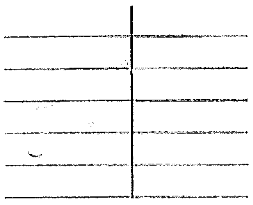
Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2018-06-02
Sal (1)	U10(20)
Tid	14-18
Kurskod	TDDD48
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Automatisk planering Skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	6
Jour/Kursansvarig Ange vem som besöker salen	Mikael Nilsson besöker salen (Jonas Kvarnström - examinator)
Telefon under skrivtiden	0708-593291(Mikael), 0704-737579 (Jonas)
Besöker salen ca klockan	ca 15-15:30
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDD48 Automated Planning 2018-06-02

Important Instructions: Read before you begin!

- Though the questions are in English, feel free to answer in Swedish if you prefer!
  Det går bra att skriva på svenska!
- Please, write clearly (block letters if necessary), and use the *lined* side of the paper except for diagrams! Checked paper works well for math but tends to make text difficult to read...



- *Clear and comprehensible* explanations and motivations are always required. This does not necessarily mean that each answer should be a long essay. What is important is that all the relevant facts are present and clearly explained.
- To see if you have succeeded, turn the tables and **try to misunderstand.** Specifically, after you write an answer, go back and see if you can creatively misinterpret the answer. If so, *clarify, clarify, clarify*!
- A good way to show your knowledge is to write explanations that can be understood by someone *else* who *does not already know* the correct answer.

  If your answer explains a topic well enough that a fellow student could *learn* something new from the answer and/or apply it in practice, you have probably succeeded.

  Conversely, if we ask how HTNs work, saying that "there are methods and operators and you construct a tree using patterns and there can be constraints too, and there's no goal" does not provide much in terms of *useful* information, and certainly will not give you a full score.
- Concrete examples or counterexamples may be useful as part of a motivation. If so, please make sure you include all relevant information about the example you have chosen to use. What is relevant naturally depends on how you use the example.
- When asked to provide examples or illustrate something through a planning problem instance, save time by *keeping the example as small as possible.*

Notation and Terminology

To avoid unnecessary confusion, we will hint at some of the terminology that you have hopefully learned during the course. **Please be careful to use the correct words!**

- **Predicate** (name) – at, on, raining
- **Object** (name) – A, B, red, truck1, helicopter8, location4
- **Variable** – *truck*, *location*
- **Atom** – at(truck1, location4), at(*truck*, location4), raining
- **Ground atom** – at(truck1, location4), raining
- **Fact** – typically a ground atom
- **Literal** – atom or negated atom
- **Ground literal**
- **Formula** – combination of atoms using connectives
- **State** – a full specification of the "configuration" of the world (to the extent it is modeled in the problem specification)
- **Goal formula, goal state, set of goal states, goal fact, ...**
- $h^*(s)$ – the cost of an optimal plan reaching a goal state from state s

1 Delete Relaxation

The first question concerns **delete relaxation** in classical planning. We use the standard *logistics domain* as defined in Appendix A; please familiarize yourself with it before continuing.

Let $P = \langle \Sigma, s_0, g \rangle$ be a classical planning problem in the logistics domain, where:

- The objects are the two packages $\{\mathsf{pkg}_1, \mathsf{pkg}_2\}$, the trucks $\{\mathsf{trk}_1, \mathsf{trk}_2\}$ and the three locations $\{\mathsf{loc}_1, \mathsf{loc}_2, \mathsf{loc}_3\}$, none of which are airports.
- The initial state s_0 specifies that $\mathsf{at}(\mathsf{pkg}_1, \mathsf{loc}_1)$, $\mathsf{at}(\mathsf{pkg}_2, \mathsf{loc}_2)$, $\mathsf{at}(\mathsf{trk}_1, \mathsf{loc}_2)$, and $\mathsf{at}(\mathsf{trk}_2, \mathsf{loc}_2)$. Also, all locations are in the same city: $\{\mathsf{same\text{-}city}(l, l') \mid l, l' \in \{\mathsf{loc}_1, \mathsf{loc}_2, \mathsf{loc}_3\}\}$. All other facts are initially false. For example, no packages are loaded into vehicles.
- The goal is $g = \{\mathsf{at}(\mathsf{pkg}_1, \mathsf{loc}_3), \mathsf{at}(\mathsf{pkg}_2, \mathsf{loc}_3), \mathsf{at}(\mathsf{trk}_1, \mathsf{loc}_2), \mathsf{at}(\mathsf{trk}_2, \mathsf{loc}_2)\}$.

Let $P' = \langle \Sigma', s_0', g' \rangle$ be the delete-relaxed version of P. **All questions below should be answered in the context of the delete-relaxed problem!**

a) Is the state $\{\mathsf{at}(\mathsf{pkg}_1, \mathsf{loc}_2), \mathsf{at}(\mathsf{pkg}_2, \mathsf{loc}_2), \mathsf{at}(\mathsf{trk}_1, \mathsf{loc}_2), \mathsf{at}(\mathsf{trk}_2, \mathsf{loc}_2)\}$ reachable from s_0 in P'? If so, show an action sequence that reaches this state. If not, provide a clear explanation and motivation. **(1 point)**

b) Suppose that in P', starting in state s_0, we execute the action sequence
$\langle \mathsf{drive\text{-}truck}(\mathsf{trk}_1, \mathsf{loc}_2, \mathsf{loc}_1), \mathsf{load\text{-}package}(\mathsf{pkg}_1, \mathsf{trk}_1, \mathsf{loc}_1),$
$\mathsf{drive\text{-}truck}(\mathsf{trk}_1, \mathsf{loc}_1, \mathsf{loc}_3), \mathsf{unload\text{-}package}(\mathsf{pkg}_1, \mathsf{trk}_1, \mathsf{loc}_3)\rangle$.
This results in a state sequence $\langle s_0, s_1, s_2, s_3, s_4 \rangle$ where s_0 is the initial state and the other four states are created by applying the actions shown above. Specify these four states. You may omit all same-city atoms: While they are important parts of each state, they are never modified. **(2 points)**

2 Landmarks

Consider again the original planning problem P from Question 1, without delete relaxation.

a) Provide two distinct fact landmarks that need to be achieved in this planning problem, and motivate why they are landmarks. **(1 point)**

b) Can you provide another two fact landmarks that need to be achieved in this planning problem? If so, motivate why they are landmarks. If not, motivate why this is impossible. **(2 points)**

3 Choosing Heuristics

Suppose you are given a classical planning problem $P = \langle \Sigma, s_0, g \rangle$, where $\Sigma = \langle S, A, \gamma \rangle$.

Suppose you are also given three distinct heuristic functions h_A, h_B and h_C, and that these result in the following heuristic values for s_0 and its four successors $s_1, \ldots, s_4$ in Σ. While the full search space contains many additional states that can be reached from $s_1, \ldots, s_4$, we will only consider this small part.

State s_i	$h^*(s_i)$	$h_A(s_i)$	$h_B(s_i)$	$h_C(s_i)$
s_0	**10**	8	9	9
s_1	**9**	6	5	7
s_2	**8**	3	9	7
s_3	**3**	2	1	1
s_4	**2**	1	2	2

Each heuristic function has various properties that makes it more or less suitable for use with different search strategies. Determining for certain which one is the best for a particular strategy would require more information about the entire search space, but the table above still shows certain trends from which we can generalize.

a) Which of the three heuristic functions would appear to be the most suitable for use with *satisficing hill climbing search* (not the "enforced" variation)? We need a clear motivation for why this is the case. **(2 points)**

b) Which of the three heuristic functions would appear to be the most suitable for use with *optimal A* search*? We need a clear motivation for why this is the case. **(2 points)**

4 Search Techniques

Recall that the FF (FastForward) planner pioneered the use of *helpful actions*, a specially designated subset of the applicable actions in any particular search state.

FF used these actions in a particular way together with enforced hill climbing. However, other planners have used them differently, through another treatment of the *search (priority) queue*.

a) How are *dual queues* used to achieve a balanced use of helpful actions? For full points, you need to provide a clear description of the actual use of dual queues and how they are integrated in the search process, at a level close to pseudo-code. **(2 points)**

b) What is the difference between dual queues and *boosted queues*? **(1 point)**

5 Motion Planning

Many forms of motion planning build on the use of a *local planner* that knows how to generate *local plans* for a particular type of vehicle. As the local planner is typically quite limited in its ability to connect arbitrary points, there must also be a *higher level planner* that can repeatedly call the local planner. For example, the higher level planner can be a *probabilistic roadmap* (PRM) planner.

- PRM planners begin with a pre-processing phase where the starting point and goal are not known. Explain what the PRM planner does during this phase, how it decides when to stop, and what the resulting output is.

  We will not require the description to be perfect in every detail, but it should be sufficiently detailed to demonstrate that you grasp the most important concepts, close to high-level pseudo-code for a full score.

- When the PRM planner does receive a starting point and goal, how does it create a motion plan? How is the result of the pre-processing phase used in this process?

(4 points)

6 Planning with Incomplete Information

In one of the later lectures, we discussed a number of different planning problems involving incomplete information about the world, including the Stochastic Shortest Path Problem (SSPP).

a) What *is* the Stochastic Shortest Path Problem?

   You might want to explain this by contrasting it to the classical planning problem and explaining the *differences and similarities* in the respective *state transition systems* of these two problems. Leading questions: What knowledge do we have before we start? What can we observe? What assumptions do we make about actions and their outcomes? What type of objective do we model in an SSPP problem instance? What kind of solution structure might we get?

   (2 points)

b) The Stochastic Shortest Path Problem is different from the general MDP (Markov Decision Process) problem. Nevertheless, an SSPP problem instance can be solved by a general MDP solver.

   How do we transform the SSPP problem before giving it to an MDP solver, so that the solver cannot generate a policy that requires executing an infinite number of useful actions? How does this transformation guarantee that any MDP solution corresponds to an SSPP solution that achieves the SSPP-specific objectives?

   (Part of this transformation may consist of turning costs into negative rewards, but this is not the part we are interested in here.)

   (2 points)

A The Logistics Domain

The standard **logistics domain** contains a set of *packages* that should be transported to their destinations. A package can be transported by *truck* between any two *locations* in the same city, and by *airplane* between special *airport* locations in different cities. Since trucks cannot deliver packages directly to other cities, and airplanes cannot visit arbitrary locations, delivering a package might require using a truck to move it to an airport, using an airplane to move it to another city and then once again using a truck to get the package to its final destination.

We assume the following types of objects:

- thing, with subtypes package and vehicle
- vehicle, with subtypes truck and airplane
- location, with subtype airport

A model of this domain may include the following **operators**. For Simple Task Network planning, these operators correspond directly to **primitive tasks**.

- load-truck(package, truck, location) loads a package into a truck, given that they are both at the same location. The truck can hold an arbitrary number of packages.
- unload-truck(package, truck, location) unloads a package from a truck in the current location.
- load-plane(package, airplane, location) loads a package into a plane, given that they are both at the same location. The plane can hold an arbitrary number of packages.
- unload-plane(package, airplane, location) unloads a package from an airplane in the current location.
- drive-truck(truck, location, location) drives a truck between locations in the same city.
- fly-plane(plane, airport, airport) flies a plane between two airports in different cities.

We assume the following **predicates** are available:

- at(thing,loc) – the package or vehicle thing is at the location loc. Note that a package "at" a certain location cannot be "in" a vehicle in the same state.
- in(x,vehicle) – the package x is in the given vehicle.
- same-city(loc1,loc2) – the given locations are in the same city. Note that every location must be in the same city as itself.

We assume a **typed** domain model where each operator parameter and predicate parameter is given a specific type and cannot take on values outside that type. For example, if t_1 is a truck, an expression such as at(t_1, t_1) is not merely false but incorrect. Nevertheless, we provide the following **type predicates** that may be useful in some situations: thing(x), package(x), vehicle(x), truck(x), airplane(x), location(x) and airport(x).

We can then define the operators more formally as follows:

- load-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: at(*pkg*, *loc*) ∧ at(*trk*, *loc*)
  Effects: ¬at(*pkg*, *loc*), in(*pkg*, *trk*)
- load-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: at(*pkg*, *loc*) ∧ at(*plane*, *loc*)
  Effects: ¬at(*pkg*, *loc*), in(*pkg*, *plane*)
- unload-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: in(*pkg*, *trk*) ∧ at(*trk*, *loc*)
  Effects: ¬in(*pkg*, *trk*), at(*pkg*, *loc*)
- unload-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: in(*pkg*, *plane*) ∧ at(*plane*, *loc*)
  Effects: ¬in(*pkg*, *plane*), at(*pkg*, *loc*)
- drive-truck(truck *trk*, location *from*, location *to*)
  Precondition: at(*trk*, *from*) ∧ same-city(*from*, *to*)
  Effects: ¬at(*trk*, *from*), at(*trk*, *to*)
- fly-airplane(airplane *plane*, airport *from*, airport *to*)
  Precondition: at(*plane*, *from*)
  Effects: ¬at(*plane*, *from*), at(*plane*, *to*)