

1. (a)

- $\alpha$ - $\beta$  pruning processes min and max nodes differently:
  - it keeps track of  $\alpha$ , the best score for max along the path to the current state: at a max node update  $\alpha$  whenever a successor produced a larger value. [2]
  - it prunes further successors of max nodes if the current max score for the node becomes larger or equal than  $\beta$  [1]
  - it keeps track of  $\beta$ , the best score for min along the path to the current state: at a min node update  $\beta$  whenever a successor produced a smaller value. [2]
  - it prunes further successors of min nodes if the current min score for the node becomes less or equal than  $\alpha$  [1]
- By keeping track of  $\alpha$  and  $\beta$  it is possible to eliminate nodes that would not be chosen by a parent node because a better max or min score has already been found. So the result does not change. [3]

Total: [9]

1. (b)

- Non-deterministic games can be dealt with by introducing chance nodes after max and min nodes in the game tree. [2]
- For a chance node, recursively process the successor nodes as max or min nodes depending on who makes the next move. Return the weighted average of the values returned using the successor probabilities as weights. [3]
- $\alpha$ - $\beta$  fails because a chance node considers all successors, i.e. needs the actual values of the evaluation function. The evaluation function does not just order the game moves anymore, but indicates the expected gain. Without evaluating all successors, the average value cannot be determined or bound without additional assumptions. [3]

Total: [8]

1.(c)

- For max and min nodes the  $\alpha$ - $\beta$  pruning can be done as before. [1]
- The evaluation function can be bound by the interval  $[-20, 20]$ , so at a chance node we have the following estimates for the average:

$$\frac{1}{6}(V_1 + \dots + V_{l-1} + V_l + 20(6-l)) \leq \alpha$$

$$\frac{1}{6}(V_1 + \dots + V_{l-1} + V_l - 20(6-l)) \geq \beta$$

where  $V_k$  are the values returned by successor  $k$  and we are currently exploring successor  $l$ . [4]

- Hence, we can prune the search at a chance node if either

$$V_l \leq 6\alpha - V_1 - \dots - V_{l-1} + 20(6-l)$$

$$V_l \geq 6\beta - V_1 - \dots - V_{l-1} + 20(6-l)$$

This can easily be done by keeping track of the sum of the returned values and the lower/upper limit estimates of the unexpanded nodes. [3]

Total: [8]

2. (a)

- Variables:  $n$  variables  $E_i$  for each exam  
 Domain:  $m$  time slots  $t_i$  for each available time slot  
 Constraints: for each pair of exams for which there is at least one student who has to take them both add a constraint which says the corresponding  $E_i$  cannot have the same value.  
 (Exchange of variables and domain is possible.) [4]
  - The constraint graph consists of a node for each exam and an edge for each pair of exams for which there is at least one student who has to take them both (equivalent to the map colouring problem). [2]
- Total: [6]

2. (b)

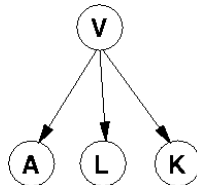
- Backtracking is a depth-first search algorithm of a tree where nodes list the values assigned to variables so far and the edges represent assigning a value to one of the unassigned variables from its domain. For constraint satisfaction problems with forward checking we introduce the following improvements: [3]
    - Fix the order of assignment of variables to fix the depth of the search tree as it does not matter in which sequence variables are assigned. [2]
    - Check for constraint violations at each node and stop expanding a node which violates at least one constraint as assigning further variables will not eliminate the violation. [2]
    - Keep track of the remaining legal values for unassigned variables and terminate the search when any variable has no legal values left. Any further assignment cannot lead to a valid assignment for this variable. [2]
  - Heuristics:
    - Always choose the least constraining value: minimise the number of restrictions on the values of any still unassigned variables. [2]
    - Always choose the most-constrained variable: choose the variable which has the smallest number of valid values left [2]
- Total: [13]

2. (c)

- The constraint solver will find a solution which satisfies all constraints, but this does not mean that the exams are finished as soon as possible as this is not part of the constraint system and in general very hard to enforce in this approach. [2]
  - One could formulate the problem as an optimisation problem where the states are all possible variable assignments. Then one could use hill-climbing / simulated annealing to find a solution by specifying the neighbours of one state as all states with only one different variable-value assignment. The target function is a weighted sum of the number of unsatisfied constraints and the length of the time span for all exams. It is important to give a higher weight to the number of unsatisfied constraints as otherwise the optimisation algorithm may find a local minimum which does not satisfy all constraints. (Variations possible). [4]
- Total: [6]

3. (a)

- $A, L$  and  $K$  are conditionally independent given  $V$ . [2]
- Bayesian Network:



Correct as local semantics mean that children are conditionally independent of their non-descendants given their parents. [4]

Total: [6]

3. (b)

- $P(v|a, \neg l, \neg k) = \frac{P(a, \neg l, \neg k|v)}{P(a, \neg l, \neg k)}$  [2]  
(Bayes' theorem)
- $= \frac{P(a|v)P(\neg l|v)P(\neg k|v)P(v)}{P(a|v)P(\neg l|v)P(\neg k|v)P(v) + P(a|\neg v)P(\neg l|\neg v)P(\neg k|\neg v)P(\neg v)}$  [4]  
(global semantics of Bayesian network)
- $= \frac{(.9)(.8)(.9)(.2)}{(.9)(.8)(.9)(.2) + (.2)(.2)(.2)(.8)}$  [2]  
(values derived from table)
- $= \frac{1296}{1360} \approx .9529$  [1]
- In the table all entries must be between 0.0 and 1.0 and  $P(v) + P(\neg v) = 1.0$ . [2]

Total: [11]

3. (c)

- Experimentally  $P(l|k) = .2$ ,  $P(l|a, k) = .1$ , [2]
- The network makes the assumption  $P(L|K) = P(L|A, K)$  [2]
- This is clearly not fulfilled. A suitable network would be a complete network with an additional link between  $A$  and  $L$  (either direction). [2]
- As we do not have any information about whether any of the e-mails contain a virus, it has no implication on the structure of the network from Question 3(a). [2]

Total: [8]

4. (a)

- Prove that  $\mathbf{X} \Leftrightarrow \mathbf{Y}$  is valid, by first negating it. [2]
- Then convert the resulting sentence to CNF, i.e. represent the negated sentence as a conjunction of disjunctions of positive or negative literals. [2]
- Finally apply the resolution rule

$$\frac{p_1 \vee \dots \vee p_j \vee \dots \vee p_m, \quad q_1 \vee \dots \vee q_k \vee \dots \vee q_n}{(p_1 \vee \dots \vee p_{j-1} \vee p_{j+1} \dots \vee p_m \vee q_1 \dots \vee q_{k-1} \vee q_{k+1} \dots \vee q_n)\sigma}$$

where  $p_j\sigma = \neg q_k\sigma$  to the clauses until an empty clause (contradiction) is found. [2]

Total: [6]

4. (b)

- **A:**  $\forall x \forall y \text{Cat}(x) \wedge \text{Cat}(y) \wedge \neg \text{EatsFish}(x) \wedge \neg \text{EatsFish}(y) \Rightarrow x = y$  [4]
- **B:**  $\forall x \text{Cat}(x) \wedge \neg \text{At}(x, \text{Cardiff}) \Rightarrow \text{EatsFish}(x)$  [3]
- Both sentences are hard to express in propositional logic because we would have to provide instantiations of the predicates for each object. [1]

Total: [8]

4. (c)

- Convert  $C$  to CNF:  
 $\forall x \neg [\exists y P(x, y)] \vee Q(x)$   
 $\forall x [\forall y \neg P(x, y)] \vee Q(x)$   
 C1:  $\neg P(x, y) \vee Q(x)$  [3]
- Convert  $\neg D$  to CNF:  
 $\neg [\forall x, y P(x, y) \Rightarrow Q(x)]$   
 $\neg [\forall x, y \neg P(x, y) \vee Q(x)]$   
 $\exists x, y \neg [\neg P(x, y) \vee Q(x)]$   
 $\exists x, y P(x, y) \wedge \neg Q(x)$   
 D1:  $P(G, H)$  and D2:  $\neg Q(G)$  [5]
- Resolve C1 with D1,  $\{x/G, y/H\}$  giving  
 R:  $Q(G)$  [2]
- Resolving D2 with R gives empty clause. [1]

Total: [11]