**CMPSCI 683** **Fall 2006**
Prof. Victor Lesser October 26, 2006

***MID-TERM EXAM***

*The answers to these questions should be specific and to the point; we are not looking for essays! The exam is broken into roughly two equal parts: a set of 8 short questions worth 48 points, and one long integrated question that has 4 parts worth 52 points.*

## 1. Short questions (48 points, 6 Points each)

a) What is the type of knowledge exploited in RBFS that speeds up the search over IDA* which is not used in IDA*?

Exploits information gathered from previous searches about minimum f so as to focus further searches. It uses information about the best alternative path of its ancestors and uses this to backtrack and search more promising looking nodes. This results in not regenerating the entire tree. This is in contrast to IDA* which re-generates nodes at every iteration. Each backtracking in RBFS is equivalent to an iteration of IDA*. So the speed gain results from only regenerating nodes when they look more promising than the current node.

b) The utility for state in an MDP is defined by $U(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s')$

When the reward is only in terms of the current state r(s), how does this utility formula change if reward is specified in terms of r (s',a, s)?

$U(s) = \max_a \sum_{s'} [\, P(s'|s,a)\, r(s',a,s) + \gamma\, P(s'|s,a)\, U(s')$

c) Why in anytime A*, is it important to check immediately after generating a node whether it is goal node while this is only done in A* when the node is taken off the open list?

By checking the node list in anytime A* early you can find whether you have an intermediate solution that is better than you current best solution. This creates the anytime nature of algorithm. It also allows you to prune the open list and focus the search more.

d) Give an informal sketch of the proof showing why the following variant of A*, called MaxA*, always produces the maximum value. In MaxA*, the heuristic function, h, has to always be an overestimate of the true value of reaching the goal node; and at every point you are expanding the node on the open list with highest f. The stopping criterion is similar to A*.

Let us suppose this does not produce the optimal solution. Then the system will stop with some non-optimal solution x whose f(x) < f(x') where x' is the optimal score. Given that at each point you are taking the node with the highest value, then at some point y' along the x' path, f(y') <=

f(x) but we know that f(y') is always an overestimate f(x'). Thus f(x') <= f(y') which implies that f(x') <= f(x) which is a contradiction with f(x) < f(x').

e) The following is a systematic mini-conflict search algorithm for a CSP. Does this algorithm exploit either value and/or variable ordering heuristics? If so, explain how each type of heuristic is exploited in the algorithm.

There is no variable ordering used explicitly in the algorithm. The mini-conflict heuristic is used to do value ordering so as to minimize the likelihood that future assignments will conflict.

```
Procedure INFORMED-BACKTRACK (VARS-LEFT VARS-DONE)
   If all variables are consistent, then solution found, STOP.
   Let VAR = a variable in VARS-LEFT that is in conflict (chosen randomly).
   Remove VAR from VARS-LEFT.
   Push VAR onto VARS-DONE.
   Let VALUES = list of possible values for VAR ordered in ascending
            order according to number of conflicts with variables
            in VARS-LEFT. ; min-conflict heuristic
   For each VALUE in VALUES, until solution found:
            If VALUE does not conflict with any variable that is in VARS-DONE,
            then Assign VALUE to VAR.
                     Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE)
            end if
   end for
end procedure
Begin program
   Let VARS-LEFT = list of all variables, each assigned an initial state
   Let VARS-DONE = nil
   Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE)
End program
```

f) In the HEARSAY-II Speech Understanding System, there were a number of heuristics and strategies used to control the search. Explain why a control strategy was used where the search process at the lower levels of abstraction was more like a beam search and at higher levels of abstraction more like a best-ordered (opportunistic) search.

At the lower levels in the blackboard the hypotheses confidence values were not very good. A highly rated hypothesis could be wrong a reasonable amount of time while a medium rated hypothesis could often be right. The reason was that evaluation of lower level hypotheses only used very local knowledge (constraints) in its evaluation. This lack of confidence in the rating meant that prematurely throwing out low-level hypotheses as a result of their having a medium rating could eliminate valuable hypotheses that were important in constructing more encompassing hypotheses at higher levels. Thus, at the lower-levels a more breadth-first type of search was appropriate. It was only after more encompassing higher-level hypotheses were constructed was there sufficient confidence in hypothesis' rating was it worthwhile to search in a more depth-first manner.

g) What type of relationship, if any, do you see between a POMDP and an HMM?

The Viterbi algorithm as used in HMMs based on the partial observation sequence of the first n steps generates the likelihood of being in a particular state at time step n. This is precisely the belief state calculation in a POMDP.

h) Briefly explain the intent behind the following ideas used in local search: random restart and simulated annealing. Are there any differences in the issues in search control they are each trying to solve? Additionally, can they be used together, and if so, would there be any advantage?

Both approaches are dealing with the problem of finding local minimum in a local stochastic search. Simulated annealing approaches the problem by taking a certain percentage of moves that are not optimal from a local perspective to get off a local minimum. While random restart tries to start the search in different parts of the search space with the hope that either you will not run into the a local optimum from one of the random starting points, or you will find a better local minimum than the one you have currently found. Both approaches can profitably be used together since they represent different mechanisms for avoiding getting stuck on a local optimum.

## PART 2 (52 points total)

Suppose you were doing a www-based search where you were trying to find $n$ documents relevant to your query $q$. Assume there is a set of relevant data bases X, where for each database $x$ you searched cost money $m_x$ and time $t_x$ and has a probability of $P(k|q,x)$ of finding $k$ relevant documents. The goal of the search strategy is to find $n$ documents by some time $T$ that minimizes the amount of money spent and that you are confident that at least $y$ percentage of time you will succeed in accomplishing the goal. It is assumed you are doing a sequential search (one database at a time).

a) (24 points) Explain in some detail how the construction of a search strategy could be framed as an A* search, an RTA* search, and MDP. It is okay if the approach does not lead to an optimal strategy if there is no way to generate an optimal strategy given the particular search framework. I want you to creatively make approximations to fit the problem to a particular framework if there is not a straight forward mapping.

A* --- let l = arg $\max_k$ **((Sum[ from k to maxdocuments]P(k|q,x)) = > y)** ; l is then the number of relevant documents gained y % of the time by accessing database x and would be the approximation of applying an operator to database x. The legal operators at each point would be the databases that have not yet been accessed. If l = 0 then we don't access that database. The A* search would find the goal state with the minimum cost. A state would contain the number of documents accumulated, the cost incurred and the time spent. A goal state would be one where the time is <= T and the number of documents retrieved y% of the time will be => than n. One example of a heuristic is based on computing off line the minimum cost for producing k documents with likelihood y ignoring time. This would always be a lower estimate of finding a sequence of databases to get to a goal state since this does not include time and may involve databases already accessed.

RTA* --- take as a horizon getting min (1/2, or remaining documents) in time minimum(T/2, time left). Take first data base on search path. Access first data base. Record time taken and number of relevant documents retrieved. Repeat search process given number of relevant documents still to find and time left.

MDP -- maps directly into an MDP policy search. A node is number of documents found, time accrued, and databases already accessed since the possible operators at each state are the databases not yet accessed. The cost incurred doesn't need to kept in the state since it can be computed from the databases accessed. The transition probability to new states is based on probability of finding a specific number of relevant documents. There is no reward until a terminal state is reached. Non-goal terminal states are states where time is beyond deadline and it has a reward of 0. Goal terminal states are those that have => then n documents, time less than equal to T and it has a reward of 1/(accumulated costs incurred). In this formulation, the notion of y% of the time satisfying the goal does not make sense since each state represents the actual results of the database access. Like A* it is solving a slightly different problem. If the reward was set to 1 then it would maximize the likelihood of getting n documents irrespective of cost. Setting the reward of 1/(accumulated costs incurred) is establishing some balance between likelihood of getting n documents and the cost.

b) (15 points) Discuss the relative strengths and weaknesses of the different approaches from the perspective of computational costs in constructing the strategy and the optimality of the given strategy.

A* uses expected value to define an ordering that may or may not lead to a valid solution because of its taking out the probabilistic nature of data base access. It produces a fixed schedule for accessing the databases independent of the result of each database access. It is unclear whether A* or RTA* produce better solutions or even produce acceptable solutions more often. RTA* takes advantage of feedback in what has happen in each access but it does not do an end-end analysis which may lead to accessing data bases that won't allow a solution to be found because for instance too much time has been used up-front. MDP will take advantage of the results of database access to decide what to do and takes an end-to-end perspective that is not possible in RTA*. However, it generates an optimal policy for a problem that is different than one specified; how close it is to the solution to the "real" problem is open for conjecture.

c) (8 points) Briefly discuss what the implications would be of $t_x$ not being fixed, but rather for accessing each database $x$ there is $P(t_x=j \mid q,x)$ for the A* and MDP approaches.

A* could just associate with each operator the maximum time for accessing each data base. This leads to guaranteeing the solution will not take more time but it can eliminate a lot of feasible alternatives and may cause no valid solutions to be generated. You could take a 90% time criterion but again that is trading off the likelihood the solution you generate will be valid when executed.

MDP can accommodate this additional uncertainty but it will explode the number of states in the policy space since there is now a cross product of next states based on (time taken*number of documents retrieved)

d) (5 points) What happens if you cannot find a strategy to find the documents by time ***T*** no matter how much money you spend? How could you modify your approach in the A* and MDP approaches to find the minimum ***T*** where you can find such a strategy?

In both approaches you would have to keep track of the states that were eliminated because of T being too large, and then repeat the search such that you try a new T that is the smallest of the times in the nodes eliminated. You may need to repeat this a number of times until you find an acceptable solution.