Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2013-06-01
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och <u>ringa in</u> vilken sal som avses	TER4
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.	Ja, någon gång efter 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	linjerat
Antal exemplar i påsen	

Exam: TDDD48 Automated Planning 2013-06-01

Important Notes

Read the following before you begin!

- Though the questions are in English, you may **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.
- 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.

1 Fundamental Concepts

a) When is an action *relevant*? When/where is the concept of relevance used in planning, and in what way is it useful? **(2 points)**

b) *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 plan (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 Heuristics and Search Guidance

a) 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)**

b) Describe the general *relaxation principle*. In particular: Given a problem, what characterizes a *relaxed* problem? How, exactly, can a relaxed problem be used to calculate an admissible heuristic value for a particular state in the original problem? Why does this method (calculation) guarantee admissibility? **(2 points)**

c) 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)**

d) What is a *landmark* (not an *action landmark*)? Explain the general definition of a landmark. Then give an example of a concrete landmark from a problem instance in a simple standard planning domain such as for example Logistics, Blocks World or Towers of Hanoi. Explain why this is a landmark and make sure you provide sufficient information about the problem instance so that this can be verified. **(2 points)**

e) 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)**

3 Partial-Order Planning

We now consider partial-order planning using the **elevator domain** defined in Appendix A. For simplicity this domain only has a single lift, but there are still opportunities for partial ordering during the actual planning process! You should use the following problem instance:

```
(define (problem mixed-f6-p3-u0-v0-g0-a0-n0-A0-B0-N0-F0-r0)
    (:domain elevator)
    (:objects p0 p1 p2 - passenger
              f0 f1 f2 f3 f4 f5 - floor)
    (:init
        (above f0 f1) (above f0 f2) (above f0 f3) (above f0 f4) (above f0 f5)
        (above f1 f2) (above f1 f3) (above f1 f4) (above f1 f5)
        (above f2 f3) (above f2 f4) (above f2 f5)
        (above f3 f4) (above f3 f5)
        (above f4 f5)
        (origin p0 f0) (destin p0 f4)
        (origin p1 f3) (destin p1 f1)
        (origin p2 f5) (destin p2 f1)
        (lift-at f0)
    )
    (:goal (and (served p0) (served p1) (served p2)))
)
```

You should do the following (also make sure you read the 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 (corresponding to what we discussed during the lectures). This is the partial plan that corresponds to the first node ("root node") generated in the search space. Make sure that all relevant aspects of the actions involved are clearly indicated in the plan. **(1 point)**

b) Show *all* the immediate successors of the initial partial plan in the partial-order search space. That is, demonstrate all the different ways in which a standard PSP-like partially ordered planner might modify π_0 in a single step. If this should result in several almost identical partial plans, you may save time by showing a set of representative example partial plans and explaining (very clearly!) in text which additional successors exist. **(2 points)**

c) Show a complete solution plan for this problem instance. 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.

Note: As you see, there are quite a lot of true instances of `above()`. To reduce space requirements, you may illustrate all true instances of `above()` simply as "`above(...)`" in the same place where you would normally have written out all instances explicitly.

4 Planning Graphs

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.

a) 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. There is no need to make it *complete* up to level n – simply add sufficient information to illustrate your answer. However, the information that *is* there should be correct. **(2 points)**

b) Why is backward search in a planning graph more efficient than simply doing standard backward search in the standard state space? **(1 point)**

c) The Fast Forward heuristic is based on the use of planning graphs. Explain how the planning problem is modified before a planning graph is constructed and 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)**

A The Elevator Domain

The following is a variation of the standard *elevator domain*, which contains a set of *passengers* that should be transported to their destination *floors*. For simplicity, we only use a single elevator/lift. 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.

Each passenger is represented by a distinct object (p1, p2, p3, ...). Since classical planners generally do not support numeric values, floors are also represented as distinct objects (f1, f2, f3, ...) together with an above predicate, rather than simply using numbers and ">" (greater than).

Note that there is no predicate directly specifying the current floor of a passenger. Instead, there are fixed (constant) predicates specifying the origin and destin(ation) of the passenger, together with two other predicates that are modified by operators: boarded and served. If the passenger has not boarded and is not served, he/she must be at the origin. If the passenger has boarded but is not served, he/she is in the lift. If the passenger is not currently boarded and is served, he/she is at the destination. This is essentially a clever way of encoding control knowledge in the fundamental structure of the domain without the use of explicit temporal control formulas, with preconditions ensuring one cannot board or debark an elevator unnecessarily.

```
(define (domain elevator)
  (:requirements :strips)
  (:types object
          passenger - object  ;; Passengers p1, p2, ...
          floor - object)     ;; Floors f1, f2, ...

(:predicates
    (origin ?person - passenger ?floor - floor)
    ;; initially, ?person is at ?floor
    ;; (remains true even after the person has moved)

    (destin ?person - passenger ?floor - floor)
    ;; the destination of ?person is ?floor

    (above ?floor1 - floor  ?floor2 - floor)
    ;; ?floor2 is located above ?floor1
    ;; (for example, f4 may be above f3, f2 and f1)

    (boarded ?person - passenger)
    ;; true if ?person is on board the lift

    (served ?person - passenger)
    ;; true if ?person has arrived at her destination

    (lift-at ?floor - floor)
    ;; current position of the lift is at ?floor
)
```

Actions are shown on the following page.

```
(:action board
  :parameters (?f - floor ?p - passenger)
  :precondition (and (lift-at ?f) (origin ?p ?f))
  :effect (boarded ?p))

(:action depart
  :parameters (?f - floor ?p - passenger)
  :precondition (and (lift-at ?f) (destin ?p ?f) (boarded ?p))
  :effect (and (not (boarded ?p)) (served ?p)))

(:action up ;; Moves the lift up
  :parameters (?f1 - floor ?f2 - floor)
  :precondition (and (lift-at ?f1) (above ?f1 ?f2))
  :effect (and (lift-at ?f2) (not (lift-at ?f1))))

(:action down ;; Moves the lift down
  :parameters (?f1 - floor ?f2 - floor)
  :precondition (and (lift-at ?f1) (above ?f2 ?f1))
  :effect (and (lift-at ?f2) (not (lift-at ?f1))))
)
```