Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-10-23
Sal (1)	G36
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	ja
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

Exam: TDDD48 Automated Planning 2014-10-23

Important Notes

Read the following 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. Write an explanation that can be understood and applied by someone who does not already know the answer!
- 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

We know the following:

- A sequential solution plan π for a classical problem instance P is *redundant* iff it is possible to remove one or more actions from π in such a way that the remaining actions, *in their original order*, still form a solution that achieves the goal. For example, the solution $[a_1, a_2, a_3, a_4, a_5, a_6]$ is redundant if $[a_1, a_2, a_3, a_6]$ is also a solution.
- A solution plan π for a given problem instance is *minimal* iff there is no other solution for P that contains strictly fewer actions.

The standard **logistics domain**, where a set of packages should be transported to their destinations, is defined in Appendix A. You should do the following:

a) Create a classical planning problem instance for this domain and show a *redundant solution* π for this problem instance. Show *why* the solution is redundant – in other words, show which actions can be removed from the solution. **(1 point)**

   Note: You must provide formulas defining the exact initial state and goal of this problem instance (a vague description in natural language is not sufficient). To save time, make the problem instance *small*.

b) Create a classical planning problem instance for this domain and show a solution π for this problem instance, such that π is *not* redundant but still not minimal. Explain / motivate clearly why the solution is not redundant and why it is not minimal. **(1 point)**

   Note: You must provide formulas defining the exact initial state and goal of this problem instance (a vague description in natural language is not sufficient). To save time, make the problem instance *small*.

Let $P_1 = (O, s_0, g_1)$ and $P_2 = (O, s_0, g_2)$ be two classical planning problems that share the same operators and initial state. Let $\pi_1 = [a_1, \ldots, a_n]$ be a solution of length n for P_1, and let $\pi_2 = [b_1, \ldots, b_n]$ be a solution of the same length for P_2.

c) Suppose that the concatenated action sequence $\pi_1 \cdot \pi_2 = [a_1, \ldots, a_n, b_1, \ldots, b_n]$ happens to be *executable* starting at state s_0. Can we then be certain that it is also a *solution* for P_2 (achieves the goals of P_2)? Motivate clearly why this is or is not the case. If you want to provide an example, you may use the logistics domain or any other domain. **(1 point)**

2 Heuristics and Search Guidance

a) Is a *pattern database heuristic* a form of *relaxation heuristic*? Is it a form of *delete relaxation heuristic*? In each case, motivate why / why not. **(2 points)**

b) Explain: What is a *landmark*? (Note that there is a similar concept called an *action landmark*. We are asking about a standard (non-action) landmark.)

Then construct a simple problem instance in the logistics domain (Appendix A), with an initial state and a goal, and show two distinct landmarks for this problem instance. Explain *why* these are landmarks. Make sure you provide sufficient information about the problem instance so that this can be verified. **(2 points)**

c) Give an example of how landmarks can be used in the definition of a heuristic function for state-space planning. Keep the description general as opposed to applying it to a specific state or problem instance! **(1 point)**

d) Recall that the Fast Forward heuristic $h_{FF}(s)$ is based on the use of *planning graphs*. When $h_{FF}(s)$ is calculated for a specific state s that is encountered during the search process, a planning graph is generated – but not for the original search problem.

Explain how h_{FF} "modifies" a planning problem before a planning graph is constructed.

Give at least one example of why this speeds up the construction of the graph.

Also explain approximately how the heuristic value is then extracted from the planning graph. **(2 points)**

3 Partial-Order Planning

We now consider partial-order planning. We 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:

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. **(1 point)**

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! **(1 point)**

c) Show a partial plan for this problem in which one or more threats appear. Indicate all threats clearly. (Note that you might be able to extend this plan into a solution below.) **(1 point)**

d) Show a complete solution plan for this problem instance, making good use of both trucks to efficiently resolve all goals. Make sure that actions are not temporally constrained relative to each other unless this is necessary in order to achieve the goal. **(1 point)**

Note: For each action in a partially ordered plan, you must clearly show each precondition *above* the action and each *effect* below the action. You must also indicate all other relevant structural features in the plan: precedence constraints (solid arrows), causal links (dashed arrows) and threats.

4 Neo-Classical Planning

Recall that planning techniques that generate classical plans for classical planning domains, but that use alternative and non-trivial representations of search spaces, are called *neo-classical*. Such techniques include SAT planning (planning based on translations into propositional satisfiability) and GraphPlan (based on planning graphs).

a) In SAT planning, we *may* use *complete exclusion axioms*. What is the purpose of such axioms and when would we want to avoid using them? **(1 point)**

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**.

- 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 $\mathsf{at}(t_1, t_1)$ 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*)