply AC-3 to the network.

Exercise 18

Consider the graph-coloring problem on a four-variable network where the domains contain only two possible colors $D_1 = D_2 = D_3 = D_4 = \{red, blue\}$ and the constraints are $x_1 \neq x_2$, $x_2 \neq x_3$, $x_3 \neq x_4$ and $x_4 \neq x_1$. Explain why the network is currently arc-consistent but not path-consistent.

Exercise 19

Consider that the domains and the constraints of the ordered constraint graph given below are as follows:

$D_1 = \{\text{red, white, black}\}$
$D_2 = \{\text{green, white, black}\}$
$D_3 = \{\text{red, white, blue}\}$
$D_4 = \{\text{white, blue, black}\}$
$R_{12} : x_1 = x_2$
$R_{13} : x_1 = x_3$
$R_{34} : x_3 = x_4$

Apply directional arc-consistency (DAC) using the ordering $\{x_1, x_2, x_3, x_4\}$. Is the resulting network full arc consistent? Is the resulting network backtrack-free? Justify your answers.

Exercise 20

Consider the GSAT procedure (greedy moves only) for finding satisfying assignments for Boolean formulas in clausal form, and the following formula F consisting of the clauses:
$(x_1 \vee \neg x_2), (x_1 \vee \neg x_3), (x_1 \vee \neg x_4), \ldots, (x_1 \vee \neg x_1 00)$

a) What is the probability that a truth assignment selected uniformly at random from all possible truth assignments will satisfy F?

b) Assuming that the initial random truth assignment select by GSAT does not satisfy F, how many flips will GSAT take to reach a satisfying assignment?

c) Give a pair of binary clauses, C1 and C2, such that when C1 and C2 are added to F, GSAT will almost certainly not be able to find a satisfying assignment for the resulting set of clauses, when starting from a random initial assignment. Briefly explain why GSAT would have trouble finding a satisfying assignment of F with the two added clauses.

Exercise 21

Consider the WALKSAT procedure with p=0.5. Consider an arbitrary satisfiable formula as input.

a) Starting at a random assignment, will the procedure eventually reach a satisfying assignment? Explain your answer. (Assume that the procedure does not restart. That is, the procedure either continues flipping variables indefinitely or stops at a satisfying assignment.)

b) In the walk step, assume that the procedure simultaneously flips all variables in the randomly selected unsatisfiable clause. Will the procedure eventually reach a satisfying assignment (again, no restarts)? Explain your answer.

c) In the walk step, assume that the procedure simultaneously flips one variable in each unsatisfied clause. Will the procedure eventually reach a satisfying assignment (no restarts)? Explain your answer.

Chapter 2

Automated Planning

2.1 Planning Domain Definition Language

PDDL Refresher

PDDL stands for Planning Domain Definition Language and is a programming language used to describe the formal representation of a planning problem and the instances of said problem we want to solve.

A traditional PDDL problem is composed by one domain file and one or multiple instance files. The domain file describes the problem domain, the domain variables, domain actions and the effects of said actions, as seen in Figure 2.1a. An instance file describes one specific instance of the problem to be solved, as seen in Figure 2.1b.

```
(define (domain switch)
    (:requirements :strips)

    (:predicates (switch_is_on)
                 (switch_is_off))

    (:action switch_on
       :precondition (switch_is_off)
       :effect (and (switch_is_on)
                    (not (switch_is_off)))
    )

    (:action switch_off
       :precondition (switch_is_on)
       :effect (and (switch_is_off)
                    (not (switch_is_on)))
    )
)
```

(a) Domain description file for the problem of switching on and off lights.

```
(define (problem turn_it_off)

    (:domain switch)

    (:init
       (switch_is_on)
    )

    (:goal (switch_is_off)))
```

(b) Example of an instance file for the problem of switching on and off lights.

Figure 2.1: Domain and instance example files for the problem of switching a light on and off.

As can be seen in the domain file, Figure 2.1a, the domain file starts with a definition of the domain name using the keyword **domain**. Each domain is then define by a set of requirements, using the keyword **:requirements**, that define a set of PDDL features that the domain uses, example of requirements are:

- **:strips** – allows the usage of add and delete effects in actions;
- **:typing** – allows the usage of object typing, creating new types that can be used in the domain;
- **:disjunctive-preconditions** – allows the usage of the keyword **or** in goals or preconditions to create disjunctions
  $(or\,(< condition_1 >)\,...\,(< condition_n >))$;
- **:equality** – allows the usage o the = sign to compare objects;
- **:existential-preconditions** – allows the usage of the keyword **exists** in goals and preconditions to verify if there is an object of the given type that satisfies a given condition
  $(exists\,(< var > - < type >)\,(< condition >))$;
- **:universal-preconditions** – allows the usage of the keyword **forall** in action preconditions and effects to apply the same affect to all the objects of the given type
  $(forall\,(< var > - < type >)\,(< condition >))$;
- **:conditional-effects** – allows the usage of the keyword **when** in action effects to apply a given effect when the condition holds true
  $(when\,(< antecedent\text{-}condition >)\,(< effect\text{-}condition >))$;
- **:negative-preconditions** – allows the usage of the keyword **not** in action preconditions to test if a condition does not hold;
- **:action-costs** – allows the usage of the **:functions** section that allows the definition of functions that return a numerical value, as well as the usage of the keywords **increase**, **decrease**, **assign**, among other numeric effects keywords;
- **:adl** – combines the requirements :strips, :typing, :disjunctive-preconditions, :equality, :existential-preconditions, :universal-preconditions, :conditional-effects.

The keyword **:predicates** defines the set of state variables, which are statements that are either true or false and describe the state of the world. These predicates are affected by the effects of actions and can only be changed with action effects. In the instance file we define which predicates of the domain are true at the beginning of the problem and predicates that are not present in the instance file are considered false when a solver starts solving an instance. Predicates can either have or not have parameters, thus defining which objects of the world the predicates affect.

The keyword **:action** defines an action of the domain. An action is responsible for changing the predicates of an instance, thus changing the state of the world. An action is composed by three sections:

1. **:parameters** - this section defines the objects the action affects, thus defining which predicates can be affected by the action, this section can be omitted when the action is applicable to all the objects of the domain;

```
(define (domain example)
    (:requirements :adl :action-costs)

    (:types truck location)

    (:predicates (at ?t - truck ?l - location)
                 (moved-through ?t - truck ?l - location))

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

    (:action move_truck
      :parameters (?t - truck ?start ?finish - location)
      :precondition (and (at ?t ?start)
                         (not (at ?t ?finish))
      :effect (and (at ?t ?finish)
                   (not (at ?t ?start))
                   (moved-through ?t ?finish)
                   (increase (total-cost) (move-cost ?start ?finish)))
    )
)
```

Figure 2.2: Example of the usage of functions in a PDDL domain file.

2. **:precondition** - this section defines a conjunction or disjunction of predicates that must be satisfied so that the action can be applied;

3. **:effect** - this section defines what are the consequences for the domain, taking the form of either a conjunction or disjunctions of predicates that are set to either true or false values, thus changing the state of the world.

With the use of the **:action-costs** requirement a new keyword can be used that defines a new section of the domain, **:functions**. This section defines a set of statements, that like with predicates can have parameters or not, and return a value associated with the instantiation of that function. The value associated with a given function can be altered with effects such as **increase**, **decrease**, **assign** and all functions available in the domain must be initially instantiated in the instance file, describing the initial value of each function and which functions are available in a specific instance. Figure 2.2 shows an example of the declaration of a function and its usage in the effect of an action and Figure 2.3 shows an example of the usage of functions in an instance file.

As can be seen in the instance file in Figure 2.1b, the instance is described by a problem name given by the keyword **problem**, a domain section given by the keyword **:domain** that defines what is the domain the instance belongs to, the initial state of the world in the section declared by the keyword **:init** and the goal to achieve in this instance given by the keyword **:goal**.

The **:init** section is a sequence with all the predicates that are true in the initial state of the world and, if functions are used in the domain, the values of the functions of the domain. To set the value of the functions we use the expression:
$(= (< function\text{-}name >) < value >)$

```
(define (problem move-truck-1)
    (:domain move-truck)

    (:objects
       t1 - truck
       l0 l1 l2 l3 - location
    )

    (:init
       (at t1 l0)
       (moved-through t1 l0)
       (= (total-cost) 0)
       (= (move-cost l0 l1) 3)
       (= (move-cost l0 l2) 2)
       (= (move-cost l0 l3) 5)
       (= (move-cost l1 l0) 3)
       (= (move-cost l1 l2) 1)
       (= (move-cost l1 l3) 2)
       (= (move-cost l2 l0) 2)
       (= (move-cost l2 l1) 1)
       (= (move-cost l2 l3) 4)
       (= (move-cost l3 l0) 5)
       (= (move-cost l3 l1) 2)
       (= (move-cost l3 l2) 4)
    )

    (:goal
       (and
            (moved-through t1 l0)
            (moved-through t1 l1)
            (moved-through t1 l2)
            (moved-through t1 l3))
    )

    (:metric minimize (total-cost))
)
```

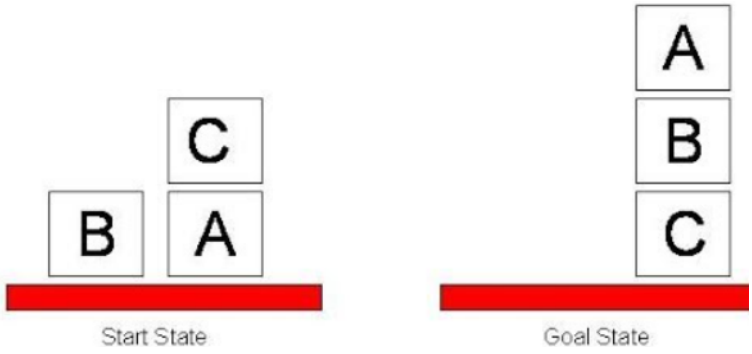
Figure 2.3: Example of the usage of functions and metric in a PDDL instance file.

The **:goal** section can be just a predicate as in Figure2.1b or can be a conjunction or disjunction of predicates.

Finally, if the domain makes use of action costs, there is one last section that can be add through the keyword **:metric** that defines a metric to be optimized while solving the problem. This section is defined as
(**metric** <direction> <expression>),
where *<direction>* is either **minimize** or **maximize** and defines if we are maximizing or minimizing the following expression and *<expression>* is any numeric expression, such as the return of a domain function or a more complex algebraic expression.

Exercise 1 - Blocks World

Consider the Blocks World. The goal is to build one or more vertical stacks of blocks. Only one block may be moved at a time: it may either be placed on the table or placed atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved. Using PDDL, formalize this domain, and create an instance for the following start and goal states:



Exercise 2 - Briefcase

Consider the Briefcase World. The objective is to transport a number of objects from their start to their goal locations, using a briefcase. Each location is accessible from each other location, objects can be put into the briefcase or taken out of the briefcase. When a move is made between locations, then all objects inside the briefcase are also moved. Model this domain using PDDL, and create an instance for 5 objects $(o_0, ..., o_4)$, and 6 locations $(l_0, ..., l5)$. The briefcase starts at l_4. The starting locations of the objects are as follows: (o_0, l_1), (o_1, l_2), (o_2, l_3), (o_3, l_0), (o_4, l_4). The goal locations for the objects are: (o_0, l_0), (o_1, l_1), (o_2, l_4), (o_3, l_3), (o_4, l_5). The briefcase should end at l_0.

Exercise 3 - Elevator

Consider a building with multiple elevators and you are tasked with developing a model to control the elevators as they take passengers from floor to floor. Each elevator can move up and down between the current floor and the next floor and takes only one passenger at a time. Consider also that before moving to a new floor, the elevator needs to wait while the passenger either enters the elevator or leaves the elevator.

Model this problem using PDDL and create an instance with 5 floors $(n_1, \ldots, n_5)$, 3 passengers $(p_1, \ldots, p_3)$, and 2 elevators (e_1, e_2). In this instance the elevators can move between all the floors and elevator 1 starts in floor 1, while elevator 2 starts in floor 5.

Passenger 1 starts in floor 2 with passenger 2 and passenger 3 starts in floor 4. The of this instance is to bring all the passengers to floor 1.

Exercise 4 - TSP

Consider the travelling salesman problem (TSP), a problem where you must define the route a salesman must take that allows him to pass only once by all the cities, taking the shortest route possible, and ends in the same city that he started.

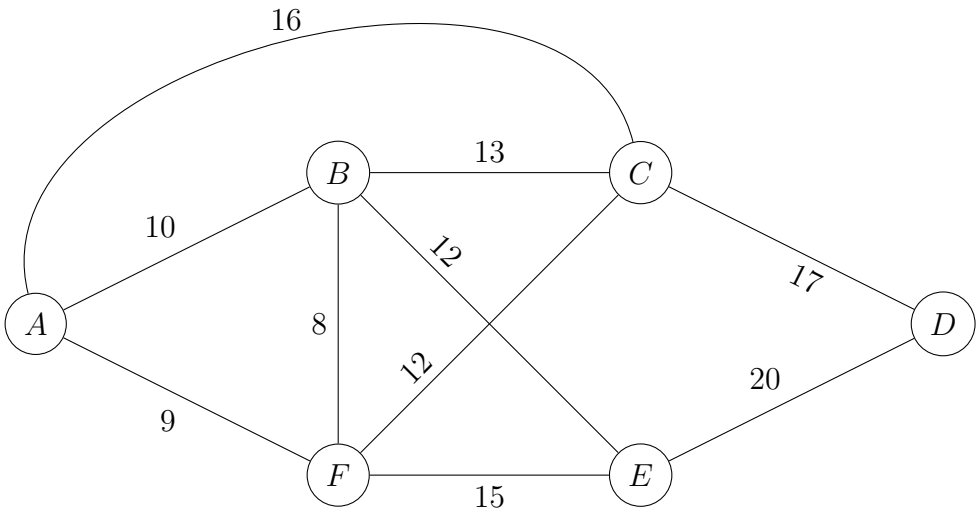


Figure 2.4: Movement cost and connection between cities in the travelling salesman problem.

Figure 2.4 shows how each city connects with the others and the cost of moving from one city to a connect city.

Model the travelling salesman problem using PDDL, for the city network in Figure 2.4, knowing that the salesman starts in city A.

2.2 Planning Algorithms

Exercise 1

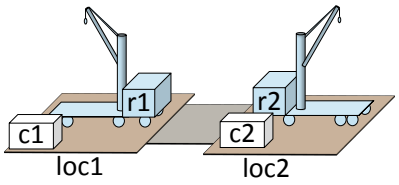
Consider a planning problem involving two robots whose actions are controlled by a single actor. In the action templates, r is a robot, l and m are locations, and c is a container. Both robots may have the same location.

take(r, l, c)
  pre: loc$(r) = l$, pos$(c) = l$,
    cargo$(r) =$ nil
  eff: cargo$(r) = c$, pos$(c) \leftarrow r$

put(r, l, c)
  pre: loc$(r) = l$, pos$(c) = r$
  eff: cargo$(r) \leftarrow$ nil, pos$(c) \leftarrow l$

move(r, l, m)
  pre: loc$(r) = l$
  eff: loc$(r) \leftarrow m$

(a) action templates



$s_0 = \{$loc(r1) = loc1, loc(r2) = loc2,
cargo(r1) = nil, cargo(r2) = nil,
pos(c1) = loc1, pos(c2) = loc2$\}$

$g = \{$pos(c1) = loc2, pos(c2) = loc2$\}$

(b) initial state and goal

a) If we run Forward-search on this problem, how many iterations will the shortest execution traces have, and what plans will they return? For one of them, give the sequence of states and actions chosen in the execution trace.

b) If we run Backward-search on this problem, how many iterations will the shortest execution traces have, and what plans will they return? For one of them, give the sequence of goals and actions chosen in the execution trace.

c) Compute the value of $h^{FF}(s_0)$.

d) Compute the value of $h^{sl}(s_0)$.

Exercise 2

Consider a state-variable version of the problem of swapping the values of two variables. The set of objects is B = Variables ∪ Numbers, where Variables = {foo, bar, baz}, and Numbers = {0, 1, 2, 3, 4, 5}. There is one action template:

assign(x_1, x_2, n)
pre: value(x_2) = n
eff: value(x_1) ← n

where Range(x_1) = Range(x_2) = Variables, and Range(n) = Numbers.

The initial state and goal are

s_0 ={value(foo)=1,value(bar)=5,value(baz)=0};

g = {value(foo) = 5, value(bar) = 1}.

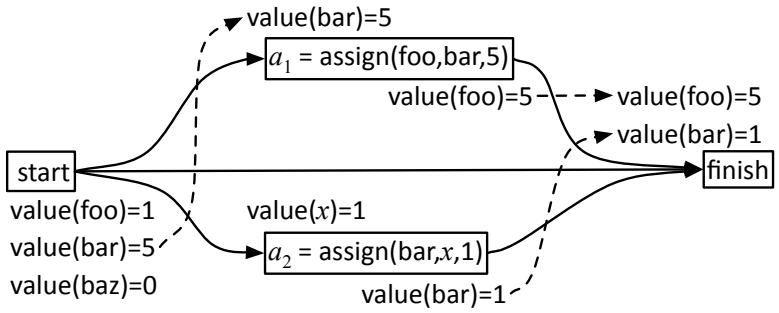
At s_0, suppose GBFS is trying to choose between the actions a_1 = assign(baz,foo,1) and a_2 = assign(foo,bar,5). Let $s_1 = \gamma(s_0, a_1)$ and $s_2 = \gamma(s_0, a_2)$. Compute each pair of heuristic values below, and state whether or not they will produce the best choice.

a) $h^{FF}(s_1)$ and $h^{FF}(s_2)$.

b) $h^{sl}(s_1)$ and $h^{sl}(s_2)$.

Exercise 3

This figure shows a partial plan for the variable-swapping problem in the previous exercise.



a) How many threats are there? What are they? What are their resolvers?

b) Can PSP generate this plan? If so, describe an execution trace that will produce it. If no, explain why not.

c) In PSP's search space, how many immediate successors does this partial plan have?

d) How many solution plans can PSP produce from this partial plan?

e) How many of the preceding solution plans are minimal?

f) Trace the operation of PSP if we start it with the plan in the figure. Follow whichever of PSP's execution traces finds the shortest plan.

Exercise 4

Blocks world is a well-known classical planning domain in which some children's blocks, Blocks = {a,b,c,...}, are arranged in stacks of varying size on an infinitely large table, table. To move the blocks, there is a robot hand, hand, that can hold at most one block at a time.

pickup(x)
  pre: loc(x) = table, top(x) = nil, holding = nil
  eff: loc(x) ← hand, holding ← x

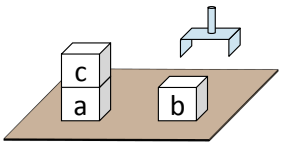
putdown(x)
  pre: holding = x
  eff: loc(x) ← table, holding ← nil

unstack(x, y)
  pre: loc(x) = y, top(x) = nil, holding = nil
  eff: loc(x) ← hand, top(y) ← nil, holding ← x

stack(x, y)
  pre: holding = x, top(y) = nil
  eff: loc(x) ← y, top(y) ← x, holding ← nil

Range(x) = Range(y) = *Blocks*

(a) action templates



$Objects = Blocks \cup \{\text{hand}, \text{table}, \text{nil}\}$
$Blocks = \{\text{a}, \text{b}, \text{c}\}$

$s_0 = \{\text{top(a)} = \text{c}, \text{top(b)} = \text{nil}, \text{top(c)} = \text{nil}, \text{holding} = \text{nil}, \text{loc(a)} = \text{table}, \text{loc(b)} = \text{table}, \text{loc(c)} = \text{a}\}$

$g = \{\text{loc(a)} = \text{b}, \text{loc(b)} = \text{c}\}$

(b) objects, initial state, and goal

The figure (a) gives the action templates. For each block x, loc(x) is x's location, which may be table, hand, or another block; and top(x) is the block (if any) that is on x, with top(x) = nil if nothing is on x. Finally, holding tells what block the robot hand is holding, with holding = nil if the hand is empty.

a) Why are there four action templates rather than just two?

b) Is the holding state variable really needed? Why or why not?

c) In the planning problem in the figure (b), how many states satisfy g?

d) Give necessary and sufficient conditions for a set of atoms to be a state.

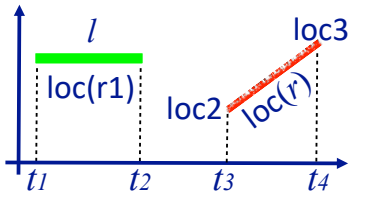
e) Is every blocks world planning problem solvable? Why or why not?

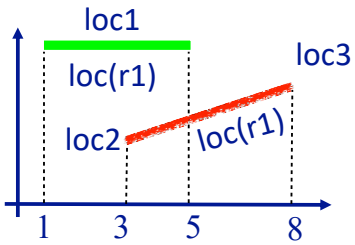
Exercise 5

Consider the following timeline $(\mathcal{T}, \mathcal{C})$ = ({[t_1,t_2] loc(r1) = loc1, [t_3,t_4] loc(r) : (l, l')}, $\{t_1 < t_2, t_3 < t_4\}$)

a) Note that the temporal assertion [t_3,t_4] loc(r) : (l, l') has no causual support, i.e. what causes r to be at l at time t_3? Provide three different ways of resolving this flaw.

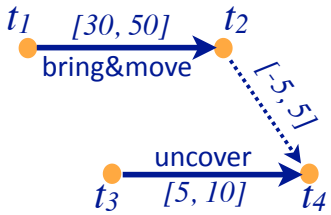
b) Note that $\mathcal{T}$ contains a pair of possibly-conflicting temporal assertions, as illustrated by the figure. Provide five different sets of separation constraints to resolve this flaw.





Exercise 6

Consider the following Simple Temporal Network (STN).



a) Apply the path-consistency algorithm to the STN to obtain the minimal network.

b) Classify the variables t_i as being controllable or contingent.

Exercise 7

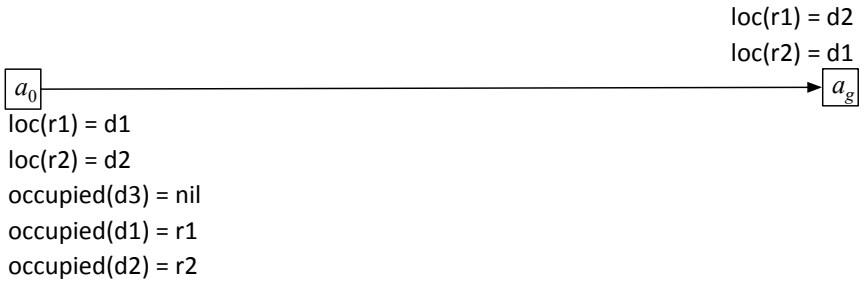
Consider a simple planning problem in which there are two robots and three loading docks, that is, B = Robots ∪ Docks, where Robots = {r1, r2} and Docks = {d1, d2, d3}. There are no rigid relations. There is one action template,

move(r, d, d')
pre: loc(r) = d, occupied(d') = nil
eff: loc(r) ← d', occupied(d) = nil, occupied(d') = r

where r ∈ Robots and d,d' ∈ Docks. The initial state and the goal are:

s_0 ={loc(r1)=d1, loc(r2)=d2, occupied(d1) = r1, occupied(d2) = r2, occupied(d3) = nil};
g = {loc(r1) = d2, loc(r2) = d1}.

In the context of Plan-Space Search (PSP), the figure below is the initial partial plan containing dummy actions a_0 and a_g that represent s_0 and g.



a) Identify the two flaws in the initial plan.

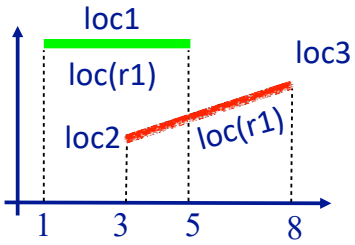
b) Resolve the two flaws adding to the figure the appropriate actions and causal links.

Exercise 8

Consider the following timeline
$(\mathcal{T}, \mathcal{C})$ = ({[t_1,t_2] loc(r1) = loc1, [t_3,t_4] loc(r) : (l, l')}, {$t_1 < t_2$, $t_3 < t_4$})

a) Note that the temporal assertion [t_3,t_4] loc(r) : (l, l') has no causal support, i.e. what causes r to be at l at time t_3? Provide **three** different ways of resolving this flaw.

b) Note that $\mathcal{T}$ contains a pair of possibly-conflicting temporal assertions, as illustrated by the figure. Provide **five** different sets of separation constraints to resolve this flaw.



Exercise 9

Run algorithm Path-Consistency (PC) on the following temporal network. Show that it adds the constraints $r_{13} = [2,3]$, $r_{24} = [3,4]$ and $r_{45} = [2, 3]$.