Yeditepe University

CSE480/591 Special Topics: Optimization with Metaheuristics

Quiz 2

Date: 7 November 2025 11:20 - 11:50 (30 mins)

Policy:

- No books, notes, or unauthorized materials are allowed.
- Any form of cheating or misconduct will be reported for disciplinary action.
- The quiz has two options. Choose one and answer only that one. Circle the option you will solve.

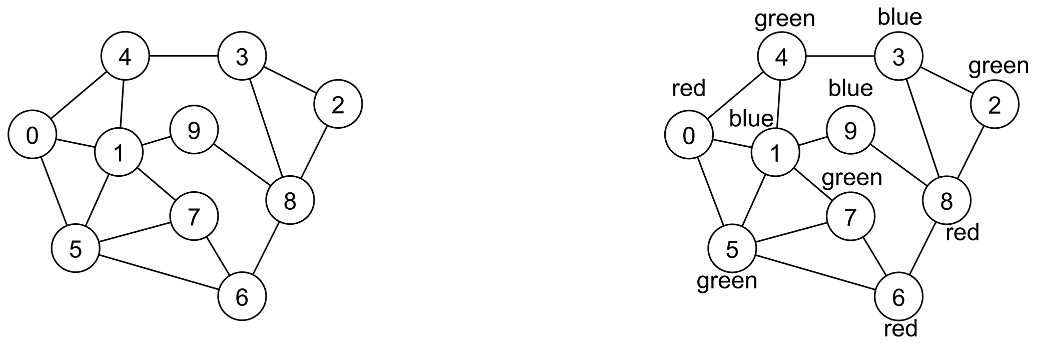
Question A (100 points) A Local Search Algorithm for Graph Coloring Problem

Definition of Graph Coloring Problem. For a graph $G = (V, E)$, a function

$$c : V \to \{1, 2, \dots, k\}$$

assigns a color to each vertex $v \in V$ such that for every edge $(u, v) \in E$, the condition $c(u) \neq c(v)$ holds. The smallest value of k for which this condition is satisfied is called the *chromatic number* of the graph, denoted by $\chi(G)$.

In this question, assume $k = 3$ is fixed. The sample graph G with 10 vertices is shown in Figure 1a.



(a) Sample graph G (b) Initial random coloring with 3 colors

Figure 1: Graph Coloring Example for Question A

Assume that an initial random coloring is given using three colors — **red = 1**, **blue = 2**, and **green = 3** — as shown in Figure 1b.

Let the cost function $f(S)$ measure the number of *conflicting edges* in a given solution S, where a conflict occurs when two adjacent vertices share the same color.
For example, the cost of the coloring in Figure 1b is $f(S) = 3$, since vertices (1, 9), (5, 7), and (6, 8) have the same color.

Your task is to design a **stochastic hill climbing algorithm** to reach an optimal solution satisfying $f(S) = 0$ when $k = 3$.

a. [**20 pts**] **Representation:** Describe how an individual (solution) is represented. Explain briefly why this representation is suitable for the problem.

Each solution (individual) is represented by a color vector:

$$S = [c(v_1), c(v_2), \ldots, c(v_{10})], \quad c(v_i) \in \{1, 2, 3\}.$$

This representation is simple and efficient since each position represents a vertex, and colors can be changed directly to explore the neighborhood. For example, the initial solution in Figure 1b can be represented as :

$$S_{initial} = [1, 2, 3, 2, 3, 3, 1, 3, 1, 2]$$

b. [**10 pts**] **Cost Function:** Formulate the cost function $f(S)$. Using the given example, compute $f(S)$ for the initial coloring to verify its correctness.

The cost is defined as the number of conflicting edges:

$$f(S) = |\{(u, v) \in E : c(u) = c(v)\}|.$$

In the initial random coloring (Figure 1b), there are conflicts between (1,9), (5,7), and (6,8), giving:

$$f(S_{initial}) = 3.$$

c. [**40 pts**] **Algorithm Design:** Write the pseudocode of a *Stochastic Hill Climbing* algorithm that uses a neighborhood operator and employs a stopping criterion of **4 iterations**.

The neighborhood operator selects one vertex at random and recolors it with a different color chosen from the remaining $k - 1$ colors. This produces a new solution that differs from the current one by the color assignment of a single vertex:

$$S' = S \text{ with } c'(v_i) \neq c(v_i), \quad v_i \in V.$$

Alternative 2 for neighborhood operator: swap colors of two randomly selected vertices

$$S' = S \text{ with } c'(v_i) = c(v_j) \text{ and } c'(v_j) = c(v_i), \quad v_i, v_j \in V, \text{ and } i \neq j.$$

```
Input: Graph G=(V,E), #colors k=3
Output: Best coloring found

1. S ← RandomColoring(V, k)
2. best ← S
3. For iter = 1 to 4 do
4.     v ← random vertex from V
5.     old_color ← S[v]
6.     new_color ← random color from {1,2,3} \ {old_color}
```

```
7.      S' ← S with S'[v] = new_color
8.      If f(S') <= f(S) then
9.          S ← S'
10.         If f(S) < f(best) then best ← S
11.      Else
12.          Reject (keep S)
13. Return best
```

d. **[15 pts] Execution:** Apply your algorithm for four iterations and show the solution, cost, and accepted or unaccepted move at each iteration. Plot the convergence of the algorithm on cost versus iterations.
Assume that we obtain the initial solution in Figure 1b with RandomColor at Step 1 of the algorithm.

$$S = [1, 2, 3, 2, 3, 3, 1, 3, 1, 2]$$

- Iteration 1: Select vertex 9 at step 4 – $old_color = 2$ (Step 5) – $new_color = 3$ (Step 6) – update S[9] as 3 (Step 7) – Since f(S') = 2, both S and best are updated with S'.
- Iteration 2:

$$S = [1, 2, 3, 2, 3, 3, 1, 3, 1, 3]$$

  Select vertex 6 randomly – old color = 1 (Step 5) – new color = 2 (Step 6) – update S[6] as 2 (Step 7) – Since f(S') = 1, both S and best are updated with S'.
- Iteration 3:

$$S = [1, 2, 3, 2, 3, 3, 2, 3, 1, 3]$$

  Select vertex 7 randomly – old color = 3 (Step 5) – new color = 1 (Step 6) – update S[6] as 2 (Step 7) – Since f(S') = 0, both S and best are updated with S'.
- Iteration 4: We reached the stopping criteria with the best solution so we do not need to run the 4th iteration.

e. **[15 pts] Analysis:** Discuss whether it is possible to reach the optimal solution ($f(S) = 0$) with your designed algorithm in (c). Explain step by step under which conditions this can occur.
It is possible to reach the optimal solution if we select the vertices 9, 6 and 7 to recolor with the colors 3 (green), 2 (blue), and 1 (red), respectively.

Question B (100 points) Simulated Annealing Design for Shoe Factory JSSP

Consider the following job shop scheduling problem:
We have three jobs (shoe types) to be processed on three machines:

$$M_0 = \text{Cutting}, \quad M_1 = \text{Stitching}, \quad M_2 = \text{Finishing}.$$

Processing routes and times (in minutes) are:

$$\begin{array}{llllll} \textbf{Sneakers}\ (J_1): & M_0\,(10) & \rightarrow & M_2\,(5) & \rightarrow & M_1\,(15) \\ \textbf{Boots}\ (J_2): & M_1\,(12) & \rightarrow & M_0\,(8) & \rightarrow & M_2\,(14) \\ \textbf{Loafers}\ (J_3): & M_2\,(9) & \rightarrow & M_1\,(7) & \rightarrow & M_0\,(11) \end{array}$$

Each machine can process at most one operation at a time, jobs are non-preemptive, and the objective is to **minimize the makespan**. Therefore, the cost function $f(S)$ should return the makespan of schedule S. Using this same problem instance:

(a) **[15 pts] Individual Representation:** Propose a schedule (solution) representation that can be used in a simulated annealing (SA) algorithm for this job shop problem. Briefly justify why this representation is suitable for the problem. Use a *job-based operation list*, where each job appears m times in the list (here $m = 3$) corresponding to the number of machines. Each position in the list specifies the next operation of the corresponding job to be scheduled. For this problem:

$$\text{Representation: } [J_1, J_2, J_3, J_1, J_2, J_3, J_1, J_2, J_3]$$

This is suitable for Simulated Annealing since swapping two job entries gives a valid neighboring schedule while maintaining job precedence.

(b) **[15 pts] Initial Solution (in your representation):** Give one feasible initial individual *using exactly the representation you defined in part (a).* Make sure your solution respects the job routes given above.

A feasible initial individual can be generated by taking the job-based list in its natural order:

$$S_0 = [J_1, J_2, J_3, J_1, J_2, J_3, J_1, J_2, J_3].$$

Decoding this list produces the following schedule:

Machine	Operations (start–finish)
M_0	$J_1(0-10),\ J_2(12-20),\ J_3(20-31)$
M_1	$J_2(0-12),\ J_3(12-19),\ J_1(19-34)$
M_2	$J_3(0-9),\ J_1(10-15),\ J_2(20-34)$

$$C_{J_1} = 34, \quad C_{J_2} = 34, \quad C_{J_3} = 31 \Rightarrow f(S_0) = 34.$$

(c) **[40 pts]** Using the initial temperature $T_0 = 50$, design **Simulated Annealing Algorithm** with a neighborhood operator that is compatible with your representation in (a) and write its pseudocode. Define the acceptance rule that determines whether a new solution is accepted. Give a temperature update rule (e.g., $T \leftarrow \alpha T$) so that the algorithm terminates after 4 iterations (temperature levels).

Parameters:

$$T_0 = 50, \quad \alpha = 0.5, \quad \text{iterations} = 4.$$

Neighborhood Operator: randomly swap two job entries in the operation list. This produces a new permutation, which is decoded into a feasible schedule, and its makespan is evaluated as the new cost $f(S')$. **Acceptance Rule:**

$$\text{If } f(S') \leq f(S) \text{ accept; else accept with probability } T.$$

Cooling Schedule:

$$T \leftarrow \alpha T \quad \text{with } \alpha = 0.5.$$

Note: You do not need to calculate a probability value. Just represent it symbollically (prob) and compare with T temperature value. **Pseudocode:**

```
S ← initial operation list (S0)
best ← S
T ← 50
for level = 1 to 4 do
    generate neighbor S' by swapping two random jobs in S
    diff ← f(S') - f(S)
    if diff  0 then
        S ← S'
        if f(S) < f(best) then best ← S
    else
        prob ← U(0,100)
        if prob < T then S ← S' //acceptance prob.
    T ← 0.5 * T
return best
```

(d) **[30 pts]** Using your initial solution from part (b) and your SA setup from part (c), show how SA would proceed for 4 iterations: state the current cost, generate a neighbor and its cost, and decide accept/reject according to the acceptance rule. If computing exact makespans is long, you may indicate symbolically. Plot the algorithm's convergence.

Iter	T	$f(S)$	Action
0	50.00	34	Initial schedule
1	50.00	35	Accepted (probability compared to T)
2	25.00	33	Accepted (better)
3	12.50	33	Accepted (equal)
4	6.25	34	Rejected or accepted with probability

Convergence: $34 \to 35 \to 33 \to 33 \to 33$

The algorithm begins with broad exploration at high T, accepting some worse solutions, and gradually stabilizes as T decreases. After 4 iterations, it converges near $f(S) \approx 33$, close to the optimal makespan.