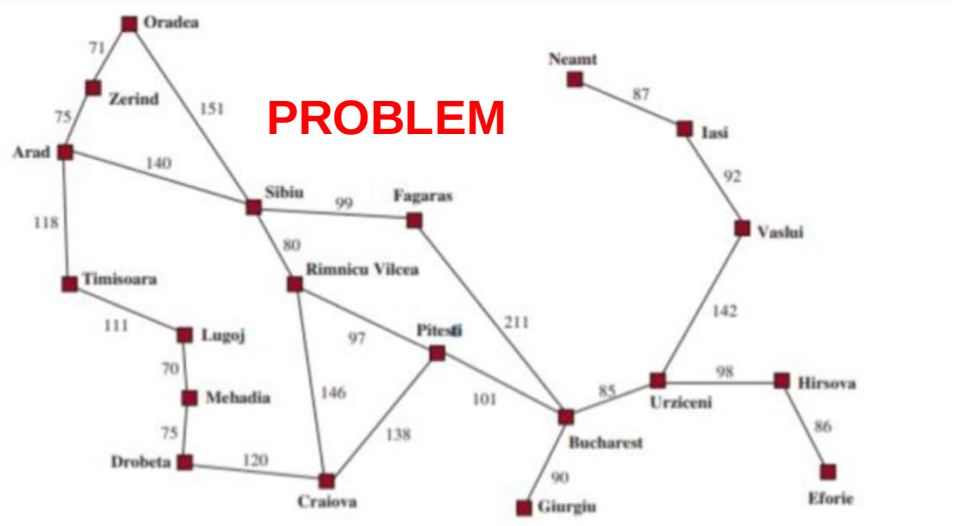
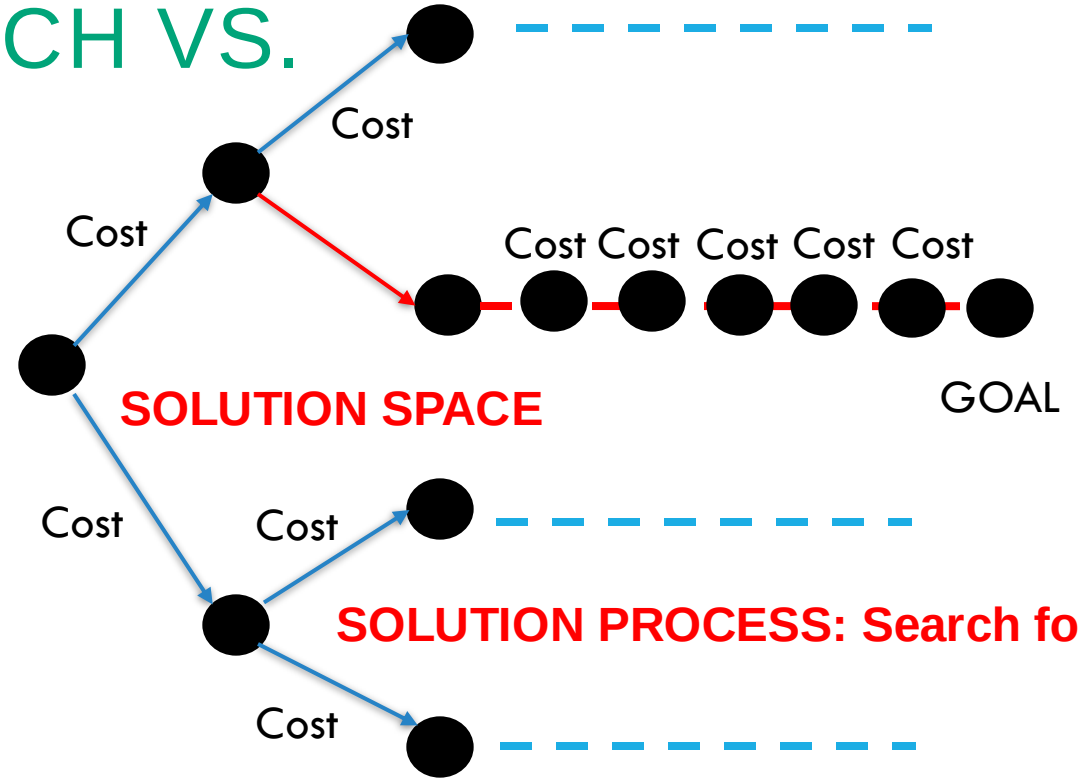
COOPERATIVE AND ADAPTIVE ALGORITHMS

Trajectory methods

TRAJECTORY SEARCH VS. TREE SEARCH





Starting from the root (initial state) travers the tree to search for the solution.

1. Guaranteed to find the solution if it exist (complete)
2. Optimal the path it finds to the solution is the shortest (optimal with respect to the path to goal from start)
3. Computational (in BIG OH) cost vs Goal cost (Actual in physical-sense).

Criteria: 1) Computational Cost to find the goal.
2) Actual Cost in the problem domain.
Cost from start node to goal node.

IN GAMES

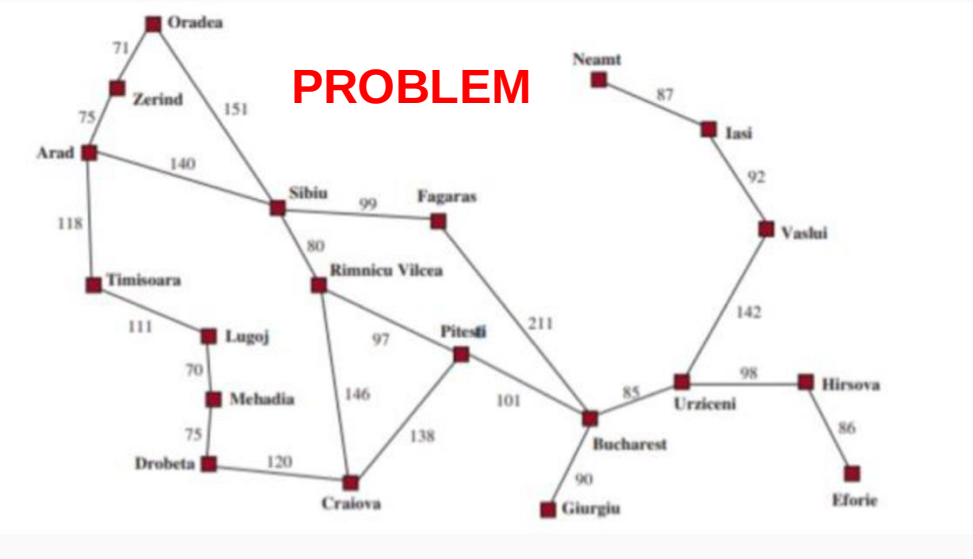
The GOAL is to search for the win state keeping in mind your intermediate choices are observed by the opponent.

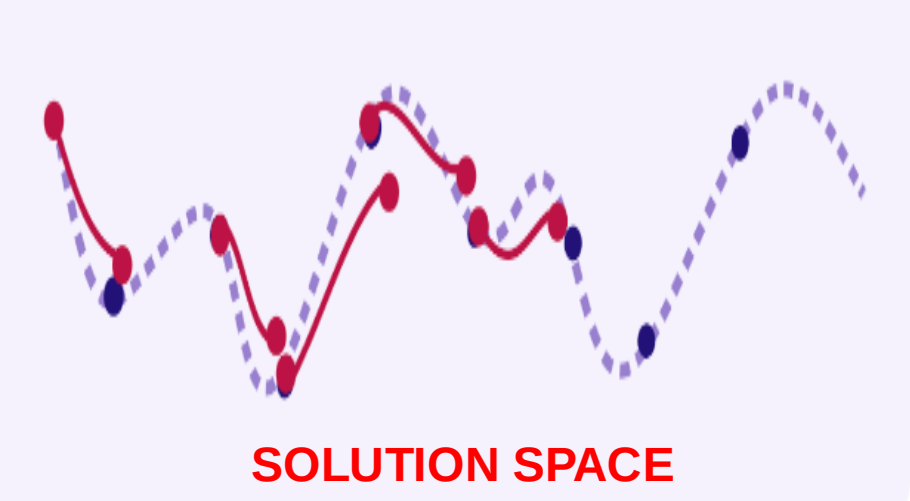
Thus:

Your choices are those that maximize your gain despite the choices your opponent will make– means, anticipate and act accordingly.

Alpha–Beta has nothing with strategy to win. It is in the computations of win, not in the actions to win.

TRAJECTORY GOAL SEARCH





The Problem is formulated as searching for the goal on cost function (not on a tree of choices).

Challenges: how to map the problem into a trajectory solutioning space?

how to map a solution step into a step on the trajectory?

how to evaluate solution goodness?

how to obtain efficiency in searching for the goal on the trajectory. Brute-Force is brutal in complex problems?

SIMULATED ANNEALING

- Mimics the physical annealing process.
- First procedure to simulate the annealing behaviour was developed by Metropolis in 1953 to generate sample states of a thermodynamic system.

V. Cerny, Thermodynamical approach to the traveling salesman problem : an

efficient simulation algorithm. J. of Optimization Theory and Applications, 45(1):41–51, 1985

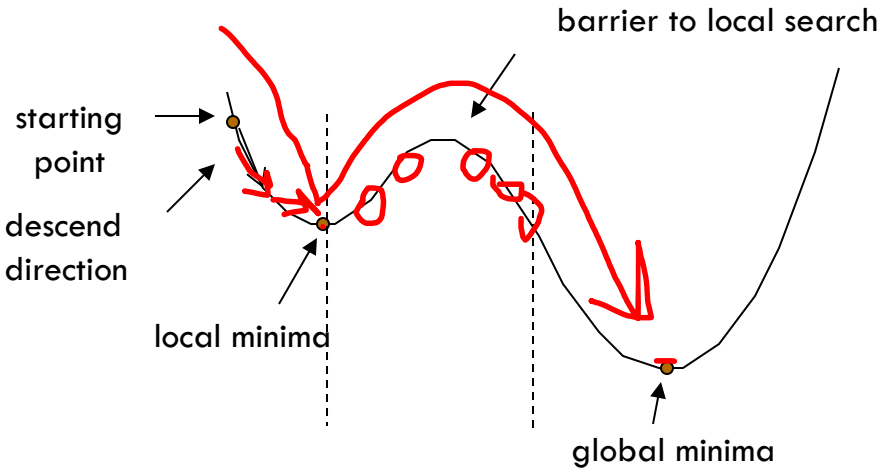
- Defined a set of conditions that, if met, ensure the random walk will sample from probability distribution at equilibrium.
- His algorithm simulates the change in energy of the material when it subjected to cooling process until it reaches a steady “frozen” state.
- He developed a formula for the probability of an increase in energy.

NATURAL VS SIMULATED

Optimization Problem	Physical System
solution x	current state of the solid
cost or objective value $f(x)$	energy of current state
control parameter T	temperature
optimal solution x_{opt}	ground state
simulated annealing	gradual cooling

SIMULATED ANNEALING

- SA is applied to solve optimization problems
- SA is a stochastic algorithm
- SA escapes from local optima by allowing worsening moves
- SA is a memoryless algorithm, the algorithm does not use any information gathered during the search
- SA is applied for both combinatorial and continuous optimization problems
- SA is simple and easy to implement.
- SA is motivated by the physical annealing process



SOME APPLICATIONS

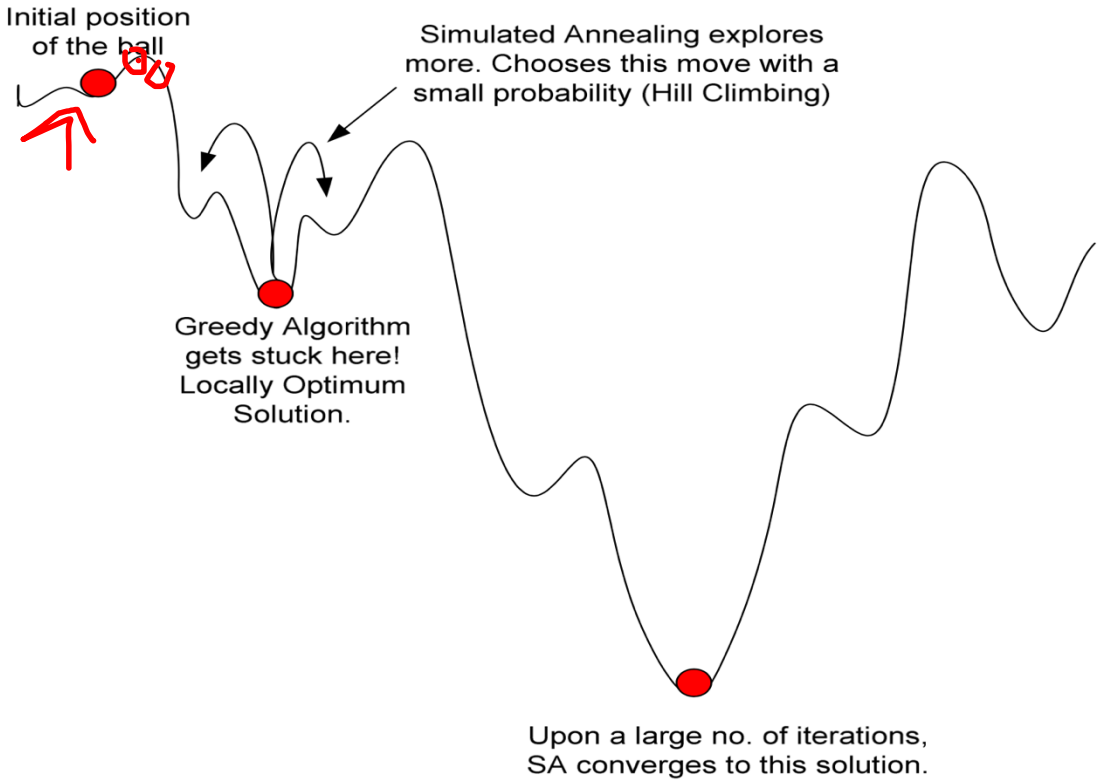
SA has demonstrated capability in problems, such as

- Traveling Salesman Problems
- Graph partitioning
- Matching prob.
- Quadratic Assignment
- Linear Arrangement
- Scheduling
- Machine Learning
- Routing
- Speech Recognition
- etc.

SIMULATED ANNEALING: HOW IT WORKS?

- The approach is part of neighborhood search approaches to move from current solution to new solutions.

- It also tries to escape from local optimum by allowing ***non-improving solutions*** but in a controlled manner.



SIMULATED ANNEALING ALGORITHM STRATEGY

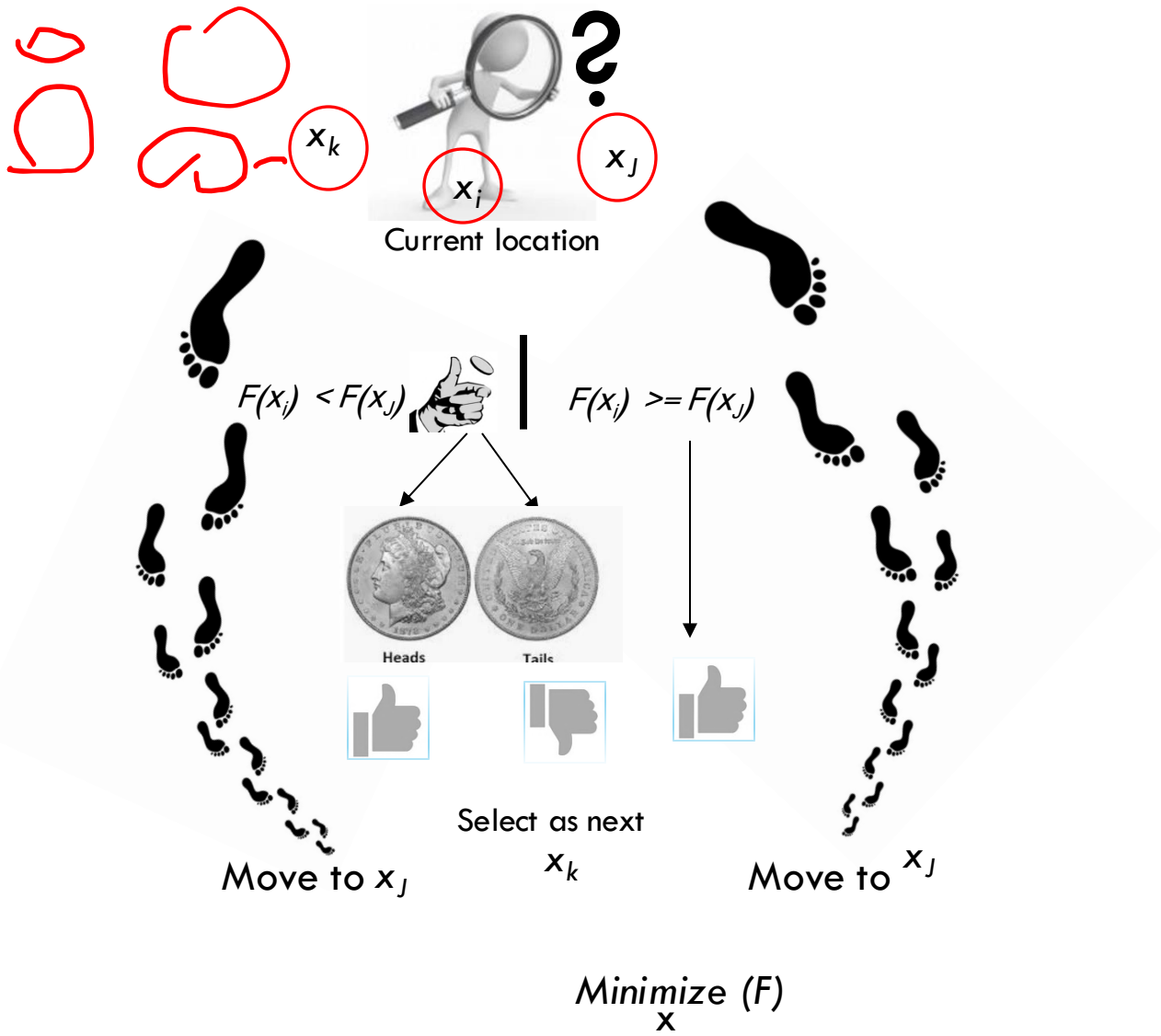
P(H/T) is 50/50

In our case temperature dependent!

Recall, in thermodynamics, a state at a temperature t has a probability of an increase in the energy magnitude ΔE given by:

$$P(\Delta E) = e^{-\Delta E / k \times t}$$

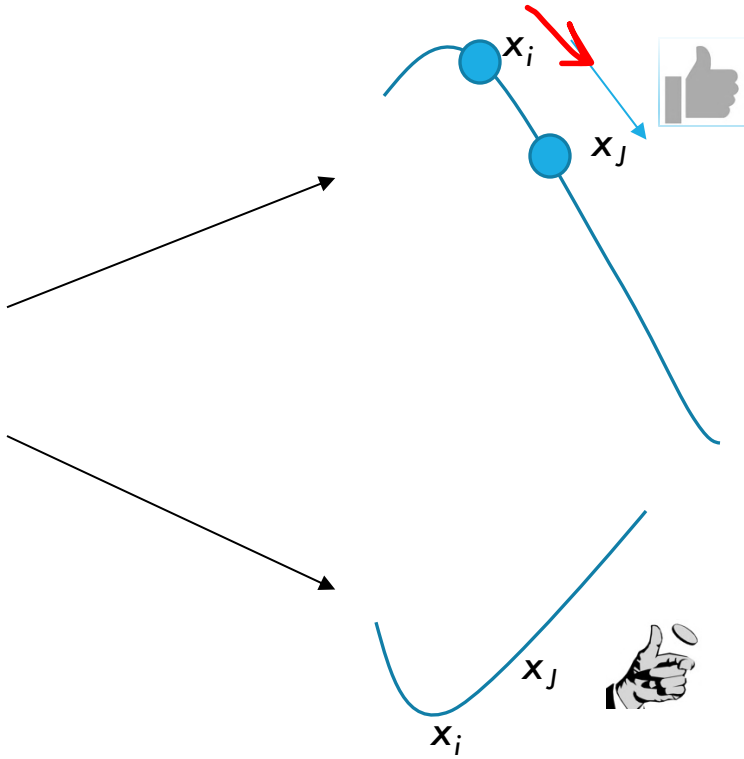
where k is the Boltzmann constant.



SIMULATED ANNEALING

The acceptance probability is defined as:

$$P = \begin{cases} 1 & if \Delta c \leq 0 \\ e^{-\Delta c / t} & if \Delta c > 0 \end{cases}$$



where:

Δc is the change in the solution cost,

t is the current temperature (control parameter, no physical role).

Notice that we dropped Boltzmann's constant

SA BASIC ALGORITHM

Set current solution to an initial solution s= s_0

Set current temperature to an initial temperature $t = t_o$

Select a temperature reduction function α

Repeat

- Repeat
  - Select a solution s_i from the neighborhood $N(s)$
  - Calculate change in cost according to the new solution ΔC
  - If ΔC <0 then accept new solution (it is improving) s=s_i
  - Else generate a random number x in the range (0,1)
    - If x < $e^{-\Delta c/t}$ then accept new solution s= s_i
- Until max no. of iterations for the temperature
- Decrease t using α

Until stopping conditions are satisfied

s is the final solution

Temp T_1

Temp T_2

$$T_1 > T_2$$

$$T_1 = T_2 + \alpha$$

$$P_2 = e^{-\frac{\Delta}{T_2}} = (e^{-\Delta})^{\frac{1}{T_2}}$$

$$P_1 = e^{-\frac{\Delta}{T_2+\alpha}} = (e^{-\Delta})^{\frac{1}{T_2+\alpha}}$$

$$P_1 > P_2$$

More chance for a worse solution to be accepted in T_1 than in T_2 .

$$T = 1$$

$$\Delta C_1 > \Delta C_2 > 0$$

$$e.g, \Delta C_1 = \Delta C_2 + \alpha$$

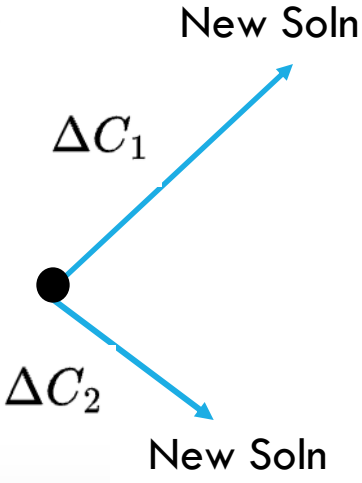
$$P_1 = e^{-\Delta C_1}$$

$$= e^{-(\Delta C_2+\alpha)}$$

$$= e^{-\Delta C_2} \times e^{-\alpha}$$

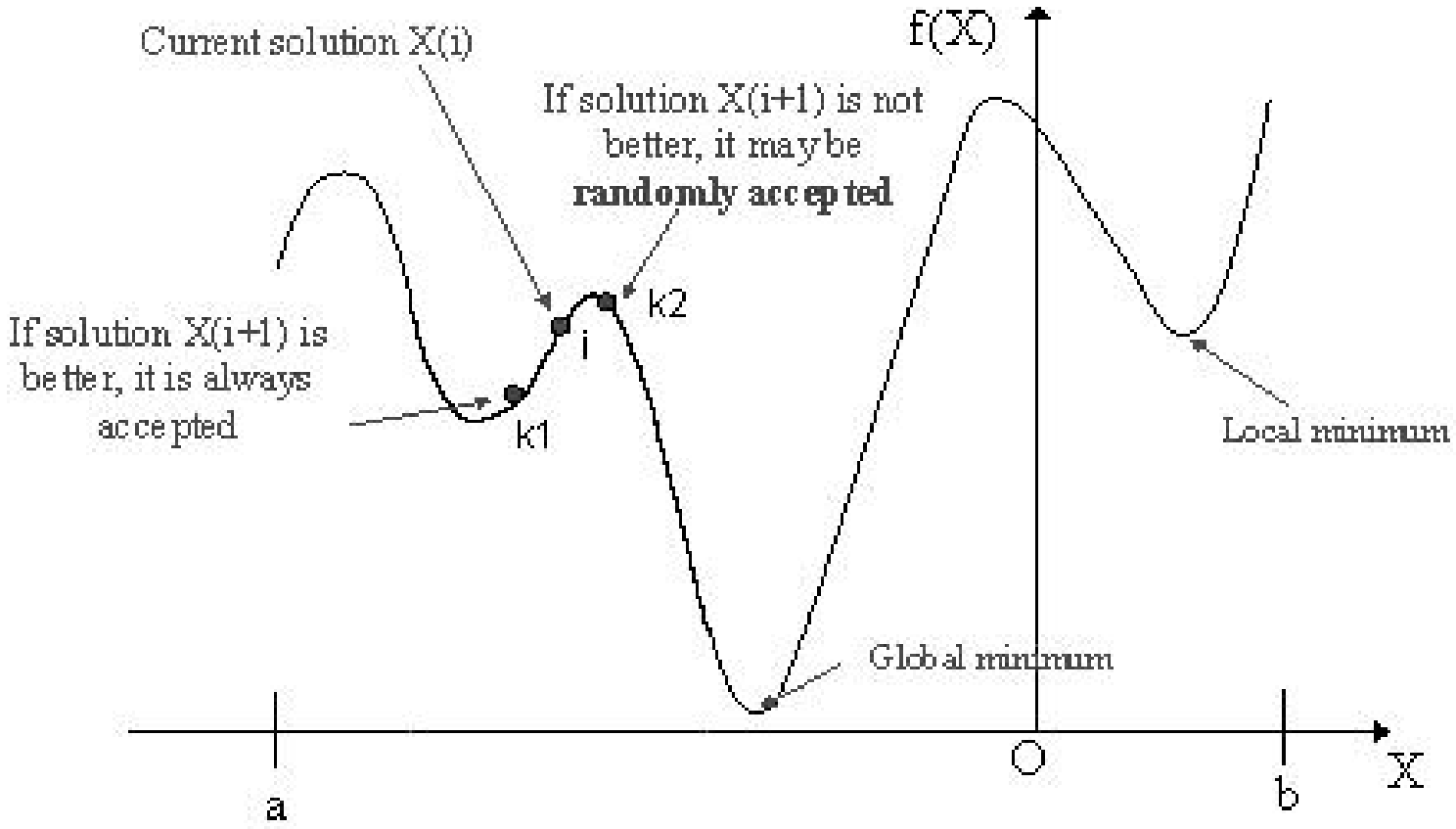
$$= P_2 \times e^{-\alpha}$$

$$P_1 < P_2$$



More chance for solution 2 to be accepted than solution 1.

SIMULATED ANNEALING



SIMULATED ANNEALING

SA **always accepts** better solutions,

SA can accept worse states according to some probability, the ***acceptance probability***,

The acceptance probability is chosen so as to ultimately move to a better solution in the search space,

The acceptance probability is dependent on:

- ✓ The current temperature,
- ✓ The cost difference.

STRATEGY

Higher temperature more attitude to accept bad moves

***i,e,* At high temperatures, explore parameter space**

At lower temperatures, tendency to reject bad moves restrict exploration- exploitation

ACCEPTANCE PROBABILITY OF WORSE SOLUTION

Change	Temperature	Acceptance Probability
0.2	0.1	0.1353
0.4	0.1	0.0183
0.6	0.1	0.0025
0.8	0.1	0.0003

Change	Temperature	Acceptance Probability
0.2	0.9	0.8007
0.4	0.9	0.6412
0.6	0.9	0.5134
0.8	0.9	0.4111

- As the temperature decreases, the probability of accepting a worse state decreases,
- At zero temperature, no worse states are accepted,

ANNEALING SCHEDULE

The value of the temperature is adjusted according to the ***annealing schedule***,

The annealing schedule is a mapping from time to temperature,

The annealing schedule is determined by:

- The initial temperature,
- The final temperature,
- The temperature decrement rule,
- Number of iterations at each temperature.

TEMPERATURE

The initial temperature:

- Must be **high enough** to allow a move to possibly any state in the search space,
- Must not be too hot [SA would behave as random search for a number of iterations],
- The maximum **change of the cost function** could be used as **a guide** to set this value,
- A good initial temperature is one that accepts about **60%** of the worse moves.

TEMPERATURE

The final temperature:

- Doesn't have to reach zero,

- Might take a very long time to reach zero when some decrement approaches are used,

- To stop the algorithm:
  - A reasonably low temperature,
  - The system is frozen, no better moves are generated neither any worse moves are getting accepted.

TEMPERATURE

Temperature decrement rules

- Linear

$$t = t - \alpha$$

- Geometric

$$t = t \times \alpha$$

- Slow decrease: [for example]

  - decreases the temperature very slowly.

$$t = t/(1+\beta t), \beta$$

TEMPERATURE-ITERATIONS

The number of iterations at each temperature:

- Enough iterations should be allowed at every temperature for the system to be stable at that temperature,
- The required number of iterations might be large for large size problems
- Usually, a constant value is used.
- In slow decrease rules, use one iteration per temperature
- We could dynamically change this value:
  - Allowing a small number of iterations at high temperatures,
  - Allowing a large number of iterations at low temperature to fully explore the local optimum.

CONVERGENCE OF SA

The behavior of SA Algorithm can be modeled using **Markov chains**.

- It has been shown that under **constant temperature** and as long as it is possible to find a sequence of exchanges that will transform any solution to another with **non−zero−probability**, the process **converges** independent of the starting solution.

In SA the temperature is **not constant**.

- The process is still a **Markov chain but non−homogeneous**. It converges if the temperature is reduced to zero slowly (with a specific logarithmic form) and the number of iterations at each temperature is large (exponential in terms of the problem size).

The convergence theory dictates a certain form of cooling schedule that is not practical to use.

- However, the results give SA a theoretical backing not always possible with heuristic methods.

SIMULATED ANNEALING

Advantages:

- Ease of use,
- Providing good solutions for a wide range of problems.

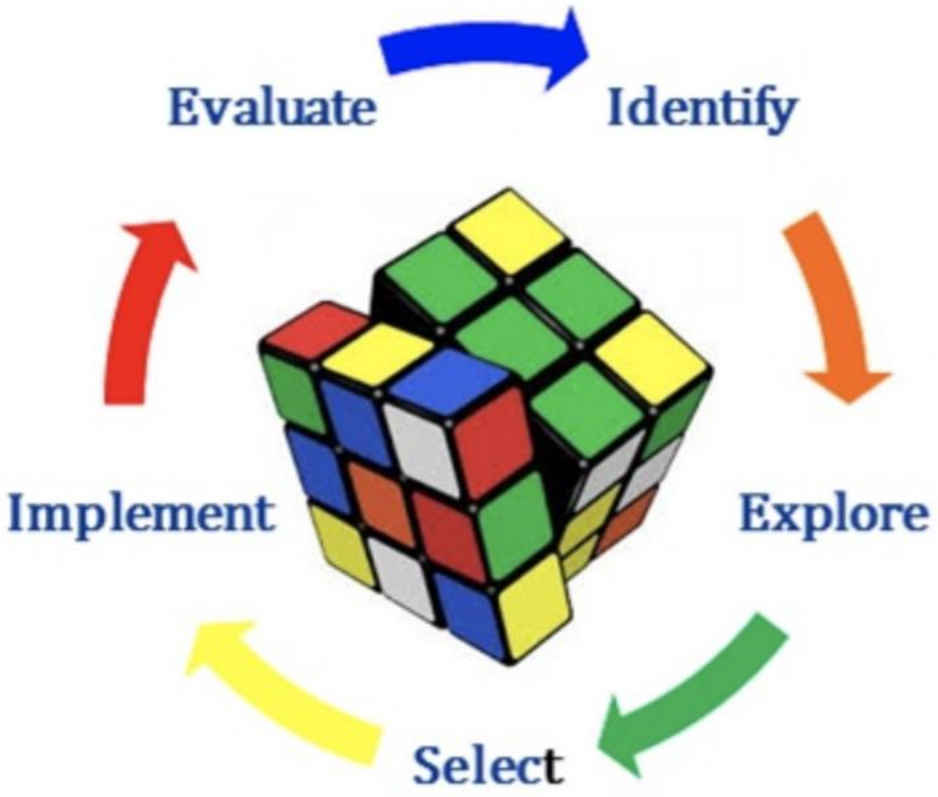
Disadvantages:

- A lot of computer time for many runs,
- A lot of tuneable parameters.

SIMULATED ANNEALING

Applied successfully to many applications:

- Non−linear function optimization,
- Travelling Salesman Problem (TSP),
- Scheduling Problems,
- FPGA placement,
- Task allocation,
- Network design,
- Graph colouring and partitioning,
- Many more …



META-HEURISTICS: SA

Traveling Salesman

SA FOR TSP

Problem:

- Given n cities,
- Find a complete tour with minimal length.

Search space is **BIG**: for 30 cities there are 30! 10^{32} possible tours.

SA FOR TSP

The solution is a permutation of n cities:

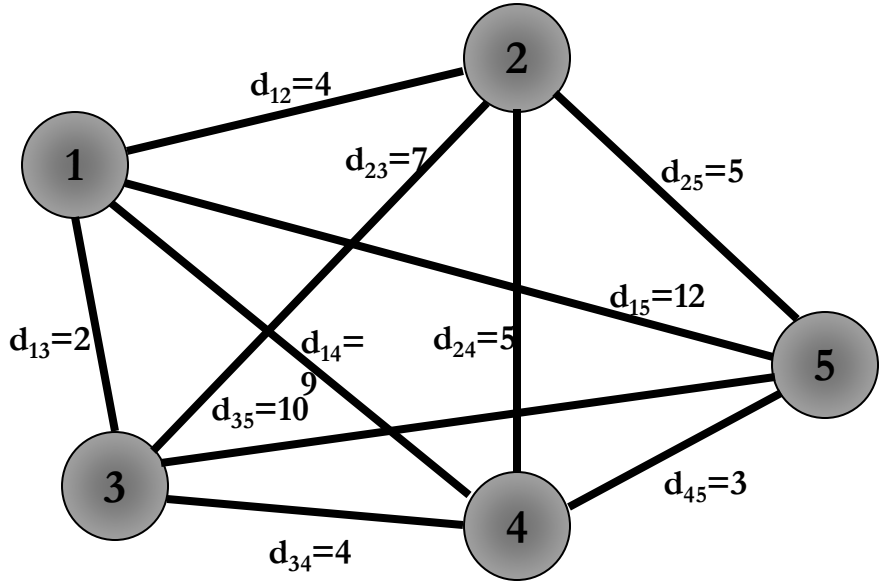
$$Sol = [\ 1,\ 2,\ 3,\ ..,\ n\ ]$$

Several choices exist for generating the required neighbourhood.

- The new solution could be generated by performing a predetermined number of swaps on the current solution,

Some adaptive implementations decrease the number of cities to swap during the run. Thus reducing the neighbourhood size as the search progresses.

SA FOR TSP - EXAMPLE



SA FOR TSP - EXAMPLE

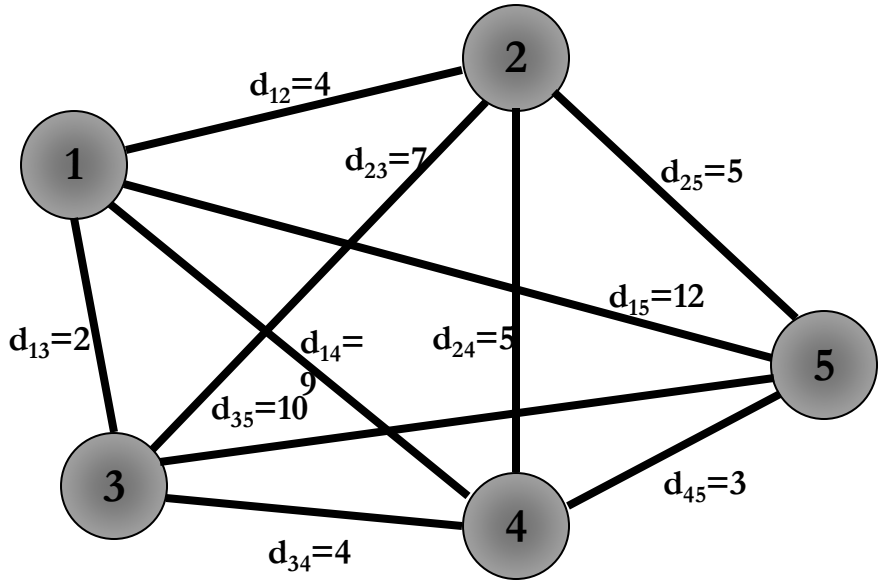
Start with a random solution :

$$Sol = [\ 1,\ 3,\ 4,\ 2,\ 5\ ]$$

The total distance would be:

$$total\_dist = \sum_{i=1}^{n-1} d_{Sol_i Sol_{i+1}} + d_{Sol_n Sol_1}$$

For a closed tour



SA FOR TSP - EXAMPLE

The tour length of the initial solution is: Sol=[1,3,4,2,5]

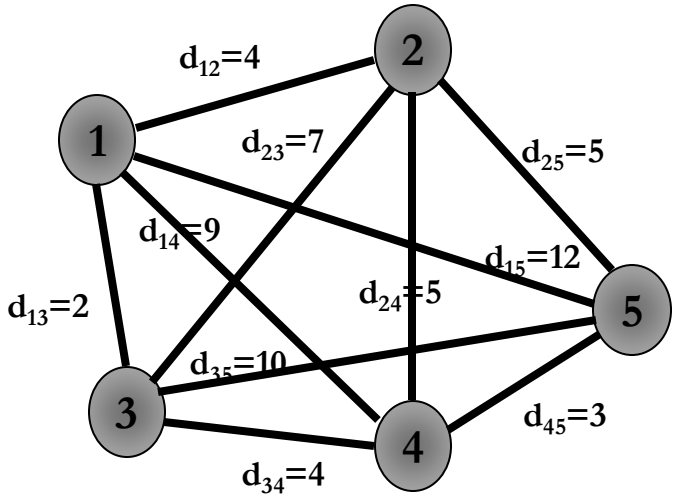
$$total\_dist = 2 + 4 + 5 + 5 + 12 = 28$$

To generate a candidate solution, select two random cities and swap them:

$$Sol = [\ 1, 2, 4, 3, 5\ ]$$

The tour length of the candidate solution is:

$$total\_dist = 4 + 5 + 4 + 10 + 12 = 35$$



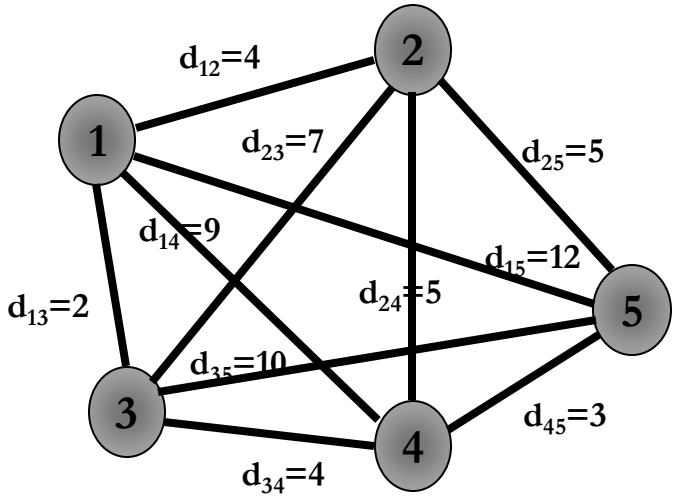
SA FOR TSP - EXAMPLE

Since the new solution has a longer tour length, it will be conditionally accepted according to a probability of (at higher temperatures, there's a higher probability of acceptance):

$$P = e^{-7/t}$$

- Assuming the new solution was not accepted, we generate a different one starting from the initial solution:

$$Sol = [\ 1,\ 3,\ 4,\ 5,\ 2\ ]$$



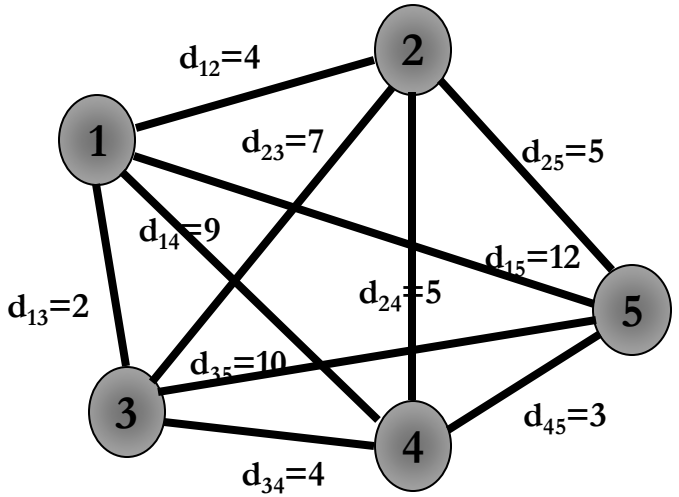
SA FOR TSP - EXAMPLE

The tour length of the candidate solution [1, 3, 4, 2, 5] is:

$$total\_dist = 2 + 4 + 3 + 5 + 4 = 18$$

Since this solution has a shorter tour length, it will get accepted,

The search continues …



SA FOR TSP

The SA was applied for the berlin52 TSP instance by having:

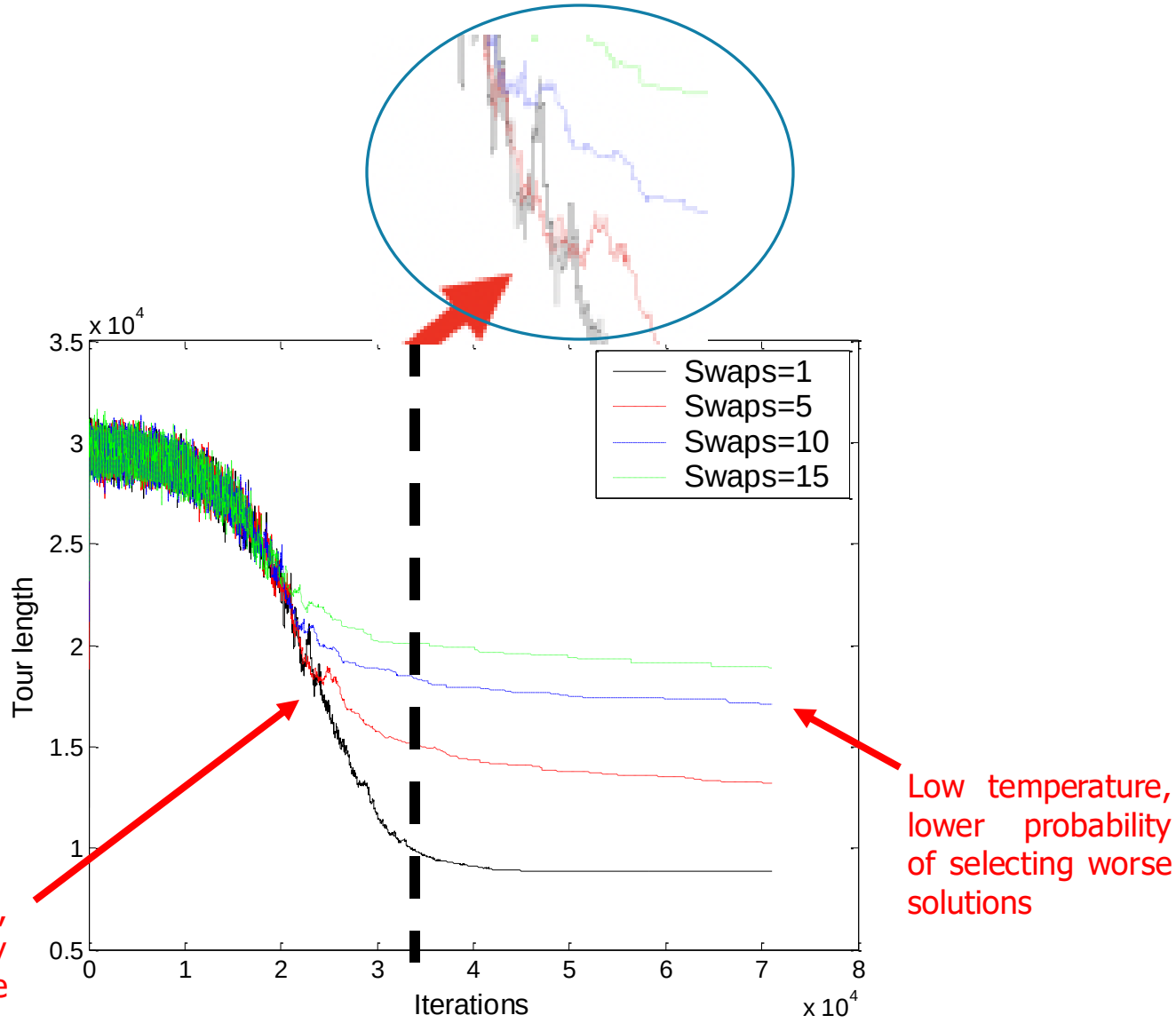
- T_initial = 10000,
- T_Final = 0.1,
- α = 0.85,
- Using 1000 iteration for each T
- Using different number of swaps for obtaining the neighbouring solution (1, 5, 10 and 15),

```
NAME: berlin52
TYPE: TSP
COMMENT: 52 locations in Berlin (Groetschel)
DIMENSION: 52
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
4 945.0 685.0
5 845.0 655.0
6 880.0 660.0
7 25.0 230.0
8 525.0 1000.0
9 580.0 1175.0
10 650.0 1130.0
11 1605.0 620.0
12 1220.0 580.0
13 1465.0 200.0
14 1530.0 5.0
15 845.0 680.0
16 725.0 370.0
17 145.0 665.0
18 415.0 635.0
19 510.0 875.0
20 560.0 365.0
21 300.0 465.0
22 520.0 585.0
23 480.0 415.0
24 835.0 625.0
25 975.0 580.0
26 1215.0 245.0
27 1320.0 315.0
28 1250.0 400.0
29 660.0 180.0
30 410.0 250.0
31 420.0 555.0
32 575.0 665.0
33 1150.0 1160.0
34 700.0 580.0
35 685.0 595.0
36 685.0 610.0
37 770.0 610.0
38 795.0 645.0
39 720.0 635.0
40 760.0 650.0
41 475.0 960.0
42 95.0 260.0
43 875.0 920.0
44 700.0 500.0
45 555.0 815.0
46 830.0 485.0
47 1170.0 65.0
48 830.0 610.0
49 605.0 625.0
50 595.0 360.0
51 1340.0 725.0
52 1740.0 245.0
EOF
```

SA FOR TSP

Number of swaps	Tour length
1	**8861**
5	13180
10	17124
15	18900

SA FOR TSP

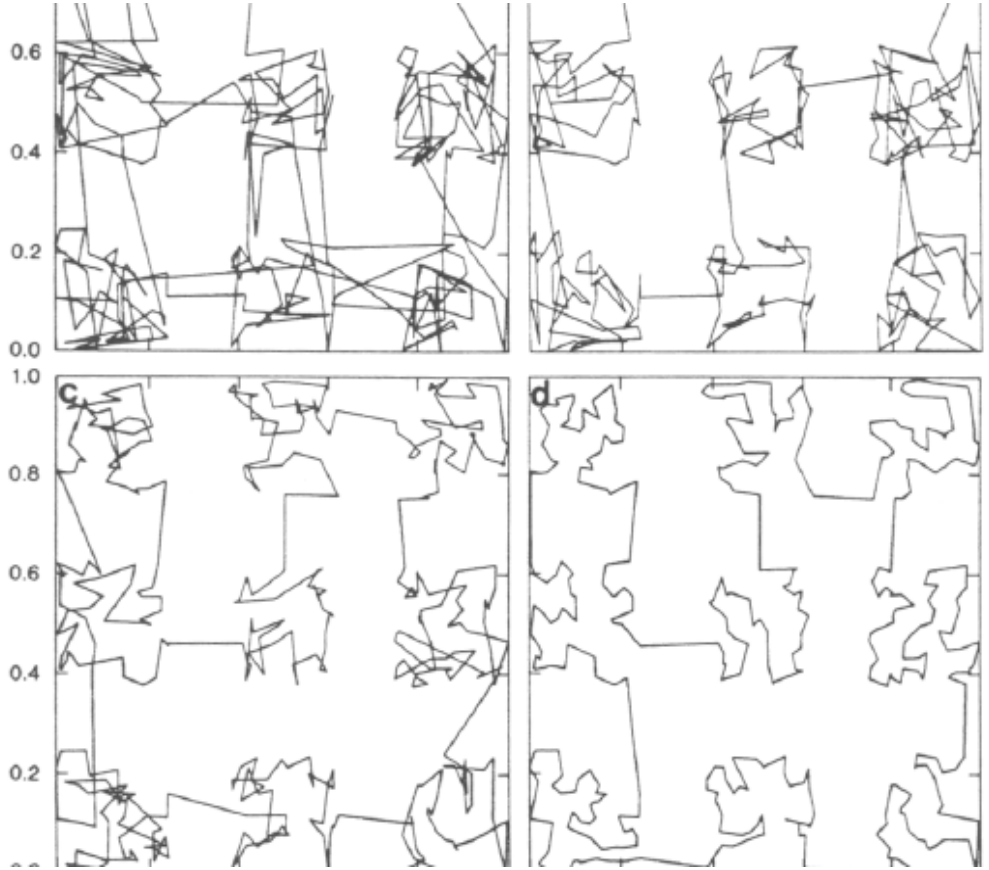


High temperature, higher probability of selecting worse solutions

Low temperature, lower probability of selecting worse solutions

SA FOR TSP

How simulated annealing proceeds …



QUICK REVIEW

SIMULATED ANNEALING

Heuristic/goal/fitness function E (energy)

Generate a move (randomly) and compute

$$\Delta E = E_{new} - E_{old}$$

If $\Delta E <= 0$, then accept the move

If $\Delta E > 0$, accept the move with probability:

Set

$$P(\Delta E) = e^{-\frac{\Delta E}{kT}}$$

T is “Temperature”

SIMULATED ANNEALING

Compare P(ΔE) with a random number from 0 to 1.

- If it's below, then accept

Temperature decreased over time

When T is higher, downward moves are more likely accepted

- T=0 means equivalent to hill climbing

When ΔE is smaller, downward moves are more likely accepted

“COOLING SCHEDULE”

Speed at which temperature is reduced has an effect

Too fast and the optima are not found

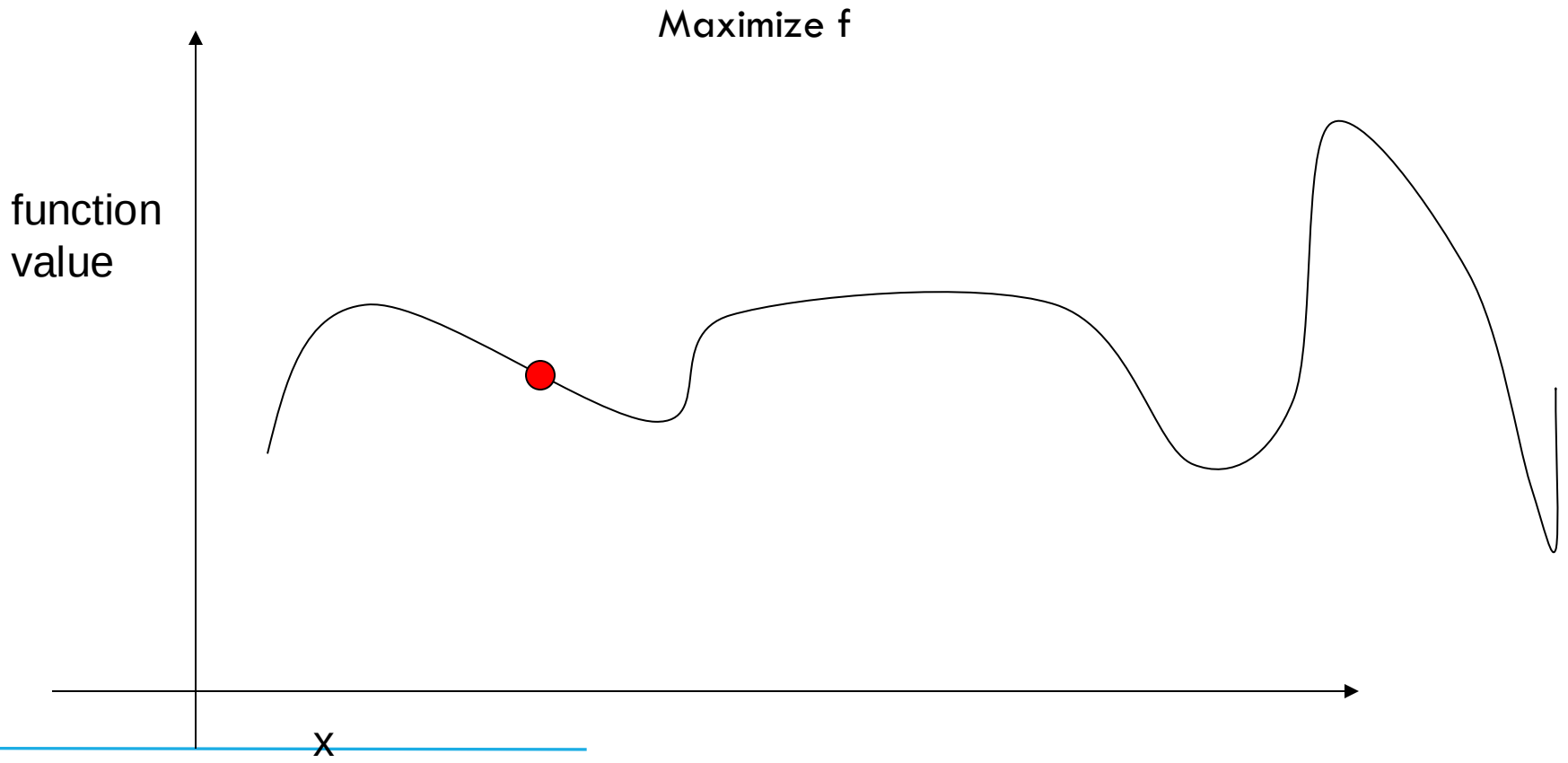
Too slow and time is wasted

SIMULATED ANNEALING

Random Starting Point

T = Very High

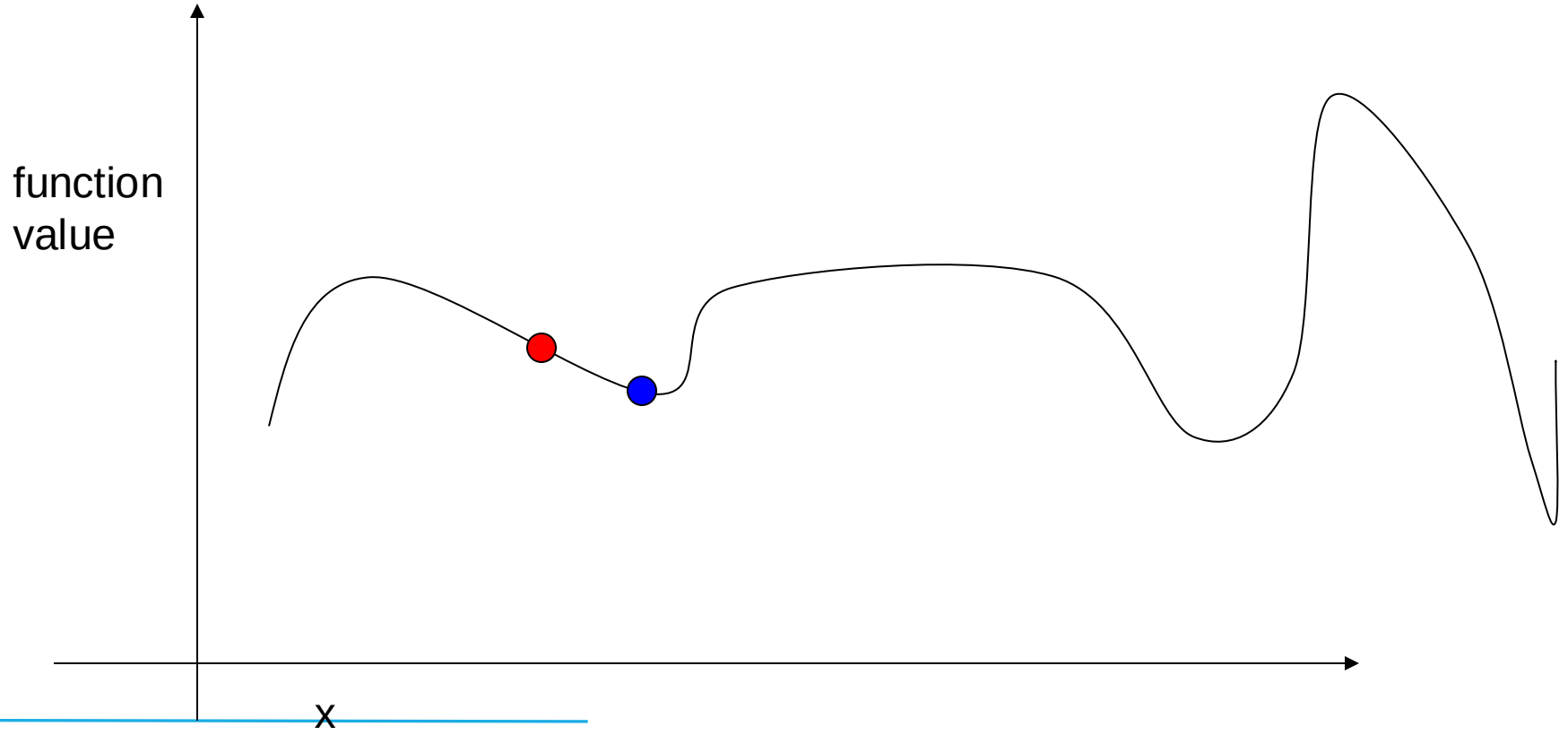
$$P(\Delta E) = e^{-\frac{\Delta E}{kT}}$$



SIMULATED ANNEALING

Random Step

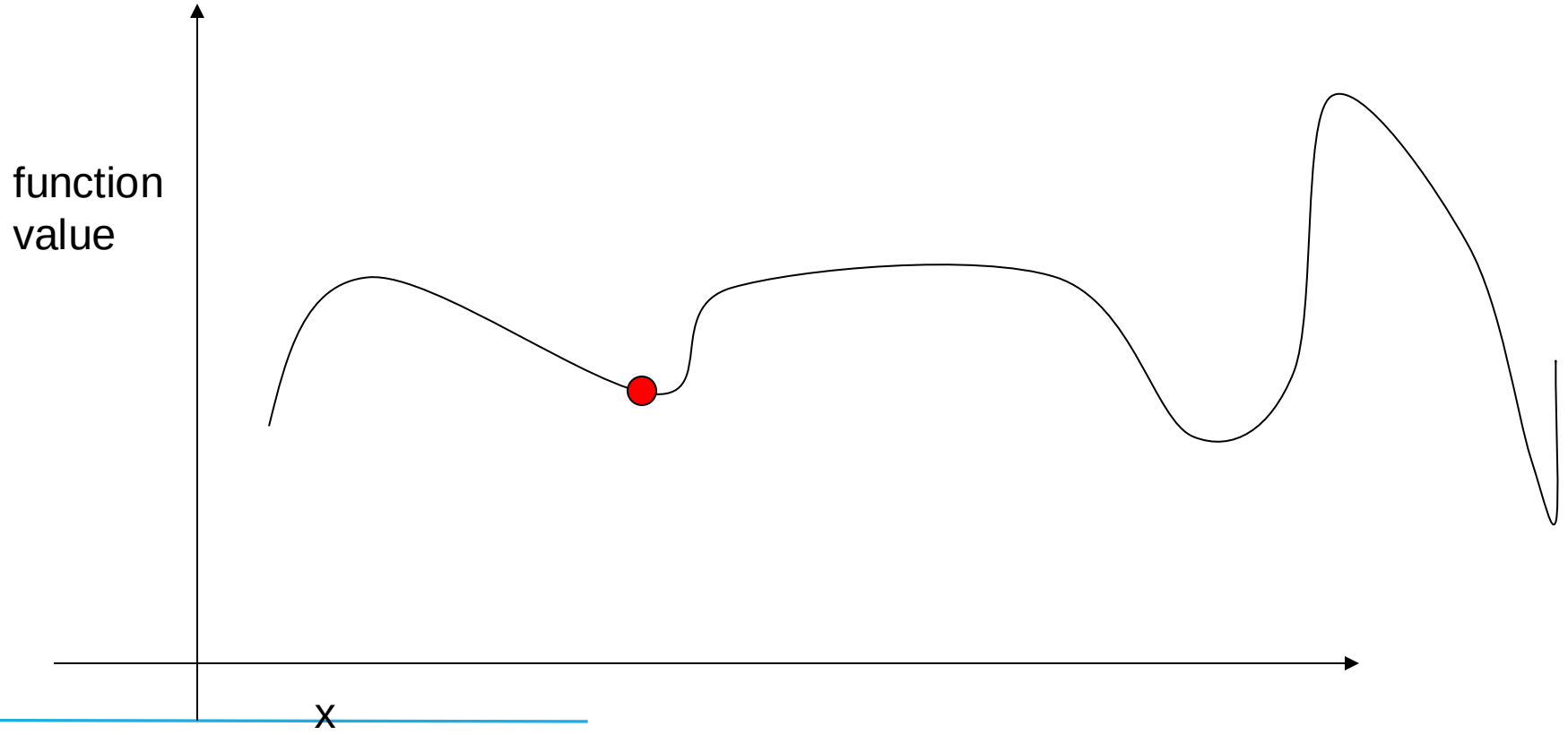
T = Very High



SIMULATED ANNEALING

Even though E is lower, accept

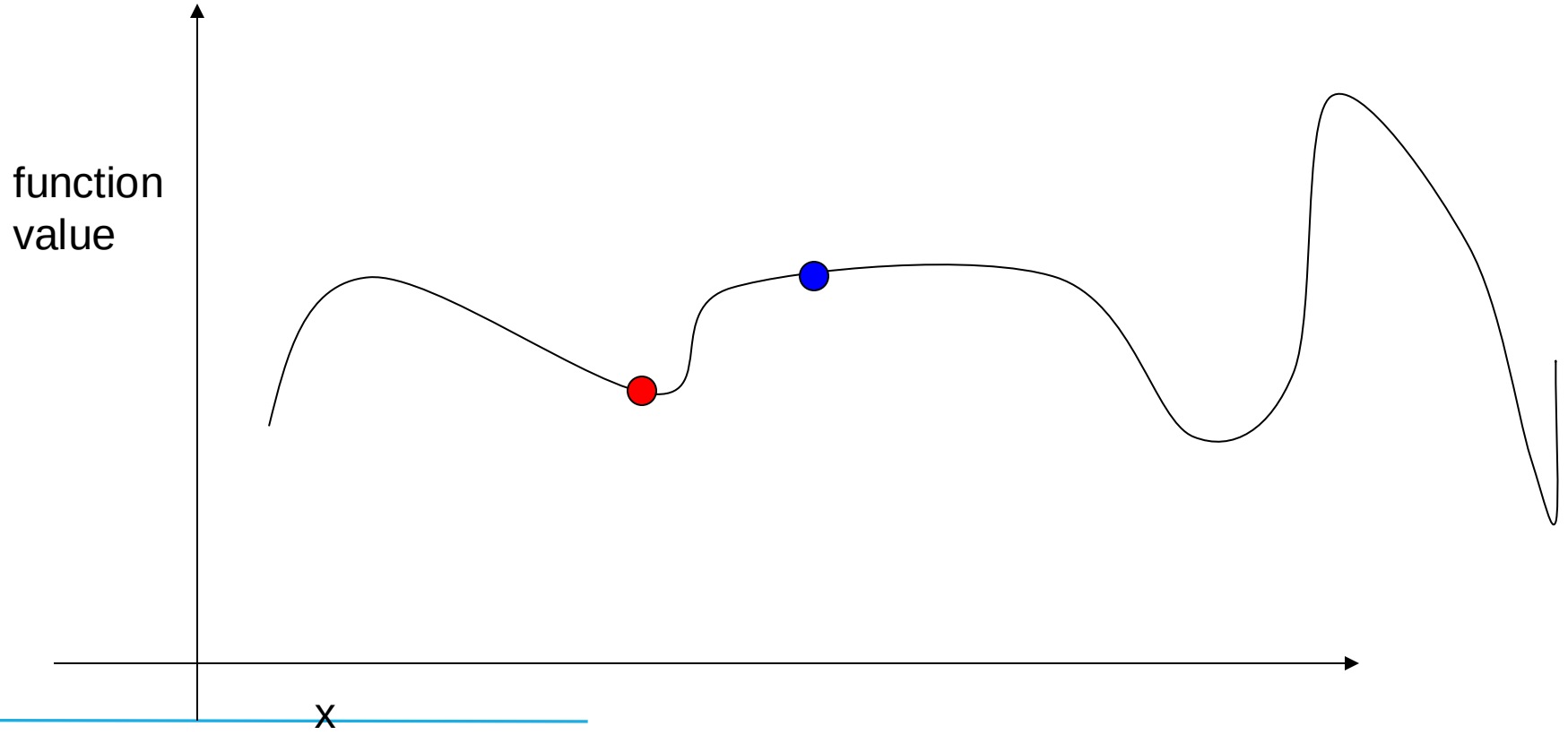
T = Very High



SIMULATED ANNEALING

Next Step; accept since higher E

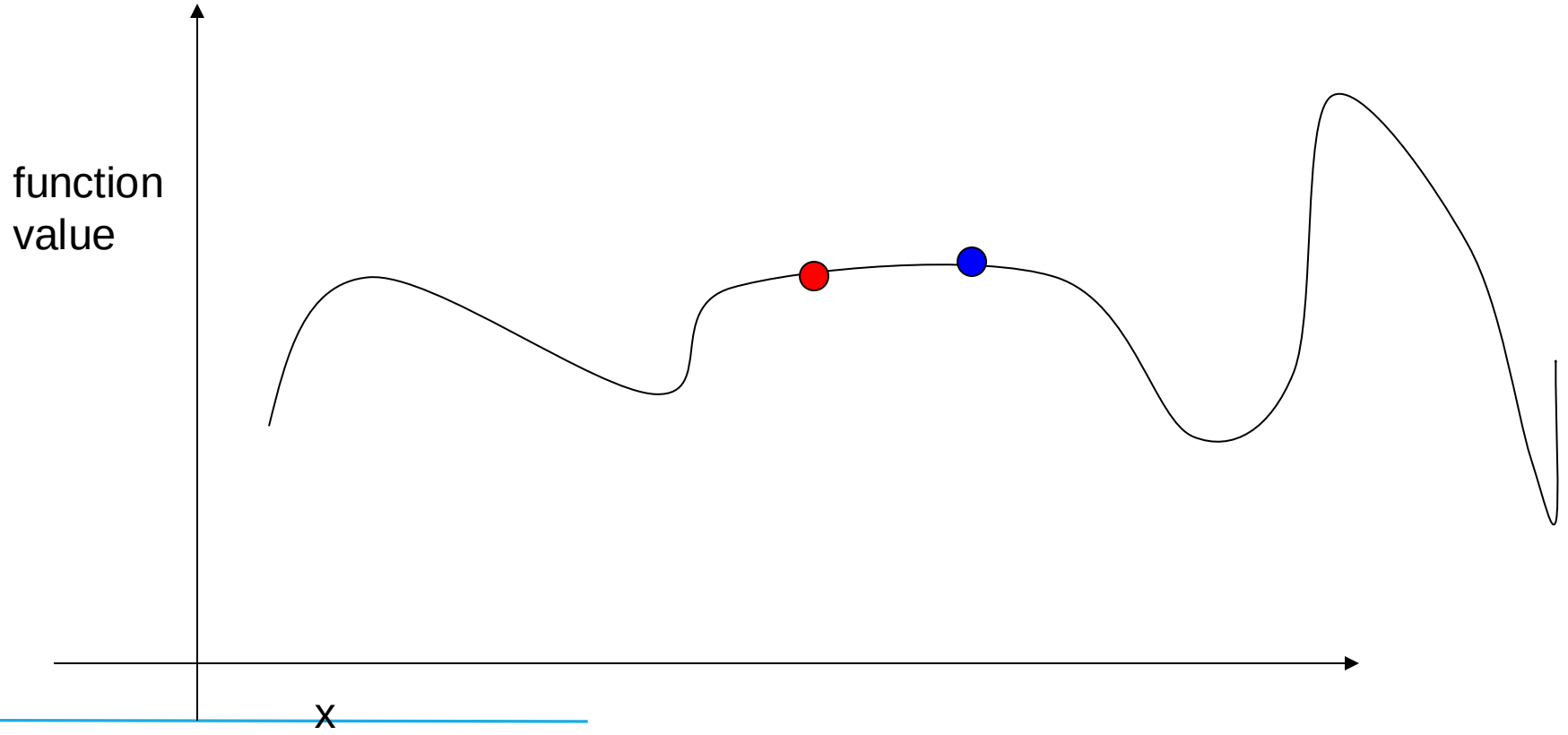
T = Very High



SIMULATED ANNEALING

Next Step; accept since higher E

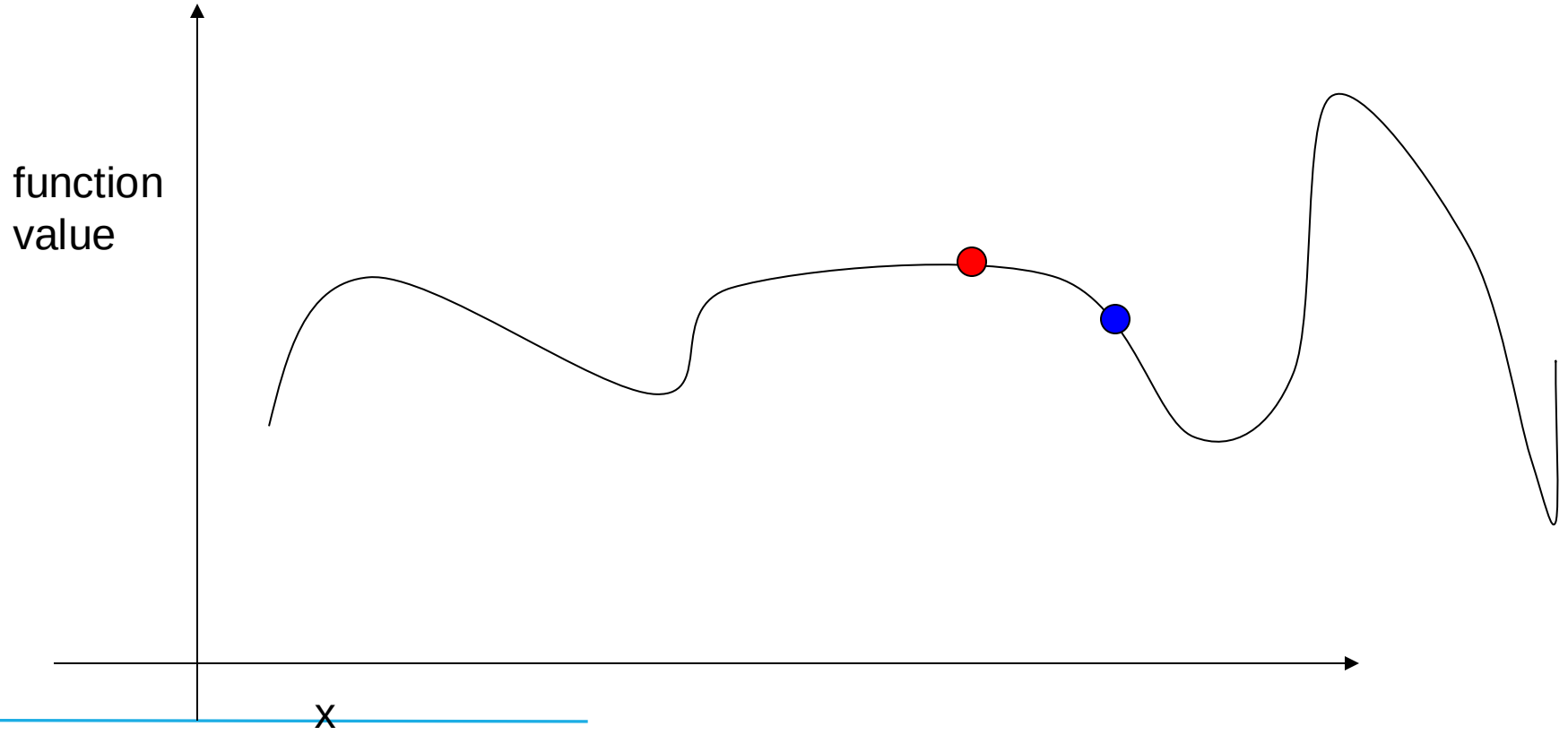
T = Very High



SIMULATED ANNEALING

Next Step; accept even though lower

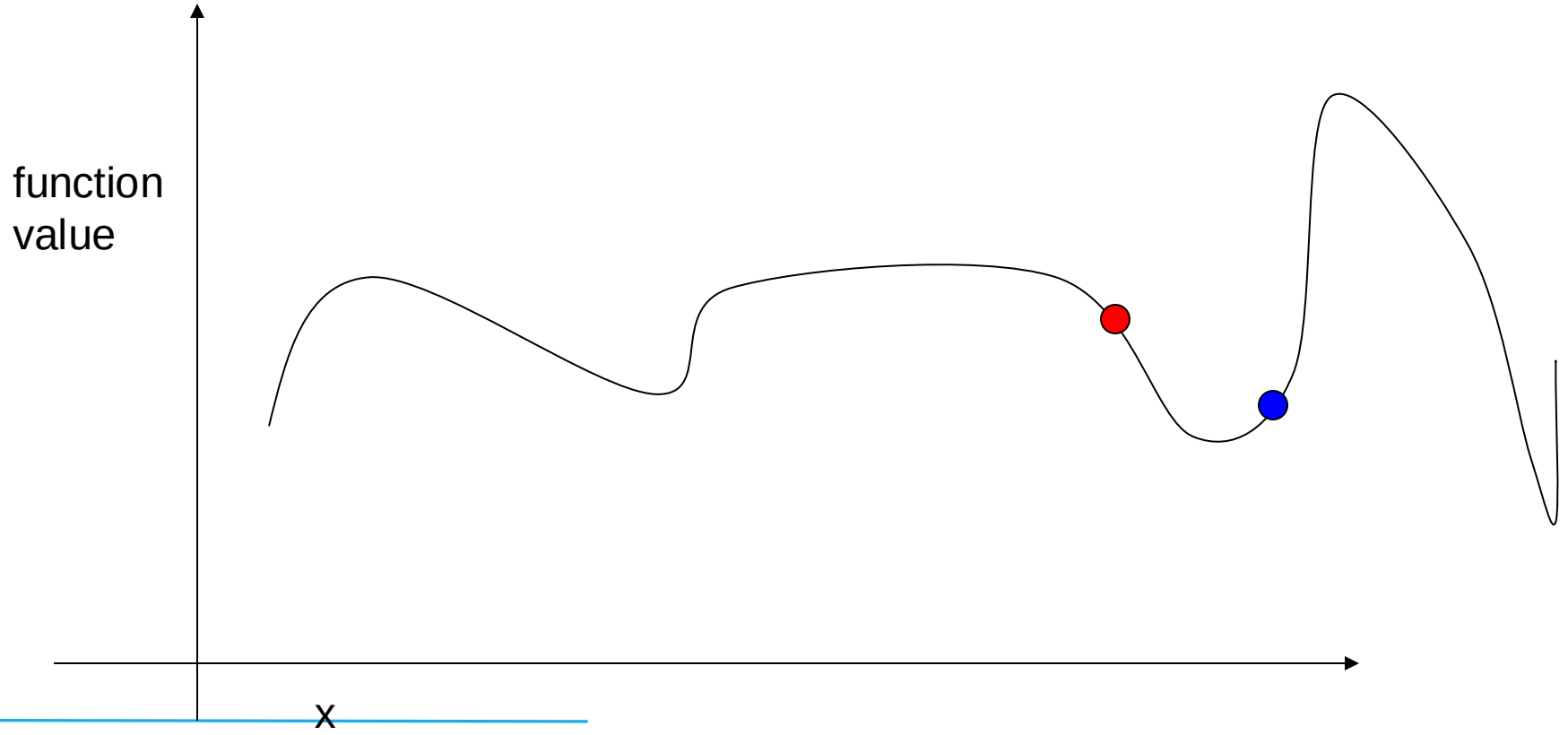
T = High



SIMULATED ANNEALING

Next Step; accept even though lower

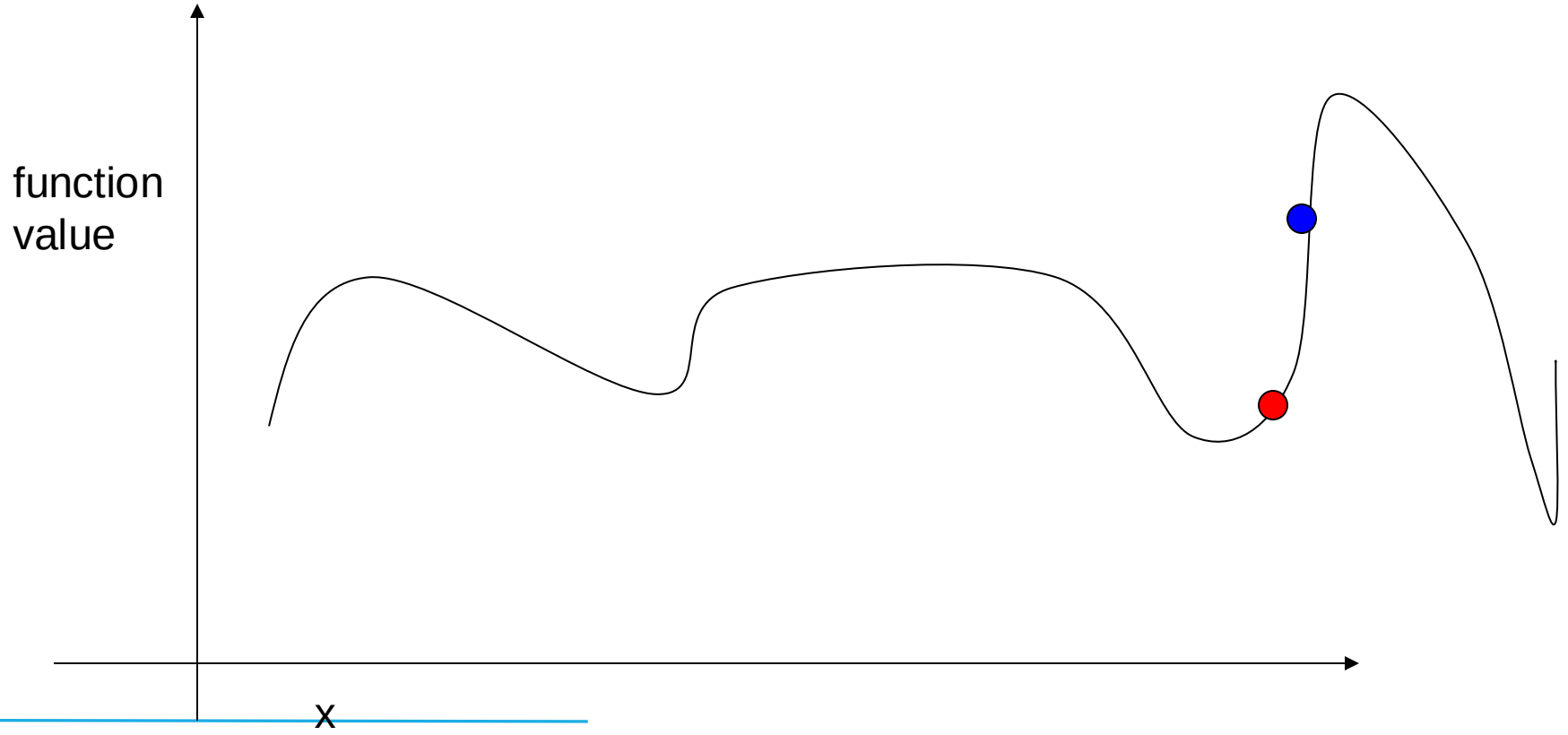
T = High



SIMULATED ANNEALING

Next Step; accept since higher

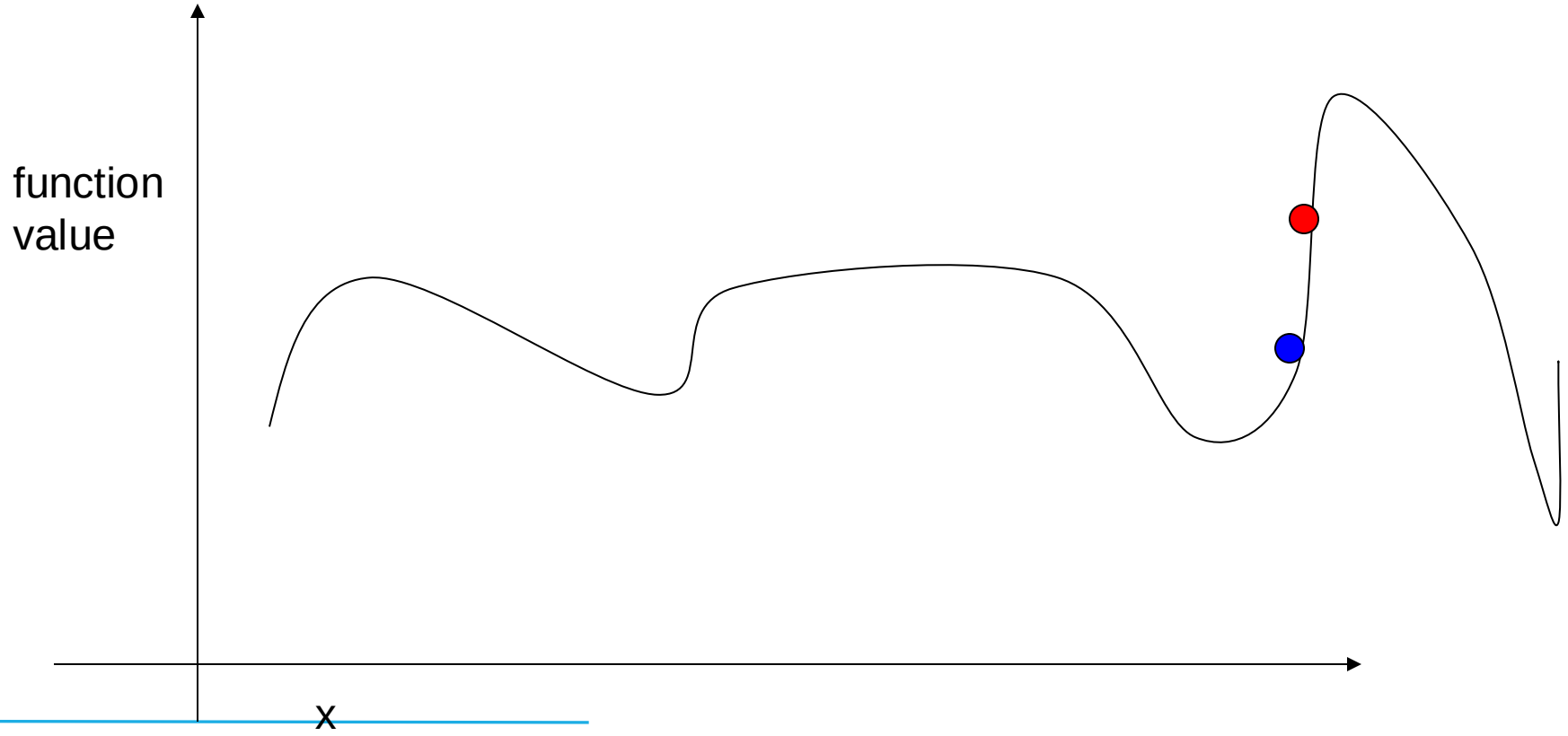
T = Medium



SIMULATED ANNEALING

Next Step; lower, but reject (T is falling)

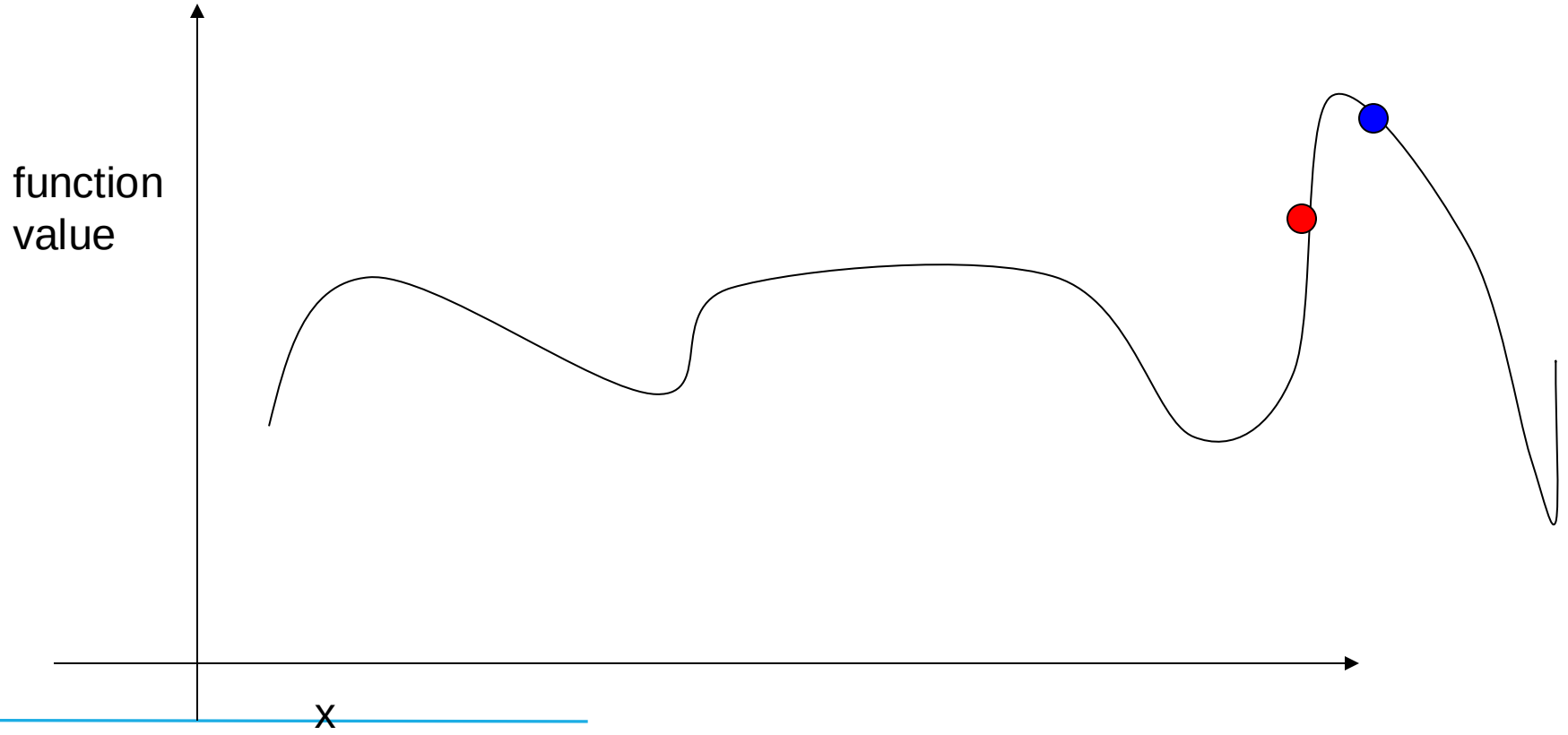
T = Medium



SIMULATED ANNEALING

Next Step; Accept since E is higher

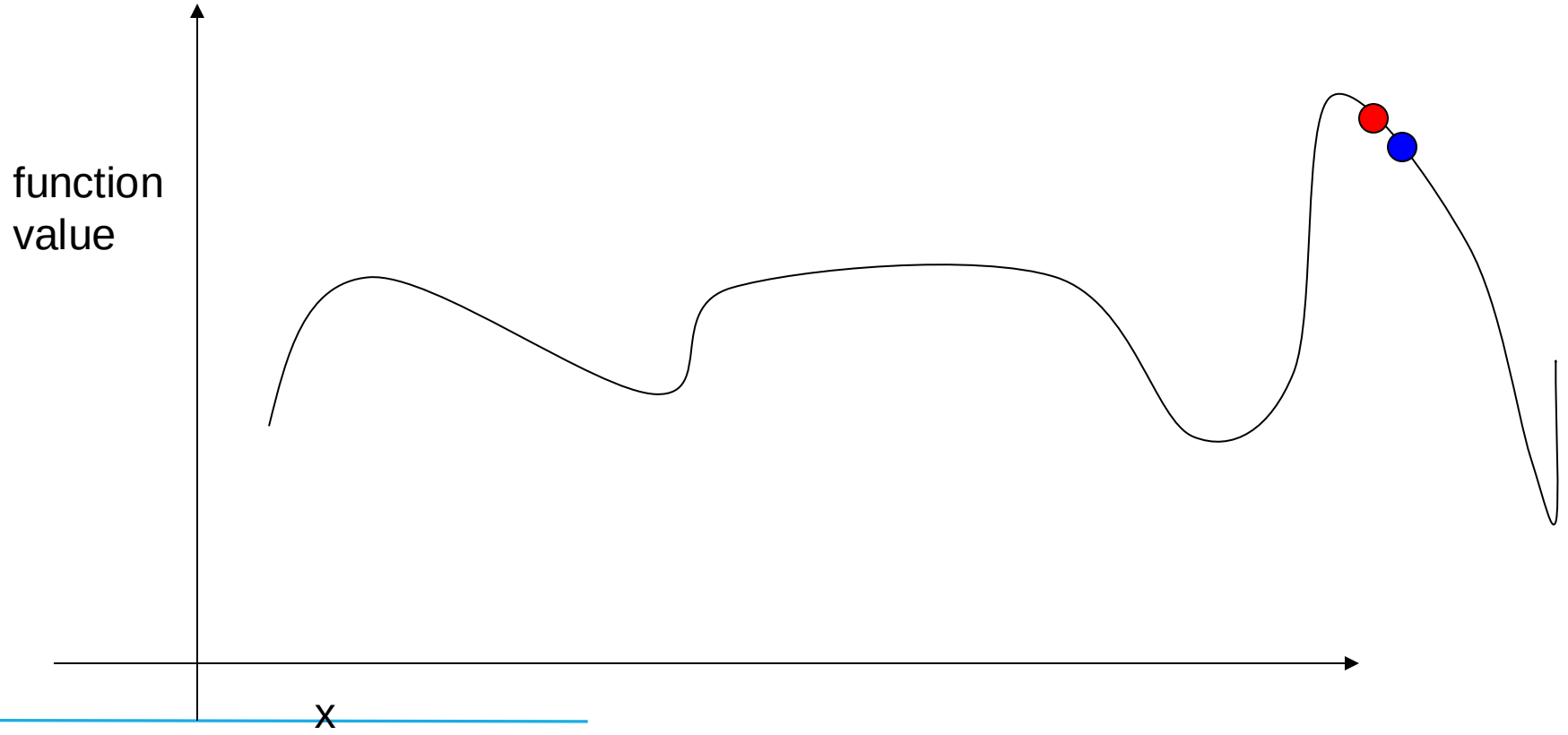
T = Medium



SIMULATED ANNEALING

Next Step; Accept since E change small

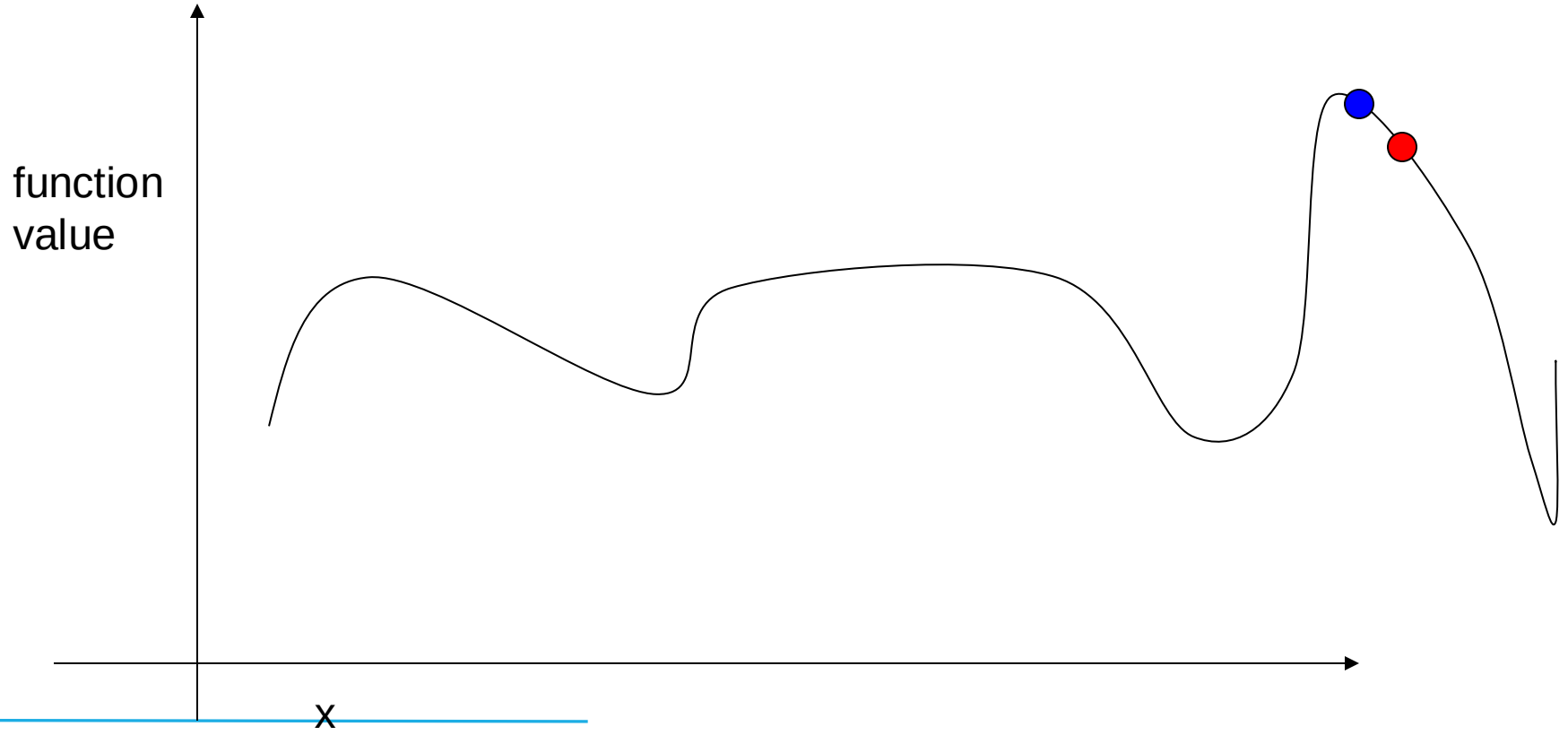
T = Low



SIMULATED ANNEALING

Next Step; Accept since E larget

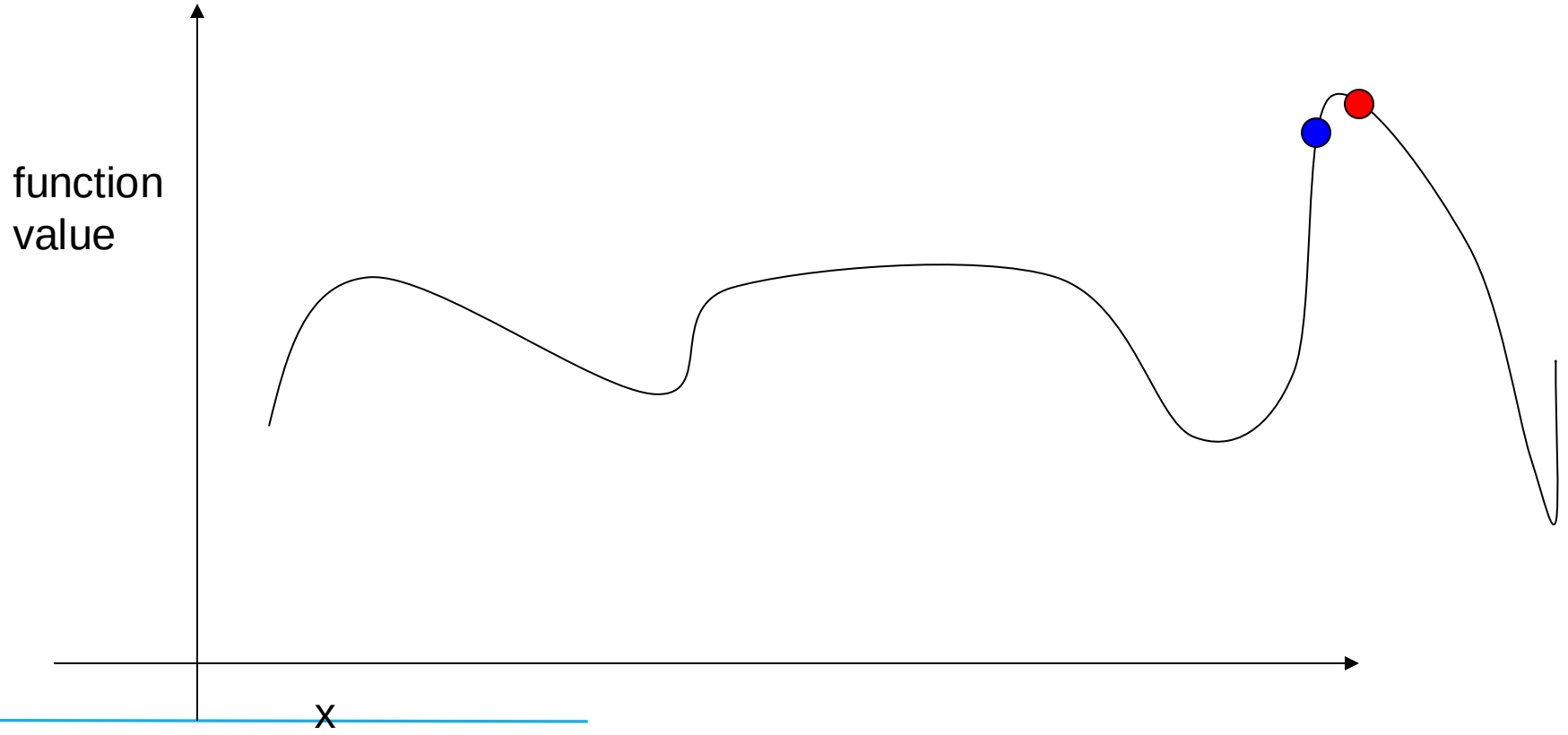
T = Low



SIMULATED ANNEALING

Next Step; Reject since E lower and T low

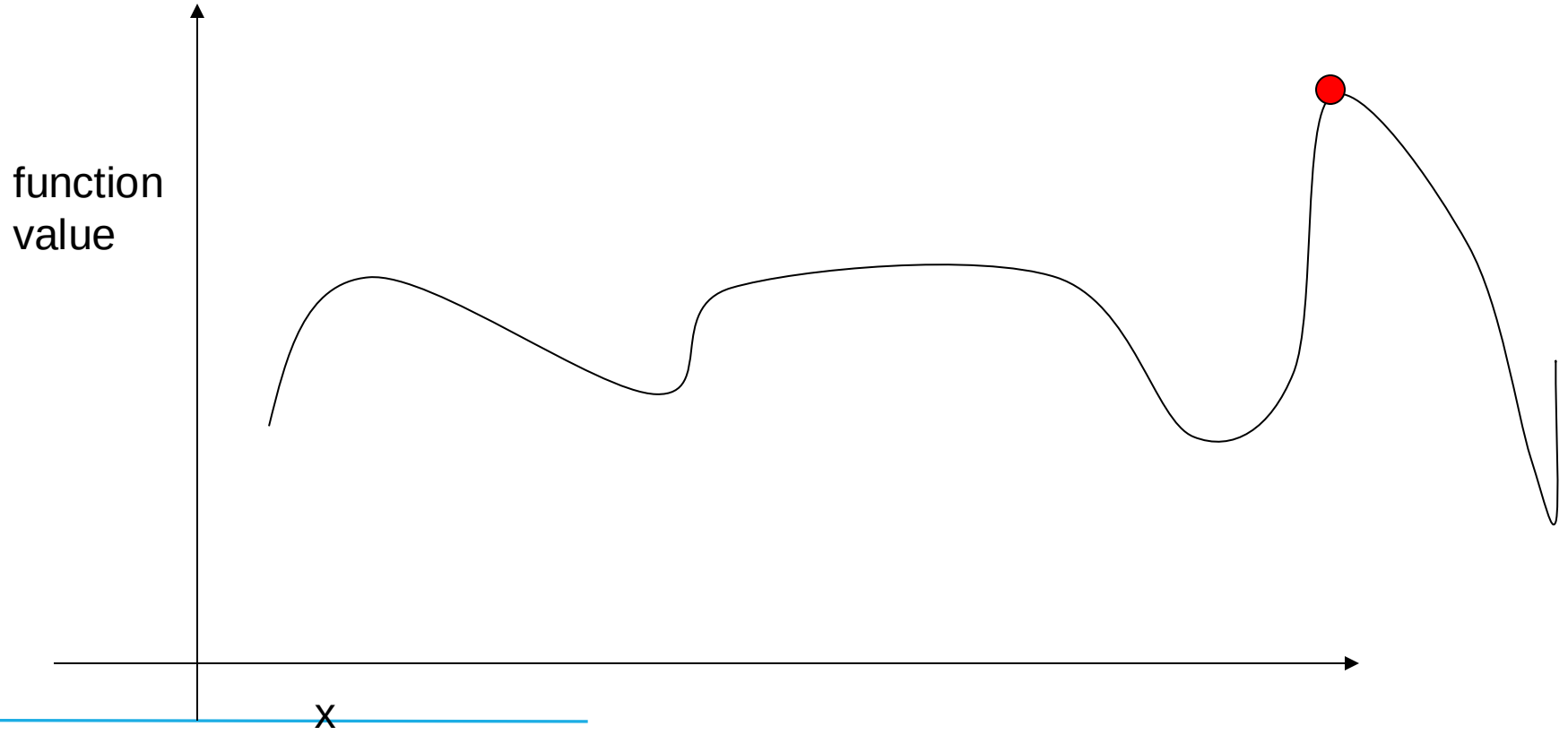
T = Low

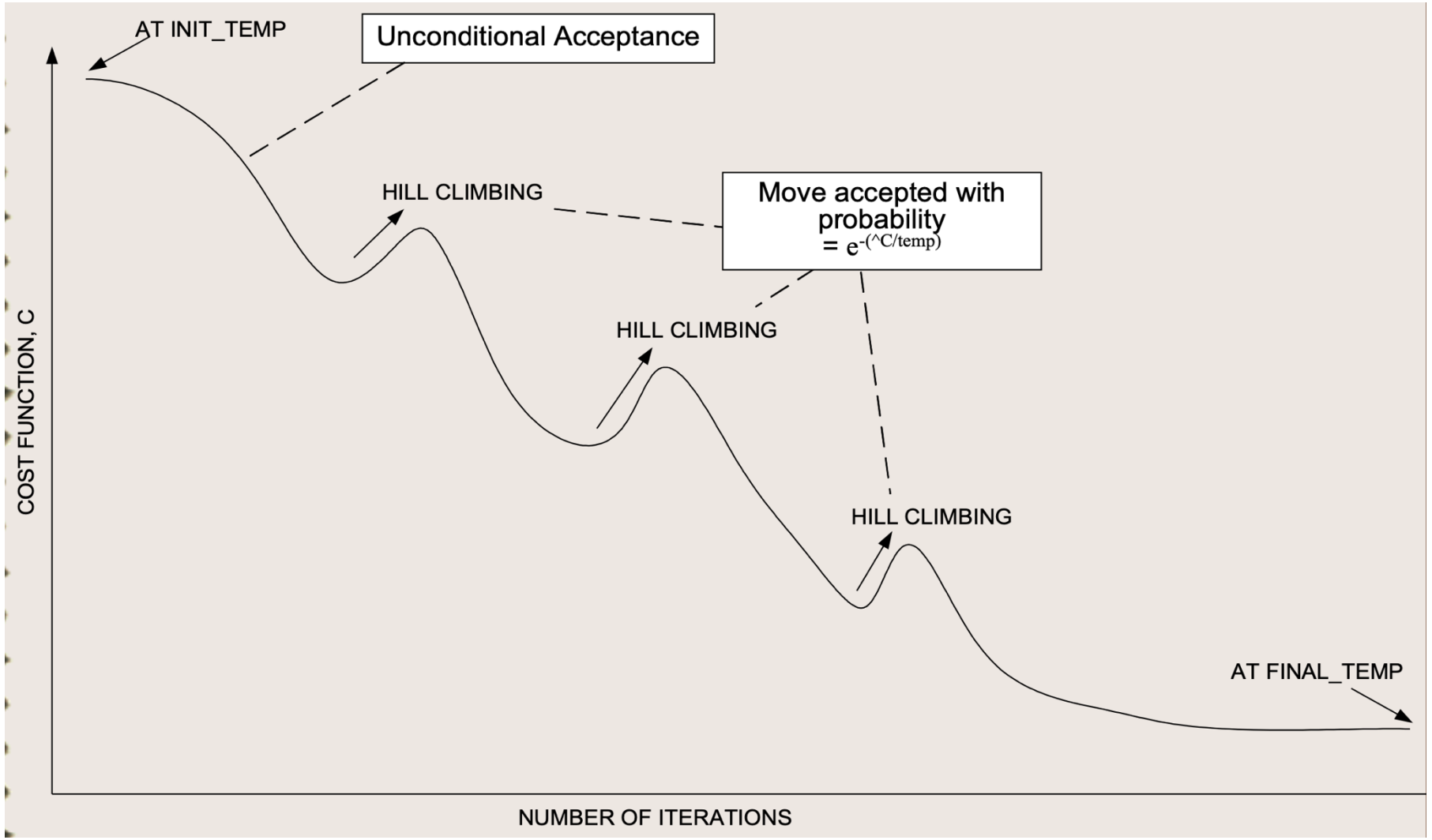


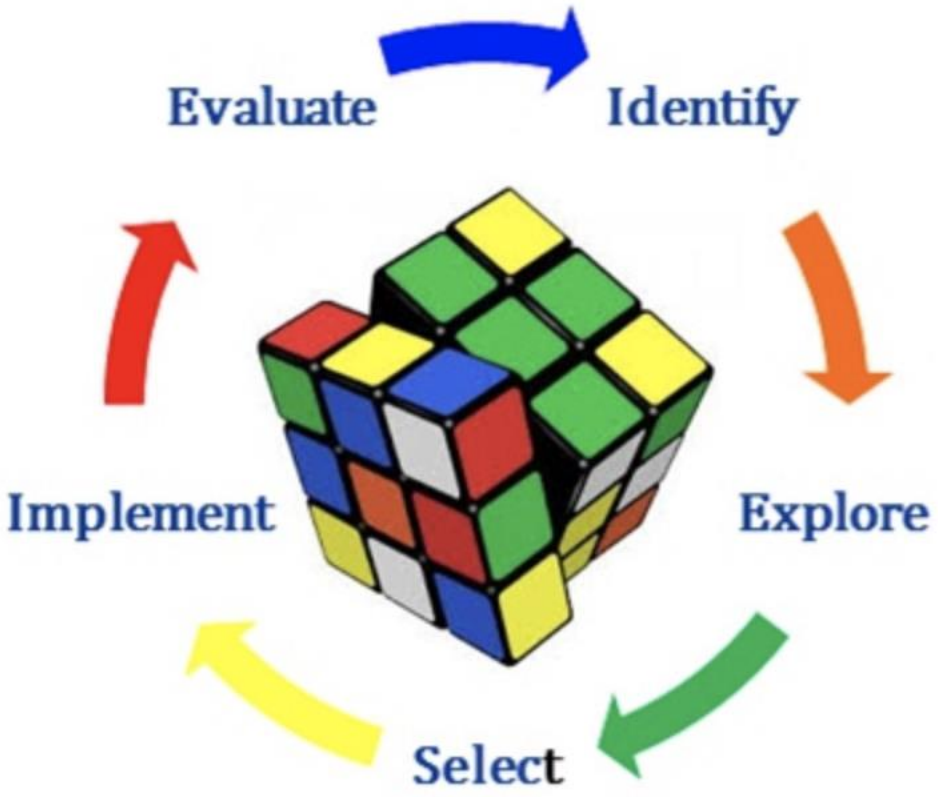
SIMULATED ANNEALING

Eventually converge to Maximum

T = Low




AT INIT_TEMP
Unconditional Acceptance
HILL CLIMBING
Move accepted with probability
= e-(^C/temp)
HILL CLIMBING
HILL CLIMBING
AT FINAL_TEMP
COST FUNCTION, C
NUMBER OF ITERATIONS



META HEURISTICS SEARCH

Adaptive SA

ADAPTATION

Adaptation in SA is in the parameters used by the algorithm:

- The *initial temperature*, the *cooling schedule* and *no. of iterations* per temperature being the most critical.
- Other components such the *cost function* and method of generating *neighborhood* solutions and *acceptance probability* affect the computational costs.

INITIAL TEMPERATURE

Finding the right temperature depends on the type of problem and sometime the instance of the problem.

One can try different values and see which leads to better solutions.

Some researchers suggested doing this adaptively using a search method such as Genetic Algorithm [7].

PROBABILITY OF ACCEPTANCE

The Boltzmann−based acceptance probability takes significant computational time (~1/3 of the SA computations).

The idea of using a lookup table where the exponential calculation are done once for a range of values for change in c and t has been suggested.

Other non−exponential probability formulas have been suggested, for example Johnson et.al.[8] suggested $P(\delta c) = 1 - \delta c / t$

COST FUNCTION

Avoid cost functions that return the same value for many states (e.g number of colors in graph coloring problem). This type of functions doesn't lead the search.

Many problems have some constraints that can be represented in the cost function using penalty terms.

One way to make the algorithm more adaptive is to have a dynamically changing weighting of the penalty terms. So in the initial phase the constraints can be relaxed more than in advanced phases.

To reduce the computational time involved, update functions have been suggested.

ADAPTIVE SA

There have been some attempts at making the selection and control of SA totally adaptive.

One such attempt proposed by Ingber in 1993 [4]. ASA automatically adjusts the algorithm parameters that control the temperature schedule.

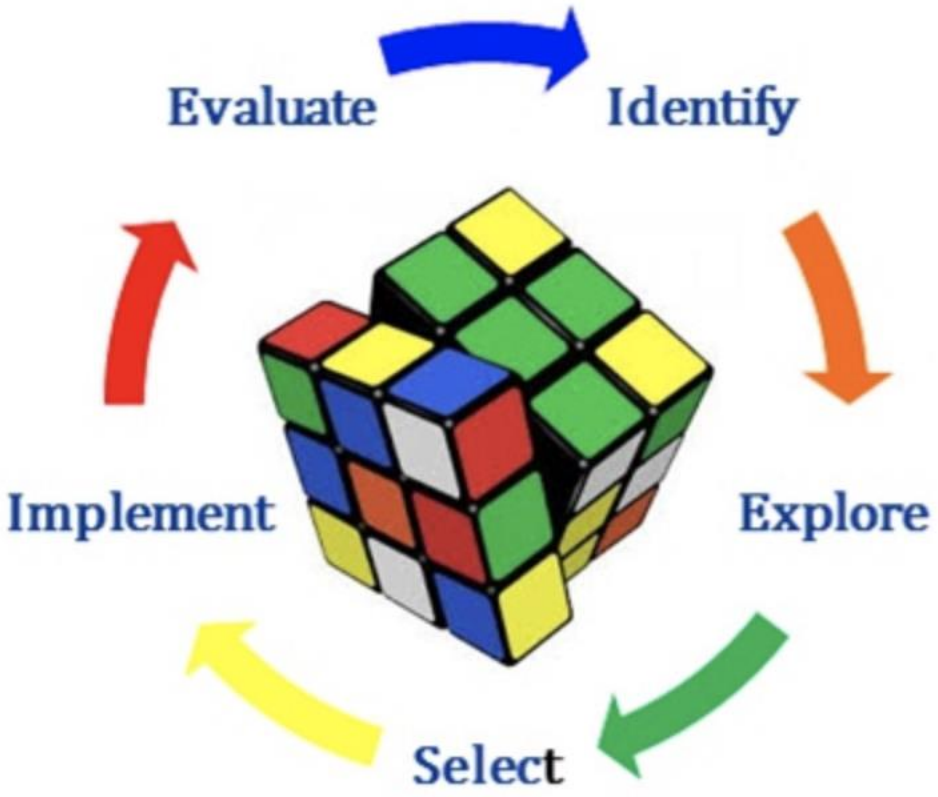
It requires the user to only specify the cooling rate α. All the other parameter are automatically determined.

ADAPTIVE SA

The method uses linear random combination of previously accepted steps and parameters to estimate new steps and parameters.

More information including source code available at:

http://www.ingber.com/



META HEURISTICS SEARCH

Cooperation and SA

COOPERATIVE SA

Cooperative SA (COSA) was proposed by Wendt et al. in 1997 [5],

COSA implements *concurrent* and *synchronous* runs of multiple SA processes,

The concurrent processes are coupled through the *cooperative transitions*,

The cooperative transition replaces the uniform distribution used to select the neighbours.

COOPERATIVE SA

The algorithm manipulates multiple solutions (a *population* of solutions) at once,

Assuming we have five solutions, we start by a randomly chosen population:

$$Pop_0 = [\ S_1,\ S_2,\ S_3,\ S_4,\ S_5\ ]$$

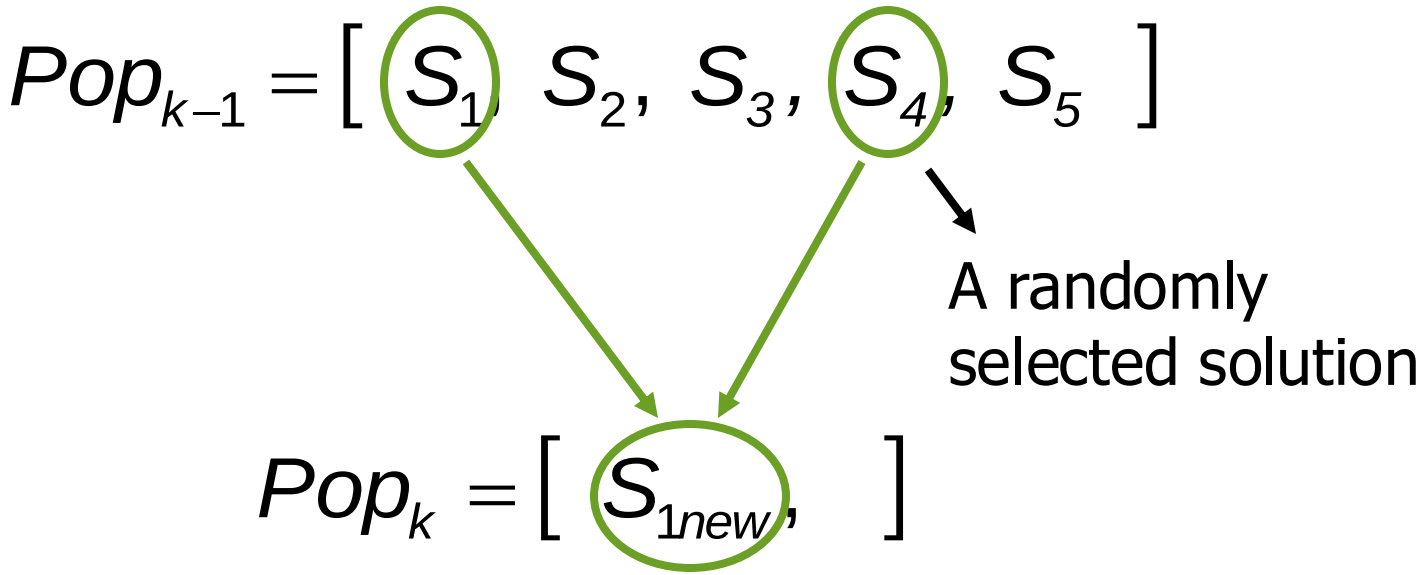
COOPERATIVE SA

The new population is iteratively produced,

Any new solution is cooperatively produced by:

- The previous value of that solution,
- The previous value of a randomly selected solution,

COOPERATIVE SA



COOPERATIVE SA

$$Pop_{k-1} = [\ S_1, S_2, S_3, S_4, S_5\ ]$$

$$Pop_k = [\ S_{1new}, S_{2new},]$$

COOPERATIVE SA

How do we find S_{1new} from S_1 and S_4?

We find the neighbours of S_1 that are closer to S_4 than S_1 itself (using some distance measure),

These neighbours constitute what is known as the *CLOSER* set.

COOPERATIVE SA

The *CLOSER* set is defined as:

$$CLOSER = \{s_k \in N(S_1) \mid d(s_k, S_4) < d(s_k, S_1)\}$$

If the *CLOSER* set is not empty, the new solution is randomly selected from it,

Otherwise, the new solution is randomly selected from the neighbourhood of S_1.

COOPERATIVE SA

The temperature is updated based on the difference of the mean fitness of the new and old populations,

$$\Delta E = E\left(Pop_k\right) - E\left(Pop_{k-1}\right)$$

$$T = \begin{cases} T & ,if \ \ \Delta E < 0 \\ \alpha T & ,otherwise \end{cases}$$

COOPERATIVE SA

Set population size *Pop*,

Initialize Pop_0,

For a determined number of transitions K

- For i = 1 to *Pop*
  - Select a random cooperator $S_j \in Pop_{k-1}$,
  - Generate S_{inew} = cotrans(S_i,S_j),
  - Accept the new solution if it' s better,
  - Otherwise, probabilistically determine if a move to be taken to the new solution and update the solution accordingly.

  end

Update Temperature,

end

COOPERATIVE SA

COSA inherits the idea of population and information exchange form Genetic Algorithms.

It uses *cooperative transitions* instead of GA crossover.

In [5] they showed that using increased cooperation had positive effect on getting better solutions (TSP, Job Shop Scheduling, select of comm. Protocols).

SA AND MACHINE LEARNING

A Reinforcement Learning Method Based on Adaptive Simulated Annealing

Amir F. Atiya, et al

Reinforcement learning (RL) is the theory of learning that acts to maximize some measure of future payoff or reward.

Usually each reward is affected by all actions taken in the past, and that presents a fairly challenging problem involving aspects of Markov processes and dynamic programming.

The RL problem is characterized by a state transition function that probabilistically specifies the next state as a function of the current state and the “action” taken.

The actions represent a “control decision” input.

It is required to find the control strategy that maximizes the expected future aggregate reward.

Because the control decision affects the instantaneous reward, as well as the state transition that will in turn affect future rewards, the resulting optimization problem is by no means simple to solve.

SA is used to find optimal RL parameters that can lead to optimal decisions.

Deep−Learning−Enabled Simulated Annealing for Topology Optimization

Changyu Deng et al

Topology optimization by distributing materials in a domain requires stochastic optimizers to solve highly complicated problems.

However, solving such problems requires millions of finite element calculations with hundreds of design variables or more involved , whose computational cost is huge and often unacceptable.

To speed up computation, here we report a method to integrate deep learning into stochastic optimization algorithm.

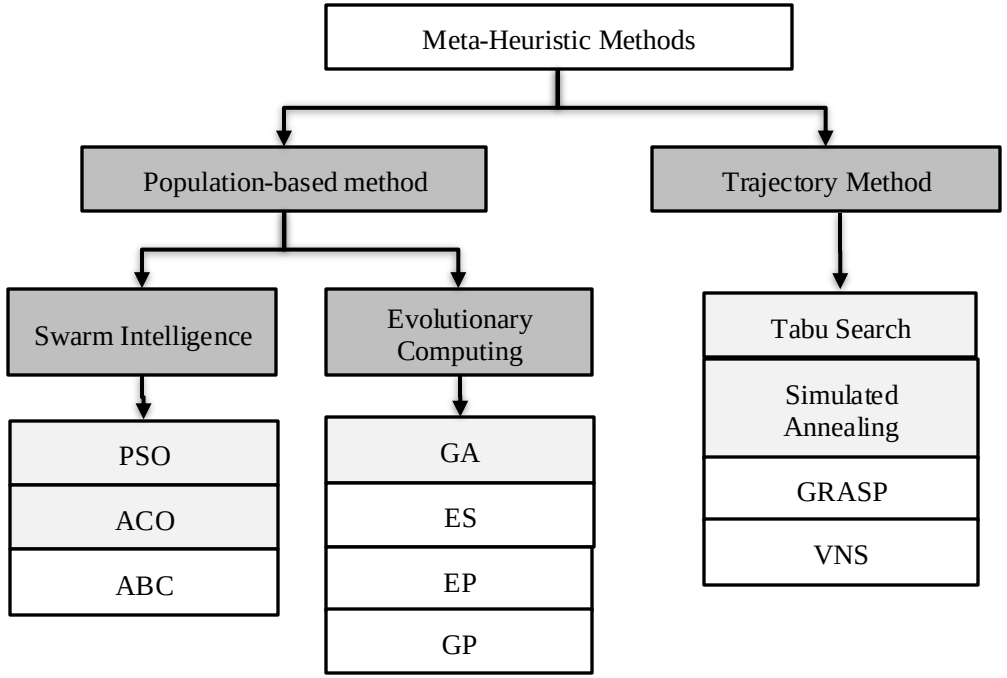
A Deep Neural Network (DNN) learns and substitutes the objective function by forming a loop with Generative Simulated Annealing (GSA).

In each iteration, GSA uses DNN to evaluate the objective function to obtain an optimized solution, based on which new training data are generated;

thus, DNN enhances its accuracy and GSA could accordingly improve its solution in next iteration until convergence. Their algorithm was tested by compliance minimization problems and reduced computational time by over two orders of magnitude. This approach sheds some light on solving large multi−dimensional optimization problems.

TABU SEARCH

META-HEURISTICS

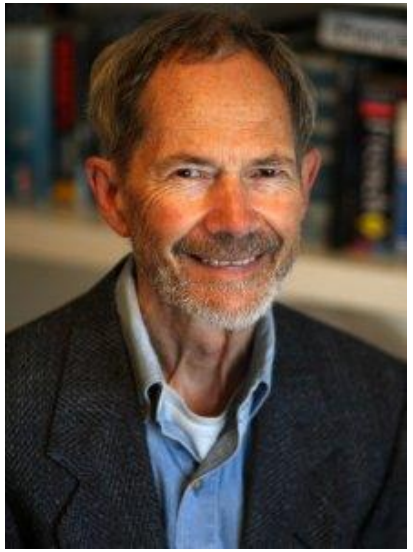


TABU SEARCH

Developed by Fred W. Glover in 1989 [2,3]

- Glover, Fred. "Tabu search—part I." *ORSA Journal on computing* 1.3 (1989): 190-206.
- Glover, Fred. "Tabu search—part II." *ORSA Journal on computing* 2.1 (1990): 4-32.

Among the most cited meta-heuristics used for optimization, [to-date: more than 12000 citations]



TABU SEARCH

It is a meta-heuristic, a general strategy for guiding and controlling inner heuristics.

Memory-based strategies are the hallmark of tabu search approaches

Tabu Search (TS) can be regarded as the combination of **local search strategy** and **memory structures**.

It uses memory structures to:

- Escape local minima,
- Implement an explorative strategy. Avoid revisiting visited nodes.

LOCAL SEARCH STRATEGY

Local Search:

- Start with an initial ***feasible*** solution,
- While termination criterion not met
  - Generate a ***neighbouring solution*** by applying a series of ***local modifications*** (or moves),
  - If the new solution is better
    - Replace the old one,

end

LOCAL SEARCH CHALLENGES

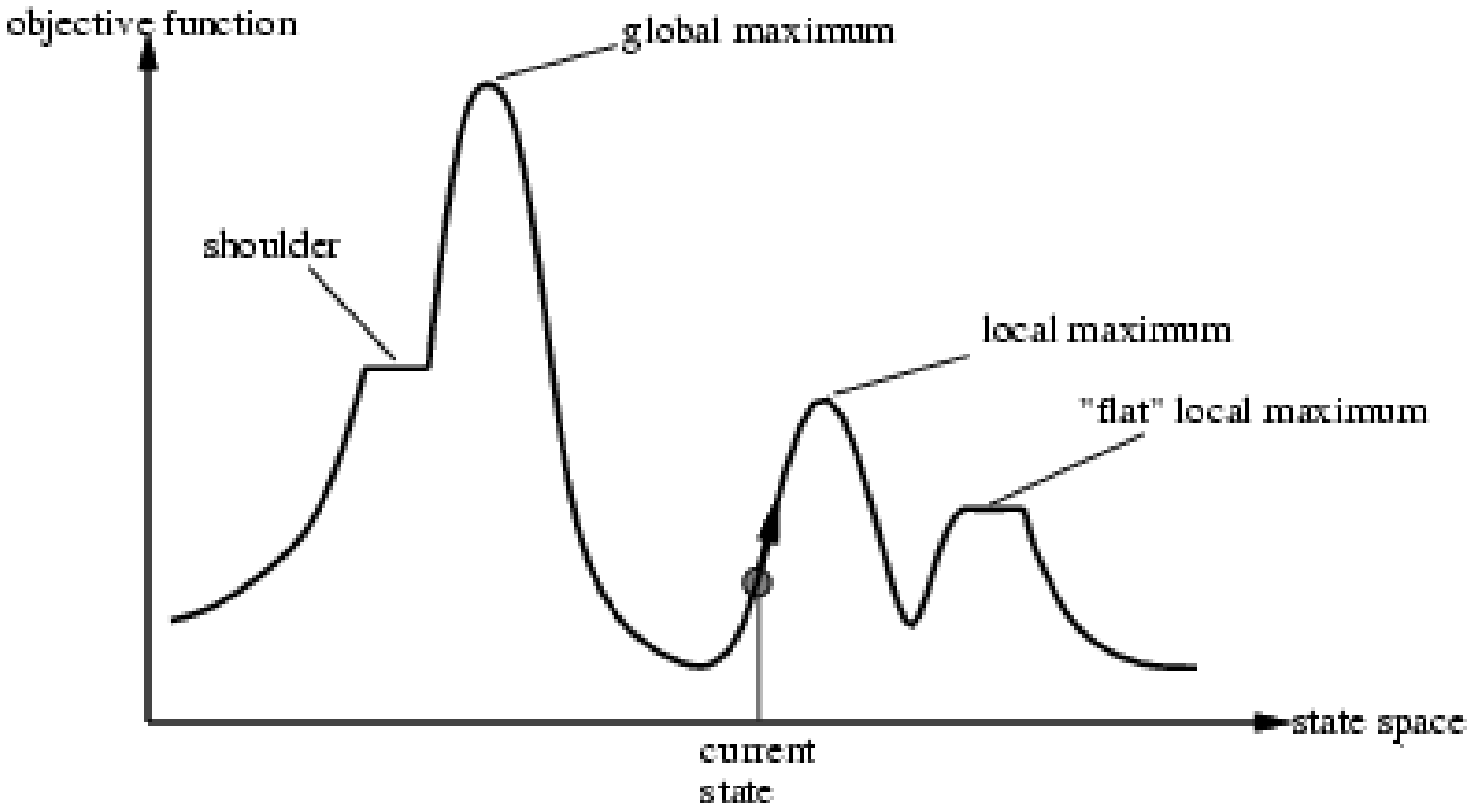
One extreme is to consider all the possible neighbors of the current solutions

- can be computationally demanding

Another extreme is to only consider one neighbor (very limited horizon)

- How can we consider a subset of the search space
- How can we escape from local optimum

LOCAL SEARCH CHALLENGES



TS BASIC IDEA

Forbid or penalize moves which take the solution, in the next iteration, to points in the solution space **previously visited** (*tabu*).

Accepting non-improving solutions deterministically in order to escape from local optima where all the neighbouring solutions are non-improving.

Key concepts:

- Memory
  - Tabu List
  - Tabu Tenure
- Move
- Solution – Initial , Current, best
- Neighborhood
- Termination Condition
- Candidate
- Aspiration list

TS- USE OF MEMORY

Uses of memory Structures:

- Short term memory based on ***recency of occurrence*** to prevent the search from revisiting previously visited solutions;
  - Recency memory can also be used to keep track of good components to return to in order to intensify the search

- Long term Memory based on ***frequency of occurrence*** of solution components from the start of the iterations.
  - It is used to diversify the search and explore unvisited areas of the solution space by avoiding frequently visited components

TS – THE SHORT TERM MEMORY

The short-term memory is known as the ***Tabu list***

This memory usually holds a fixed and limited amount of information,

- Complete solutions, rarely used because of the space requirement,
- Recent moves applied to the current solutions, in order to prevent reverse moves.

Tabu lists length refer to the number of iterations ***T*** for which we keep a certain move (or its attributes) in the list (***tabu tenure***).

The list may record *tabu* active attributes

EXAMPLE OF MOVES

- Toggle variable between 0 and 1
- Swap nodes in a routing tour
- Insert/delete edge in a graph
- Interchange variables

➢ Usually the attributes are recorded in the memory and moves involving the attributes become tabu

TS- NEIGHBORHOOD

- Like other search method, we need a neighborhood structure N(s)
- In selecting a new state, we consider neighbours that are not on the Tabu list N(s) − T(s)
- N(s) can be reduced and modified based on history and knowledge

TERMINATION CONDITIONS

Some stopping conditions could be the following:

1. no feasible solution in the neighborhood of current solution
2. reached the maximum number of iterations allowed.
3. The number of iterations since the last improvement is larger than a specified number.
4. Evidence shows that an optimum solution has been obtained.

TS- CANDIDATE LIST

Helps address some of the search space issues.

Candidate lists are used to reduce the number of solutions examined on a given iterations

They isolate regions of the neighborhood containing moves with desirable features

TS – ASPIRATION

Sometimes it's useful to allow a certain move even if it is tabu,

This could be done to prevent ***stagnation***,

Approaches used to cancel the tabus are referred to as ***aspiration criteria***

TS – BASIC ALGORITHM

Generate an initial solution

While termination criterion not met

◎Choose the best:

$$s' \in N(s) = \{N(s) - T(s)\} + A(s)$$

◎Memorize s' if it improves the best known solution

◎s = s',

◎Update T(s) and A(s)

Tabu list

Aspiration list

end

RECENCY BASED MEMORY

Recency based memory records solution attributes (or elements or components) that have changed “recently”

Selected attributes in recently visited solutions become tabu−active during their tabu tenures

Solutions containing these attributes are classified as tabu

Tabu solutions are excluded from and not revisited during the tabu tenure (a certain period of time = certain number of moves)

$$N^*(x)$$

ASPİRATİON CRİTERİA

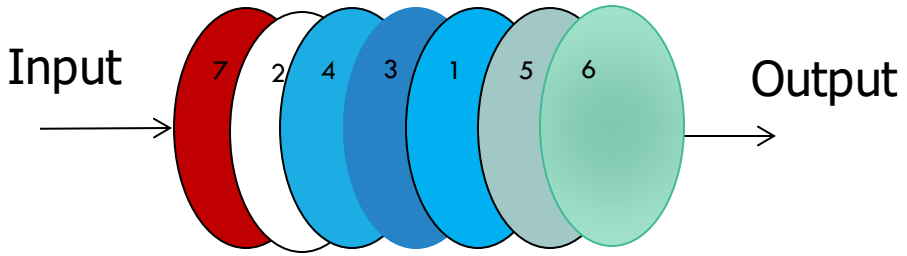
Improved−best or best solution criterion: If a tabu solution encountered at the current iteration is better than the best solution found so far, then its tabu status is overridden

Other aspiration criteria are possible, e.g. setting the tabu tenure shorter for better solutions

FILTER DESIGN

Example

INSULATING MODULES PROBLEM



Filtering Sequence

Ordering of filters determines the overall insulating performance

INSULATING MODULES PROBLEM

- **Problem definition:** Given the type of filters and a function that determines the insulating value of a given ordering, find the ordering of modules (filters) that maximizes the overall insulation

- Representation of a solution for 7 modules (abstraction):

$F(x_o)$=10 units

2	5	7	3	4	6	1

x_0

NEIGHBORHOOD SIZE

- To consider all permutation, the search space will be *n!*
  - for *n=7* it will be *5040*

- **Neighborhood structure:** swapping 2 modules

A solution has 21 neighbors: 7 choose 2 (C_2^7)

$$F(x_1)= 8$$

NEIGHBORHOOD SIZE

As the number of allowed swaps increase, the neighbourhood size increases as well

It is a computationally expensive task to generate the whole neighbourhood,

Sometimes only a reasonable part of the neighbourhood is generated.

DESIGNING TABU LIST DATA STRUCTURE

An important issue is how to design the tabu list without consuming a lot of memory,

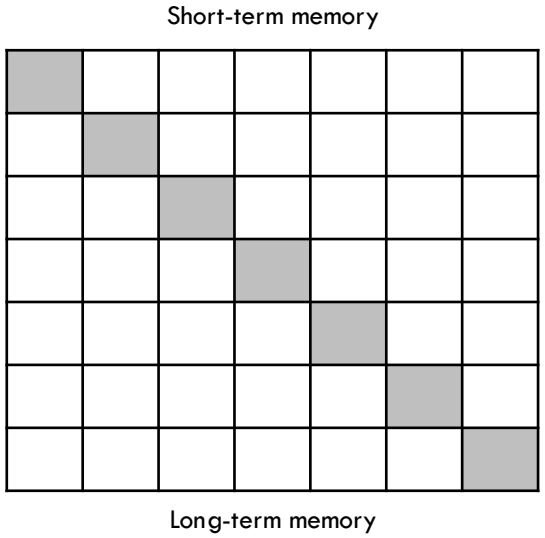
A reasonable choice is to define the tabu list as an ***nxn*** matrix with the value of the ***(i,j)*** element denoting if the swap between filters ***i*** and ***j*** is permitted.

If a swap is performed, the corresponding element in the matrix is set to ***T***, the tabu list length,

After every iteration, all the non-zero elements are decreased by 1,

Hence this swap is considered tabu for ***T*** consecutive iterations.

DESIGNING TABU LIST DATA STRUCTURE



EXAMPLE 1: RECENCY BASED MEMORY AND TABU CLASSİFİCATİON

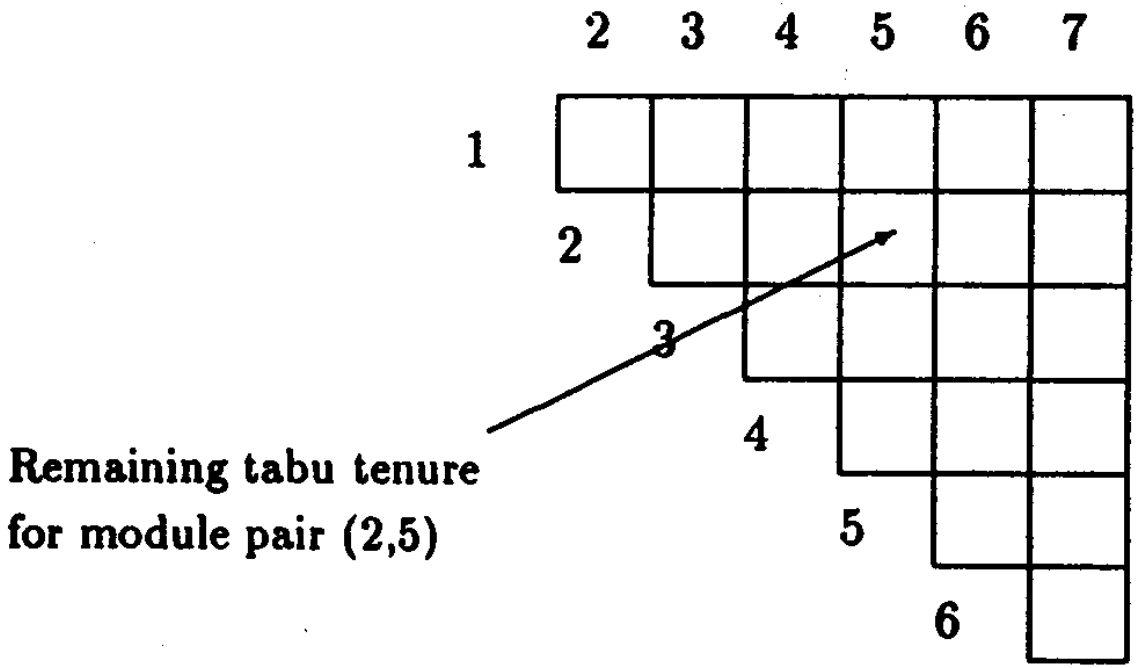
Tabu attributes are selected as most recently made swaps

Tabu tenure is set as 3 iterations

Hence, solutions involving 3 most recent swaps will be classified as tabu

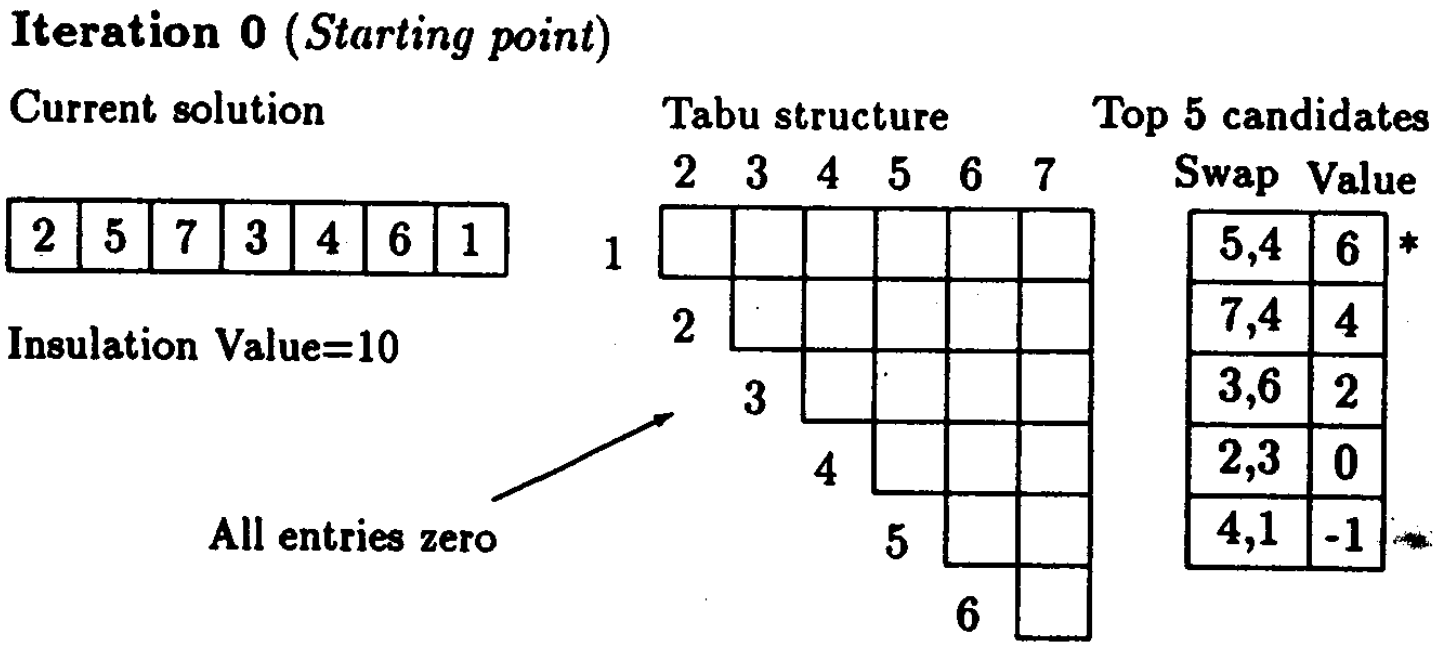
Aspiration criterion is chosen as best solution

EXAMPLE 1: INITIALIZATION OF RECENCY BASED MEMORY



An upper triangle is enough

EXAMPLE 1: ITERATİON 0



"Value" is the gain of swap

EXAMPLE 1: ITERATİON 1

Iteration 1

Current solution

2	4	7	3	5	6	1

Insulation Value=16

Tabu structure

	2	3	4	5	6	7
1						
2						
3						
4				3		
5						
6						

Top 5 candidates

Swap	Value	
3,1	2	*
2,3	1	
3,6	-1	
7,1	-2	
6,1	-4	

EXAMPLE 1: ITERATİON 2

Iteration 2

Current solution

2	4	7	1	5	6	3

Insulation Value=18

Tabu structure

	2	3	4	5	6	7
1		3				
2						
3						
4		2				
5						
6						

Top 5 candidates

Swap	Value	
1,3	-2	T
2,4	-4	*
7,6	-6	
4,5	-7	T
5,3	-9	

Moves (1,3) and (4,5) have respective tabu tenures 3 and 2

No move with a positive gain, hence best (non−tabu) move will be non−improving

EXAMPLE 1: ITERATİON 3

Iteration 3

Current solution

4	2	7	1	5	6	3

Insulation Value=14

Tabu structure

	2	3	4	5	6	7
1		2				
2			3			
3						
4				1		
5						
6						

Top 5 candidates

Swap	Value	
4,5	6	T*
5,3	2	
7,1	0	
1,3	-3	T
2,6	-6	

Move $(4,5)$ has a tabu tenure of 1 iteration

But this move results in the best solution so far

Hence its tabu status is overridden

EXAMPLE 1: ITERATİON 4

Iteration 4

Current solution

5	2	7	1	4	6	3

Insulation Value=20

Tabu structure

	2	3	4	5	6	7
1		1				
2			2			
3						
4				3		
5						
6						

Top 5 candidates

Swap	Value	
7,1	0	*
4,3	-3	
6,3	-5	
5,4	-6	T
2,6	-8	

Best move is $(1,7)$

EXAMPLES FOR TABU RESTRICTIONS

Restrictions:

- A move that involves the same exchange of positions of a tabu move
- A move that involves any of the positions that were involved in a tabu move
- Same for adding and deleting rather than exchanging.

TABU TENURE

Static: choose T to be a constant or as a guideline *(problem size)*

Dynamic: choose T to vary randomly between T_{min} and T_{max} $\sqrt{n}$

Can make the threshold T_{min} *and* T_{max} vary for attributes.

Limitations:

- Fixed length tabu lists cannot always prevent cycling,
- There could exist some cycles with lengths longer than the tabu tenure,
- Some researchers proposed the use of tabu lists with varying lengths during the search,

ASPIRATION CRITERIA

By Default

- A tabu move becomes admissible if it yields a solution that is better than any obtained solution so far

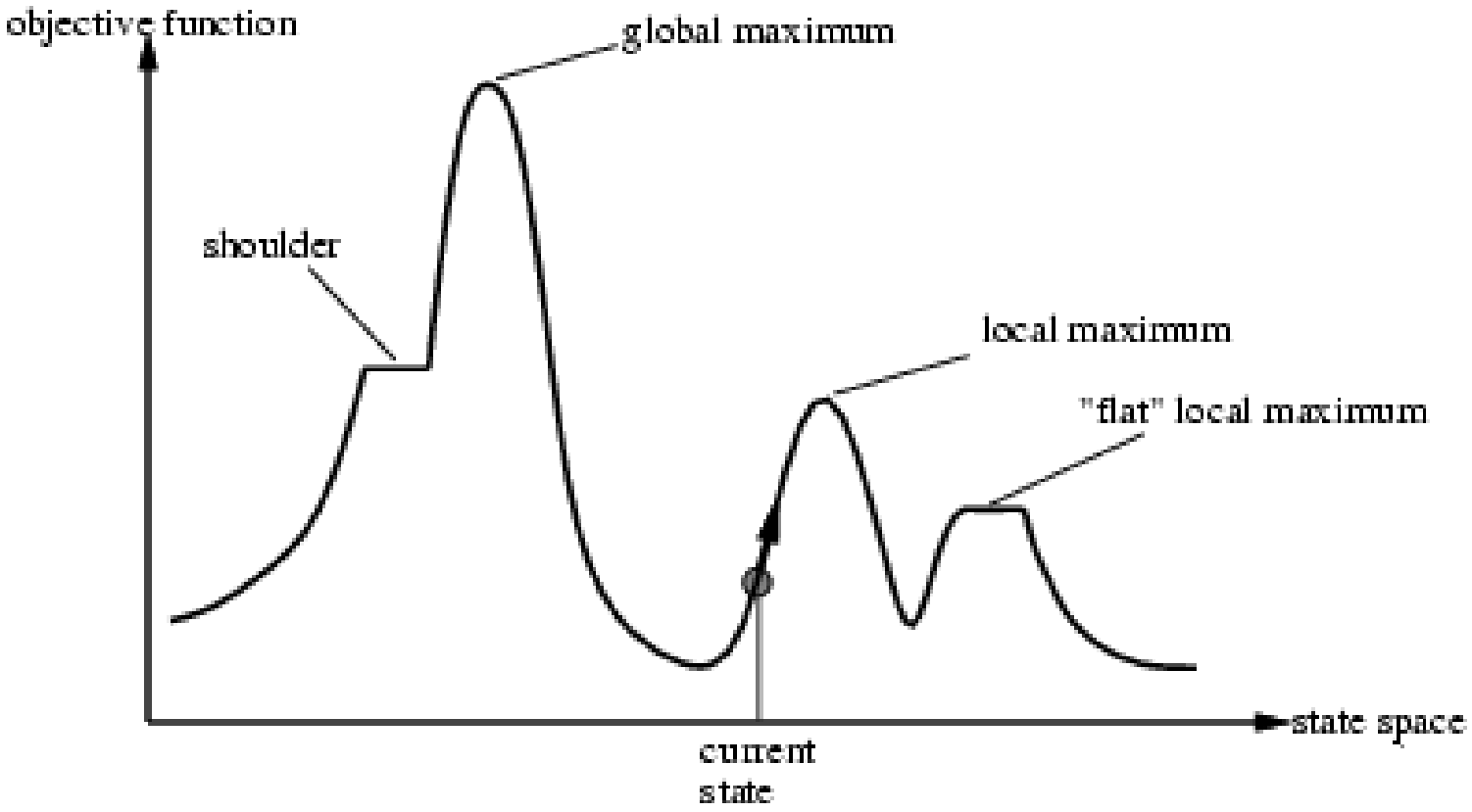
By Objective

- A tabu move becomes admissible if it yields a solution that is better than an aspiration value

By Search Direction

- A tabu move becomes admissible if the direction of the search (improving or non–improving) does not change

TABU SEARCH CHALLENGES



INTENSIFICATION AND DIVERSIFICATION

Two important concepts in Tabu Search are:

- ***Intensification***: refers to the process of exploiting a ***small portion*** of the search space, or penalizing solutions that are ***far from current solution***.

- ***Diversification***: refers to forcing the search into previously ***unexplored areas*** of the search space, or penalizing solutions that are ***close to current***.

INTENSIFICATION

Intensification is not always necessary as sometimes the search process is thorough enough

It can be based on ***recency memory***

Idea:

- The memory holds the number of consecutive iterations in which certain components always appear in the current solution without interruption.
- After a while, one could stop the current search process and move into an intensification phase,
- This phase tries to locally optimize the best known solution while keeping those components intact.

DIVERSIFICATION

Despite the use of tabu lists, TS may tend to be too local sometimes,

This problem causes TS to miss some good solutions in unexplored search space areas,

Diversification aims to overcome this problem.

Having a proper diversification strategy is very critical to the success of the TS

It is based on a long-term memory referred to as the *frequency memory*,

The memory holds the total number of iterations (since the beginning of the search) in which certain components always appear in the current solution or are involved in the selected moves,

DIVERSIFICATION

Two major approaches exist for applying the diversification process:

- ***Restart diversification***: forces components that rarely appear in the current solution and restart the search from these points,

- ***Continuous diversification***: bias the evaluation of possible moves by adding to the objective function a term related to component frequency. Thus incorporating diversification into the regular search process.

- One can use the approach of penalizing solutions that are close to current solution

N-QUEENS PROBLEM

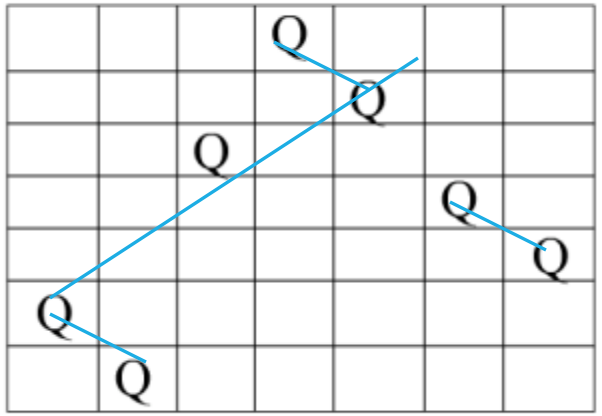
N-QUEENS PROBLEM

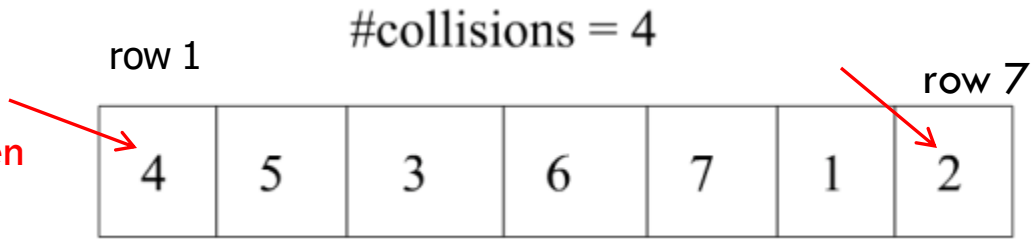
The n-queens problem consists of placing n queens on a n x n chessboard in such a way that no two queens capture each other.

In order for this to happen,

- no two queens should be placed
  - on the same row,
  - on the same column, or
  - on the same diagonal,
- If two queens are placed such that they are able to capture each other,

it is said that a "collision" has occurred.





Iteration 0 (Starting Point)

Current solution

4	5	3	6	7	1	2

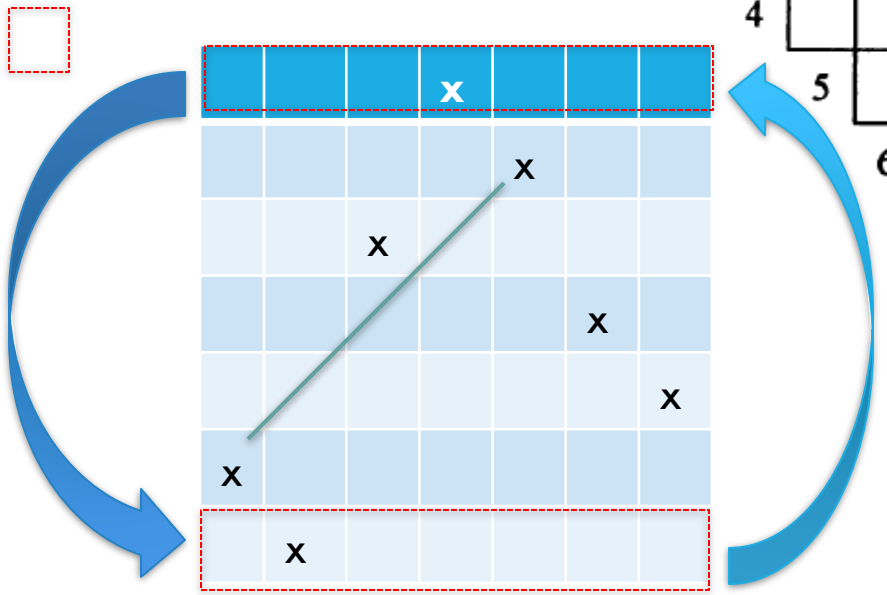
Number of collisions = 4

Tabu structure

	2	3	4	5	6	7
1						
2						
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
1	7	-2	*
2	4	-2	
2	6	-2	
5	6	-2	
1	5	-1	

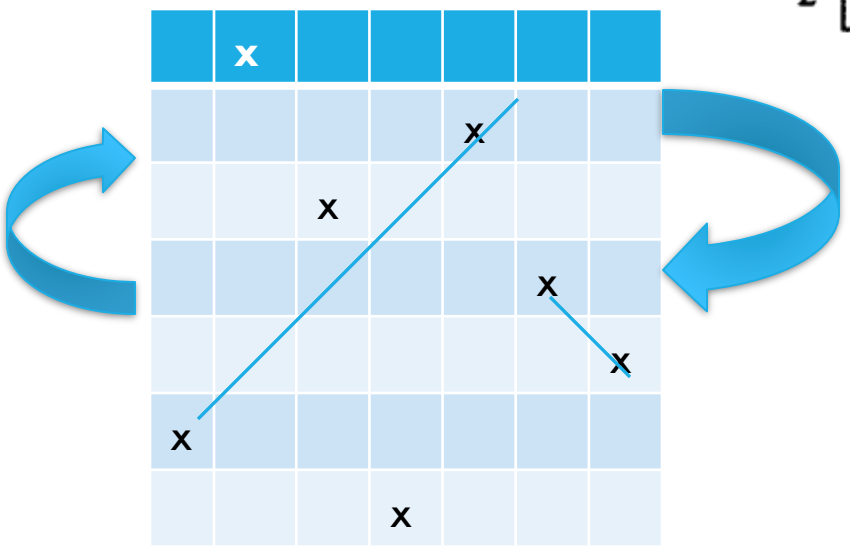


Iteration 1

Current solution

2	5	3	6	7	1	4

Number of collisions = 2



Tabu structure

	2	3	4	5	6	7
1						3
2						
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
2	4	-1	*
1	6	0	
2	5	0	
1	2	1	
1	3	1	

Iteration 2

Current solution

2	6	3	5	7	1	4

Number of collisions = 1

Tabu structure

	2	3	4	5	6	7
1						2
2			3			
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
1	3	0	*
1	7	1	T
2	4	1	T
4	5	1	
6	7	1	

Iteration 3

Current solution

3	6	2	5	7	1	4

Number of collisions = 1

Tabu structure

	2	3	4	5	6	7
1		3				1
2			2			
3						
4						
5						
6						

Top 5 candidates

Swap		Value	
1	3	0	T
1	7	0	T
5	7	1	*
6	7	1	1
1	2	2	

Iteration 4

Current solution

3	6	2	5	4	1	7

Number of collisions = 2

Tabu structure

	2	3	4	5	6	7
1		2				
2			1			
3						
4						
5						3
6						

Top 5 candidates

Swap		Value	
4	7	-1	*
5	7	-1	T
1	5	0	
2	5	0	
2	4	2	T

Iteration 5

Current solution

3	6	2	7	4	1	5

Number of collisions = 1

Tabu structure

	2	3	4	5	6	7
1		1				
2						
3						
4						3
5						2
6						

Top 5 candidates

Swap		Value	
1	3	-1	T*
5	6	-1	
5	7	0	T
1	6	0	
1	7	2	

Iteration 26

Current solution

1	3	6	2	7	5	4

Number of collisions = 1

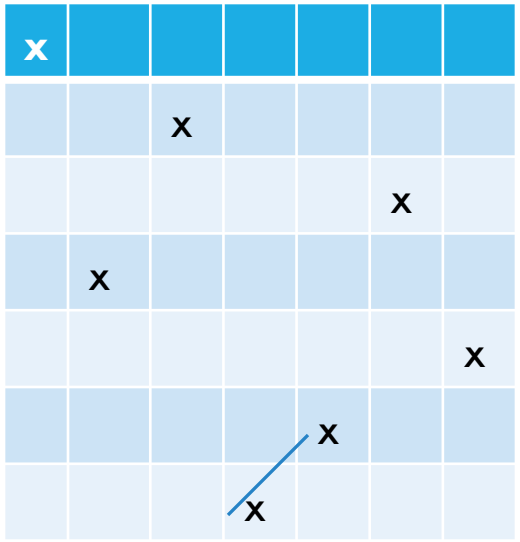
Tabu structure

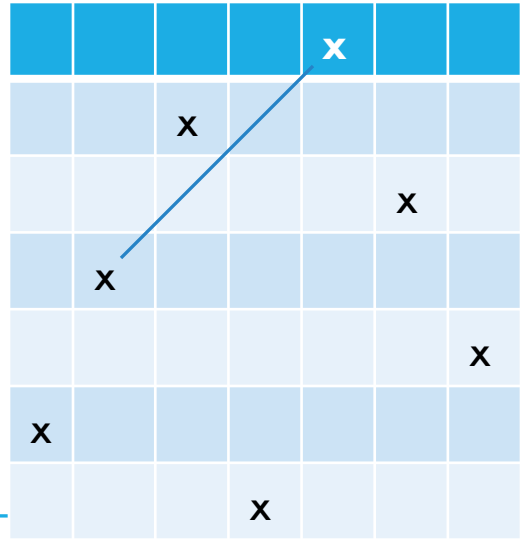
	1	2	3	4	5	6	7
1						3	
2							
3	5					2	
4		3					1
5	2			4			
6	1				2		
7		4	3	1			

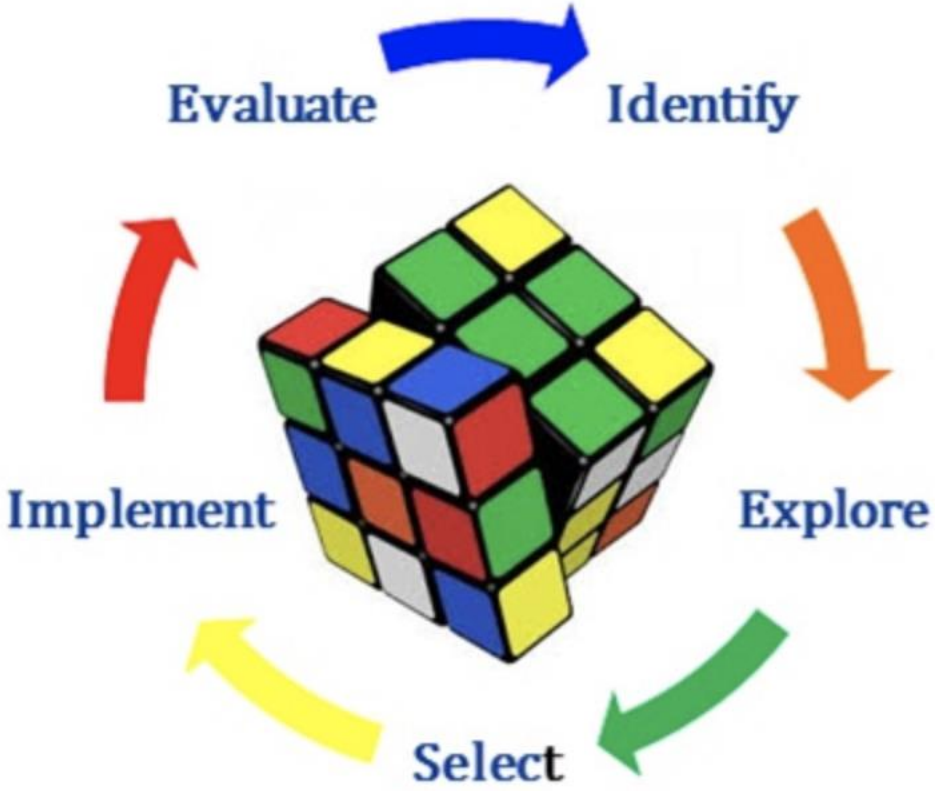
Frequency

Top 5 candidates

Swap		Value	Penalized Value	
1	6	0	1	T
1	3	1	6	
1	5	1	3	*
2	7	1	5	
3	7	1	4	



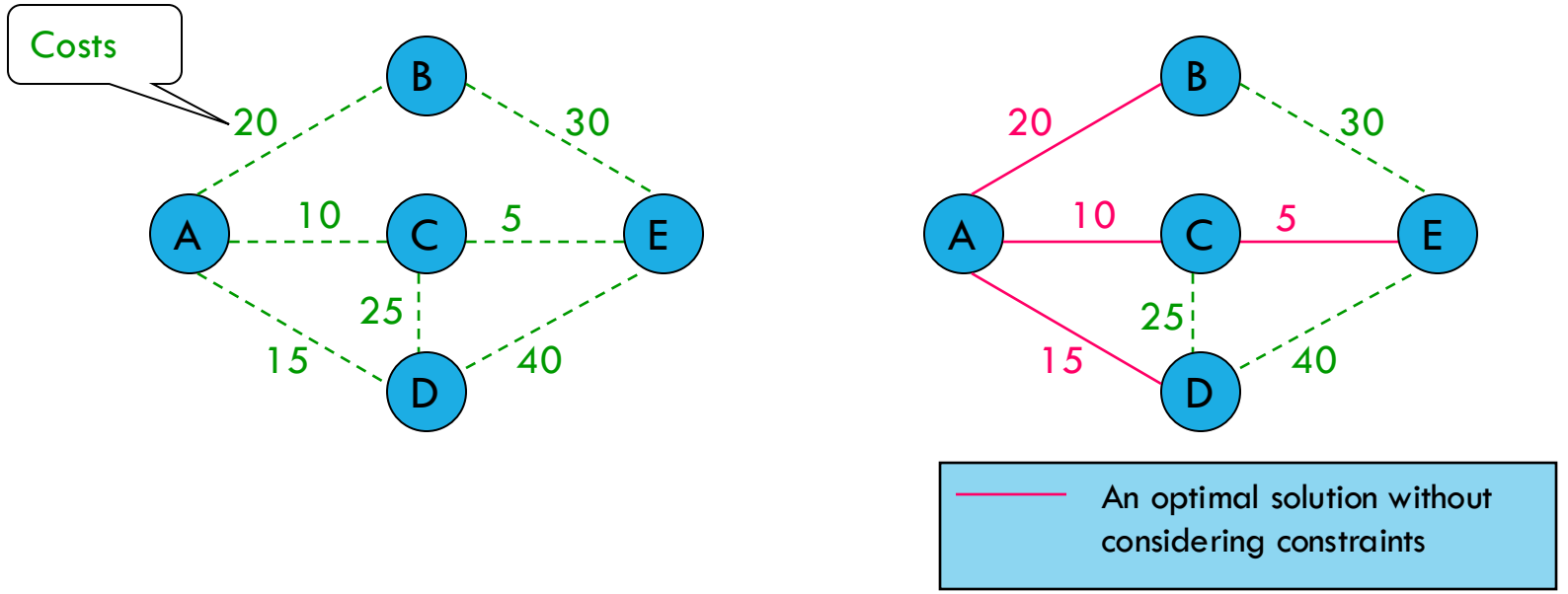




MINIMUM SPANNING TREE

EXAMPLE [5]

- Minimum spanning tree problem with constraints.
- *Objective: Connects all nodes with minimum costs*

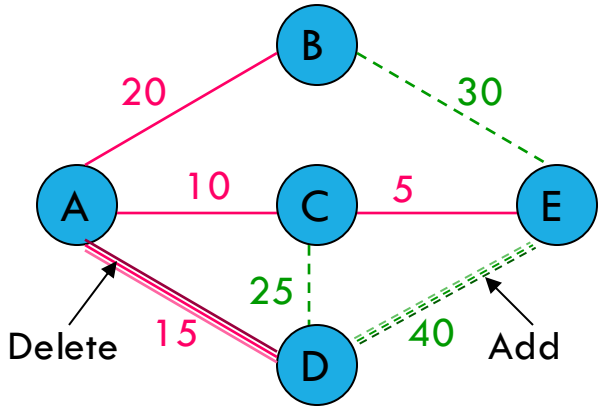


Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

EXAMPLE

Iteration 1

Cost=50+200 (constraint penalties)



Add	Delete	Cost
BE	CE	75+200=275
BE	AC	70+200=270
BE	AB	60+100=160
CD	AD	60+100=160
CD	AC	65+300=365
DE	CE	85+100=185
DE	AC	80+100=180
DE	AD	**75+0=75**

New cost = 75 (iteration 2)

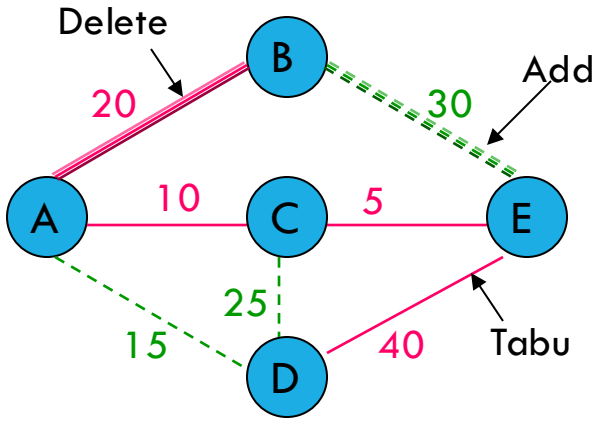
(local optimum)

Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

EXAMPLE

Tabu list: DE

Iteration 2 Cost=75



Add	Delete	Cost
AD	DE*	Tabu move
AD	CE	85+100=185
AD	AC	80+100=180
BE	CE	100+0=100
BE	AC	95+0=95
BE	AB	**85+0=85**
CD	DE*	60+100=160
CD	CE	95+100=195

** **A tabu move will be considered only if it would result in a better solution than the best trial solution found previously (Aspiration Condition)***

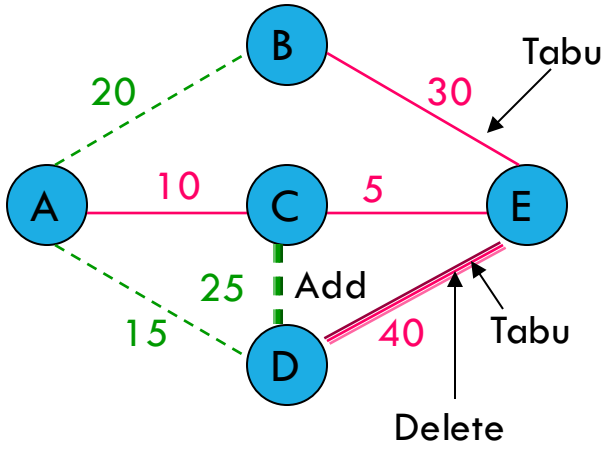
Iteration 3 new cost = 85 Escape local optimum

Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

EXAMPLE

Tabu list: DE & BE

Iteration 3 Cost=85



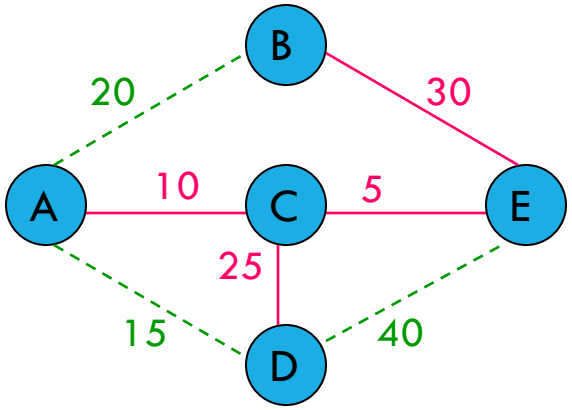
Add	Delete	Cost
AB	BE*	Tabu move
AB	CE	100+0=100
AB	AC	95+0=95
AD	DE*	60+100=160
AD	CE	95+0=95
AD	AC	90+0=90
CD	DE*	**70+0=70**
CD	CE	105+0=105

**** A tabu move will be considered only if it would result in a better solution than the best trial solution found previously (Aspiration Condition)***

Iteration 4 new cost = **70** Override tabu status

Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

EXAMPLE



Optimal Solution

Cost = 70

Additional iterations only find inferior solutions

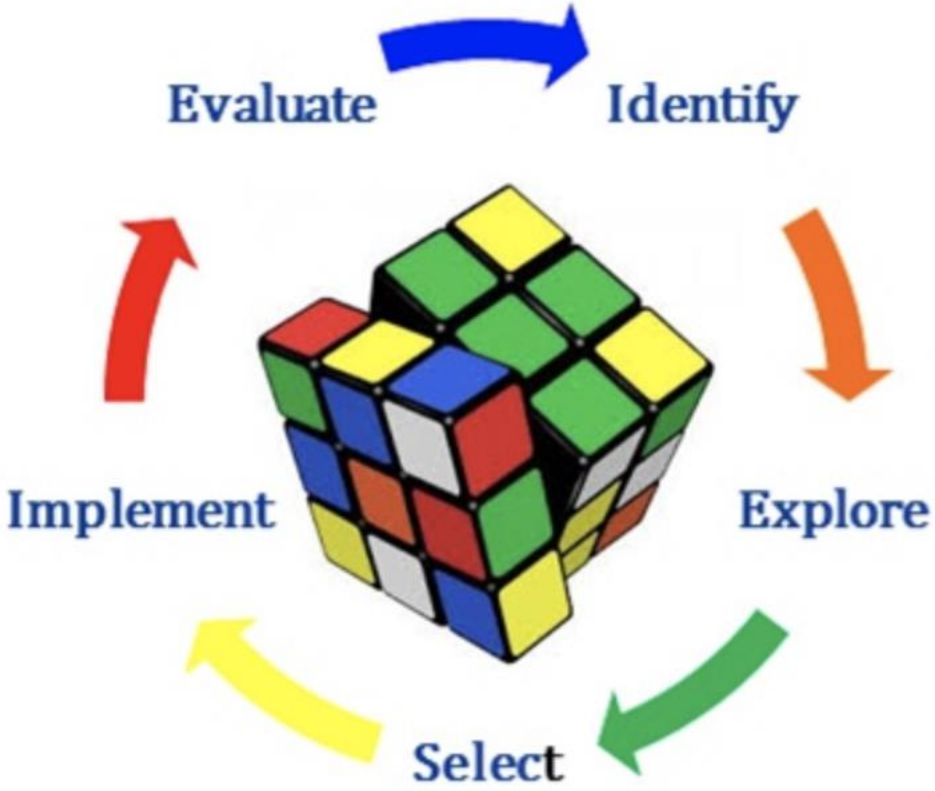
PROS AND CONS

Pros:

- Allows non-improving solution to be accepted in order to escape from a local optimum
- The use of Tabu list
- Can be applied to both discrete and continuous solution spaces
- For larger and more difficult problems (scheduling, quadratic assignment and vehicle routing), tabu search obtains solutions that rival and often surpass the best solutions previously found by other approaches [1].

Cons:

- Too many parameters to be determined
- Number of iterations could be very large
- Global optimum may not be found, depends on parameter settings



TABU SEARCH

Assignment Problem

ASSIGNMENT PROBLEM

Linear Assignment Problem formulated by Hanan and Kurtzberg [1972],

Involves the assignment of *n* objects to *n* sites.

For each assignment, there is a related cost, c_{ij} , of assigning object *i* to site *j*.

The objective is to assign each object to one and only one site in such a manner that minimizes the sum of each assignment cost, i.e., the total cost.

ASSIGNMENT PROBLEM

Mathematically, the above problem can be formulated as follows:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

over all permutations of {1, 2, . . . , n},

Subject to

$$\sum_{i=1}^{n} x_{ij}, = 1, j = 1,..n, \sum_{j=1}^{n} x_{ij} = 1, i = 1,..n$$

$$x_{ij} = 0 or 1$$

Notice that each set of assignments is a permutation of a set on n integers; hence, there are n! distinct permutations from which to choose the optimal assignment

APPLICATIONS

Assignment of offices or services in buildings (e.g. university campus, hospital, etc.),

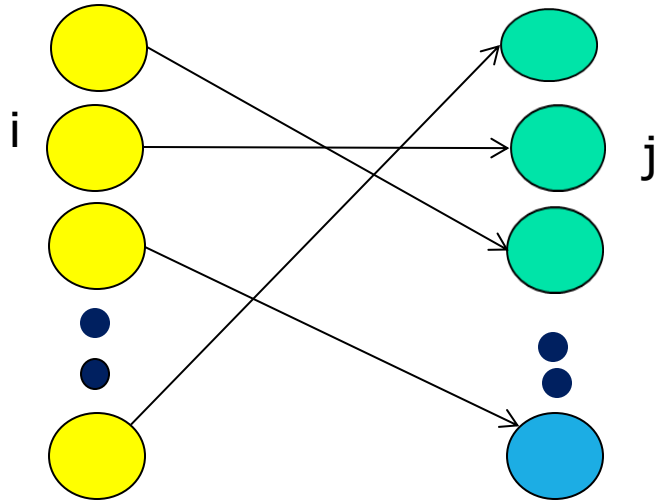
Assignment of the departure gates to airplanes in an airport, the placement of logical modules in electronic circuits,

Distribution of the files in a database

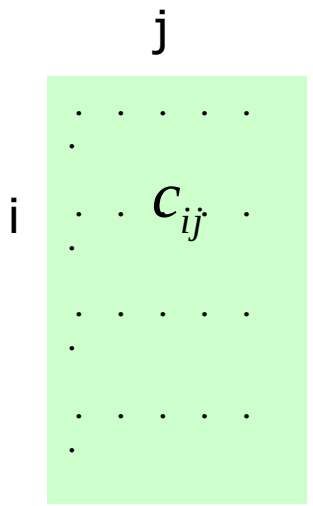
Placement of the keys of keyboards of typewriters.

LINEAR ASSIGNMENT

Set of Objects (persons, facility, etc)



Set of sites Jobs, locations, etc



Bigraph: is a **graph** whose vertices can be divided into two disjoint sets

QUADRATIC ASSIGNMENT PROBLEM (QAP)

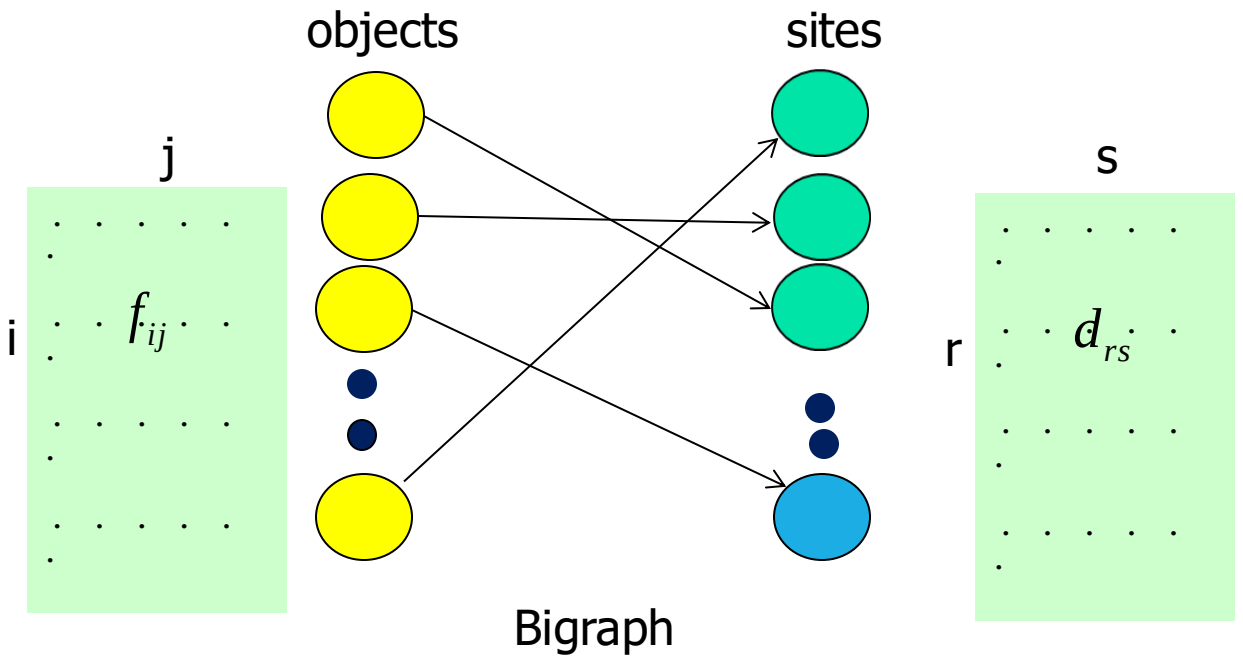
Given n objects and the flows f_{ij} between object i and object j ($i, j = 1 \ldots n$), and given n sites with distance d_{rs} between the sites r and s ($r, s = 1 \ldots n$),

The problem deals with placing the n objects on the n sites so as to minimize the sum of the products, flows × distances.

Mathematically, this is equivalent to finding a permutation **p**, whose $p_i i'th$ component denotes the object assigned to site i, which minimizes

$$\sum_{i=1}^{n} \sum_{j=1}^{n} f_{p_i p_j} d_{ij}$$

QAP



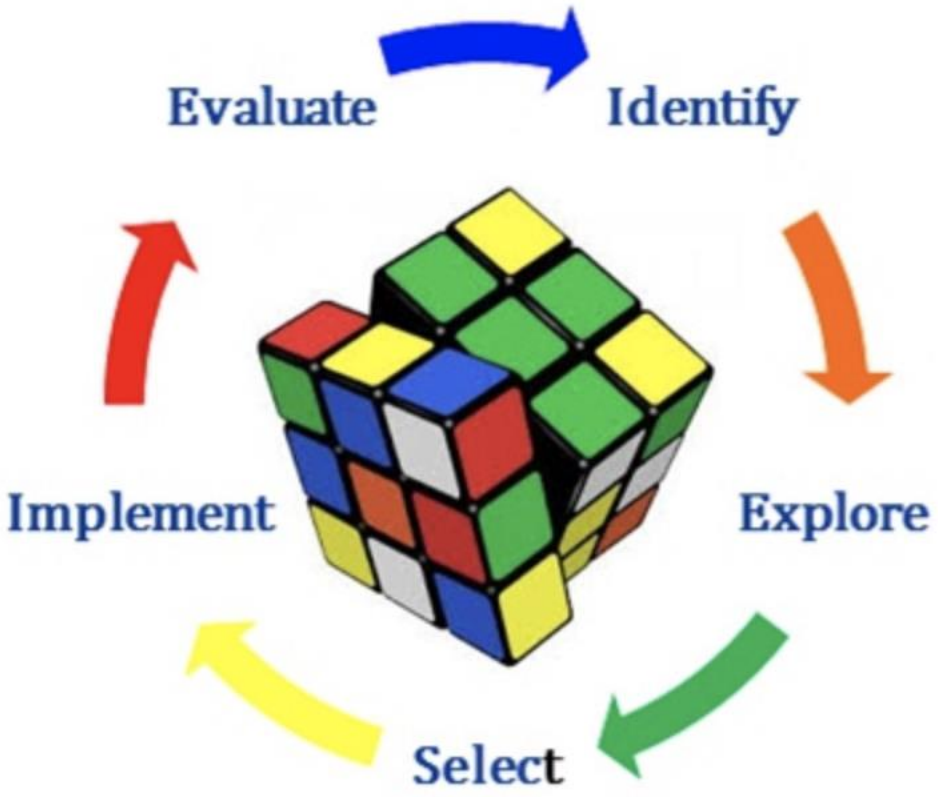
OTHER APPLICATIONS

Vehicle routing

Graph coloring

Layout planning

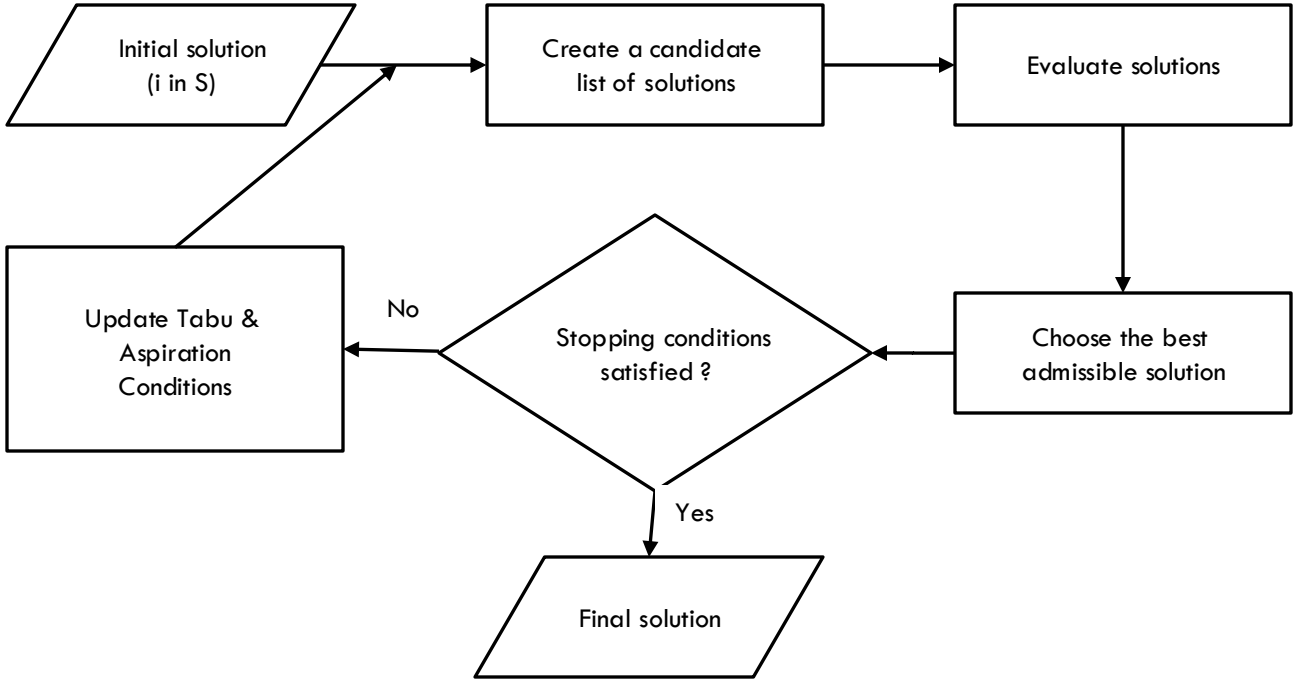
DNA sequencing



TABU SEARCH

Enhancements

FLOWCHART OF A STANDARD TABU SEARCH ALGORITHM

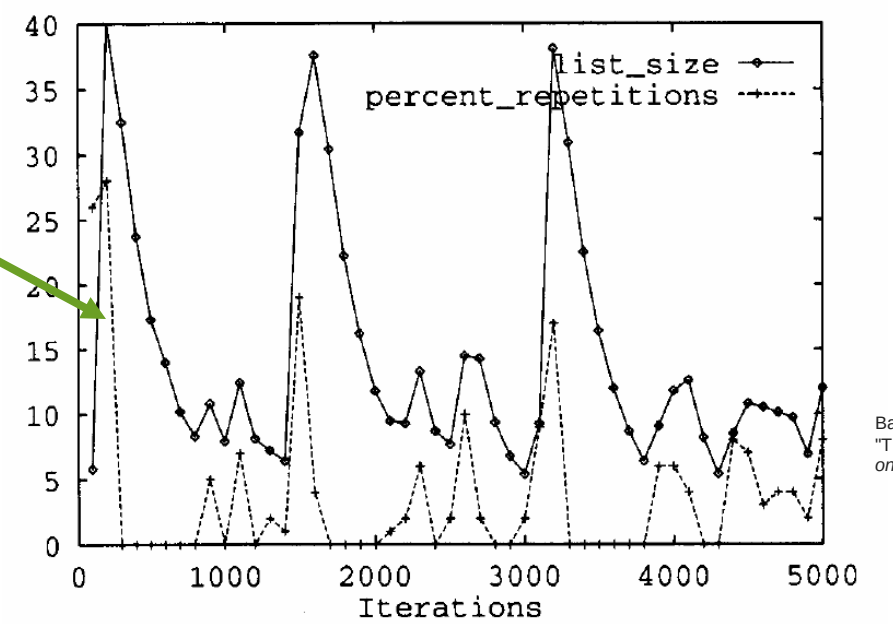


ADAPTATION

One of the most important parameters of the TS is the length of the tabu list,

- If the length is too small, the a search can be trapped into cycles,
- If the length is too large, too many moves could be prevented at each iteration.

Longer cycles are detected when using longer tabu lists



Battiti, Roberto, and Giampietro Tecchiolli. "The reactive tabu search." *ORSA journal on computing* 6.2 (1994): 126-140.

ADAPTATION

One of the adaptive approaches incorporated into TS is to allow the length of the short term memory (tabu list) to vary dynamically [introduced by Taillard [1] and Dell'Amico and Trubian [2]].

In [1], the adaptive approach was as follows:

- A range for the tabu list length is computed in advance according to the problem size,
- A new list length is randomly selected from this range every predetermined number of iterations.

1. E. Taillard. "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem". *ORSA Journal on Computing*, vol. 6, no. 2, 108-117, 1994.
2. M. Dell'Amico and M. Trubian. "Applying Tabu Search to the Job Shop Scheduling Problem". Annals of Operation Research, vol. 41, no. 3, pp. 231-252, 1993.

ADAPTATION

In [2], the adaptive approach was as follows:

- The tabu list length is restricted between L_{min} and L_{max},
- If the current solution is better than the best-so-far, the tabu list length is set to 1,
- If in an improving phase (the solution has improved over the last iteration), the tabu list length is decreased by 1,
- If not in an improving phase (the solution has deteriorated over the last iteration) the tabu list length is increased by 1,
- The values of L_{min} and L_{max} are randomly changed every A iterations.

The idea behind the approach is that if the current solution has improved the tabu list length is decreased in order to focus the search in a region of potential improvement,

On the other hand, if the current solution has deteriorated the tabu list length is increased in order to guide the search away from an apparently bad region.

COOPERATION

In Crainic et al. [5], the authors studied the idea of having *p* independent tabu searches working concurrently and exchanging information every predetermined number of iterations (*synchronous communication*).

This approach was referred to as *coordinated searches*.

The different search processes may use different:

- Initial solutions,
- Search strategies (parameter settings),
- Ways of handling the incoming solution:
  - Replacing its own best-so-far solution (forced diversification), referred to as *simple import*,
  - Replacing its own best-so-far solution only if the incoming solution is better, referred to as *conditional import*.

COOPERATION

Another implementation is found in Malek et al. [6], where the tabu lists are **reset** after any synchronization step.

An ***asynchronous communication*** approach was proposed in [10],

- Instead of exchanging information every predetermined number of iterations, each search process only broadcasts information when its best-so-far solution is updated,
- A ***central memory*** is used to handle the information exchange.
- Each search process sends its best-so-far solution to the central memory when it gets updated,
- If the sent solution is worse than the solution available in the central memory, the search process uses the stored solution,
- If its best-so-far solution is not improved for a specified number of iteration, the search process requests the solution stored in the central memory.

COOPERATION

Another approach is to implement the central memory as a pool of the best ***s*** found solutions,

When a search process requests a solution from the central memory, it gets a randomly selected solution from the pool rather than always getting the best one.

COOPERATION

General conclusions found were:

- For a fixed number of iterations, increasing the number of processors improved the obtained solution up to a certain point,

- Reducing the number of iterations between synchronization steps increases the computational times because of the increased message passing overhead,

- Conditional import always produces better or similar results to simple import.