Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-06-05
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER1
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 kl.	ca kl. 15
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	Inga
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	Valfritt
Antal exemplar i påsen	

Exam: TDDD48 Automated Planning 2014-06-06

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

a) Given a classical planning problem $P = (O, s_0, g)$, let $\Pi = \{\pi \mid \pi \text{ is applicable to } s_0\}$ be the set of all action sequences that are executable in P starting at s_0. Is Π infinite for *all*, *some* or *no* planning problems P? Explain why. **(1 point)**

b) In a classical state-space planner using forward search:

- A search state corresponds directly to a *world state*, which is a set containing exactly those atoms that are initially true.
- The initial search state corresponds to the *initial world state* specified in the planning problem instance.
- Each search state has one successor for *each action that is applicable* in the corresponding world state.

Provide the corresponding description for a planner using backward search. **(2 points)**

c) *Lifted* planning is a general technique applicable to a variety of search spaces and planning algorithms. During the lectures, we specifically discussed *lifted partial-order planning*. Explain what lifted planning is in general, and give a concrete example of why it can be useful when generating partial-order plans. In this example you should show a small lifted partial-order plan structure (or a reasonable fragment of one), contrast it against a non-lifted alternative, and explain what is better about the lifted version. **(2 points)**

2 Search Guidance

Given the size of a typical search space, a planner almost always needs some form of search guidance as opposed to doing blind search. Often such guidance takes the shape of a *heuristic function* that evaluates the "quality" of a certain search node, corresponding to one specific choice that can be made at a particular point in the search space.

a) Heuristic functions can yield *plateaus* in the search space. What is a plateau? Visualize one using part of a search space: Show a set of search nodes, a number of possible transitions between those nodes, and the type of heuristic values that characterize a plateau. Show which nodes are part of the plateau and explain in words *why* they form a plateau. **(1 point)**

b) Given an optimization problem, plateaus can in some cases be handled simply by terminating search as soon as a plateau is reached. Explain why this approach is generally not useful in classical planning. **(1 point)**

c) Is the h_{add} heuristic, also called h_0, admissible? If it is, motivate clearly why the calculations will always lead to an admissible heuristic value. If it is not, demonstrate using a small counterexample that includes a current state, a goal to be achieved, and a set of actions where the heuristic function can yield an inadmissible heuristic value. **(2 points)**

d) Many heuristics work by solving *subproblems* of the actual problem being solved, and then combining the costs of these subproblems in some way. For example, a single subproblem for the h_2 heuristic is constructed by selecting two literals from the "real" goal, and then (under)estimating the cost of achieving only these two literals. The heuristic then solves one such subproblem for *every* pair of goal literals, and returns the maximum of all calculated costs.

   What is the corresponding way in which subproblems are constructed for a *pattern database heuristic*? Describe the general steps and ideas involved. Assume that the problem is already in state-variable representation as opposed to classical representation, so that an initial conversion phase (as described during the lectures) is not necessary. **(2 points)**

3 Neoclassical Planning

Neoclassical planning methods include techniques based on planning graphs as well as propositional satisfiability (SAT).

a) Recall that planning based on planning graphs iterates over two phases: Forward graph expansion, where two new graph layers (each of a distinct type) is added, and backward graph search.

   Suppose we do forward graph expansion to level n. What information can we extract from the resulting planning graph? In other words, how can we interpret the information present in the different levels of a particular planning graph? You should construct a simple planning graph to refer to in your explanation.

   Note: There is no need to make the graph *complete* up to level n – simply add sufficient information to illustrate your answer. However, the information that *is* there should be correct. Also, the graph itself should only be an illustration to your *descriptions* of what the graph includes. **(2 points)**

b) SAT planners require *frame axioms* (for example, *explanatory* frame axioms) to ensure that the SAT solver gives correct results that correspond to valid plans.

   Explain how planning could go wrong if frame axioms were not included in the SAT translation of a planning problem. Also demonstrate using a concrete example. The example does not have to involve an entire SAT assignment and SAT formula processed from beginning to end, as this would take considerable time to generate by hand. It is sufficient to illustrate the most central aspects of how the planning process could return invalid plans given a lack of frame axioms. **(2 points)**

4 Markov Decision Processes

a) The states and transitions involved in a classical planning problem can be described using a specific type of state transition system. This is also the case for the states and transitions involved in an MDP, a (fully observable) Markov Decision Process.

However, there is a key difference in expressivity between classical planning problems and MDP problems, which is also reflected in the definition of their respective state transition systems. What is this difference (what can you model in an MDP but not in classical planning)? How is the increased expressivity represented in the state transition system of an MDP? **(1 point)**

b) Markov Decision Processes are characterized by the *Markov property*. What is the Markov property? **(1 point)**

c) Markov Decision Processes often use a *discount factor*. Which property of a policy is calculated using the discount factor, and how is the discount factor used in the calculations? You can express the answer as a formula or in words, as long as the description clearly shows how the discount factor is used. Also explain why the use of a discount factor is not only a mathematical trick but in many cases results in a better model of our preferences/goals/desires than if the discount factor had not been used. **(2 points)**

d) *Policy iteration* and *value iteration* are two standard iterative methods for finding suitable policies for an MDP. What is the main difference between these methods (in terms of what they *do* in each iteration)? What consequences can this difference have in terms of performance and convergence towards an optimal policy for an MDP? **(2 points)**