5

Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-06-05
Sal (2)	TB TB1
Tid	14-18
Kurskod	TDDD48
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Automatisk planering Skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	4
Jour/Kursansvarig Ange vem som besöker salen	Jonas Kvarnström
Telefon under skrivtiden	0704-737579
Besöker salen ca klockan	ca kl. 15
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, anna.grabska.eklund@liu.se, ankn. 2362
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDD48 Automated Planning 2015-06-05

Important Instructions: Read before you begin!

- Though the questions are in English, feel free to **answer in Swedish** if you prefer!
- *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.
- Make an effort to write explanations that can be understood by someone 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, then 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.
- 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*!
- 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*.

1 Fundamental Concepts: Relaxation

a) **Relaxation.** Let $\mathscr{P} = \langle O, s_0, g \rangle$ be an arbitrary classical planning problem. When is $\mathscr{P}' = \langle O', s_0', g' \rangle$ a *relaxed* version of $\mathscr{P}$? That is, what criterion or criteria must be satisfied for $\mathscr{P}'$ to be a relaxation of $\mathscr{P}$?

Note that we are discussing relaxation in general, as opposed to some specific type of relaxation. Note also that you should describe the *exact* requirements for relaxation. That is, your definition should be *sufficient and necessary*, rather than covering a special case. **(2 points)**

b) **Delete relaxation.** We use the standard logistics domain as defined in Appendix A:

- The problem instance contains the packages $\{p_1, p_2\}$, the trucks $\{t_1, t_2\}$ and the three locations $\{l_1, l_2, l_3\}$, none of which are airports.
- Let O be the set of operators in this domain.
- The initial state is
  $s_0 = \{\mathsf{at}(p_1, l_1), \mathsf{at}(p_2, l_2), \mathsf{at}(t_1, l_2), \mathsf{at}(t_2, l_2)\} \cup \{\mathsf{same\text{-}city}(l, l') \mid l, l' \in \{l_1, l_2, l_3\}\}$,
  where the latter part indicates that all locations are in the same city. No packages are loaded into vehicles and we do not use type predicates.
- The goal (which will actually not be used in this question) is
  $g = \{\mathsf{at}(p_1, l_3), \mathsf{at}(p_2, l_3), \mathsf{at}(t_1, l_2), \mathsf{at}(t_2, l_2)\}$.

This yields a problem instance $\mathscr{P} = \langle O, s_0, g \rangle$. Let $\mathscr{P}' = \langle O', s_0', g' \rangle$ be the *delete-relaxed* version of $\mathscr{P}$. Then:

- Show the exact sequence of states s_0', s_1', s_2', s_3' generated by executing the action sequence $\langle \mathsf{load\text{-}truck}(p_2, t_2, l_2), \mathsf{drive\text{-}truck}(t_2, l_2, l_1), \mathsf{unload\text{-}truck}(p_2, t_2, l_1) \rangle$ in the delete-relaxed problem instance $\mathscr{P}'$. That is, show s_0' and then each of the three consecutive states resulting from each new action being executed.

  You should omit instances of same-city, since they are identical in all states.

  You may shorten the answer by representing non-initial states s_1', s_2', s_3' as *differences* compared to s_0'.
- Generate a delete-relaxed plan beginning in s_0' and reaching precisely the state $s = \{\mathsf{at}(t_1, l_2), \mathsf{at}(t_2, l_2), \mathsf{in}(p_1, t_1), \mathsf{in}(p_2, t_2)\} \cup \{\mathsf{same\text{-}city}(l, l') \mid l, l' \in \{l_1, l_2, l_3\}\}$, or prove that such a plan does not exist in the delete-relaxed problem instance.

(2 points)

2 Heuristics and Search Guidance

a) **Landmarks.** Explain: What is a *disjunctive action landmark*?

Then show three distinct (disjunctive or non-disjunctive) action landmarks for the logistics problem instance in question 1b, and explain *why* these are landmarks.

The explanation should not repeat the definition of a landmark. Instead it must show why the landmarks *satisfy* the definition. **(2 points)**

b) **Pattern databases.** Describe the general ideas behind standard pattern database (PDB) heuristics, assuming that a *single* pattern is used. Specifically: **(3 points)**

- PDB heuristics include a pre-processing phase that is specific to a problem instance but is performed before search begins. Assume as given a problem instance $\mathcal{P} = \langle O, s_0, g \rangle$ in the state variable representation, with no additional PDB-specific information provided. Describe what the pre-processing phase does with this problem instance, step by step. Also describe what the resulting output will consist of.

- A planner must be able to calculate a heuristic value $h(s)$ for every state s encountered during the search process. How is $h(s)$ calculated for the standard PDB heuristic (still using a single pattern)? How is the result of the pre-processing phase used during this calculation?

c) **Pattern databases.** Some variations of PDB heuristics remain admissible despite the fact that they combine *multiple* heuristic estimates based on *multiple* different patterns.

Describe a technique for achieving this that is applicable to *every* combination of admissible heuristic estimates, regardless of whether pattern databases are used or not.

Also, describe a technique for doing this that is specifically based on limiting which patterns can be selected and used in the combined heuristic function. What are the limitations / restrictions that are applied when patterns are selected, how are the heuristic values calculated and combined, and why is the resulting value admissible? **(2 points)**

3 Partial-Order Planning

We now consider partial-order planning. We again use the standard **logistics domain** as defined in Appendix A, with a problem instance containing the packages $\{p_1, p_2\}$, the trucks $\{t_1, t_2\}$ and the three locations $\{l_1, l_2, l_3\}$, none of which are airports.

In the initial state, we know that $\mathsf{at}(p_1, l_1)$, $\mathsf{at}(p_2, l_2)$, $\mathsf{at}(t_1, l_2)$, and $\mathsf{at}(t_2, l_2)$. All locations are in the same city: $\{\mathsf{same\text{-}city}(l, l') \mid l, l' \in \{l_1, l_2, l_3\}\}$. No packages are loaded into vehicles.

The goal is that $\mathsf{at}(p_1, l_3)$, $\mathsf{at}(p_2, l_3)$, $\mathsf{at}(t_1, l_2)$, and $\mathsf{at}(t_2, l_2)$.

You should do the following (see also notes below):

a) Show the *initial partial plan* π_0 generated for this problem instance by a typical partial-order planner, such as the PSP planner in the course book. This is the partial plan that corresponds to the first node ("root node") generated in the search space.

   Identify all *flaws* in this partial plan, and show *why* they are flaws (explain what a flaw is and show why the flaws you have identified satisfy the criteria). **(2 points)**

b) Show all the immediate successors of the initial partial plan. That is, demonstrate all the different ways in which a standard PSP-like partially ordered planner might extend π_0 in a single step.

   For this particular task you do not have to illustrate each successor plan graphically. Instead, you can explain clearly in writing how each successor would extend π_0. However, you still need to indicate *all* changes that would be made, including new constraints and relations! **(2 points)**

c) Show a partial plan for this problem in which one or more *threats* appear. Indicate all threats clearly. Explain in text *why* they are threats – which criteria have to be satisfied for the threat to exist. **(2 points)**

Note: For each action in a partial plan, you must clearly show each precondition *above* the action and each effect *below* the action (or to the left / to the right, if you choose this direction).

You must also indicate all other relevant structural features in the plan. Explain your notation and omit nothing without proper explanation. (For example, the lecture notes explicitly stated that certain kinds of arrows were omitted for clarity; you can also omit these *as long as you say they are omitted and explain how one can infer which arrows should actually be there*!)

Hint: You may benefit from quickly sketching plans on a separate paper first.

4 Planning with Incomplete Information

For this question, I would like to remind everyone that good answers should clearly and unambiguously explain concepts as if the reader had not already heard of them.

a) Explain the meaning of, and the differences between, three types of planning: *Fully observable*, *partially observable* and *non-observable*.

   Hint: What can or can't we observe? When would such observations be made? When could they be useful? **(2 points)**

b) Explain the meaning of, and the differences between, *non-deterministic planning* and *probabilistic planning*. **(2 points)**

c) Both classical planning and planning using Markov Decision Processes can be characterized in terms of state transition systems. What is the most significant difference between the state transition system (structure) used for classical planning and the one used for Markov Decision Processes? **(1 point)**

A The Logistics Domain

The **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**.

- 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 a location must be in the same location 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 $\text{at}(t_1, t_1)$ 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 shown on the following page:

- load-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: at(*pkg*, *loc*) $\wedge$ at(*trk*, *loc*)
  Effects: $\neg$at(*pkg*, *loc*), in(*pkg*, *trk*)

- load-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: at(*pkg*, *loc*) $\wedge$ at(*plane*, *loc*)
  Effects: $\neg$at(*pkg*, *loc*), in(*pkg*, *plane*)

- unload-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: in(*pkg*, *trk*) $\wedge$ at(*trk*, *loc*)
  Effects: $\neg$in(*pkg*, *trk*), at(*pkg*, *loc*)

- unload-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: in(*pkg*, *plane*) $\wedge$ at(*plane*, *loc*)
  Effects: $\neg$in(*pkg*, *plane*), at(*pkg*, *loc*)

- drive-truck(truck *trk*, location *from*, location *to*)
  Precondition: at(*trk*, *from*) $\wedge$ same-city(*from*, *to*)
  Effects: $\neg$at(*trk*, *from*), at(*trk*, *to*)

- fly-airplane(airplane *plane*, airport *from*, airport *to*)
  Precondition: at(*plane*, *from*)
  Effects: $\neg$at(*plane*, *from*), at(*plane*, *to*)