Reference Book: "Artificial Intelligence A Modern Approach", Third Edition

Disclaimer: Following are few samples question. Do not depend on the following question for exampreparation. These are given for you to understand general question patterns.

Chapter 2

Q2.1 Suppose that the performance measure is concerned with just the first T time steps of the environment and ignores everything thereafter. Show that a rational agent's action may depend not just on the state of the environment but also on the time step it has reached.

Ans:

2.1 This question tests the student's understanding of environments, rational actions, and performance measures. Any sequential environment in which rewards may take time to arrive will work, because then we can arrange for the reward to be "over the horizon." Suppose that in any state there are two action choices, a and b, and consider two cases: the agent is in state s at time T or at time $T-1$. In state s, action a reaches state s' with reward 0, while action b reaches state s again with reward 1; in s' either action gains reward 10. At time $T-1$, it's rational to do a in s, with expected total reward 10 before time is up; but at time T, it's rational to do b with total expected reward 1 because the reward of 10 cannot be obtained before time is up.

Students may also provide common-sense examples from real life: investments whose payoff occurs after the end of life, exams where it doesn't make sense to start the high-value question with too little time left to get the answer, and so on.

The environment state can include a clock, of course; this doesn't change the gist of the answer—now the action will depend on the clock as well as on the non-clock part of the state—but it does mean that the agent can never be in the same state twice.

Q 2.2 Let us examine the rationality of various vacuum-cleaner agent functions.

a. Show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational under the assumptions listed on page 38.

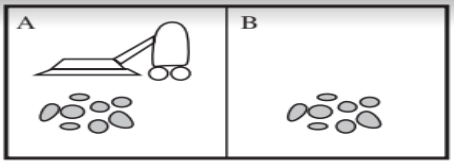


Figure 2.2 A vacuum-cleaner world with just two locations.

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

b. Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require internal state?

c. Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn? If not, why not?

Ans:

2.2 Notice that for our simple environmental assumptions we need not worry about quantitative uncertainty.

a. It suffices to show that for all possible actual environments (i.e., all dirt distributions and initial locations), this agent cleans the squares at least as fast as any other agent. This is trivially true when there is no dirt. When there is dirt in the initial location and none in the other location, the world is clean after one step; no agent can do better. When there is no dirt in the initial location but dirt in the other, the world is clean after two steps; no agent can do better. When there is dirt in both locations, the world is clean after three steps; no agent can do better. (Note: in general, the condition stated in the first sentence of this answer is much stricter than necessary for an agent to be rational.)

b. The agent in (a) keeps moving backwards and forwards even after the world is clean. It is better to do $NoOp$ once the world is clean (the chapter says this). Now, since the agent's percept doesn't say whether the other square is clean, it would seem that the agent must have some memory to say whether the other square has already been cleaned. To make this argument rigorous is more difficult—for example, could the agent arrange things so that it would only be in a clean left square when the right square was already clean? As a general strategy, an agent *can* use the environment itself as a form of **external memory**—a common technique for humans who use things like appointment calendars and knots in handkerchiefs. In this particular case, however, that is not possible. Consider the reflex actions for $[A, Clean]$ and $[B, Clean]$. If either of these is $NoOp$, then the agent will fail in the case where that is the initial percept but the other square is dirty; hence, neither can be $NoOp$ and therefore the simple reflex agent is doomed to keep moving. In general, the problem with reflex agents is that they have to do the same thing in situations that look the same, even when the situations are actually quite different. In the vacuum world this is a big liability, because every interior square (except home) looks either like a square with dirt or a square without dirt.

c. If we consider asymptotically long lifetimes, then it is clear that learning a map (in some form) confers an advantage because it means that the agent can avoid bumping into walls. It can also learn where dirt is most likely to accumulate and can devise an optimal inspection strategy. The precise details of the exploration method needed to construct a complete map appear in Chapter 4; methods for deriving an optimal inspection/cleanup strategy are in Chapter 21.

Q2.4 For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties listed in Section 2.3.2.

• Playing soccer.

• Exploring the subsurface oceans of Titan.

• Shopping for used AI books on the Internet.

• Playing a tennis match.

• Practicing tennis against a wall.

• Performing a high jump.

• Knitting a sweater.

• Bidding on an item at an auction.

Ans: Check solution manual for Answers.

Q2.5 (for this question you may also need to present the diagrams for agents), Q2.6, Q 2.10, Q2.11

Ans: Check solution manual for Answers.

Chapter 3

Q 3.1 Explain why problem formulation must follow goal formulation.

Ans: In goal formulation, we decide which aspects of the world we are interested in, and which can be ignored or abstracted away. Then in problem formulation we decide how to manipulate the important aspects (and ignore the others). If we did problem formulation first we would not know what to include and what to leave out. That said, it can happen that there is a cycle of iterations between goal formulation, problem formulation, and problem solving until one arrives at a sufficiently useful and efficient solution.

Q 3.2 Your goal is to navigate a robot out of a maze. The robot starts in the center of the maze facing north. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall.

a. Formulate this problem. How large is the state space?

b. In navigating a maze, the only place we need to turn is at the intersection of two or more corridors. Reformulate this problem using this observation. How large is the state space now?

c. From each point in the maze, we can move in any of the four directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using these actions. Do we need to keep track of the robot's orientation now?

d. In our initial description of the problem we already abstracted from the real world, restricting actions and removing details. List three such simplifications we made.

Ans:

a. We'll define the coordinate system so that the center of the maze is at (0, 0), and the maze itself is a square from (−1, −1) to (1, 1).

Initial state: robot at coordinate (0, 0), facing North.

Goal test: either $|x| > 1$ or $|y| > 1$ where (x, y) is the current location.

Successor function: move forwards any distance d; change direction robot it facing.

Cost function: total distance moved.

The state space is infinitely large, since the robot's position is continuous.

b. The state will record the intersection the robot is currently at, along with the direction it's facing. At the end of each corridor leaving the maze we will have an exit node. We'll assume some node corresponds to the center of the maze.

Initial state: at the center of the maze facing North.

Goal test: at an exit node.

Successor function: move to the next intersection in front of us, if there is one; turn to face a new direction.

Cost function: total distance moved.

There are 4n states, where n is the number of intersections.

c. Initial state: at the center of the maze.

Goal test: at an exit node.

Successor function: move to next intersection to the North, South, East, or West.

Cost function: total distance moved.

We no longer need to keep track of the robot's orientation since it is irrelevant to predicting the outcome of our actions, and not part of the goal test. The motor system that executes this plan will need to keep track of the robot's current orientation, to know when to rotate the robot.

d. State abstractions:

(i) Ignoring the height of the robot off the ground, whether it is tilted off the vertical.

(ii) The robot can face in only four directions.

(iii) Other parts of the world ignored: possibility of other robots in the maze, the weather in the Caribbean.

Action abstractions:

(i) We assumed all positions we safely accessible: the robot couldn't get stuck or

damaged.

(ii) The robot can move as far as it wants, without having to recharge its batteries.

(iii) Simplified movement system: moving forwards a certain distance, rather than controlled each individual motor and watching the sensors to detect collisions.

Q 3.9 The missionaries and cannibals problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution.

Ans: Here is one possible representation: A state is a six-tuple of integers listing the number of missionaries, cannibals, and boats on the first side, and then the second side of the river. The goal is a state with 3 missionaries and 3 cannibals on the second side. The cost function is one per action, and the successors of a state are all the states that move 1 or 2 people and 1 boat from one side to another.

Q3.10 Define in your own words the following terms: state, state space, search tree, search node, goal, action, transition model, and branching factor.

Ans: A state is a situation that an agent can find itself in. We distinguish two types of states: world states (the actual concrete situations in the real world) and representational states (the abstract descriptions of the real world that are used by the agent in deliberating about what to do).

A state space is a graph whose nodes are the set of all states, and whose links are actions that transform one state into another.

A search tree is a tree (a graph with no undirected loops) in which the root node is the start state and the set of children for each node consists of the states reachable by taking any action.

A search node is a node in the search tree.

A goal is a state that the agent is trying to reach.

An action is something that the agent can choose to do.

A successor function described the agent's options: given a state, it returns a set of (action, state) pairs, where each state is the state reachable by taking the action.

The branching factor in a search tree is the number of actions available to the agent.

Q3.11 What's the difference between a world state, a state description, and a search node? Why is this distinction useful?

Ans: A world state is how reality is or could be. In one world state we're in Arad, in another we're in Bucharest. The world state also includes which street we're on, what's currently on the radio, and the price of tea in China. A state description is an agent's internal description of a world state. Examples are In(Arad) and In(Bucharest). These descriptions are necessarily approximate, recording only some aspect of the state.

We need to distinguish between world states and state descriptions because state description are lossy abstractions of the world state, because the agent could be mistaken about how the world is, because the agent might want to imagine things that aren't true but it could make true, and because the agent cares about the world not its internal representation of it. Search nodes are generated during search, representing a state the search process knows how to reach. They contain additional information aside from the state description, such as the sequence of actions used to reach this state. This distinction is useful because we may generate different search nodes which have the same state, and because search nodes contain more information than a state representation.

Q3.15 Consider a state space where the start state is number 1 and each state k has two successors: numbers 2k and 2k + 1.

a. Draw the portion of the state space for states 1 to 15.

b. Suppose the goal state is 11. List the order in which nodes will be visited for breadth first search, depth-limited search with limit 3, and iterative deepening search.

c. How well would bidirectional search work on this problem? What is the branching factor in each direction of the bidirectional search?

d. Does the answer to (c) suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search?

e. Call the action going from k to 2k Left, and the action going to 2k + 1 Right. Can you find an algorithm that outputs the solution to this problem without any search at all?

Ans:

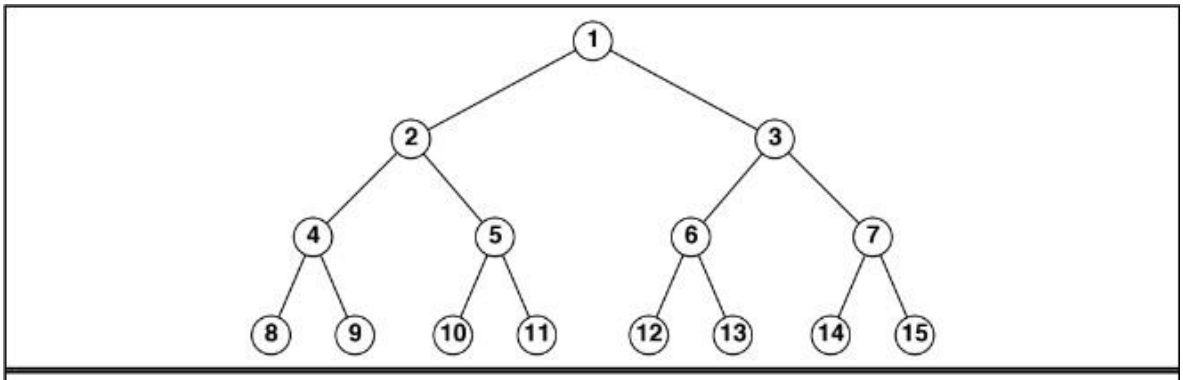


Figure S3.1 The state space for the problem defined in Ex. 3.15.

a. See Figure S3.1.

b. Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
Depth-limited: 1 2 4 8 9 5 10 11
Iterative deepening: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11

c. Bidirectional search is very useful, because the only successor of n in the reverse direction is $\lfloor (n/2) \rfloor$. This helps focus the search. The branching factor is 2 in the forward direction; 1 in the reverse direction.

d. Yes; start at the goal, and apply the single reverse successor action until you reach 1.

e. The solution can be read off the binary numeral for the goal number. Write the goal number in binary. Since we can only reach positive integers, this binary expansion beings with a 1. From most- to least- significant bit, skipping the initial 1, go Left to the node $2n$ if this bit is 0 and go Right to node $2n + 1$ if it is 1. For example, suppose the goal is 11, which is 1011 in binary. The solution is therefore Left, Right, Right.

3.17 On page 90, we mentioned **iterative lengthening search**, an iterative analog of uniform cost search. The idea is to use increasing limits on path cost. If a node is generated whose path cost exceeds the current limit, it is immediately discarded. For each new iteration, the limit is set to the lowest path cost of any node discarded in the previous iteration.

a. Show that this algorithm is optimal for general path costs.

b. Consider a uniform tree with branching factor b, solution depth d, and unit step costs. How many iterations will iterative lengthening require?

c. Now consider step costs drawn from the continuous range $[\epsilon, 1]$, where $0 < \epsilon < 1$. How many iterations are required in the worst case?

d. Implement the algorithm and apply it to instances of the 8-puzzle and traveling salesperson problems. Compare the algorithm's performance to that of uniform-cost search, and comment on your results.

Ans:

a. The algorithm expands nodes in order of increasing path cost; therefore the first goal it encounters will be the goal with the cheapest cost.

b. It will be the same as iterative deepening, d iterations, in which O(bd) nodes are

generated.

c. d/ε

d. Implementation not shown.

3.18 Describe a state space in which iterative deepening search performs much worse than depth-first search (for example, O(n2) vs. O(n)).

Ans: Consider a domain in which every state has a single successor, and there is a single goal at depth n. Then depth-first search will find the goal in n steps, whereas iterative deepening search will take

1+2+3+ ⋯ + n = O(n2) steps.

3.21 Prove each of the following statements, or give a counterexample:

a. Breadth-first search is a special case of uniform-cost search.

b. Depth-first search is a special case of best-first tree search.

c. Uniform-cost search is a special case of A∗ search.

Ans:

a. When all step costs are equal, g(n) a depth(n), so uniform-cost search reproduces breadth-first search.

b. Breadth-first search is best-first search with f(n) = depth(n); depth-first search is best-first search with f(n) = −depth(n); uniform-cost search is best-first search with f(n) = g(n).

c. Uniform-cost search is A∗ search with h(n)=0.

3.27 n vehicles occupy squares $(1,1)$ through $(n,1)$ (i.e., the bottom row) of an $n \times n$ grid. The vehicles must be moved to the top row but in reverse order; so the vehicle i that starts in $(i,1)$ must end up in $(n-i+1,n)$. On each time step, every one of the n vehicles can move one square up, down, left, or right, or stay put; but if a vehicle stays put, one other adjacent vehicle (but not more than one) can hop over it. Two vehicles cannot occupy the same square.

a. Calculate the size of the state space as a function of n.

b. Calculate the branching factor as a function of n.

c. Suppose that vehicle i is at (x_i, y_i); write a nontrivial admissible heuristic h_i for the number of moves it will require to get to its goal location $(n-i+1,n)$, assuming no other vehicles are on the grid.

d. Which of the following heuristics are admissible for the problem of moving all n vehicles to their destinations? Explain.

(i) $\sum_{i=1}^{n} h_i$.
(ii) $\max\{h_1, \ldots, h_n\}$.
(iii) $\min\{h_1, \ldots, h_n\}$.

Ans:

a. n^{2n}. There are n vehicles in n^2 locations, so roughly (ignoring the one-per-square constraint) $(n^2)^n = n^{2n}$ states.

b. 5^n.

c. Manhattan distance, i.e., $|(n-i+1) - x_i| + |n - y_i|$. This is exact for a lone vehicle.

d. Only (iii) $\min\{h_1, \ldots, h_n\}$. The explanation is nontrivial as it requires two observations. First, let the *work* W in a given solution be the total *distance* moved by all vehicles over their joint trajectories; that is, for each vehicle, add the lengths of all the steps taken. We have $W \geq \sum_i h_i \geq\geq n \cdot min\{h_1, ..., h_n\}$. Second, the total work we can get done per step is $\leq n$. (Note that for every car that jumps 2, another car has to stay put (move 0), so the total work per step is bounded by n.) Hence, completing all the work requires at least $n \cdot min\{h_1, ..., h_n\}/n = min\{h_1, ..., h_n\}$ steps.

3.28 Invent a heuristic function for the 8-puzzle that sometimes overestimates, and show how it can lead to a suboptimal solution on a particular problem. (You can use a computer to help if you want.) Prove that if h never overestimates by more than c, A* using h returns a solution whose cost exceeds that of the optimal solution by no more than c.

Ans:

3.28 The heuristic $h = h_1 + h_2$ (adding misplaced tiles and Manhattan distance) sometimes overestimates. Now, suppose $h(n) \leq h^*(n) + c$ (as given) and let G_2 be a goal that is suboptimal by more than c, i.e., $g(G_2) > C^* + c$. Now consider any node n on a path to an optimal goal. We have

$$\begin{aligned} f(n) &= g(n) + h(n) \\ &\leq g(n) + h^*(n) + c \\ &\leq C^* + c \\ &\leq g(G_2) \end{aligned}$$

so G_2 will never be expanded before an optimal goal is expanded.

3.29 Prove that if a heuristic is consistent, it must be admissible. Construct an admissible heuristic that is not consistent.

Ans:

3.29 A heuristic is consistent iff, for every node n and every successor n' of n generated by any action a,

$$h(n) \leq c(n, a, n') + h(n')$$

One simple proof is by induction on the number k of nodes on the shortest path to any goal from n. For $k = 1$, let n' be the goal node; then $h(n) \leq c(n, a, n')$. For the inductive case, assume n' is on the shortest path k steps from the goal and that $h(n')$ is admissible by hypothesis; then

$$h(n) \leq c(n, a, n') + h(n') \leq c(n, a, n') + h^*(n') = h^*(n)$$

so $h(n)$ at $k + 1$ steps from the goal is also admissible.

Chapter 4:

Book Exercise: Q4.1, Q4.3, 4.13

Solution: Check Solution Manual

Question: Solve genetic algorithm for 1 generation for 8 queen problem where you need to show all the stages of genetic algorithm.

Note: Question may contain any other problem description in a similar setting.

Answer: Check class lecture

Question: You may need to solve a problem using hill climbing or simulated annealing.

Answer: Check lecture notes

Chapter 5:

Book Exercise: Q5.1, Q5.4, Q5.6, Q5.9, Q5.12, Q5.13

Solution: Check solution manual

Question: A game tree will be given and you need to perform alpha beta pruning

Answer: Check lecture notes

Chapter 6

Question: Q6.1, Q6.2, Q6.3, Q6.4 , Q6.7, Q6.8, Q6.9

Solution: Check solution manual

Question: Problem related to crypto arithmetic, graph node coloring.

Answer: Check lecture note

6.6 Show how a single ternary constraint such as “A + B = C” can be turned into three binary constraints by using an auxiliary variable. You may assume finite domains. (Hint: Consider a new variable that takes on values that are pairs of other values, and consider constraints such as “X is the first element of the pair Y .”) Next, show how constraints with more than three variables can be treated similarly. Finally, show how unary constraints can be eliminated by altering the domains of variables. This completes the demonstration that any CSP can be transformed into a CSP with only binary constraints.

Solution:

6.6 The problem statement sets out the solution fairly completely. To express the ternary constraint on A, B and C that $A + B = C$, we first introduce a new variable, AB. If the domain of A and B is the set of numbers N, then the domain of AB is the set of pairs of numbers from N, i.e. $N \times N$. Now there are three binary constraints, one between A and AB saying that the value of A must be equal to the first element of the pair-value of AB; one between B and AB saying that the value of B must equal the second element of the value of AB; and finally one that says that the sum of the pair of numbers that is the value of AB must equal the value of C. All other ternary constraints can be handled similarly.

Now that we can reduce a ternary constraint into binary constraints, we can reduce a 4-ary constraint on variables A, B, C, D by first reducing A, B, C to binary constraints as shown above, then adding back D in a ternary constraint with AB and C, and then reducing this ternary constraint to binary by introducing CD.

By induction, we can reduce any n-ary constraint to an $(n-1)$-ary constraint. We can stop at binary, because any unary constraint can be dropped, simply by moving the effects of the constraint into the domain of the variable.

Problem:

6.4 Give precise formulations for each of the following as constraint satisfaction problems:

a. Rectilinear floor-planning: find non-overlapping places in a large rectangle for a number of smaller rectangles.

b. Class scheduling: There is a fixed number of professors and classrooms, a list of classes to be offered, and a list of possible time slots for classes. Each professor has a set of classes that he or she can teach.

c. Hamiltonian tour: given a network of cities connected by roads, choose an order to visit all cities in a country without repeating any.

Solution:

6.4 **a.** For rectilinear floor-planning, one possibility is to have a variable for each of the small rectangles, with the value of each variable being a 4-tuple consisting of the x and y coordinates of the upper left and lower right corners of the place where the rectangle will be located. The domain of each variable is the set of 4-tuples that are the right size for the corresponding small rectangle and that fit within the large rectangle. Constraints say that no two rectangles can overlap; for example if the value of variable R_1 is $[0, 0, 5, 8]$, then no other variable can take on a value that overlaps with the $0, 0$ to $5, 8$ rectangle.

b. For class scheduling, one possibility is to have three variables for each class, one with times for values (e.g. MWF8:00, TuTh8:00, MWF9:00, ...), one with classrooms for values (e.g. Wheeler110, Evans330, ...) and one with instructors for values (e.g. Abelson, Bibel, Canny, ...). Constraints say that only one class can be in the same classroom at the same time, and an instructor can only teach one class at a time. There may be other constraints as well (e.g. an instructor should not have two consecutive classes).

c. For Hamiltonian tour, one possibility is to have one variable for each stop on the tour, with binary constraints requiring neighboring cities to be connected by roads, and an AllDiff constraint that all variables have a different value.

Problem:

6.2 Consider the problem of placing k knights on an n × n chessboard such that no two knights are attacking each other, where k is given and k ≤ n2.

a. Choose a CSP formulation. In your formulation, what are the variables?

b. What are the possible values of each variable?

c. What sets of variables are constrained, and how?

d. Now consider the problem of putting as many knights as possible on the board with-out any attacks. Explain how to solve this with local search by defining appropriate ACTIONS and RESULT functions and a sensible objective function.

Solution:

6.2

a. Solution A: There is a variable corresponding to each of the n^2 positions on the board.
Solution B: There is a variable corresponding to each knight.

b. Solution A: Each variable can take one of two values, {occupied,vacant}
Solution B: Each variable's domain is the set of squares.

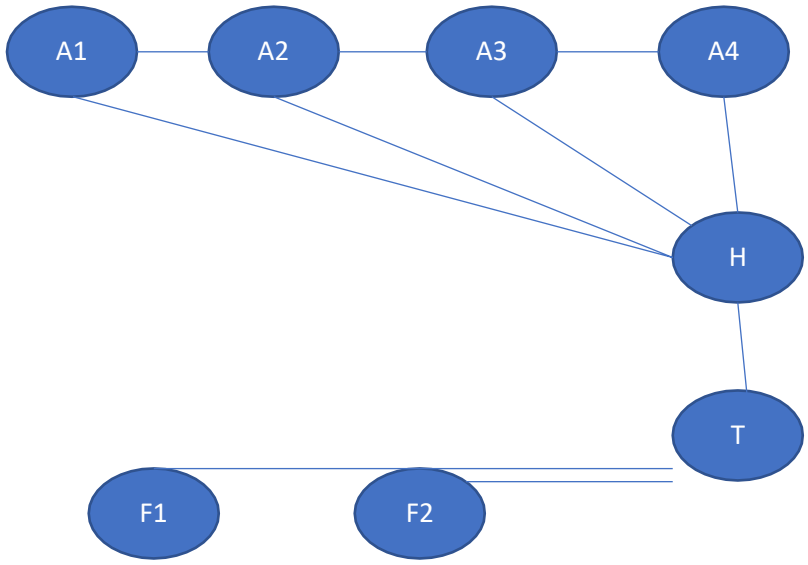
c. Solution A: every pair of squares separated by a knight's move is constrained, such that both cannot be occupied. Furthermore, the entire set of squares is constrained, such that the total number of occupied squares should be k.
Solution B: every pair of knights is constrained, such that no two knights can be on the same square or on squares separated by a knight's move. Solution B may be preferable because there is no global constraint, although Solution A has the smaller state space when k is large.

d. Any solution must describe a *complete-state* formulation because we are using a local search algorithm. For simulated annealing, the successor function must completely connect the space; for random-restart, the goal state must be reachable by hillclimbing from some initial state. Two basic classes of solutions are:
Solution C: ensure no attacks at any time. Actions are to remove any knight, add a knight in any unattacked square, or move a knight to any unattacked square.
Solution D: allow attacks but try to get rid of them. Actions are to remove any knight, add a knight in any square, or move a knight to any square.

Problem:

6.8 Consider the graph with 8 nodes A1, A2, A3, A4, H, T, F1, F2. Ai is connected to Ai+1 for all i, each Ai is connected to H, H is connected to T, and T is connected to each Fi. Find a 3-coloring of this graph by hand using the following strategy: intelligent backtracking, the variable order A1, H, A4, F1, A2, F2, A3, T, and the value order R, G, B.

Solution:



a) A1 = R.
b) H = R conflicts with A1.
c) H = G.
d) A4 = R.
e) F1 = R.
f) A2=R conflicts with A1, A2=G conflicts with H, so A2 = B
g) F2 = R
h) A3 = R conflicts with A4, A3 = G conflicts with H. A3 = B conflicts with A2. So back tack. Conflicts set is {A2, H, A4}, so jump to A2. Add {H, A4} to A2's conflict set.
i) A2 has no more values, so backtrack. Conflict set is {A1, H, A4} so jump back to A4. Add {A1,H} to A4's conflict set.
j) A4 = G conflicts with H so A4=B

k) F1=R
l) A2 = R conflict with A1, A2 = G conflicts with H, so A2 = B
m) F2=R
n) A3 = R
o) T = T conflicts with F1 and F2. T = G conflicts with G, so T = B
p) success