Optimization with Metaheuristics

CSE480 & CSE591- Week 5 - Simulated Annealing Example & Tabu Search

Asst. Prof. Gizem Süngü Terci

October 30, 2025

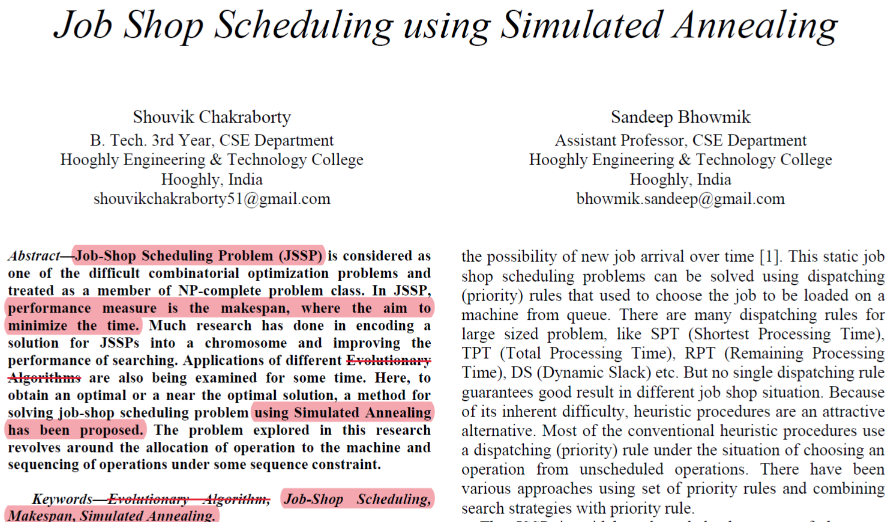
YEDİTEPE UNIVERSITY

Finding an Example Paper for Simulated Annealing

1. After learning how Simulated Annealing works, we want to see its performance on real problems.
2. We pick a familiar problem: **Job Shop Scheduling Problem (JSSP)**.
3. Go to **Google Scholar** to perform a literature search.
4. Use keywords: *"Job Shop Scheduling Problem" AND "Simulated Annealing"*.
5. One relevant paper we identify:
   - **Chakraborty & Bhowmik (2013)**
     Job shop scheduling using simulated annealing
     Presented at IC3A Conference.
6. It is a peer-reviewed conference paper and has been cited by many related studies, which indicates its relevance and recognition in the field.

Reference: Chakraborty, S. and Bhowmik, S., 2013, January. Job shop scheduling using simulated annealing. In First International Conference on Computation and Communication Advancement (Vol. 1, No. 1, pp. 69-73).

Example for Simulated Annealing



Job Shop Scheduling using Simulated Annealing

Shouvik Chakraborty
B. Tech. 3rd Year, CSE Department
Hooghly Engineering & Technology College
Hooghly, India
shouvikchakraborty51@gmail.com

Sandeep Bhowmik
Assistant Professor, CSE Department
Hooghly Engineering & Technology College
Hooghly, India
bhowmik.sandeep@gmail.com

***Abstract*—Job-Shop Scheduling Problem (JSSP) is considered as one of the difficult combinatorial optimization problems and treated as a member of NP-complete problem class. In JSSP, performance measure is the makespan, where the aim to minimize the time. Much research has done in encoding a solution for JSSPs into a chromosome and improving the performance of searching. Applications of different ~~Evolutionary Algorithms~~ are also being examined for some time. Here, to obtain an optimal or a near the optimal solution, a method for solving job-shop scheduling problem using Simulated Annealing has been proposed. The problem explored in this research revolves around the allocation of operation to the machine and sequencing of operations under some sequence constraint.**

***Keywords*—~~Evolutionary Algorithm,~~ Job-Shop Scheduling, Makespan, Simulated Annealing.**

the possibility of new job arrival over time [1]. This static job shop scheduling problems can be solved using dispatching (priority) rules that used to choose the job to be loaded on a machine from queue. There are many dispatching rules for large sized problem, like SPT (Shortest Processing Time), TPT (Total Processing Time), RPT (Remaining Processing Time), DS (Dynamic Slack) etc. But no single dispatching rule guarantees good result in different job shop situation. Because of its inherent difficulty, heuristic procedures are an attractive alternative. Most of the conventional heuristic procedures use a dispatching (priority) rule under the situation of choosing an operation from unscheduled operations. There have been various approaches using set of priority rules and combining search strategies with priority rule.

Components of the Paper

1. **Problem & class:** Job-Shop Scheduling Problem (JSSP)
2. **Objective/metric:** Minimize **makespan** (completion time of last job).
3. **Method: Simulated Annealing** (SA) is proposed to solve JSSP.
4. **Scope/assumptions:** Non-preemptive operations; one machine per job at a time; fixed job routes; precedence constraints. Not known yet.
5. **Keywords:** Evolutionary algorithm (? out of scope); JSSP; makespan; SA.

What Does the Introduction Include?

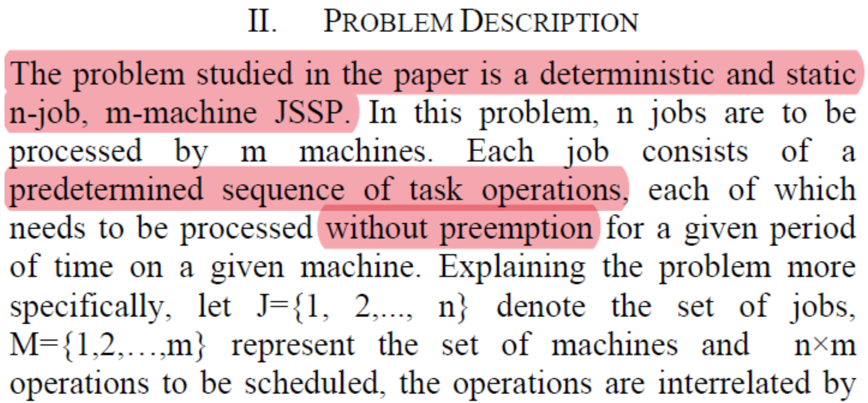
Purpose of Introduction: Explain why JSSP is difficult and why a metaheuristic (SA) is needed.

Key Points from the Paper:

- JSSP: schedule n jobs on m machines.
- Objective: minimize the makespan — total completion time of all jobs.
- Hardness: classified as **NP-complete**; search space grows exponentially with jobs/machines.
- Classical dispatching rules (SPT, TPT, RPT, DS) **do not guarantee good results** in all cases.
- Heuristics and metaheuristics are attractive due to large, constrained search space.
- Research gap: need an efficient method to handle local minima and search effectiveness.

Motivation (their claim):
Simulated Annealing can overcome local minima and improve scheduling effectiveness within reasonable time.

Problem Definition



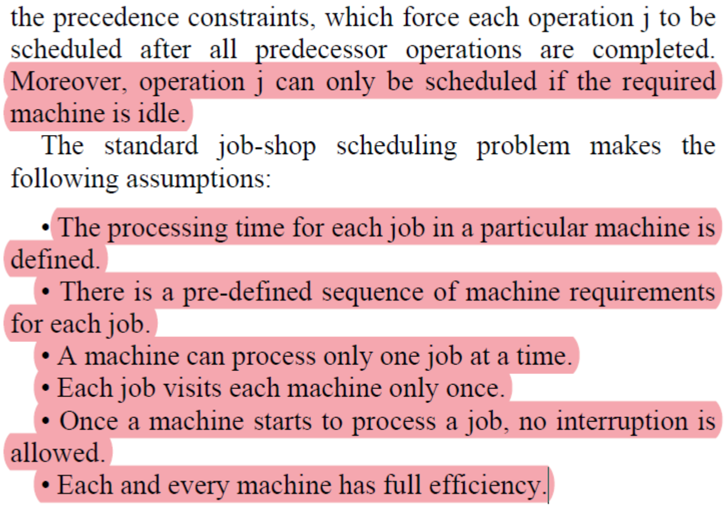
II. PROBLEM DESCRIPTION

The problem studied in the paper is a deterministic and static n-job, m-machine JSSP. In this problem, n jobs are to be processed by m machines. Each job consists of a predetermined sequence of task operations, each of which needs to be processed without preemption for a given period of time on a given machine. Explaining the problem more specifically, let J={1, 2,..., n} denote the set of jobs, M={1,2,…,m} represent the set of machines and n×m operations to be scheduled, the operations are interrelated by

Key Concepts in JSSP Problem Description

- **Static:** All jobs are available at time 0; no new jobs arrive later.
- **Deterministic:** Processing times are fully known in advance.
- **n-job, m-machine:** n jobs must be processed using m machines.
- **Predetermined operation sequence:** Each job has a fixed order of machines (routing constraints).
- **Without preemption:** Once started on a machine, a job must finish before being moved.

Constraints in Problem Definition



the precedence constraints, which force each operation j to be scheduled after all predecessor operations are completed. Moreover, operation j can only be scheduled if the required machine is idle.

The standard job-shop scheduling problem makes the following assumptions:

- The processing time for each job in a particular machine is defined.
- There is a pre-defined sequence of machine requirements for each job.
- A machine can process only one job at a time.
- Each job visits each machine only once.
- Once a machine starts to process a job, no interruption is allowed.
- Each and every machine has full efficiency.

Problem Instance

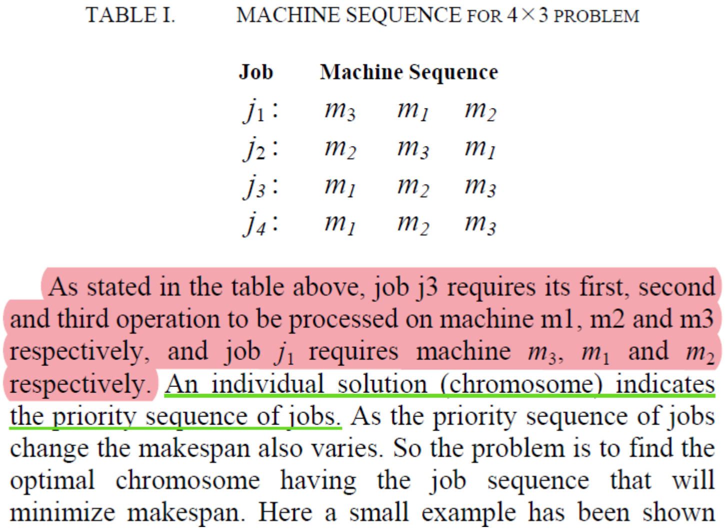


TABLE I. MACHINE SEQUENCE FOR 4×3 PROBLEM

Job	**Machine Sequence**		
j_1 :	m_3	m_1	m_2
j_2 :	m_2	m_3	m_1
j_3 :	m_1	m_2	m_3
j_4 :	m_1	m_2	m_3

As stated in the table above, job j3 requires its first, second and third operation to be processed on machine m1, m2 and m3 respectively, and job j_1 requires machine m_3, m_1 and m_2 respectively. An individual solution (chromosome) indicates the priority sequence of jobs. As the priority sequence of jobs change the makespan also varies. So the problem is to find the optimal chromosome having the job sequence that will minimize makespan. Here a small example has been shown

Problem Instance

- When a problem instance is given, a description is needed by text. (Most of mistakes in Quiz 1 is about it.)
- What Does "Individual" Mean in metaheuristics?
  - In metaheuristics, an **individual** = one **candidate solution**.
  - Here, each individual is a **chromosome** representing a schedule.
  - Encoding: **job priority sequence** (e.g., $[j_2, j_1, j_3, j_4]$)
  - Different individuals $\Rightarrow$ different schedules $\Rightarrow$ different makespan values.

Goal: Find the individual with the **minimum makespan**.

Key Concepts of Solutions in Metaheuristics

- **Search Space:** The set of all possible configurations the algorithm may explore (valid + invalid solutions).

$$\mathcal{X} = \{x_1, x_2, ..., x_N\}$$

- **Solution Space:** The subset of the search space containing only **feasible** solutions.

$$\mathcal{F} \subseteq \mathcal{X}$$

- **Individual/Chromosome (Candidate Solution):** One specific configuration of decision variables. Represented by a data structure encoding the solution.

$$x \in \mathcal{X}$$

- **Neighborhood:** Set of solutions obtainable by a small change to the current individual.

$$N(x) \subseteq \mathcal{X}$$

We evaluate one **individual** at a time, move through its **neighborhood**, but aim to find the best solution in the **feasible solution space**.

Individual 1 in Problem Definition

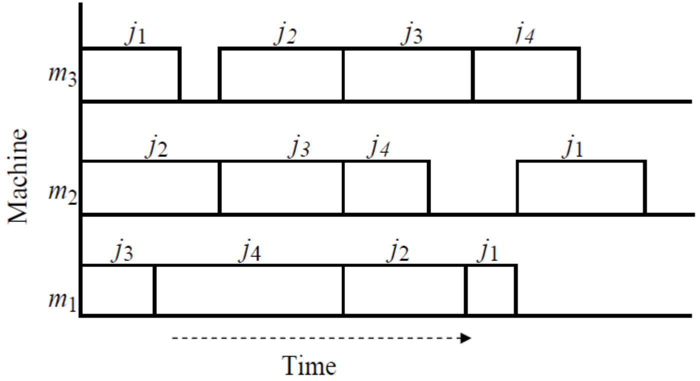


Figure 1. Gantt chart while building the schedule for a 4×3 job-shop scheduling problem. The ‘x’ axis represe nts execution time and ‘y’ axis represents machine. The priority sequence considered here is { j2 , j1 , j3, j4 }.

Individual 2 in Problem Definition

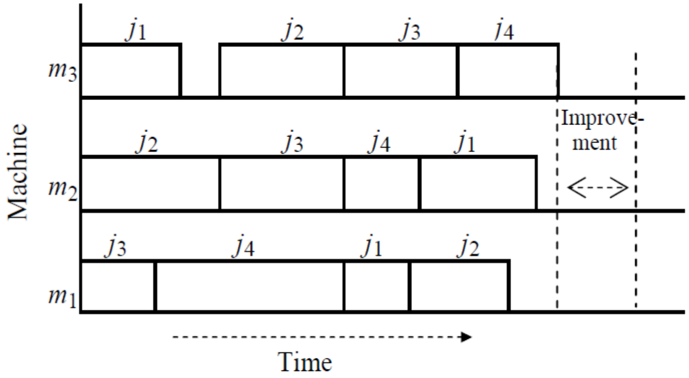
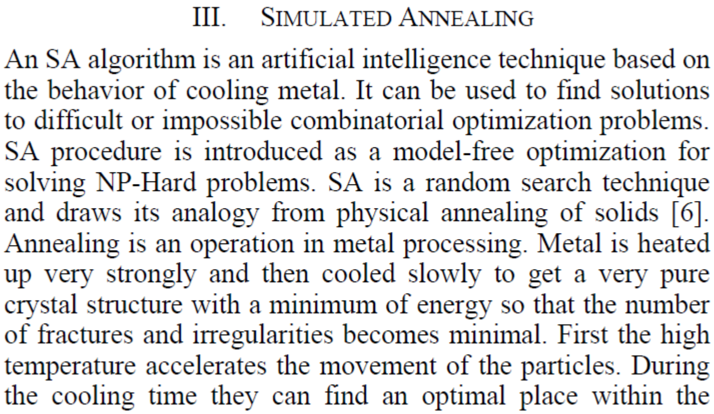


Figure 2. Gantt chart while building the schedule for a 4×3 job-shop scheduling problem. The 'x' axis represents execution time and 'y' axis represents machine. The priority sequence considered here is { j1 , j2 , j3, j4 }.

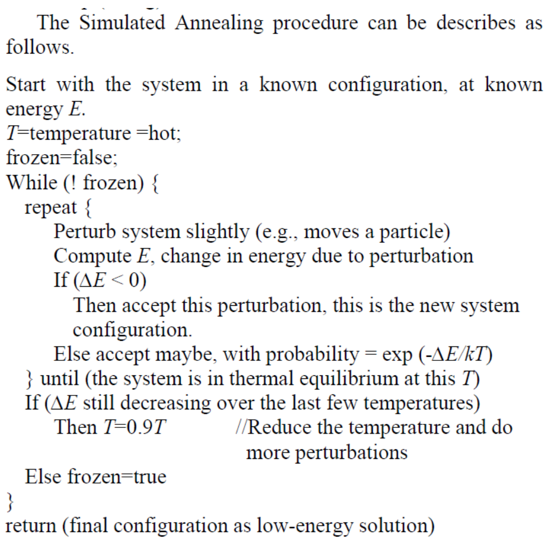
Definition of Simulated Annealing



III. SIMULATED ANNEALING

An SA algorithm is an artificial intelligence technique based on the behavior of cooling metal. It can be used to find solutions to difficult or impossible combinatorial optimization problems. SA procedure is introduced as a model-free optimization for solving NP-Hard problems. SA is a random search technique and draws its analogy from physical annealing of solids [6]. Annealing is an operation in metal processing. Metal is heated up very strongly and then cooled slowly to get a very pure crystal structure with a minimum of energy so that the number of fractures and irregularities becomes minimal. First the high temperature accelerates the movement of the particles. During the cooling time they can find an optimal place within the

Simulated Annealing



The Simulated Annealing procedure can be describes as follows.

```
Start with the system in a known configuration, at known
energy $E$.
$T$=temperature =hot;
frozen=false;
While (! frozen) {
   repeat {
        Perturb system slightly (e.g., moves a particle)
        Compute $E$, change in energy due to perturbation
        If ($\Delta E < 0$)
           Then accept this perturbation, this is the new system
           configuration.
        Else accept maybe, with probability = exp ($-\Delta E/kT$)
   } until (the system is in thermal equilibrium at this $T$)
   If ($\Delta E$ still decreasing over the last few temperatures)
        Then $T$=0.9$T$          //Reduce the temperature and do
                                  more perturbations
   Else frozen=true
}
return (final configuration as low-energy solution)
```

Simulated Annealing

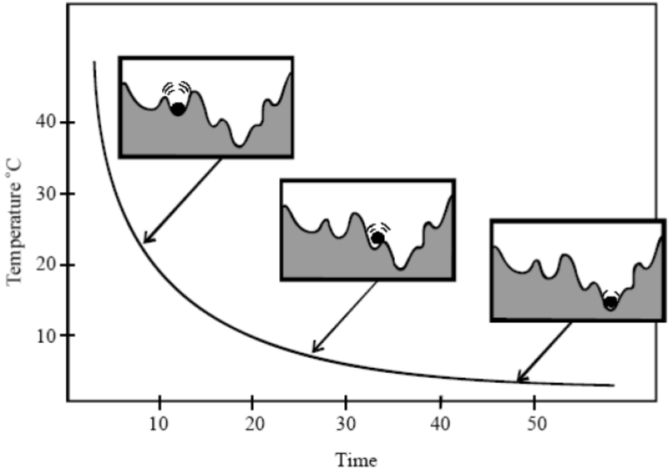
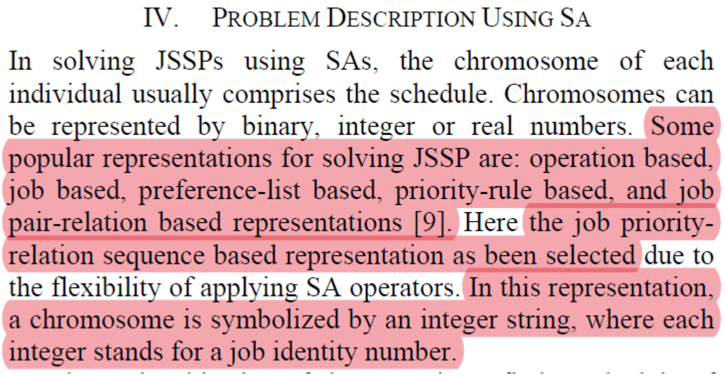


Figure 3. The method of Simulated Annealing.

JSSP Definition using SA



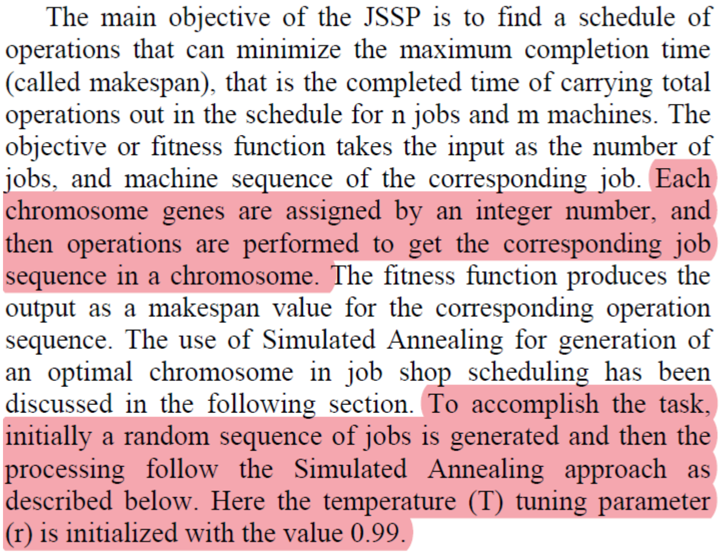
IV. Problem Description Using SA

In solving JSSPs using SAs, the chromosome of each individual usually comprises the schedule. Chromosomes can be represented by binary, integer or real numbers. Some popular representations for solving JSSP are: operation based, job based, preference-list based, priority-rule based, and job pair-relation based representations [9]. Here the job priority-relation sequence based representation as been selected due to the flexibility of applying SA operators. In this representation, a chromosome is symbolized by an integer string, where each integer stands for a job identity number.

Popular Representations for JSSP

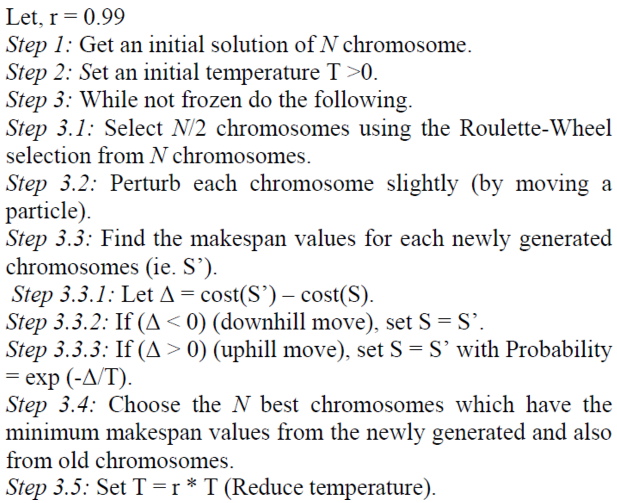
- **Operation-based:** Sequence of all $n \cdot m$ operations.
  Example: $[J2, J3, J1, J2, J3, J1, J3, J1, J2]$
- **Job-priority based:** Each job appears once (priority order).
  Example: $[J1, J3, J2]$
- **Preference-list based:** Jobs have machine preference rankings.
  Example: $J1 : [M2 > M3 > M1]$
- **Priority-rule based:** A dispatching rule selects jobs dynamically.
  Example: Rule = SPT (Shortest Processing Time first)
- **Job-pair-relation based:** Pairwise ordering constraints.
  Example: $(J1 \prec J2), (J3 \prec J1)$

JSSP Definition using SA



The main objective of the JSSP is to find a schedule of operations that can minimize the maximum completion time (called makespan), that is the completed time of carrying total operations out in the schedule for n jobs and m machines. The objective or fitness function takes the input as the number of jobs, and machine sequence of the corresponding job. Each chromosome genes are assigned by an integer number, and then operations are performed to get the corresponding job sequence in a chromosome. The fitness function produces the output as a makespan value for the corresponding operation sequence. The use of Simulated Annealing for generation of an optimal chromosome in job shop scheduling has been discussed in the following section. To accomplish the task, initially a random sequence of jobs is generated and then the processing follow the Simulated Annealing approach as described below. Here the temperature (T) tuning parameter (r) is initialized with the value 0.99.

JSSP Definition using SA



Let, r = 0.99

Step 1: Get an initial solution of *N* chromosome.

Step 2: Set an initial temperature T >0.

Step 3: While not frozen do the following.

Step 3.1: Select *N*/2 chromosomes using the Roulette-Wheel selection from *N* chromosomes.

Step 3.2: Perturb each chromosome slightly (by moving a particle).

Step 3.3: Find the makespan values for each newly generated chromosomes (ie. S').

Step 3.3.1: Let Δ = cost(S') – cost(S).

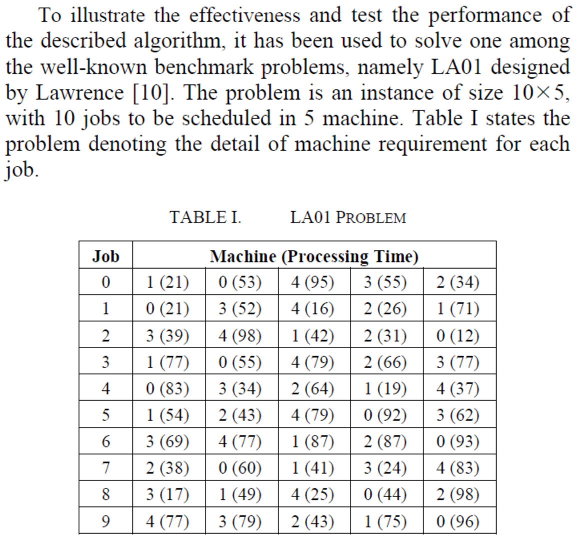
Step 3.3.2: If (Δ < 0) (downhill move), set S = S'.

Step 3.3.3: If (Δ > 0) (uphill move), set S = S' with Probability = exp (-Δ/T).

Step 3.4: Choose the *N* best chromosomes which have the minimum makespan values from the newly generated and also from old chromosomes.

Step 3.5: Set T = r * T (Reduce temperature).

Problem Instance LA01

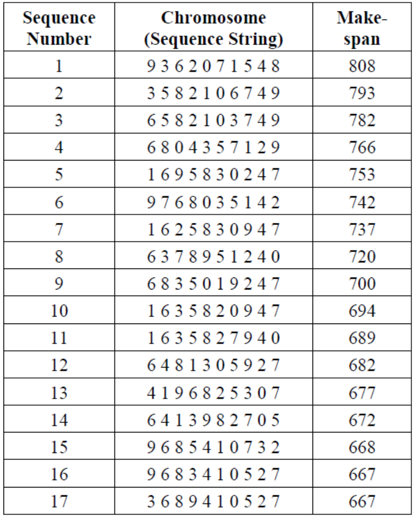


To illustrate the effectiveness and test the performance of the described algorithm, it has been used to solve one among the well-known benchmark problems, namely LA01 designed by Lawrence [10]. The problem is an instance of size 10×5, with 10 jobs to be scheduled in 5 machine. Table I states the problem denoting the detail of machine requirement for each job.

TABLE I. LA01 PROBLEM

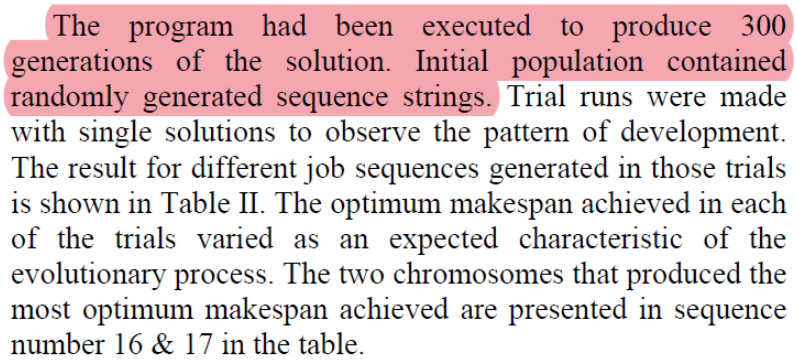
Job	**Machine (Processing Time)**				
0	1 (21)	0 (53)	4 (95)	3 (55)	2 (34)
1	0 (21)	3 (52)	4 (16)	2 (26)	1 (71)
2	3 (39)	4 (98)	1 (42)	2 (31)	0 (12)
3	1 (77)	0 (55)	4 (79)	2 (66)	3 (77)
4	0 (83)	3 (34)	2 (64)	1 (19)	4 (37)
5	1 (54)	2 (43)	4 (79)	0 (92)	3 (62)
6	3 (69)	4 (77)	1 (87)	2 (87)	0 (93)
7	2 (38)	0 (60)	1 (41)	3 (24)	4 (83)
8	3 (17)	1 (49)	4 (25)	0 (44)	2 (98)
9	4 (77)	3 (79)	2 (43)	1 (75)	0 (96)

Experimental Results and Discussions



Sequence Number	Chromosome (Sequence String)	Make-span
1	9 3 6 2 0 7 1 5 4 8	808
2	3 5 8 2 1 0 6 7 4 9	793
3	6 5 8 2 1 0 3 7 4 9	782
4	6 8 0 4 3 5 7 1 2 9	766
5	1 6 9 5 8 3 0 2 4 7	753
6	9 7 6 8 0 3 5 1 4 2	742
7	1 6 2 5 8 3 0 9 4 7	737
8	6 3 7 8 9 5 1 2 4 0	720
9	6 8 3 5 0 1 9 2 4 7	700
10	1 6 3 5 8 2 0 9 4 7	694
11	1 6 3 5 8 2 7 9 4 0	689
12	6 4 8 1 3 0 5 9 2 7	682
13	4 1 9 6 8 2 5 3 0 7	677
14	6 4 1 3 9 8 2 7 0 5	672
15	9 6 8 5 4 1 0 7 3 2	668
16	9 6 8 3 4 1 0 5 2 7	667
17	3 6 8 9 4 1 0 5 2 7	667

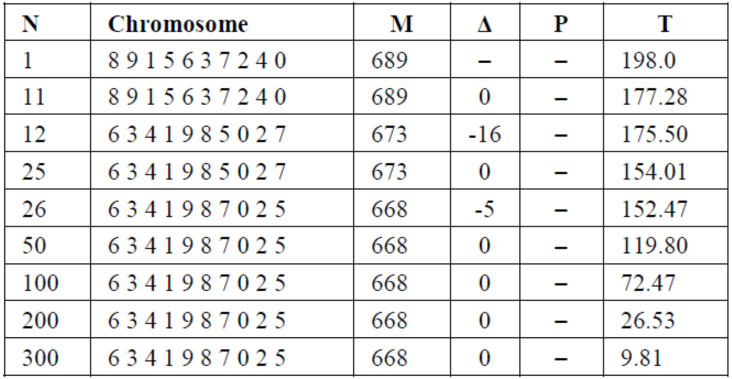
Experimental Results and Discussions



The program had been executed to produce 300 generations of the solution. Initial population contained randomly generated sequence strings. Trial runs were made with single solutions to observe the pattern of development. The result for different job sequences generated in those trials is shown in Table II. The optimum makespan achieved in each of the trials varied as an expected characteristic of the evolutionary process. The two chromosomes that produced the most optimum makespan achieved are presented in sequence number 16 & 17 in the table.

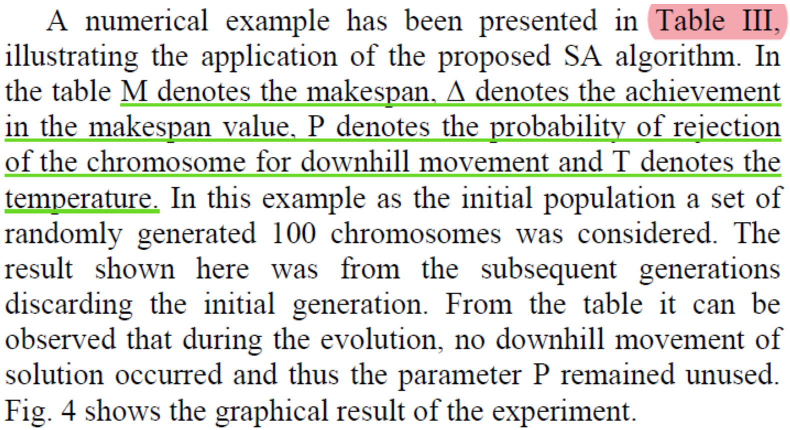
Experimental Results and Discussions

TABLE III. STEP BY STEP REPRESENTATION



N	Chromosome	M	Δ	P	T
1	8 9 1 5 6 3 7 2 4 0	689	–	–	198.0
11	8 9 1 5 6 3 7 2 4 0	689	0	–	177.28
12	6 3 4 1 9 8 5 0 2 7	673	-16	–	175.50
25	6 3 4 1 9 8 5 0 2 7	673	0	–	154.01
26	6 3 4 1 9 8 7 0 2 5	668	-5	–	152.47
50	6 3 4 1 9 8 7 0 2 5	668	0	–	119.80
100	6 3 4 1 9 8 7 0 2 5	668	0	–	72.47
200	6 3 4 1 9 8 7 0 2 5	668	0	–	26.53
300	6 3 4 1 9 8 7 0 2 5	668	0	–	9.81

Experimental Results and Discussions



A numerical example has been presented in Table III, illustrating the application of the proposed SA algorithm. In the table M denotes the makespan, Δ denotes the achievement in the makespan value, P denotes the probability of rejection of the chromosome for downhill movement and T denotes the temperature. In this example as the initial population a set of randomly generated 100 chromosomes was considered. The result shown here was from the subsequent generations discarding the initial generation. From the table it can be observed that during the evolution, no downhill movement of solution occurred and thus the parameter P remained unused. Fig. 4 shows the graphical result of the experiment.

Experimental Results and Discussions

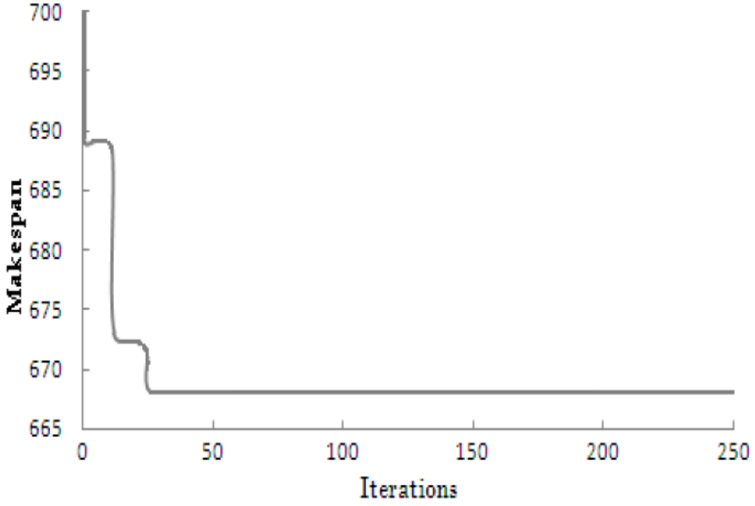


Figure 4. Graphical representation of the result obtained

My Conclusion on the SA-Based JSSP Study

- The paper is well-structured and easy to follow, but it lacks some key academic elements such as detailed citations, experimental setup clarity, and refined scientific writing style.
- Despite these limitations, its structure can be used as a reference when preparing your term project report (CSE480).
- The SA performance **stagnates early** and remains stuck in a local optimum.
- The paper does not provide a discussion on the causes of this stagnation.
- **Possible reasons:**
  - Job-priority representation reduces the search space too much $\rightarrow$ may prevent exploring diverse schedules.
  - Cooling/temperature schedule may be too aggressive $\rightarrow$ low probability of accepting uphill moves.
  - Table III suggests very few moves were accepted $\rightarrow$ there may even be a coding or acceptance logic issue.

Overall: A good introductory example of applying SA to JSSP, but improvements are necessary for stronger optimization performance and scientific rigor.

What is Tabu Search?

- **Tabu Search (TS)** is a *meta-heuristic* that guides a local search to find near-optimal solutions.
- Key idea: avoid revisiting recently explored, poor moves/solutions by maintaining a **tabu list** (short-term memory).
- It will be used to guide the movement from one solution to the next one avoiding cycling.
- Deterministically allows non-improving ("tabu") moves to escape local optima when beneficial.
- Heuristic intuition: **Hill climbing + short-term memory = Tabu Search**.
- Origins: **Fred Glover (1986)**; popularity surged in the 1990s.

CLOSED, OPEN, and TABU (Solutions & Moves)

- Treat **nodes** as candidate **solutions/states**; neighbors come from applying **moves**.
- Classical search: **CLOSED** (visited solutions) and **OPEN** (frontier); newly generated solutions found in **CLOSED** are removed from **OPEN**.
- **Tabu Search:** recently used **moves or solution attributes** are marked **tabu** (short-term memory) to avoid cycles.
- **Aspiration:** a tabu move/solution can be accepted if it improves the **best-so-far**.
- Practically, tabu lists track **move attributes** (e.g., swaps, insertions) rather than full solutions for efficiency.

Terminology

- **Tabu list:** Tracks recent moves/solutions (short-term memory).
- **Tabu tenure:** Number of iterations an item remains tabu.
- **Frequency memory (diversification):** Moves chosen too frequently without success become less likely later to push the search into new regions.

Three main strategies

1. **Neighborhood search:** Generate candidate solutions via allowable moves; pick the best next.
2. **Aspiration criteria:** Override tabu if a candidate is *better than the best-so-far*.
3. **Memory mechanism:** Maintain and update tabu list (and possibly frequency metrics) to balance *exploration* vs *exploitation*.

Main concepts — I

1. TS uses **memory** to record search information.
2. Generate a neighborhood from the current solution; accept the best candidate (even when not strictly improving).
3. This flexibility can induce **cycles**.
4. Previously visited solutions *could* be revisited.
5. To avoid cycling, use a **tabu list** to forbid recently visited moves/solutions.

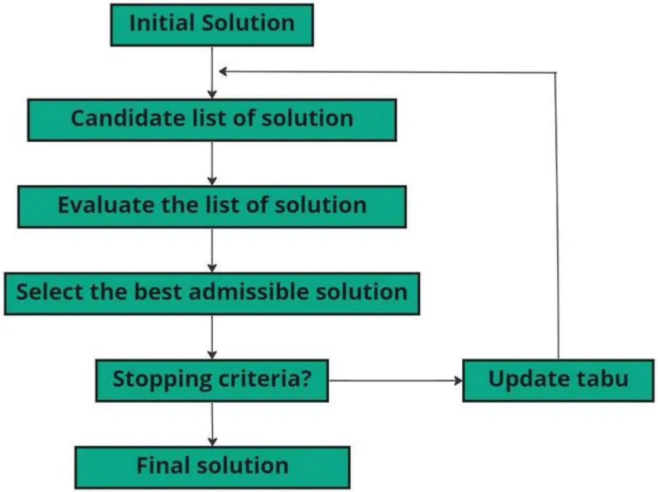
Main concepts — II

6. **Tabu list length** controls the search.
7. Larger tenure $\Rightarrow$ wider exploration, more forbiddances.
8. Smaller tenure $\Rightarrow$ focused search.
9. Tabu list updated each iteration (FIFO behavior).
10. **Tabu tenure** $=$ number of iterations a move is forbidden.
11. **Aspiration:** allow tabu moves that improve the *global best*.

Basic Tabu Search algorithm

- Initialize N (current/best), `CLOSED` (tabu list), parameters: `TabuListSize`, `TT` (tenure).
- Loop until termination:
  1. Generate children `CHILD` $\leftarrow$ `MOVEGEN`(N).
  2. Filter out tabu; append aspirational candidates.
  3. Select next N; scan neighbors to update N by best fitness.
  4. Update best solution if improved.
  5. Append N to `CLOSED`; truncate if exceeds size.

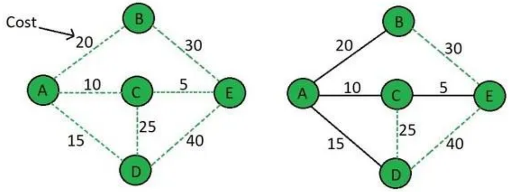
Design Flow


Initial Solution
Candidate list of solution
Evaluate the list of solution
Select the best admissible solution
Stopping criteria?
Update tabu
Final solution

Termination criteria

- Maximum iterations reached.
- Target solution quality reached.
- Time limit reached.
- No improvement for a given window.
- Combined criteria (e.g., time limit + quality threshold).

Example: Minimum Spanning Tree with Constraints



Diagram

See in left side diagram it indicates an optimal solution without constraints,

Traditional prim's or Kruskal's algorithm will give:

AB+AC+AD+CE = 20+10+15+5 = 50

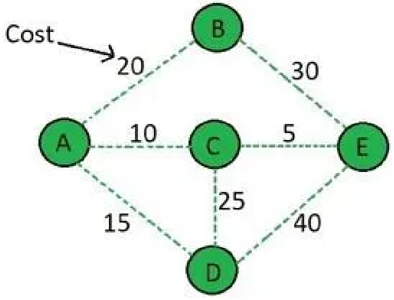
Figure: Source: Tabu Search Article in medium.com by @andy08

Objective: Connect all nodes with minimum costs

Constraints

Constraints:

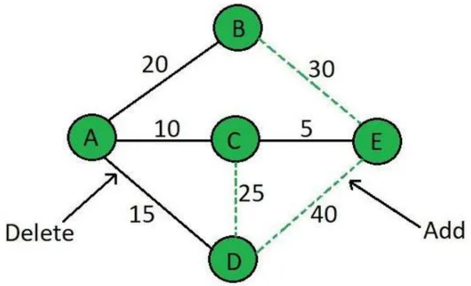
- Path AD is included with path DE then Penalty is 100.
- At most one of the three path AD, CD, and AB can be included. (Penalty of 100 if selected two of the three and Penalty 200 if all three are selected.)



Iteration 1

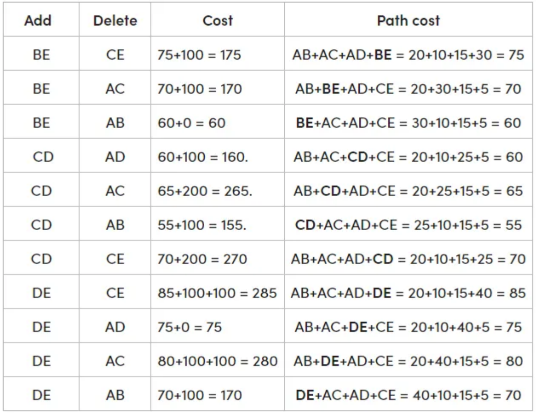
Iteration 1

Cost = 50+100 (Constraint penalties)



Iteration 1

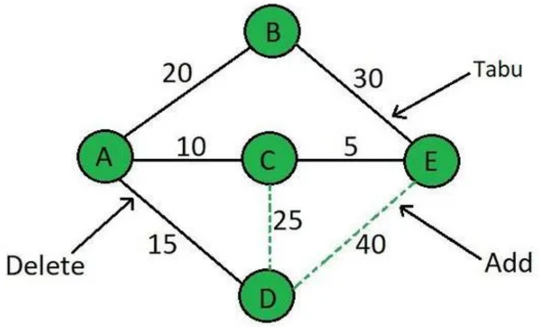
Iteration 1



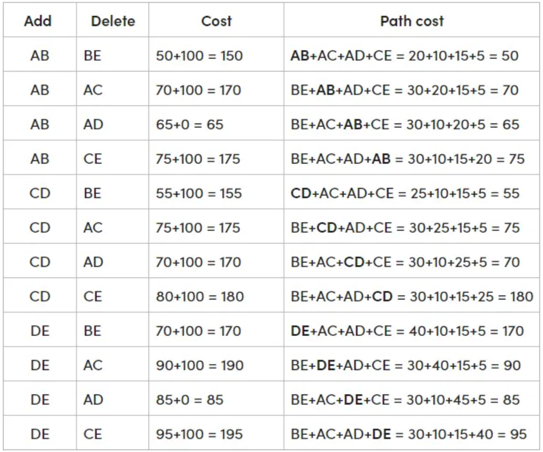
Add	Delete	Cost	Path cost
BE	CE	75+100 = 175	AB+AC+AD+**BE** = 20+10+15+30 = 75
BE	AC	70+100 = 170	AB+**BE**+AD+CE = 20+30+15+5 = 70
BE	AB	60+0 = 60	**BE**+AC+AD+CE = 30+10+15+5 = 60
CD	AD	60+100 = 160.	AB+AC+**CD**+CE = 20+10+25+5 = 60
CD	AC	65+200 = 265.	AB+**CD**+AD+CE = 20+25+15+5 = 65
CD	AB	55+100 = 155.	**CD**+AC+AD+CE = 25+10+15+5 = 55
CD	CE	70+200 = 270	AB+AC+AD+**CD** = 20+10+15+25 = 70
DE	CE	85+100+100 = 285	AB+AC+AD+**DE** = 20+10+15+40 = 85
DE	AD	75+0 = 75	AB+AC+**DE**+CE = 20+10+40+5 = 75
DE	AC	80+100+100 = 280	AB+**DE**+AD+CE = 20+40+15+5 = 80
DE	AB	70+100 = 170	**DE**+AC+AD+CE = 40+10+15+5 = 70

The best optimal path for iteration 1 is BE+AC+AD+CE = 30+10+15+5 = 60

Iteration 2



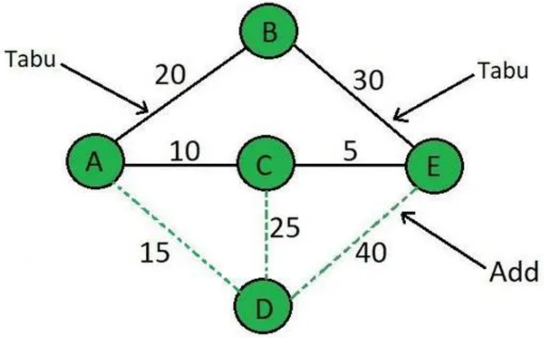
Iteration 2



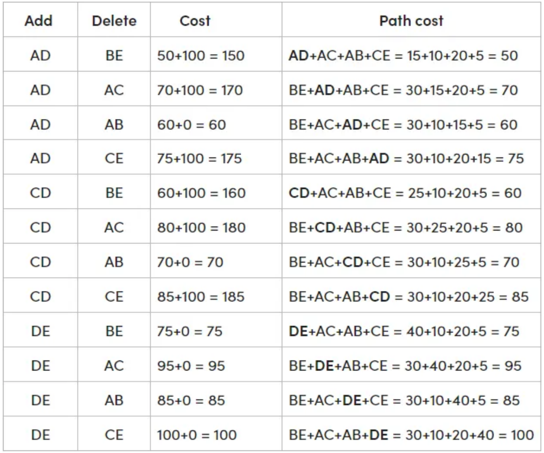
Add	Delete	Cost	Path cost
AB	BE	50+100 = 150	**AB**+AC+AD+CE = 20+10+15+5 = 50
AB	AC	70+100 = 170	BE+**AB**+AD+CE = 30+20+15+5 = 70
AB	AD	65+0 = 65	BE+AC+**AB**+CE = 30+10+20+5 = 65
AB	CE	75+100 = 175	BE+AC+AD+**AB** = 30+10+15+20 = 75
CD	BE	55+100 = 155	**CD**+AC+AD+CE = 25+10+15+5 = 55
CD	AC	75+100 = 175	BE+**CD**+AD+CE = 30+25+15+5 = 75
CD	AD	70+100 = 170	BE+AC+**CD**+CE = 30+10+25+5 = 70
CD	CE	80+100 = 180	BE+AC+AD+**CD** = 30+10+15+25 = 180
DE	BE	70+100 = 170	**DE**+AC+AD+CE = 40+10+15+5 = 170
DE	AC	90+100 = 190	BE+**DE**+AD+CE = 30+40+15+5 = 90
DE	AD	85+0 = 85	BE+AC+**DE**+CE = 30+10+45+5 = 85
DE	CE	95+100 = 195	BE+AC+AD+**DE** = 30+10+15+40 = 95

The best optimal path for iteration 2 is BE+AC+AB+CE = 30+10+20+5 = 65

Iteration 3



Iteration 3



Add	Delete	Cost	Path cost
AD	BE	50+100 = 150	**AD**+AC+AB+CE = 15+10+20+5 = 50
AD	AC	70+100 = 170	BE+**AD**+AB+CE = 30+15+20+5 = 70
AD	AB	60+0 = 60	BE+AC+**AD**+CE = 30+10+15+5 = 60
AD	CE	75+100 = 175	BE+AC+AB+**AD** = 30+10+20+15 = 75
CD	BE	60+100 = 160	**CD**+AC+AB+CE = 25+10+20+5 = 60
CD	AC	80+100 = 180	BE+**CD**+AB+CE = 30+25+20+5 = 80
CD	AB	70+0 = 70	BE+AC+**CD**+CE = 30+10+25+5 = 70
CD	CE	85+100 = 185	BE+AC+AB+**CD** = 30+10+20+25 = 85
DE	BE	75+0 = 75	**DE**+AC+AB+CE = 40+10+20+5 = 75
DE	AC	95+0 = 95	BE+**DE**+AB+CE = 30+40+20+5 = 95
DE	AB	85+0 = 85	BE+AC+**DE**+CE = 30+10+40+5 = 85
DE	CE	100+0 = 100	BE+AC+AB+**DE** = 30+10+20+40 = 100

The best optimal path for iteration 3 is BE+AC+AD+CE = 30+10+15+5 = 60

Tabu Search: Advantages

- **Escapes local optima**: accepts non-improving moves while preventing short cycles via tabu memory.
- **Exploration–exploitation balance**: tenure length + aspiration criteria tune diversification vs. intensification.
- **Flexible memory**: short-, medium-, and long-term memories (e.g., tabu list, frequency, elite set) guide search.
- **Good empirical performance**: often finds high-quality solutions on large combinatorial problems (TSP, JSSP, VRP).

Tabu Search: Disadvantages

- **Parameter tuning**: tenure size, neighborhood scope, aspiration rules can be problem-dependent.
- **Computational cost**: scanning large neighborhoods + memory checks can be expensive.
- **Memory overhead**: attributive/explicit memories add bookkeeping (tabu tables, frequency counts, elite sets).
- **Design effort**: needs a good move operator and attribute encoding; weak designs lead to poor results.
- **No optimality guarantee**: metaheuristic—may stagnate or wander without careful calibration.
- **Sensitivity**: too-short tenure $\Rightarrow$ cycling; too-long tenure $\Rightarrow$ over-diversification.