Melvin Mokhtari
Email: melvin.mokhtari@ec.iut.ac.ir
Course: Artificial Intelligence
Instructor: Dr. Samaneh Hosseini

Assignment 1
ID: 9831143
Semester: Winter 2023
Date: March 15, 2023

Problem 1

Solution:

Part 1

Uninformed search, also known as blind search, is a search algorithm that operates without any prior knowledge or heuristics. It explores the search space systematically without any guidance or direction. On the other hand, informed search, also known as heuristic search, uses some prior knowledge, domain-specific heuristics, or other intelligent strategies to guide the search process towards the goal.
Here are some advantages of uninformed search:

1. Simplicity: Uninformed search algorithms are usually simple and easy to implement.
2. Completeness: They are guaranteed to find a solution if one exists.
3. Memory efficiency: They require minimal memory usage, making them suitable for large search spaces.
4. Time efficiency: They can be efficient in small search spaces.
5. Applicability: They can be applied to a wide range of search problems without the need for domain-specific knowledge.

Here are some advantages of informed search:

1. Speed: Informed search algorithms can quickly converge to a solution by using domain-specific heuristics to guide the search.
2. Efficiency: They can save time and resources by avoiding unnecessary exploration of unpromising parts of the search space.
3. Accuracy: They can provide more accurate solutions than uninformed search algorithms.
4. Domain-specificity: They can be customized to exploit domain-specific knowledge and heuristics to solve problems more efficiently.
5. Admissibility: They can guarantee that the solution found is optimal or near-optimal, depending on the quality of the heuristic function used.

Part 2

A good algorithm for solving the TSP is the Held-Karp algorithm, also known as the Bellman-Held-Karp algorithm. It is a dynamic programming algorithm that breaks down the problem into subproblems, computing the shortest path between all pairs of cities that include the starting city and one other city, and then using this information to compute the shortest path that visits each subset of cities exactly once and returns to the starting city. The Held-Karp algorithm is considered the best algorithm for the TSP because it guarantees to find the optimal solution and has a time complexity of $O(n^2 * 2^n)$, which is an improvement over the brute-force algorithm.

Another popular algorithm for solving the TSP is the A* algorithm, which is an informed search algorithm. It uses a heuristic function in addition to the cost function to determine the next node to explore during the search. The heuristic function estimates the distance from the current node to the goal node, allowing the algorithm to prioritize exploring nodes that are closer to the goal. A* algorithm can be more efficient than other informed search algorithms in terms of time and memory usage, but the quality of the solution depends on the quality of the heuristic function.

When it comes to solving the TSP, selecting the most suitable algorithm is dependent on the specific problem instance and the desired level of optimality. In my opinion, for this particular problem, the Bellman-Held-Karp algorithm appears to be more advanced and well-suited. Thus, I would choose the Held-Karp algorithm as the preferred solution method for the reasons I have previously mentioned! :))

Problem 2

Solution:

Part 1

In this problem, as we can see in the provided figure, we have a graph with six nodes named A, B, C, D, E, and F, and we want to use only three colors—Red, Green, and Blue—to color the nodes.
The variables in this problem are the vertices of the graph, and the domain of each variable is the set of available colors. The constraints in this problem state that no two adjacent vertices can have the same color. We can define the problem as follows:

- Variables: {A, B, C, D, E, F}
- Domain: {Red, Green, Blue}
- Constraints: For any pair of adjacent nodes (u, v), where u and v are adjacent nodes in the graph, u and v cannot have the same color.

One possible solution would be:

- A: Red
- B: Green
- C: Blue
- D: Green
- E: Blue
- F: Red

Part 2

The constraints in graph coloring CSPs are binary because they relate to pairs of nodes in the graph and are based on the requirement that no two adjacent nodes can be assigned the same color.

Part 3

1. Arc consistency: In this test, we check if every value in a variable's domain is consistent with the domains of its neighbors. In this problem, we can see that all arcs are already consistent, and no values can be removed from any domain.
2. Path consistency: In this test, we extend arc consistency by considering longer paths of variables. Here we can see that all paths of length two are already consistent, and no values can be removed from any domain.
3. 4 consistency: If a graph can be colored with only three colors, it is considered to be 3-colorable and consequently 4-consistent. This is because any valid 4-coloring of the graph will also serve as a valid 3-coloring, ensuring that each variable has at least one value in its domain that can be part of a solution to the problem.

Part 4

No, it is possible to have a CSP that is (k+1)-consistent but not k-consistent. For example, consider a CSP with two variables (A and B) and domains 1, 2, and 3. The constraint between A and B is that A+B is even. This CSP is 2-consistent because any pair of variables that share a constraint have at least two values that satisfy the constraint. However, it is not 1-consistent because the only value of A that satisfies the constraint with B=1 is A=2, and the only value of A that satisfies the constraint with B=3 is A=1. Therefore, this CSP is not 1-consistent but is 2-consistent.

Problem 3

Solution: This problem can be solved using backtracking or constraint propagation algorithms. So to solve this as a CSP, we need to define the variables, domains, and constraints:

- Variables: {B, A, S, E, L, G, M, C1, C2, C3}
- Domains: {0, 1} for {G, C1, C2, C3} and {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} for others
- Constraints:

  1. All variables (except C1, C2, and C3) must have different values.
  2. E + L = S + C1 * 10
  3. S + L + C1 = E + C2 * 10
  4. A + A + C2 = M + C3 * 10
  5. B + B + C3 = A + G * 10
  6. $B \neq 0$ (since B can't be zero in a four-digit number)

Using these constraints, we can understand that this problem has binary and unary constraints, as well as global constraints (e.g., all-different constraint). The use of these constraints helps to narrow down the possible solutions, which reduces the search space and speeds up the solving process.
One possible solution for this cryptarithmetic problem as a CSP is:

- B = 7
- A = 4
- S = 8
- E = 3
- L = 5
- G = 1
- M = 9

This solution satisfies all of the constraints and makes the equation true: 7483 + 7455 = 14938.

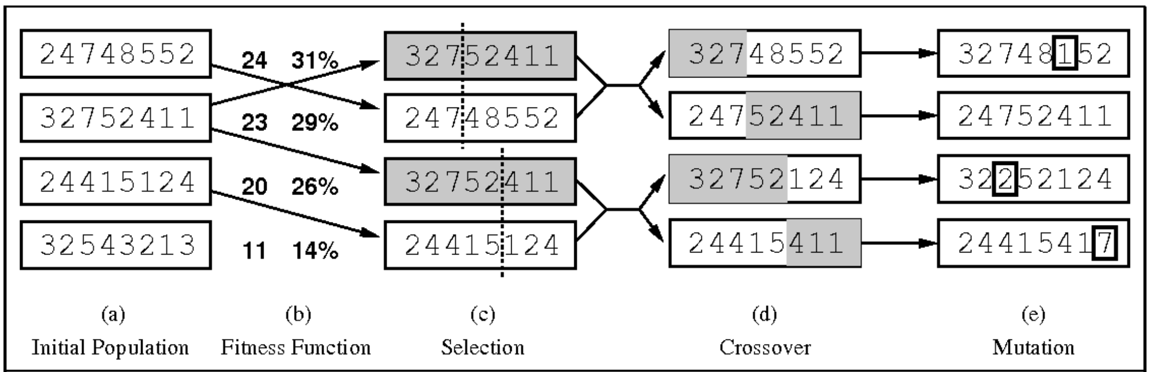
Problem 4

Solution: This question is aimed at describing the steps of solving an 8-Queens problem using the genetic algorithm theoretically. So the steps could be as follows:

1. Initialize the population: Generate an initial population of random solutions, each representing a placement of the eight queens on the chessboard. For example, one possible initial could be [0, 1, 2, 3, 4, 5, 6, 7], where each number represents the row position of a queen in the corresponding column.
2. Evaluate the fitness: Evaluate the fitness of each individual in the population using a fitness function. In the case of the 8 Queens problem, the fitness function would assign a high fitness score to solutions that satisfy the constraints of the problem (i.e., no two queens can be placed in the same row, column, or diagonal) and a low fitness score to solutions that violate the constraints.
3. Selection: Select the parents for the next generation. One common selection method is tournament selection, where a random subset of the population is selected and the individual with the highest fitness is chosen as a parent. We will repeat this process to select multiple parents.
4. Crossover: Apply the crossover operator to create new offspring. One common crossover operator for the 8 Queens problem is the uniform crossover, where each gene is randomly selected from either parent with equal probability.
5. Mutation: Apply the mutation operator to introduce new genetic variation into the offspring. One common mutation operator for the 8 Queens problem is the swap mutation, where two randomly selected genes in the chromosome are swapped.
6. Evaluate the fitness: Evaluate the fitness of the new offspring using the fitness function.
7. Elitism: Select the best individuals from the current generation to carry over to the next generation unchanged. This helps prevent the loss of the best solutions found so far.
8. Repeat steps 3–7 for a specified number of generations or until a satisfactory solution is found.

By repeating these steps for a specified number of generations, the genetic algorithm can converge on a solution that satisfies the constraints of the 8-Queens problem. After running the genetic algorithm for 100 generations, for example, one of the best possible solutions might be [3, 5, 7, 1, 6, 0, 2, 4] as shown in the question, which has a fitness value of 28. There are other solutions as well, including [0, 4, 7, 5, 2, 6, 1], [0, 6, 3, 5, 7, 1, 4], [1, 3, 5, 7, 2, 0, 6], etc.

- The figure below illustrates the primary stages of this solution for an example 8 queen problem.



The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).