Longest Path Problem

Ahnaf Faisal, 1505005
Raihanul Alam, 1505010
Mahim Mahbub, 1505022
Zahin Wahab, 1505031
Bishal Basak Papan, 1505043

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology

November 23, 2020

Outline:

1. Recap
2. Heuristic and Metaheuristic
3. Ant colony
4. Genetic Algorithm
5. GA in longest path problem
6. Simulated Annealing
7. Particle Swarm Optimization
8. References

Where we left off

- We have already talked about the **Longest Path Problem** in our previous presentations.

Where we left off

- We have already talked about the **Longest Path Problem** in our previous presentations.
- We analysed the hardness of this problem and showed that to solve the problem exactly, we need **exponential** time.

Where we left off

- We have already talked about the **Longest Path Problem** in our previous presentations.
- We analysed the hardness of this problem and showed that to solve the problem exactly, we need **exponential** time.
- We have discussed that our algorithm does not have any APX and we tried to give one.

Where we left off

- We have already talked about the **Longest Path Problem** in our previous presentations.
- We analysed the hardness of this problem and showed that to solve the problem exactly, we need **exponential** time.
- We have discussed that our algorithm does not have any APX and we tried to give one.
- Today we will propose meta-heuristics for our problem.

Where we left off

- We have already talked about the **Longest Path Problem** in our previous presentations.
- We analysed the hardness of this problem and showed that to solve the problem exactly, we need **exponential** time.
- We have discussed that our algorithm does not have any APX and we tried to give one.
- Today we will propose meta-heuristics for our problem.
- But before we proceed any further, let's just shed some light to our previous discussion.

What is a Longest Path Problem?

Optimization Version

Given a weighted graph G, find a simple path in this graph which has the maximum weight.

But Longest Path problem is NP-Complete

- From our previous discussion, we can safely state that Longest Path problem is **NP-Complete** which makes it both **NP** and **NP-Hard**.
- So, unless **P=NP**, there is no polynomial time algorithm which gives exact solution of Longest Path problem.
- But solving a **NP-hard optimization problem** like ours optimally takes a toll on running time. So we are going to relax the criterion of getting an optimal solution.
- We have tried to find an algorithm which solves the aforementioned problem approximately in polynomial time. These were algorithms that sacrificed correctness for faster running times.
- This week we will try to find a metaheuristic algorithm which solves the aforementioned problem in polynomial time. This brings us to this week's content : **Heuristic and Metaheuristic Algorithms**.

But Longest Path problem is NP-Complete

- From our previous discussion, we can safely state that Longest Path problem is **NP-Complete** which makes it both **NP** and **NP-Hard**.
- So, unless **P=NP**, there is no polynomial time algorithm which gives exact solution of Longest Path problem.
- But solving a **NP-hard optimization problem** like ours optimally takes a toll on running time. So we are going to relax the criterion of getting an optimal solution.
- We have tried to find an algorithm which solves the aforementioned problem approximately in polynomial time. These were algorithms that sacrificed correctness for faster running times.
- This week we will try to find a metaheuristic algorithm which solves the aforementioned problem in polynomial time. This brings us to this week's content : **Heuristic and Metaheuristic Algorithms**.

But Longest Path problem is NP-Complete

- From our previous discussion, we can safely state that Longest Path problem is **NP-Complete** which makes it both **NP** and **NP-Hard**.
- So, unless **P=NP**, there is no polynomial time algorithm which gives exact solution of Longest Path problem.
- But solving a **NP-hard optimization problem** like ours optimally takes a toll on running time. So we are going to relax the criterion of getting an optimal solution.
- We have tried to find an algorithm which solves the aforementioned problem approximately in polynomial time. These were algorithms that sacrificed correctness for faster running times.
- This week we will try to find a metaheuristic algorithm which solves the aforementioned problem in polynomial time. This brings us to this week's content : **Heuristic and Metaheuristic Algorithms**.

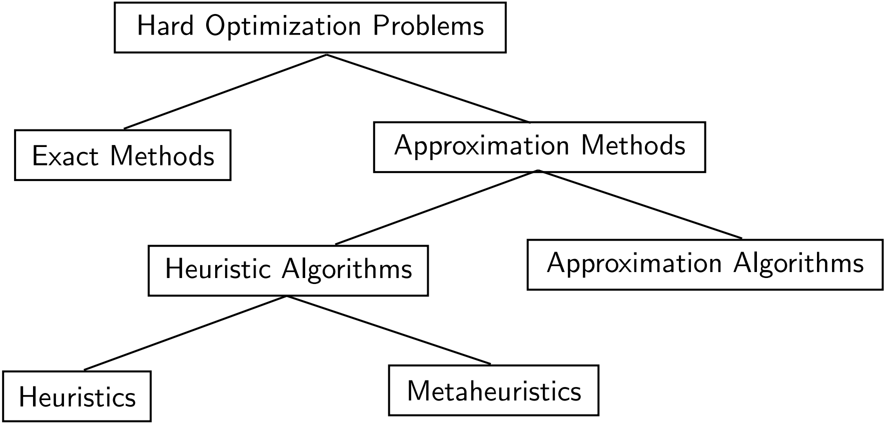
But Longest Path problem is NP-Complete

- From our previous discussion, we can safely state that Longest Path problem is **NP-Complete** which makes it both **NP** and **NP-Hard**.
- So, unless **P=NP**, there is no polynomial time algorithm which gives exact solution of Longest Path problem.
- But solving a **NP-hard optimization problem** like ours optimally takes a toll on running time. So we are going to relax the criterion of getting an optimal solution.
- We have tried to find an algorithm which solves the aforementioned problem approximately in polynomial time. These were algorithms that sacrificed correctness for faster running times.
- This week we will try to find a metaheuristic algorithm which solves the aforementioned problem in polynomial time. This brings us to this week's content : **Heuristic and Metaheuristic Algorithms**.

But Longest Path problem is NP-Complete

- From our previous discussion, we can safely state that Longest Path problem is **NP-Complete** which makes it both **NP** and **NP-Hard**.
- So, unless **P=NP**, there is no polynomial time algorithm which gives exact solution of Longest Path problem.
- But solving a **NP-hard optimization problem** like ours optimally takes a toll on running time. So we are going to relax the criterion of getting an optimal solution.
- We have tried to find an algorithm which solves the aforementioned problem approximately in polynomial time. These were algorithms that sacrificed correctness for faster running times.
- This week we will try to find a metaheuristic algorithm which solves the aforementioned problem in polynomial time. This brings us to this week's content : **Heuristic and Metaheuristic Algorithms**.

Hard Optimization Problems



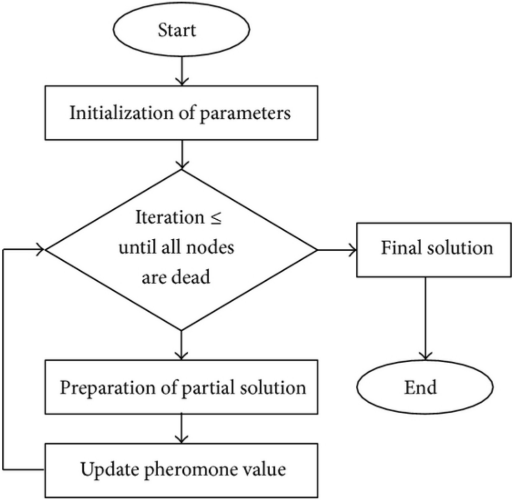
Heuristic and Metaheuristic

Heuristic	Metaheuristic
Heuristics are methods of exploration that exploit certain aspects of a problem and apply only to it.	A metaheuristic is general exploration method that applies to many problems in the same way and is often stochastic.
Heuristics are often problem-dependent.	Metaheuristics are problem-independent techniques that can be applied to a broad range of problems.
Do not guarantee to find optimal solution.	Do not guarantee to find optimal solution.

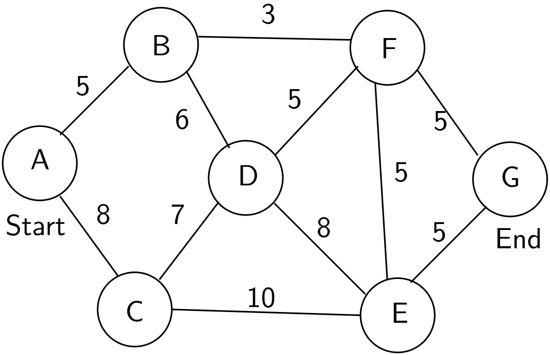
Metaheuristic Problems

- Metaheuristic algorithms
  - Trajectory Based
    - Simulated Annealing
    - Local Search
    - Random Search and Random Walk
    - Tabu Search
    - Hill Climbing
  - Population Based
    - Ant Colony Optimization
    - Particle Swarm Optimization
    - Evolutionay Algorithms
    - Bacterial Foraging
    - Artificial Bee Colony

Ant Colony Flow Chart

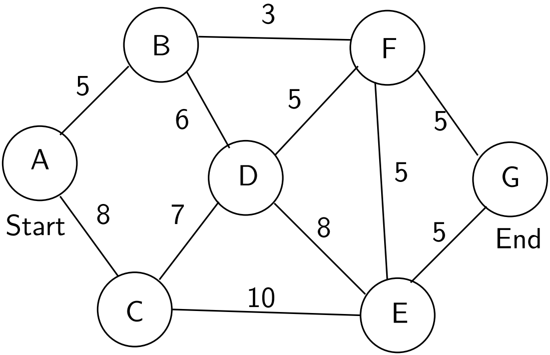


Simple Graph



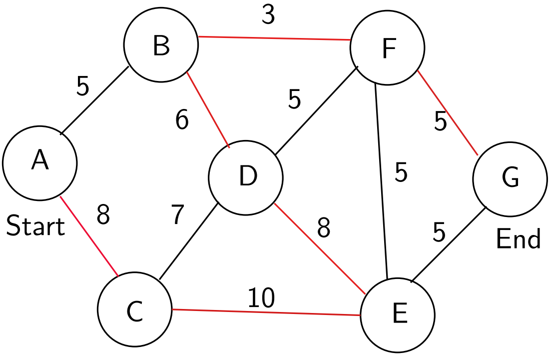
- This is a weighted undirected graph.

Simple Graph



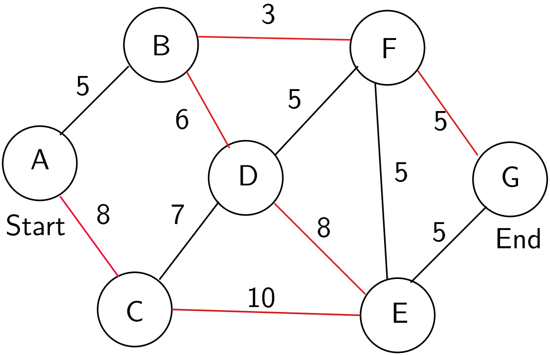
- This is a weighted undirected graph.
- We want to find the longest path between A and G.

Finding longest path



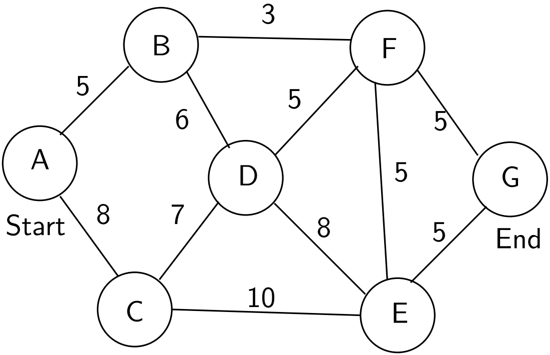
- Longest path here is A-> C-> E-> D-> B-> F-> G and path length is 40.

Finding longest path



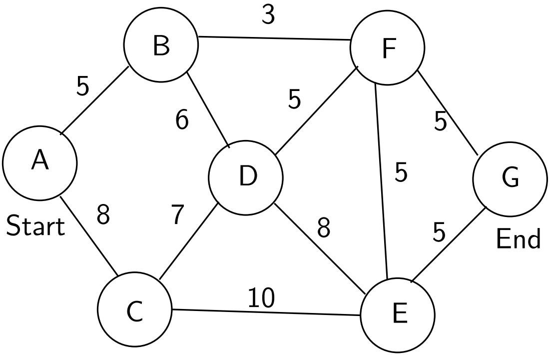
- Longest path here is A-> C-> E-> D-> B-> F-> G and path length is 40.
- Can we exactly find this path using ant colony ?

Starting with Ant colony



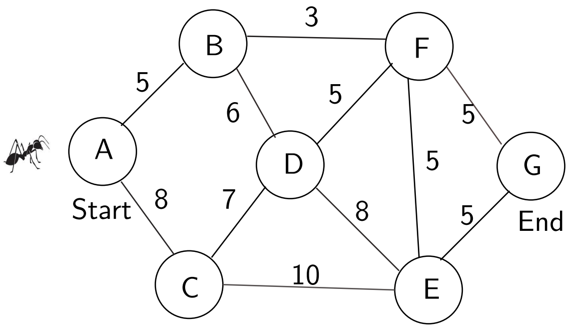
- We initialize each edge with base pheromone.

Starting with Ant colony



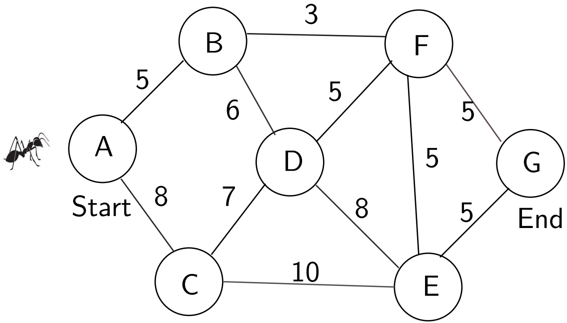
- We initialize each edge with base pheromone.
- Each edge has probability equation of format $pr = ph * \alpha + weight * \beta$ where α and β are constants given as parameters.

Starting with Ant colony



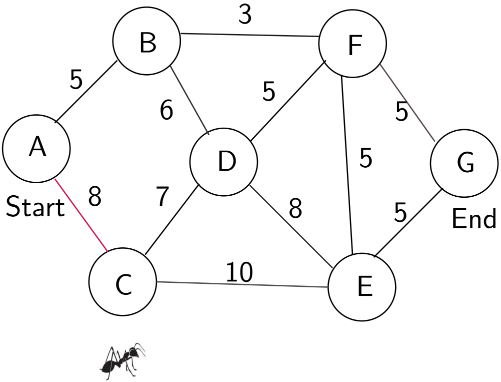
- Our very first ant at starting node.

Starting with Ant colony



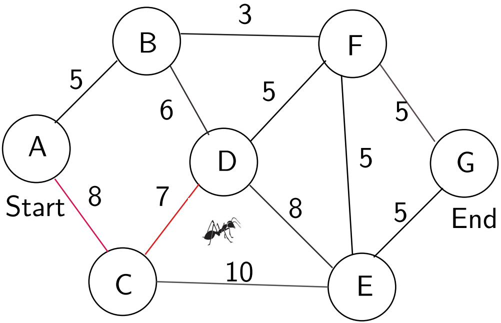
- Our very first ant at starting node.
- Which path will it go?

Starting with Ant colony



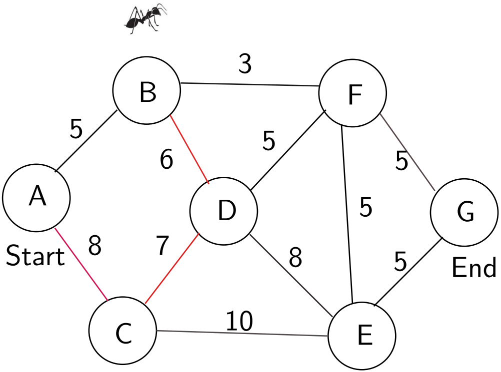
- Our Ant goes along the path with longest probability here and ends up at C.

Starting with Ant colony



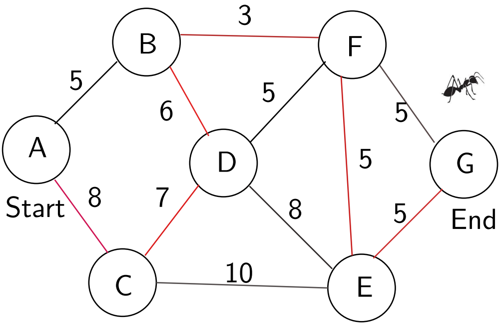
- Our Ant goes along the path CD and ends up at D.

Starting with Ant colony



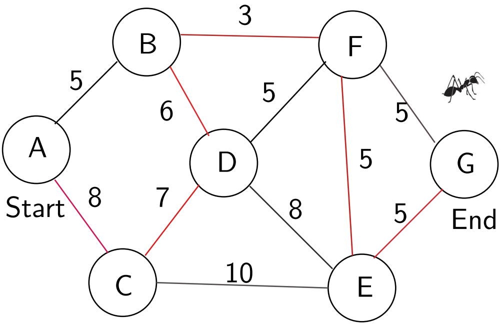
- Our Ant goes along the path DB and ends up at B.

Ending with Ant colony



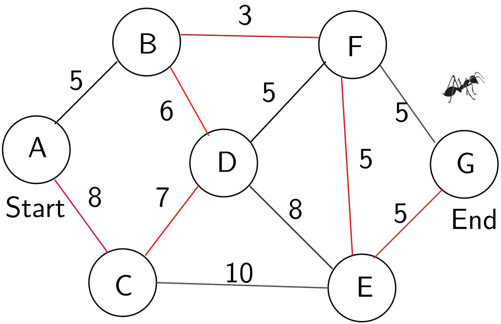
- All our ants are at finish point.

Ending with Ant colony



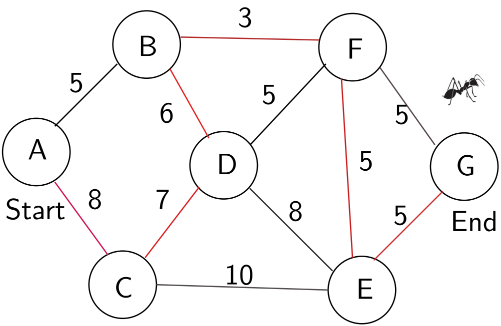
- All our ants are at finish point.
- See their paths and record the longest of them.

Ending with Ant colony



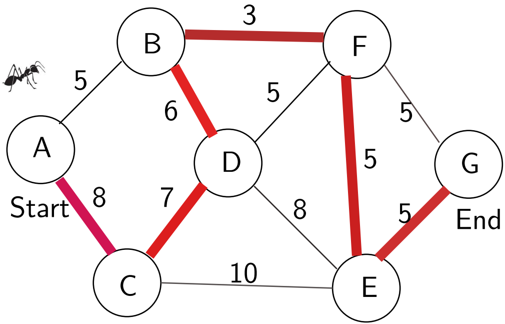
- All our ants are at finish point.
- See their paths and record the longest of them.
- Suppose the longest they have travelled is through A-> C-> D-> B-> F-> E-> G and path length is 35.

Ending with Ant colony



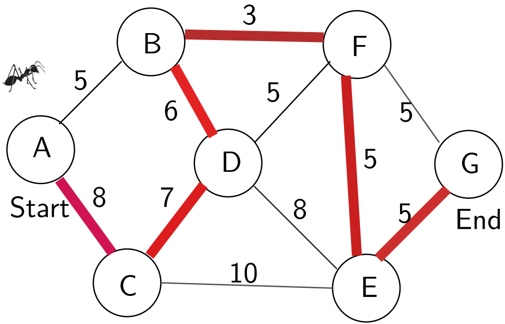
- All our ants are at finish point.
- See their paths and record the longest of them.
- Suppose the longest they have travelled is through A-> C-> D-> B-> F-> E-> G and path length is 35.
- Now lets increase pheromone along this path.

Ending with Ant colony



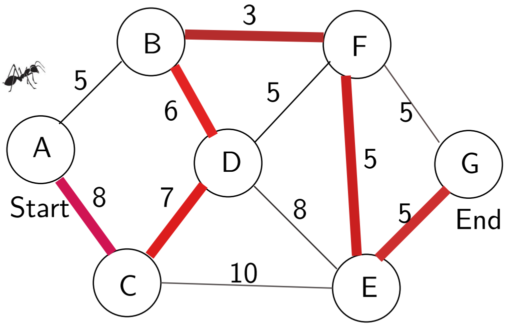
- With the pheromone increased along the local longest path,iteration 1 is finished.

Ending with Ant colony



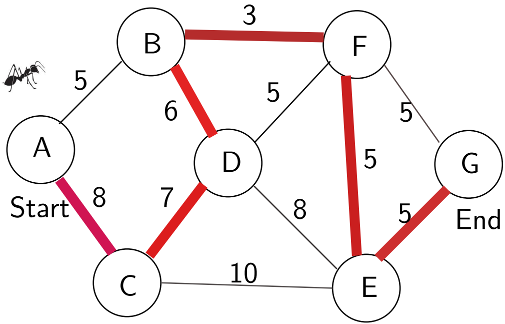
- With the pheromone increased along the local longest path,iteration 1 is finished.
- iteration 2 starts again at Node A.

Ending with Ant colony



- With the pheromone increased along the local longest path,iteration 1 is finished.
- iteration 2 starts again at Node A.
- As pheromone is more along the previous local longest path,ants might follow that path more.

Ending with Ant colony



- With the pheromone increased along the local longest path,iteration 1 is finished.
- iteration 2 starts again at Node A.
- As pheromone is more along the previous local longest path,ants might follow that path more.
- After each iteration, we compare the local longest path with global and update the global path.

Final words on Ant Colony

- We will also need to evaporate pheromone after certain times.

Final words on Ant Colony

- We will also need to evaporate pheromone after certain times.
- After our iterations have finished,we will end up with the global longest path.

Final words on Ant Colony

- We will also need to evaporate pheromone after certain times.
- After our iterations have finished,we will end up with the global longest path.
- Important fact is the algorithm is polynomial as only two loops.Iteration number and ant number.Building path can be done through BFS or DFS.

Final words on Ant Colony

- We will also need to evaporate pheromone after certain times.
- After our iterations have finished,we will end up with the global longest path.
- Important fact is the algorithm is polynomial as only two loops.Iteration number and ant number.Building path can be done through BFS or DFS.
- It might not be the actual longest path but chances are it is relatively close due to its design and how probability works.

Genetic Algorthim

- A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution.

Genetic Algorthim

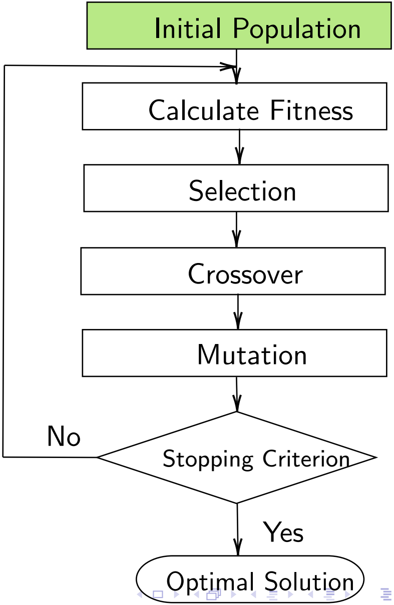
- A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution.
- This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Introduction to Genetic Algorithm

Initial Population: The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem we want to solve.

Fitness Calculation: The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals)

Selection: The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation.

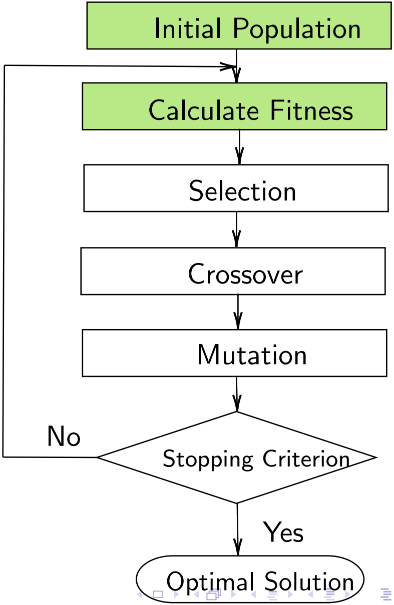


Introduction to Genetic Algorithm

Initial Population: The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem we want to solve.

Fitness Calculation: The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals)

Selection: The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation.

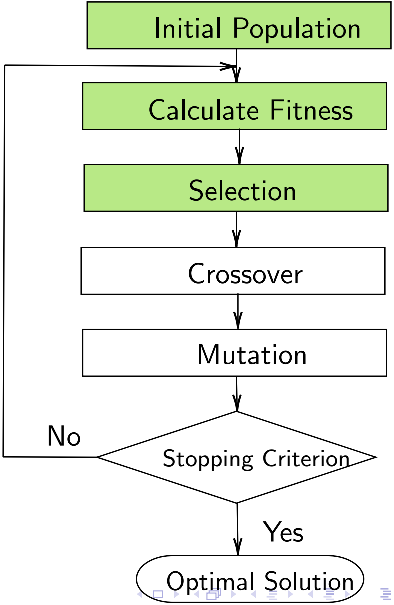


Introduction to Genetic Algorithm

Initial Population: The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem we want to solve.

Fitness Calculation: The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals)

Selection: The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation.

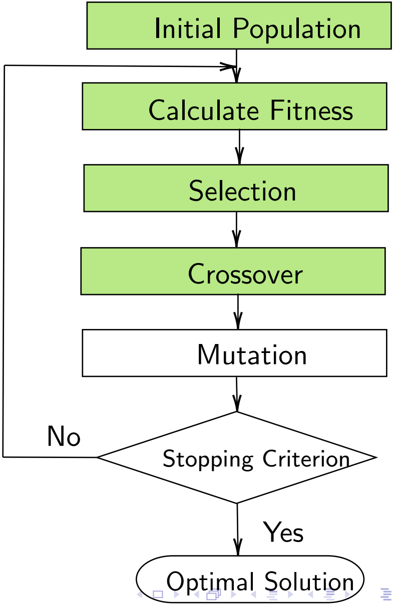


Introduction to Genetic Algorithm

Crossover: For each pair of parents to be mated, a crossover point is chosen at random from within the genes.

Mutation: In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability.

Termination: The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation).

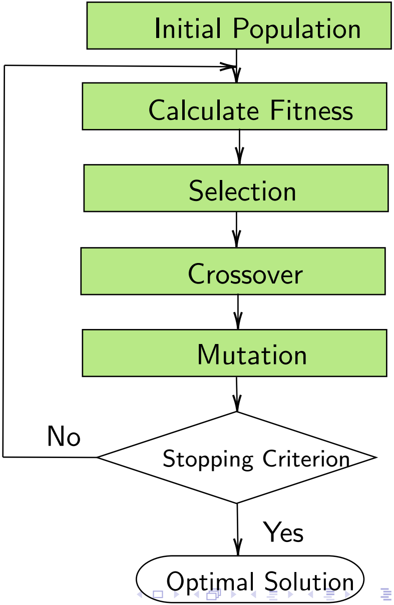


Introduction to Genetic Algorithm

Crossover: For each pair of parents to be mated, a crossover point is chosen at random from within the genes.

Mutation: In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability.

Termination: The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation).

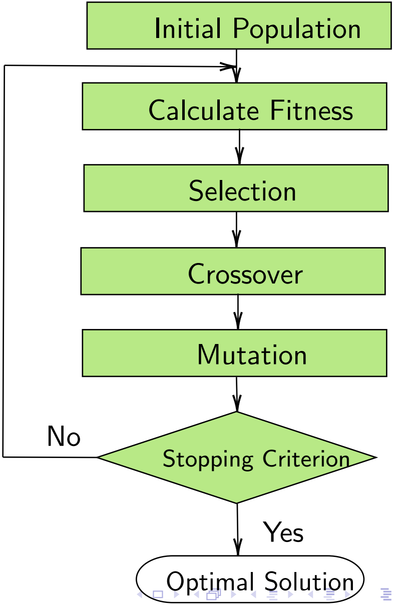


Introduction to Genetic Algorithm

Crossover: For each pair of parents to be mated, a crossover point is chosen at random from within the genes.

Mutation: In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability.

Termination: The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation).

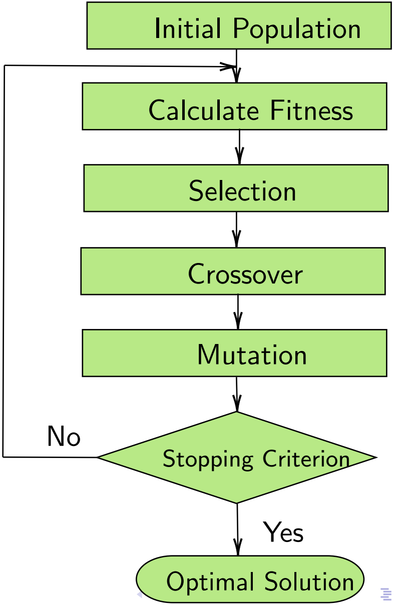


Introduction to Genetic Algorithm

Crossover: For each pair of parents to be mated, a crossover point is chosen at random from within the genes.

Mutation: In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability.

Termination: The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation).



Genetic Algorithm in Longest Path Problem

- **Portugal et al.** applied Genetic Algorithm (**GA**) in Longest path problem (**LPP**).[1]

Genetic Algorithm in Longest Path Problem

- **Portugal et al.** applied Genetic Algorithm (**GA**) in Longest path problem (**LPP**).[1]
- Four approaches were proposed
  - GA using non-intersecting paths (**GANP**)
  - GA using intersecting paths (**GAIP**)
  - GA using both pairs of paths (**GABPP**)
  - GA using a mutation mechanism (**GAMM**)

Genetic Algorithm in Longest Path Problem

- **Portugal et al.** applied Genetic Algorithm (**GA**) in Longest path problem (**LPP**).[1]
- Four approaches were proposed
  - GA using non-intersecting paths (**GANP**)
  - GA using intersecting paths (**GAIP**)
  - GA using both pairs of paths (**GABPP**)
  - GA using a mutation mechanism (**GAMM**)
- The first three approaches (**GANP,GAIP,GABPP**) proposed are based on **crossover** mechanisms, in which two parents create a set of offspring that share their genetic material.

Genetic Algorithm in Longest Path Problem

- **Portugal et al.** applied Genetic Algorithm (**GA**) in Longest path problem (**LPP**).[1]
- Four approaches were proposed
  - GA using non-intersecting paths (**GANP**)
  - GA using intersecting paths (**GAIP**)
  - GA using both pairs of paths (**GABPP**)
  - GA using a mutation mechanism (**GAMM**)
- The first three approaches (**GANP,GAIP,GABPP**) proposed are based on **crossover** mechanisms, in which two parents create a set of offspring that share their genetic material.
- The last approach (**GAMM**) is based on a **mutation** operator, in which each individual creates two offspring by perturbation of their genetic material in places specified according to the overall state of the system.

Applying GA to LPP: encoding

A question though

How will we encode path as genetic materials i.e.chromosomes?

Applying GA to LPP: encoding

A question though

How will we encode path as genetic materials i.e.chromosomes?

- Each path is represented as an ordered array of vertices with variable length.

GA: Generating initial population

- Better results and faster convergence depends a lot on initial population.

GA: Generating initial population

- Better results and faster convergence depends a lot on initial population.
- Since GAs strongly rely on the genetic material of the initial solution population, it is important to guarantee initial solutions with good quality.

GA: Generating initial population

- Better results and faster convergence depends a lot on initial population.
- Since GAs strongly rely on the genetic material of the initial solution population, it is important to guarantee initial solutions with good quality.
- In this case, this corresponds to long and diverse initial paths.

Generating initial population using Random method

- First select a random vertex of the graph

Generating initial population using Random method

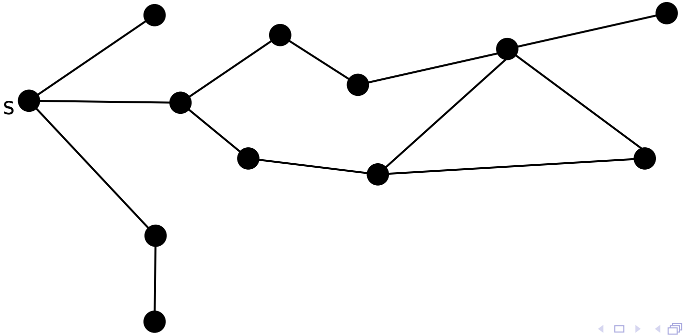
- First select a random vertex of the graph
- compute paths by choosing neighbors of the current vertex at random, as long as they were not already included in the path.

Generating initial population using Random method

- First select a random vertex of the graph
- compute paths by choosing neighbors of the current vertex at random, as long as they were not already included in the path.
- finish computing the path when there were no available neighbors left.

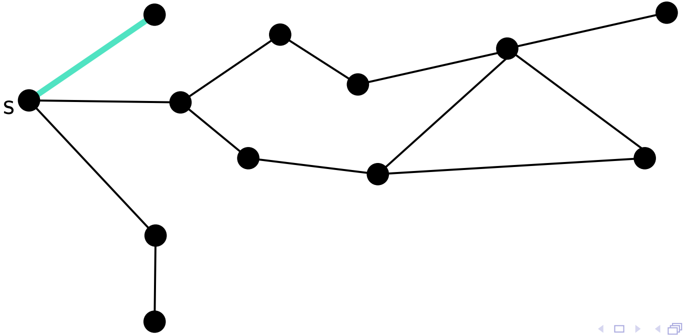
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.



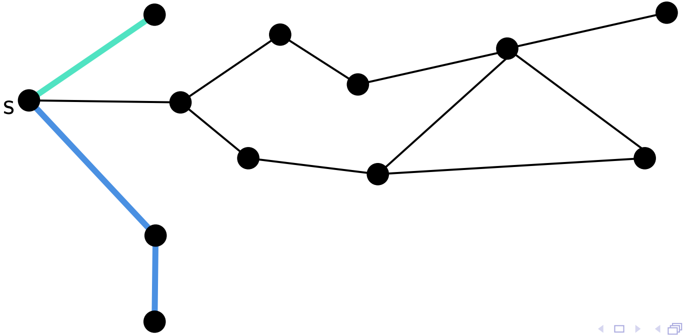
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.
- Now if we keep adding vertices to the path, maybe we will get a path like this.



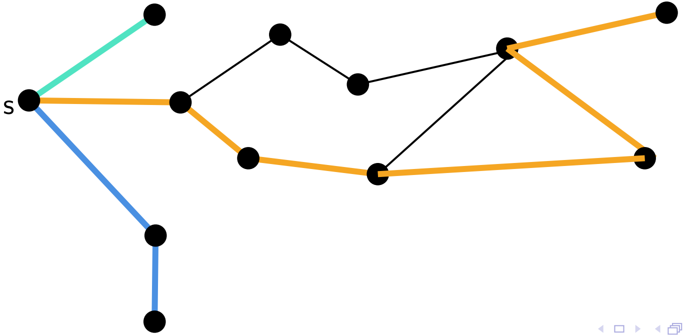
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.
- Now if we keep adding vertices to the path, maybe we will get a path like this.
- or this (a bit longer).



Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.
- Now if we keep adding vertices to the path, maybe we will get a path like this.
- or this (a bit longer).
- If we get really lucky, we might end up with this.



Generating initial population using Random method: Cons

- This proved to be an inefficient method, since most of the paths computed would be very short, mostly because degree one vertices, i.e. vertices with only one neighbor would be selected rather rapidly.

Generating initial population using Random method: Cons

- This proved to be an inefficient method, since most of the paths computed would be very short, mostly because degree one vertices, i.e. vertices with only one neighbor would be selected rather rapidly.
- Could we do it a bit more intelligently?

Generating initial population using Intelligent method

- The first vertex is still selected at random.

Generating initial population using Intelligent method

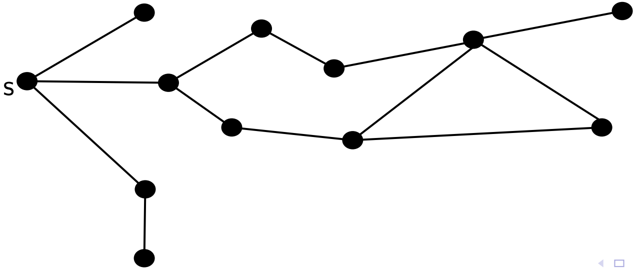
- The first vertex is still selected at random.
- The next vertices are selected with a probability according to their degree. For example, if two neighbors of a given vertex have degrees a and b, the first one would be selected with a probability of $a/(a+b)$ and the second one with $b/(a+b)$.

Generating initial population using Intelligent method

- The first vertex is still selected at random.
- The next vertices are selected with a probability according to their degree. For example, if two neighbors of a given vertex have degrees a and b, the first one would be selected with a probability of $a/(a+b)$ and the second one with $b/(a+b)$.
- If we have reached a vertex with unavailable neighbors, instead of stopping the method, we analyze whether the degree of the first vertex is greater than one and keep computing the path to the opposite direction until reaching a finishing point.

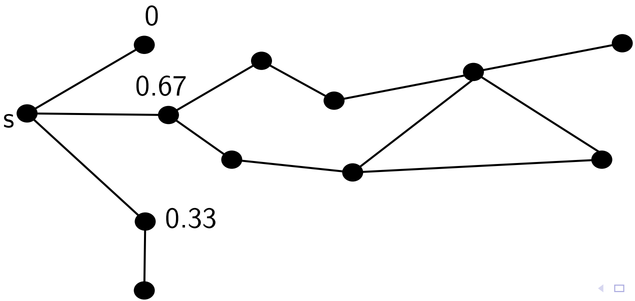
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.



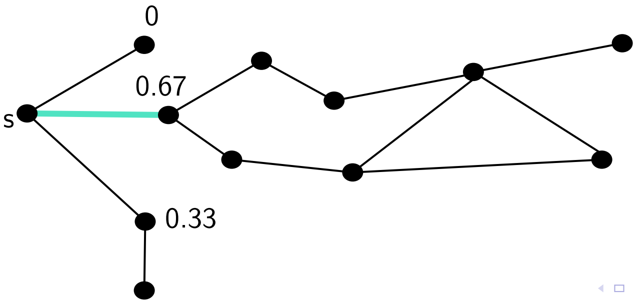
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.
- We will assign probabilities to each of the neighbours of s.



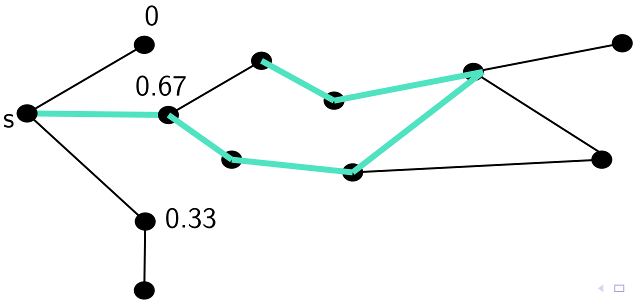
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.
- We will assign probabilities to each of the neighbours of s.
- We will compute path by taking edges to the neighbour with highest probability.



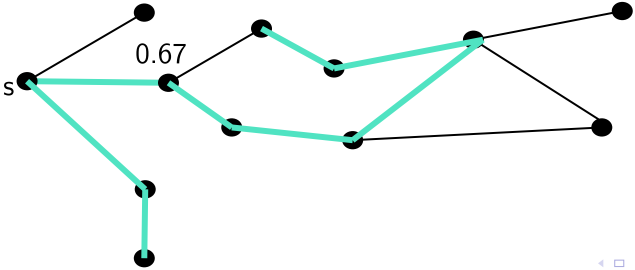
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.
- We will assign probabilities to each of the neighbours of s.
- We will compute path by taking edges to the neighbour with highest probability.
- Keep adding vertices and we will get a path like this



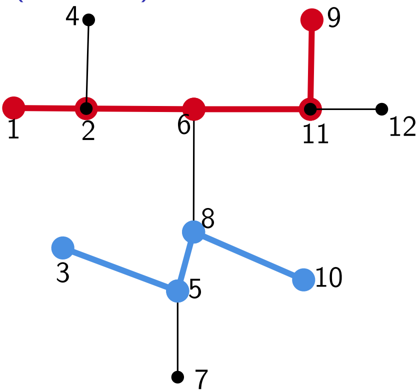
Generating initial population using Random method: Visualization

- Let us select a random starting vertex. We call it s.
- We will assign probabilities to each of the neighbours of s.
- We will compute path by taking edges to the neighbour with highest probability.
- Keep adding vertices and we will get a path like this
- As of now we can see there are no unavailable neighbours left so we go backwards from the starting vertex s and extend the path.



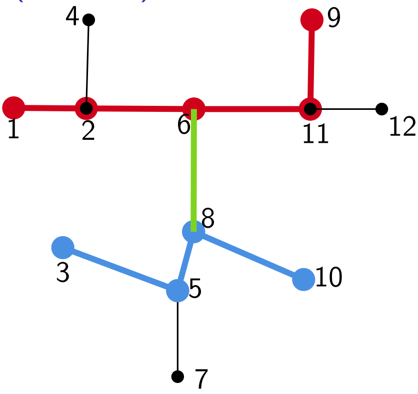
GA using non-intersecting paths (**GANP**)

- Given initial set of population, we will find pairs of non-intersecting paths (that do not have common vertices).



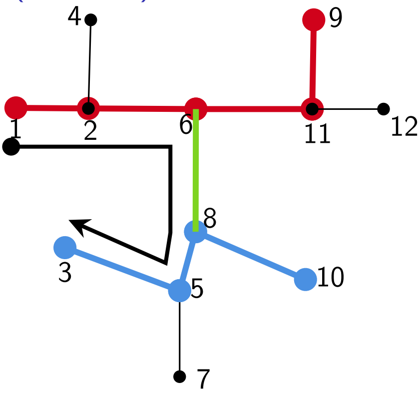
GA using non-intersecting paths (**GANP**)

- Given initial set of population, we will find pairs of non-intersecting paths (that do not have common vertices).
- Then we search for an edge that connects both paths and that edge must not be in any of the parent paths.



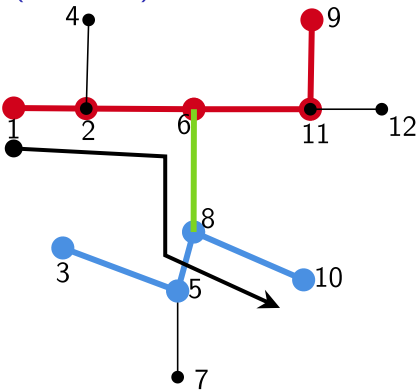
GA using non-intersecting paths (**GANP**)

- Given initial set of population, we will find pairs of non-intersecting paths (that do not have common vertices).
- Then we search for an edge that connects both paths and that edge must not be in any of the parent paths.
- We take the connector edge and construct 4 offspring paths like this.
  - 1-2-6-8-5-3

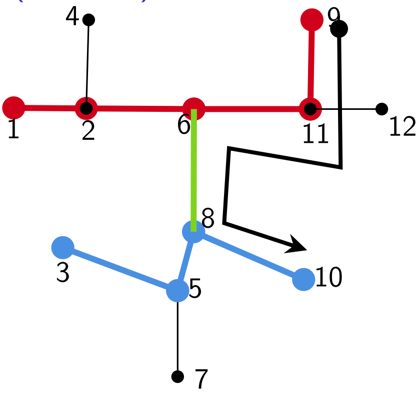


GA using non-intersecting paths (**GANP**)

- Given initial set of population, we will find pairs of non-intersecting paths (that do not have common vertices).
- Then we search for an edge that connects both paths and that edge must not be in any of the parent paths.
- We take the connector edge and construct 4 offspring paths like this.
  - 1-2-6-8-5-3
  - 1-2-6-8-10



GA using non-intersecting paths (**GANP**)

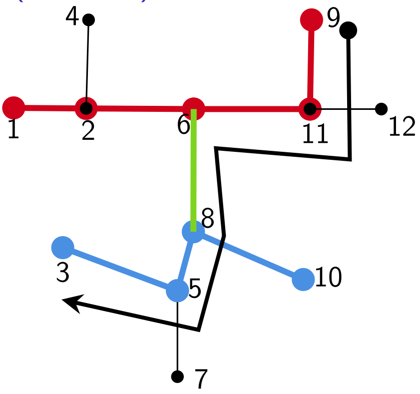


- Given initial set of population, we will find pairs of non-intersecting paths (that do not have common vertices).
- Then we search for an edge that connects both paths and that edge must not be in any of the parent paths.
- We take the connector edge and construct 4 offspring paths like this.

  - 1-2-6-8-5-3
  - 1-2-6-8-10
  - 9-11-6-8-10

GA using non-intersecting paths (**GANP**)

- Given initial set of population, we will find pairs of non-intersecting paths (that do not have common vertices).
- Then we search for an edge that connects both paths and that edge must not be in any of the parent paths.
- We take the connector edge and construct 4 offspring paths like this.
  - 1-2-6-8-5-3
  - 1-2-6-8-10
  - 9-11-6-8-10
  - 9-11-6-8-5-3



GA using non-intersecting paths (**GANP**):Convergence

- First all parents and descendants are gathered

GA using non-intersecting paths (**GANP**):Convergence

- First all parents and descendants are gathered
- then remove identical solutions

GA using non-intersecting paths (**GANP**):Convergence

- First all parents and descendants are gathered
- then remove identical solutions
- compute their path cost (fitness function)

GA using non-intersecting paths (**GANP**):Convergence

- First all parents and descendants are gathered
- then remove identical solutions
- compute their path cost (fitness function)
- finally select solutions with a probability according to their cost. The higher the cost, the more likely it is that a given solution is chosen.

GA using non-intersecting paths (**GANP**):Convergence

- First all parents and descendants are gathered
- then remove identical solutions
- compute their path cost (fitness function)
- finally select solutions with a probability according to their cost. The higher the cost, the more likely it is that a given solution is chosen.
- After this step, the new generation is created and the process can start again.

GA using non-intersecting paths (**GANP**):Convergence

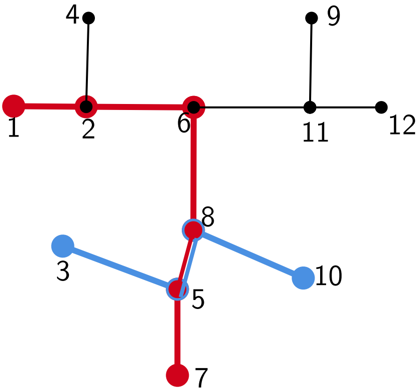
- First all parents and descendants are gathered
- then remove identical solutions
- compute their path cost (fitness function)
- finally select solutions with a probability according to their cost. The higher the cost, the more likely it is that a given solution is chosen.
- After this step, the new generation is created and the process can start again.
- As the process goes on, it is expected to run slower in the beginning and faster along time because probability of getting disconnected pairs decreases over time.

GA using non-intersecting paths (**GANP**):Convergence

- First all parents and descendants are gathered
- then remove identical solutions
- compute their path cost (fitness function)
- finally select solutions with a probability according to their cost. The higher the cost, the more likely it is that a given solution is chosen.
- After this step, the new generation is created and the process can start again.
- As the process goes on, it is expected to run slower in the beginning and faster along time because probability of getting disconnected pairs decreases over time.
- After a few runs, the algorithm converges to a final solution.

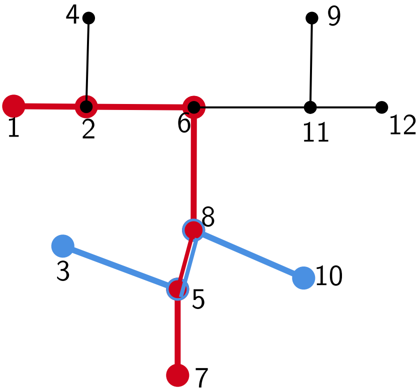
GA using intersecting paths (**GAIP**)

- Given initial set of population, we will find pairs of that intersect once (which have a common vertex, a common edge or a common set of edges). Two cases occur here:



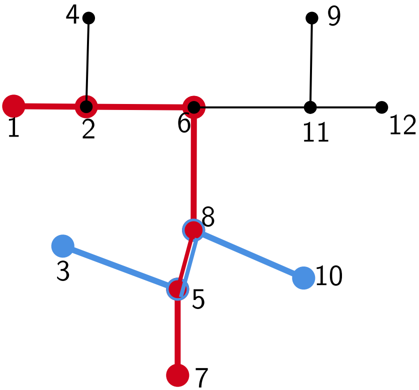
GA using intersecting paths (**GAIP**)

- Given initial set of population, we will find pairs of that intersect once (which have a common vertex, a common edge or a common set of edges). Two cases occur here:
    - **Crossroad:** if they only have a common vertex, which means that this vertex has at least degree 4, and four offspring can be generated.



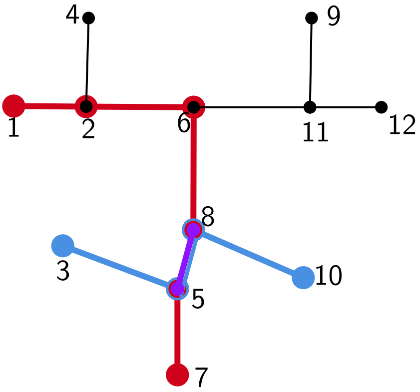
GA using intersecting paths (**GAIP**)

- Given initial set of population, we will find pairs of that intersect once (which have a common vertex, a common edge or a common set of edges). Two cases occur here:
  - **Crossroad:** if they only have a common vertex, which means that this vertex has at least degree 4, and four offspring can be generated.
  - **Junction:** if they have a common edge or set of edges, which is the most usual case. They can generate only two descendants.



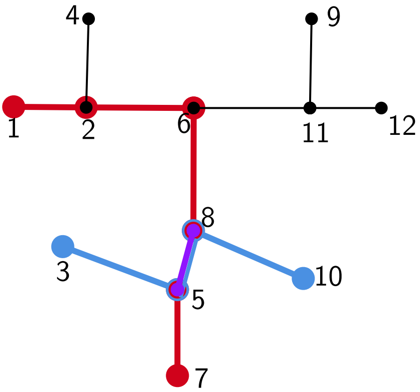
GA using intersecting paths (**GAIP**)

- Given initial set of population, we will find pairs of that intersect once (which have a common vertex, a common edge or a common set of edges). Two cases occur here:
  - **Crossroad:** if they only have a common vertex, which means that this vertex has at least degree 4, and four offspring can be generated.
  - **Junction:** if they have a common edge or set of edges, which is the most usual case. They can generate only two descendants.
- Here junction is shown with two intersecting paths (1-2-6-8-5-7) and (3-5-8-10) with common edge (5,8).



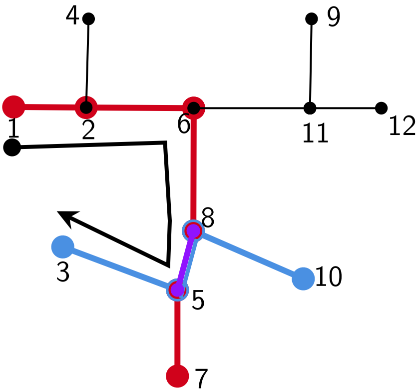
GA using intersecting paths (**GAIP**)

- Two offspring, which also incorporate the junction, can be generated:



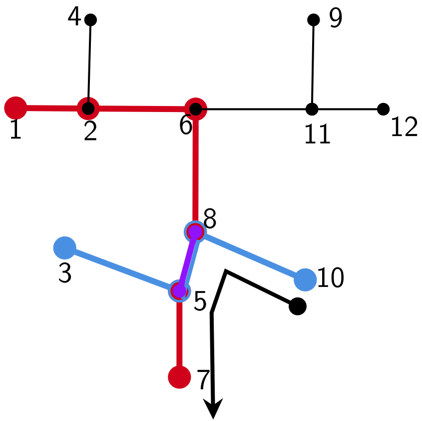
GA using intersecting paths (**GAIP**)

- Two offspring, which also incorporate the junction, can be generated:
  - 1-2-6-8-5-3



GA using intersecting paths (**GAIP**)

- Two offspring, which also incorporate the junction, can be generated:
    - 1-2-6-8-5-3
    - 10-8-5-7



GA using intersecting paths (**GAIP**):Convergence

- as the process goes on, it is expected to run faster in the beginning and slower along time, due to the detection of longer paths over time, which increases the probability of having intersecting paths and consequently more crossovers are required.

GA using intersecting paths (**GAIP**):Convergence

- as the process goes on, it is expected to run faster in the beginning and slower along time, due to the detection of longer paths over time, which increases the probability of having intersecting paths and consequently more crossovers are required.
- After a few runs, the algorithm converges to a final solution.

GA using both pairs of paths (**GABPP**)

- The third algorithm basically combines the two previous approaches.

GA using both pairs of paths (**GABPP**)

- The third algorithm basically combines the two previous approaches.
- Assuming that we have the initial solution population, a search takes place to find pairs of paths that intersect once and pairs of disconnected paths.

GA using both pairs of paths (**GABPP**)

- The third algorithm basically combines the two previous approaches.
- Assuming that we have the initial solution population, a search takes place to find pairs of paths that intersect once and pairs of disconnected paths.
- Rest steps are like previous algorithms.

GA using a mutation mechanism (**GAMM**)

- This algorithm uses a mutation technique to generate descendants.

GA using a mutation mechanism (**GAMM**)

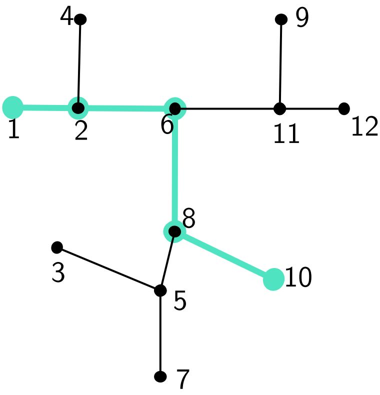
- This algorithm uses a mutation technique to generate descendants.
- A variable is initialized to measure the perturbation pressure, which is related to the rate of solution improvement obtained.

GA using a mutation mechanism (**GAMM**)

- This algorithm uses a mutation technique to generate descendants.
- A variable is initialized to measure the perturbation pressure, which is related to the rate of solution improvement obtained.
- Two offspring are generated per path consisting in two mutated solutions that result from the perturbation applied in the original path and in a flipped version of the original path,

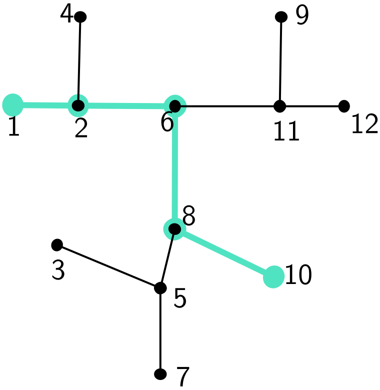
GA using a mutation mechanism (**GAMM**)

- A path and its flipped version are considered. In this case [1-2-6-8-10] and [10-8-6-2-1] are taken.



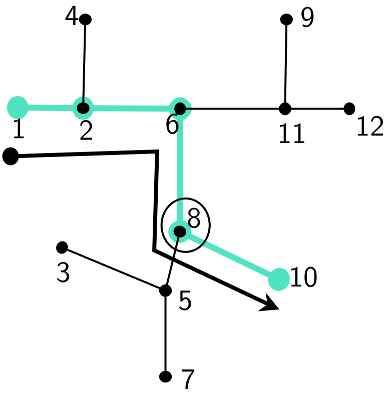
GA using a mutation mechanism (**GAMM**)

- A path and its flipped version are considered. In this case [1-2-6-8-10] and [10-8-6-2-1] are taken.
- The perturbation starts in a vertex that must have at least degree 3, since it cannot go backwards or to the next vertex of the original path.



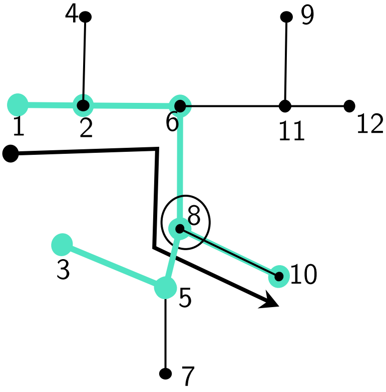
GA using a mutation mechanism (**GAMM**)

- A path and its flipped version are considered. In this case [1-2-6-8-10] and [10-8-6-2-1] are taken.
- The perturbation starts in a vertex that must have at least degree 3, since it cannot go backwards or to the next vertex of the original path.
- For the path [1-2-6-8-10], the 3-degree vertex 8 is chosen.



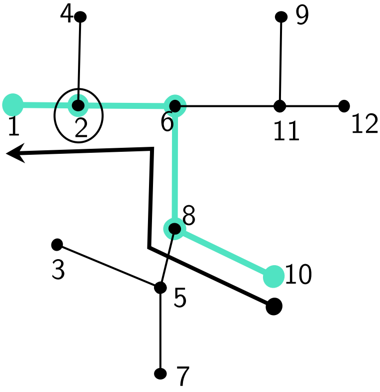
GA using a mutation mechanism (**GAMM**)

- A path and its flipped version are considered. In this case [1-2-6-8-10] and [10-8-6-2-1] are taken.
- The perturbation starts in a vertex that must have at least degree 3, since it cannot go backwards or to the next vertex of the original path.
- For the path [1-2-6-8-10], the 3-degree vertex 8 is chosen.
- And new path is explored from vertex 8 and we get an offspring [1-2-6-8-5-3].



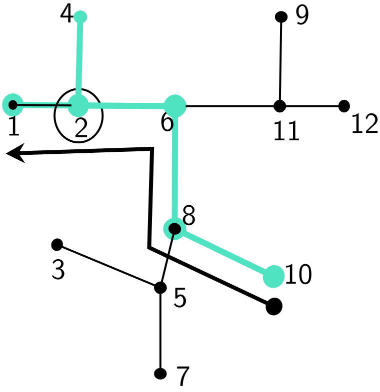
GA using a mutation mechanism (**GAMM**)

- A path and its flipped version are considered. In this case [1-2-6-8-10] and [10-8-6-2-1] are taken.
- The perturbation starts in a vertex that must have at least degree 3, since it cannot go backwards or to the next vertex of the original path.
- For the path [1-2-6-8-10], the 3-degree vertex 8 is chosen.
- And new path is explored from vertex 8 and we get an offspring [1-2-6-8-5-3].
- For the path [10-8-6-2-1], the 3-degree vertex 2 is chosen.

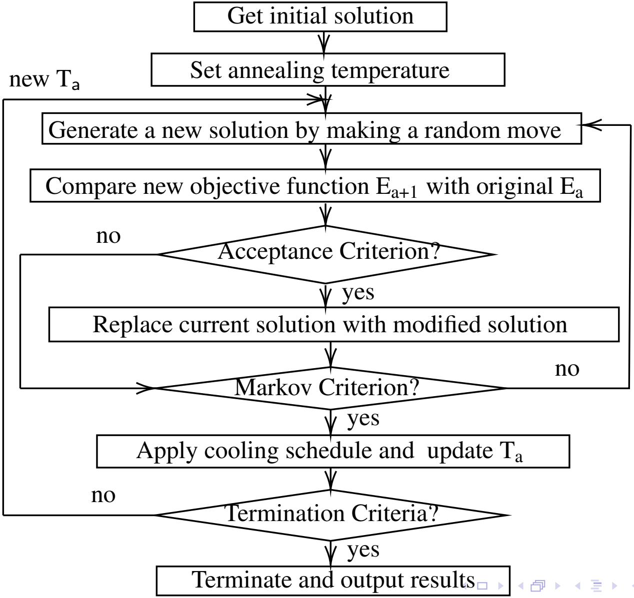


GA using a mutation mechanism (**GAMM**)

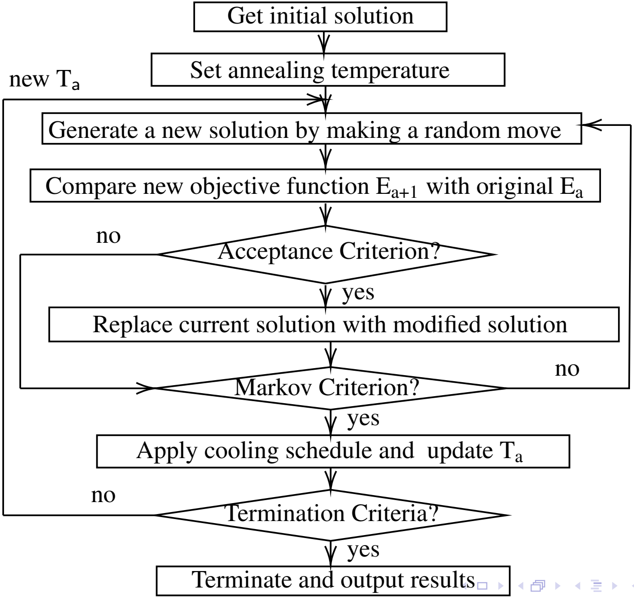
- A path and its flipped version are considered. In this case [1-2-6-8-10] and [10-8-6-2-1] are taken.
- The perturbation starts in a vertex that must have at least degree 3, since it cannot go backwards or to the next vertex of the original path.
- For the path [1-2-6-8-10], the 3-degree vertex 8 is chosen.
- And new path is explored from vertex 8 and we get an offspring [1-2-6-8-5-3].
- For the path [10-8-6-2-1], the 3-degree vertex 2 is chosen.
- And new path is explored from vertex 2 and we get an offspring [10-8-6-2-4].



Simulated Annealing



Simulated Annealing



Solving LP using SA

- We set the initial and the final temperature.

Solving LP using SA

- We set the initial and the final temperature.
- We set an iteration limit in each temperature.

Solving LP using SA

- We set the initial and the final temperature.
- We set an iteration limit in each temperature.
- For each temperature, we start from a random solution (path).

Solving LP using SA

- We set the initial and the final temperature.
- We set an iteration limit in each temperature.
- For each temperature, we start from a random solution (path).
- In each iteration, we find a neighbor of a solution (path) by edge swapping or choosing another path randomly.

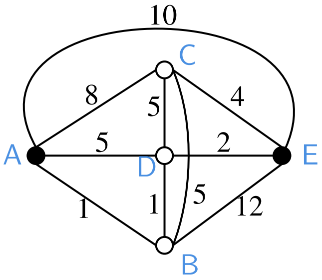
Solving LP using SA

- We set the initial and the final temperature.
- We set an iteration limit in each temperature.
- For each temperature, we start from a random solution (path).
- In each iteration, we find a neighbor of a solution (path) by edge swapping or choosing another path randomly.
- If the length of path P in iteration $i+1$ is less than the path length in iteration i, then we decide considering P by comparing the probability with a random value.

Solving LP using SA

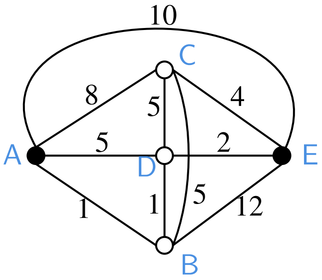
- We set the initial and the final temperature.
- We set an iteration limit in each temperature.
- For each temperature, we start from a random solution (path).
- In each iteration, we find a neighbor of a solution (path) by edge swapping or choosing another path randomly.
- If the length of path P in iteration $i + 1$ is less than the path length in iteration i, then we decide considering P by comparing the probability with a random value.
- We store and update the current best solution accordingly.

Solving LP using SA



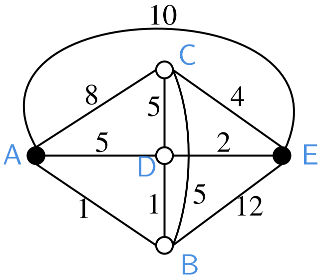
- In the graph shown, the source vertex is A and the destination vertex is E.

Solving LP using SA



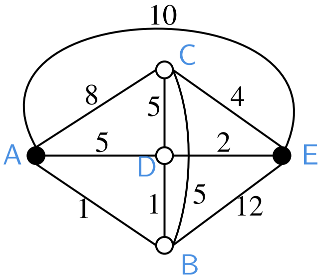
- In the graph shown, the source vertex is A and the destination vertex is E.
- Let the initial temperature be 20 and the final temperature be 10.
- In each temperature, we choose 4 different paths.

Solving LP using SA



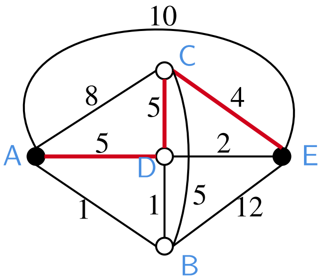
- In the graph shown, the source vertex is A and the destination vertex is E.
- Let the initial temperature be 20 and the final temperature be 10.
- In each temperature, we choose 4 different paths.
- If the difference between the length of the initially chosen path P and the current path S in temperature $temp$ is d, then the probability function is $e^{\frac{d}{temp}}$. If S satisfies the criteria, we set $P = S$.

Solving LP using SA



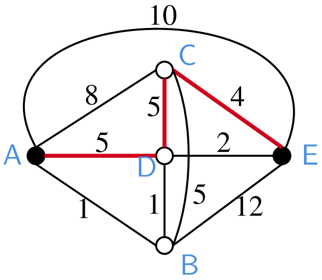
- In the graph shown, the source vertex is A and the destination vertex is E.
- Let the initial temperature be 20 and the final temperature be 10.
- In each temperature, we choose 4 different paths.
- If the difference between the length of the initially chosen path P and the current path S in temperature $temp$ is d, then the probability function is $e^{\frac{d}{temp}}$. If S satisfies the criteria, we set $P = S$.
- We decrease the temperature by 1 after each step.

Solving LP using SA



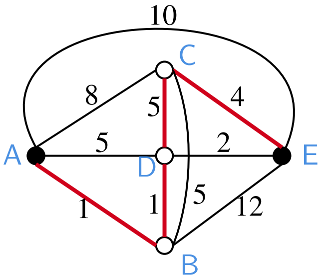
- Initially chosen random path, P $=A$-D-C-E.
- Length of P = 14

Solving LP using SA



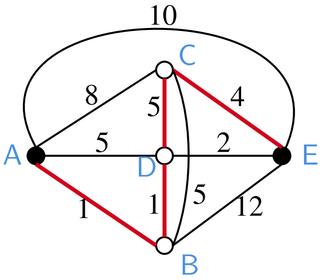
- P =A-D-C-E, length = 14
- $temp$ = 20, iteration 1:

Solving LP using SA



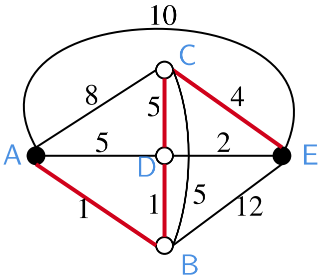
- P =A-D-C-E, length = 14
- $temp$ = 20, iteration 1:
- S =A-B-D-C-E, length = 11.

Solving LP using SA



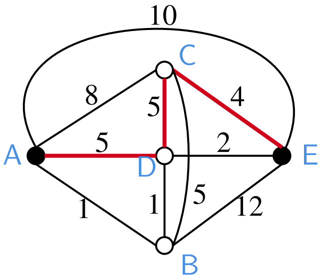
- P =A-D-C-E, length = 14
- $temp$ = 20, iteration 1:
- S =A-B-D-C-E, length = 11.
- $d = 11 - 14 = -3$.

Solving LP using SA



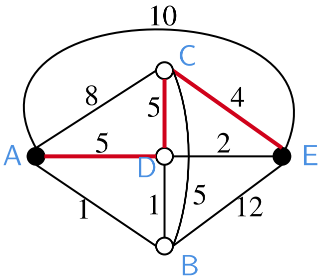
- $P =A\text{-}D\text{-}C\text{-}E$, length $= 14$
- $temp = 20$, iteration 1:
- $S =A\text{-}B\text{-}D\text{-}C\text{-}E$, length $= 11$.
- $d = 11 - 14 = -3$.
- Length of S is less than length of P.
- Random probability $= 0.90$, $e^{\frac{-3}{20}} = 0.86 < 0.90$.

Solving LP using SA



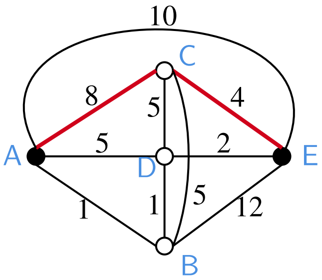
- P =A-D-C-E, length = 14
- $temp$ = 20, iteration 1:
- S =A-B-D-C-E, length = 11.
- $d = 11 - 14 = -3$.
- Length of S is less than length of P.
- Random probability = 0.90, $e^{\frac{-3}{20}} = 0.86 < 0.90$.
- Decision:P =A-D-C-E.

Solving LP using SA



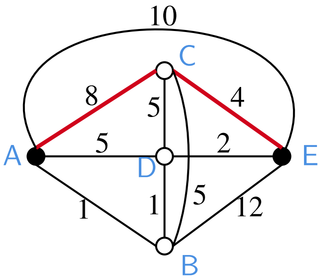
- P =A-D-C-E, length = 14
- $temp$ = 20, iteration 2:

Solving LP using SA



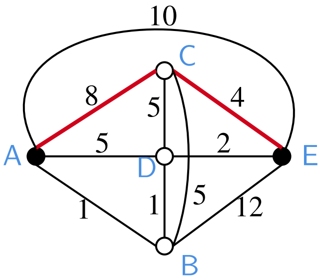
- P =A-D-C-E, length $= 14$
- $temp = 20$, iteration 2:
- S =A-C-E, length $= 12$.

Solving LP using SA



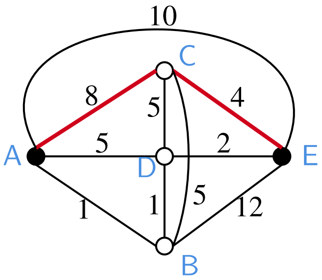
- P =A-D-C-E, length = 14
- $temp$ = 20, iteration 2:
- S =A-C-E, length = 12.
- d = 12 − 14 = −2.

Solving LP using SA



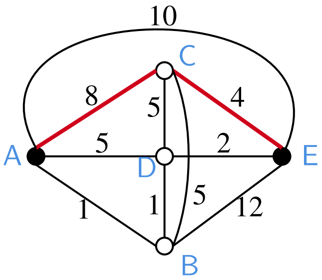
- P =A-D-C-E, length = 14
- $temp = 20$, iteration 2:
- S =A-C-E, length = 12.
- $d = 12 - 14 = -2$.
- Length of S is less than length of P.
- Random probability = 0.83, $e^{\frac{-2}{20}} = 0.90 > 0.83$.

Solving LP using SA



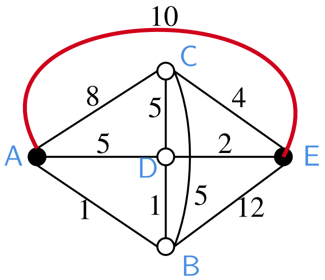
- P =A-D-C-E, length = 14
- $temp$ = 20, iteration 2:
- S =A-C-E, length = 12.
- d = 12 − 14 = −2.
- Length of S is less than length of P.
- Random probability = 0.83, $e^{\frac{-2}{20}} = 0.90 > 0.83$.
- Decision: update P =A-C-E.

Solving LP using SA



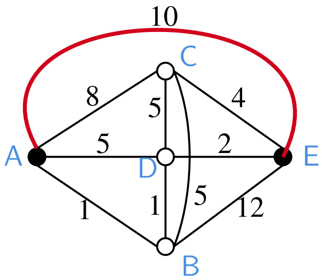
- P =A-C-E, length = 12
- $temp$ = 20, iteration 3:

Solving LP using SA



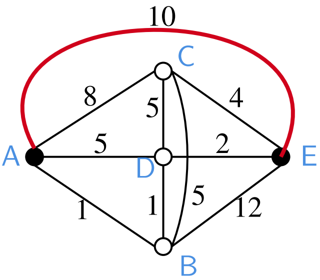
- P =A-C-E, length = 12
- $temp$ = 20, iteration 3:
- S =A-E, length = 10.

Solving LP using SA



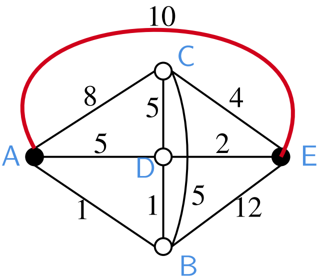
- P =A-C-E, length = 12
- $temp$ = 20, iteration 3:
- S =A-E, length = 10.
- d = 10 − 12 = −2.

Solving LP using SA



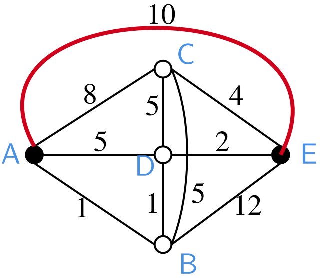
- $P = A\text{-}C\text{-}E$, length $= 12$
- $temp = 20$, iteration 3:
- $S = A\text{-}E$, length $= 10$.
- $d = 10 - 12 = -2$.
- Length of S is less than length of P.
- Random probability $= 0.68$, $e^{\frac{-2}{20}} = 0.90 > 0.68$.

Solving LP using SA



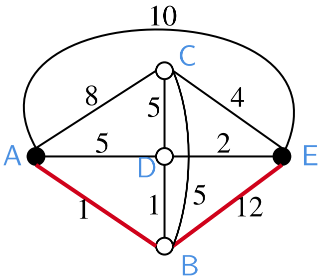
- P =A-C-E, length = 12
- $temp$ = 20, iteration 3:
- S =A-E, length = 10.
- $d = 10 - 12 = -2$.
- Length of S is less than length of P.
- Random probability = 0.68, $e^{\frac{-2}{20}} = 0.90 > 0.68$.
- Decision: update P =A-E.

Solving LP using SA



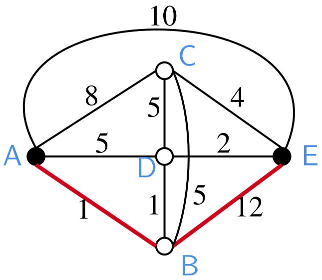
- P =A-E, length = 10
- $temp$ = 20, iteration 4:

Solving LP using SA



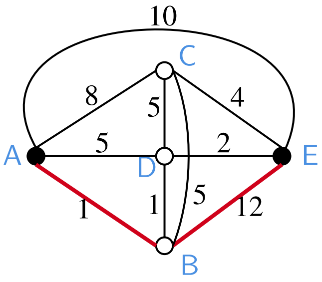
- P =A-E, length = 10
- $temp$ = 20, iteration 4:
- S =A-B-E, length = 13.

Solving LP using SA



- P = A-E, length $= 10$
- $temp = 20$, iteration 4:
- S = A-B-E, length $= 13$.
- $d = 13 - 10 = 3$.

Solving LP using SA



- P =A-E, length $= 10$
- $temp = 20$, iteration 4:
- S =A-B-E, length $= 13$.
- $d = 13 - 10 = 3$.
- Length of S is greater than length of P.
- Decision: update P =A-B-E.

PSO - Introduction

- A **Stochastic Optimization** technique related to **Swarming Theory** such as :
    1. Bird Flocking
    2. Fish Schooling

PSO - Introduction

- A **Stochastic Optimization** technique related to **Swarming Theory** such as :
  1. Bird Flocking
  2. Fish Schooling
- Proposed by **Kennedy and Eberhart (1995.)** [1]

PSO - Introduction

- A **Stochastic Optimization** technique related to **Swarming Theory** such as :
  1. Bird Flocking
  2. Fish Schooling
- Proposed by **Kennedy and Eberhart (1995.)** [1]
- Does not use gradient of the problem/function being optimized.
- Does not require function to be differentiable.

Natural Metaphor

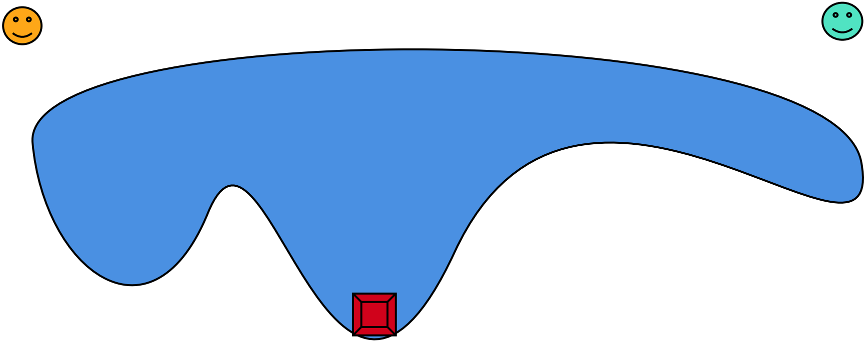
- A flock of birds (or a school of fish) searches for food.
- Want to **efficiently** find the food source.

Natural Metaphor

- A flock of birds (or a school of fish) searches for food.
- Want to **efficiently** find the food source.
- Explore food personally **and** communicate with the team.
- Follow the bird **nearest** to the food.

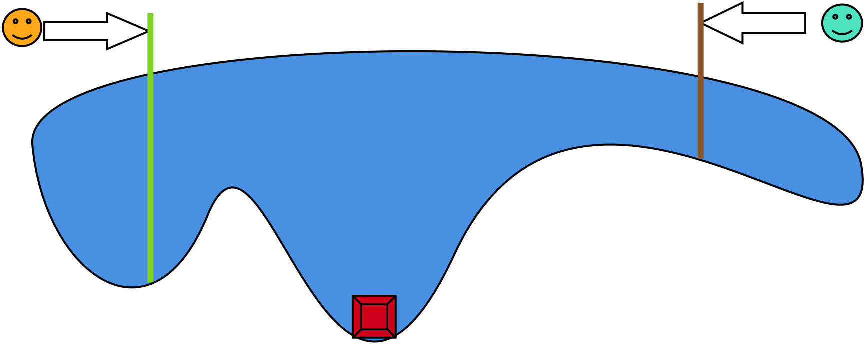
A Simplified Example

Treasure (at the bottom), *Two smily boats*, *Teamwork*, **Get Rich**.



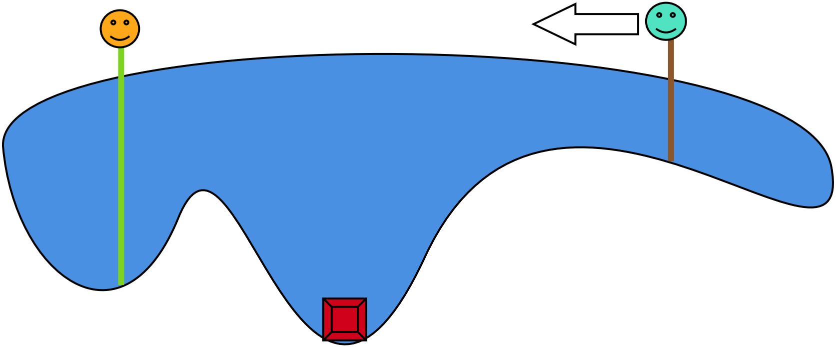
A Simplified Example

Treasure (at the bottom), *Two smily boats*, *Teamwork*, **Get Rich**.



A Simplified Example

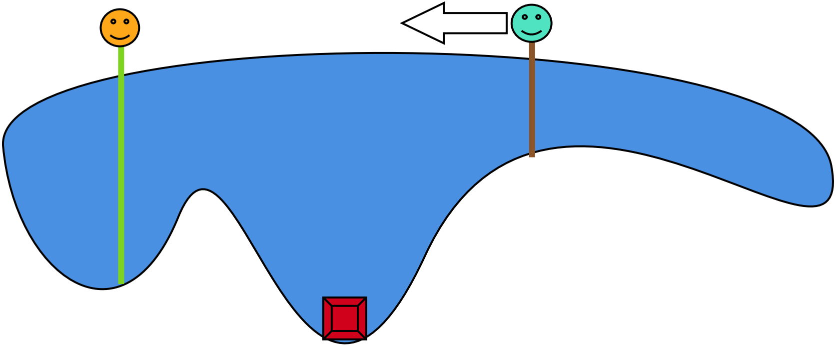
Treasure (at the bottom), *Two smily boats*, *Teamwork*, **Get Rich**.



Cyan will move towards Orange

A Simplified Example

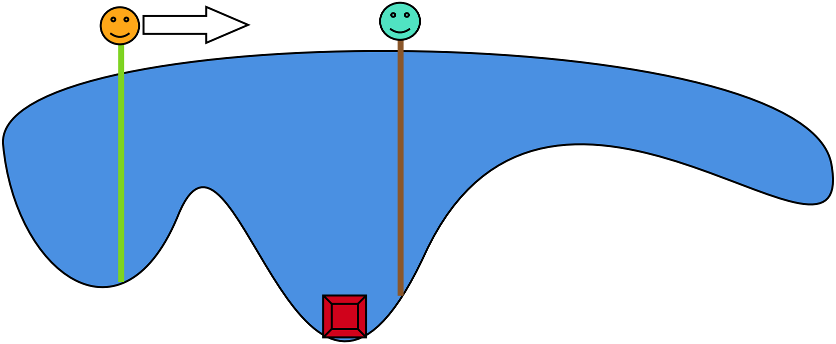
Treasure (at the bottom), *Two smily boats*, *Teamwork*, **Get Rich**.



Cyan will move towards Orange AGAIN !!

A Simplified Example

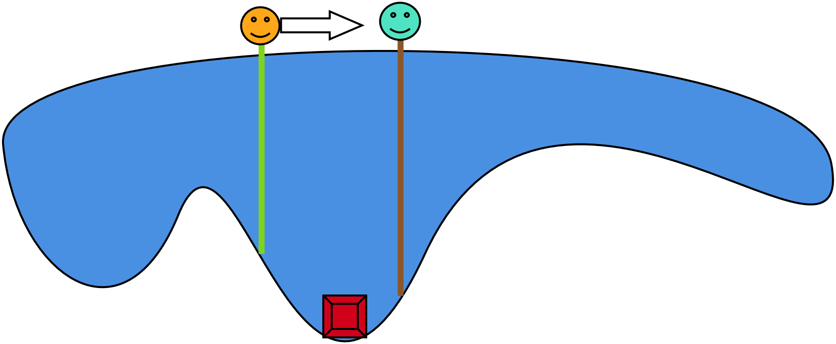
Treasure (at the bottom), *Two smily boats*, *Teamwork*, **Get Rich**.



Orange will move towards Cyan

A Simplified Example

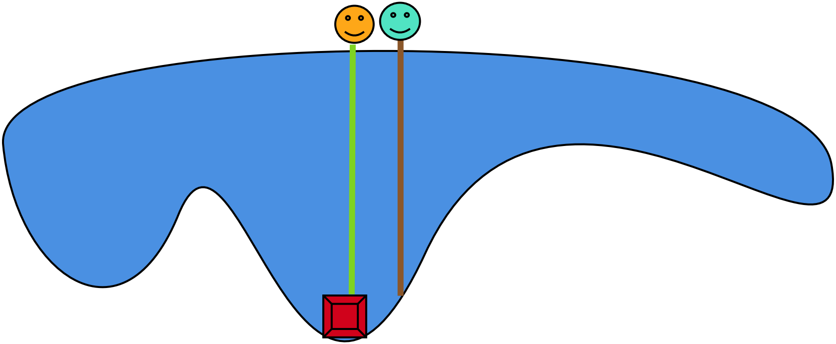
Treasure (at the bottom), *Two smily boats*, *Teamwork*, **Get Rich**.



Orange will move towards Cyan AGAIN !!

A Simplified Example

Treasure (at the bottom), *Two smily boats*, *Teamwork*, **Get Rich**.



Treasure was the friendships we made along the way

Mathematical Formulation of PSO

- **Swarm** of particles indicate population of **candidate solutions**.

Mathematical Formulation of PSO

- **Swarm** of particles indicate population of **candidate solutions**.
- Every particle is a candidate solution.

Mathematical Formulation of PSO

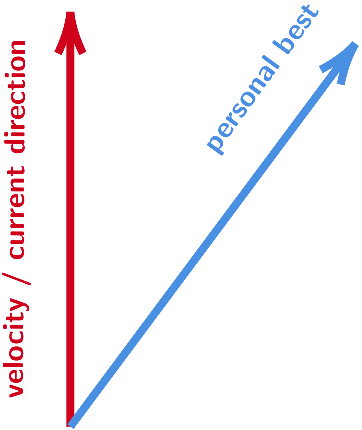
- **Swarm** of particles indicate population of **candidate solutions**.
- Every particle is a candidate solution.
- Every i^{th} particle (at a particular time-step) stores the following :
  1. **Position** x_i
  2. **Velocity** v_i
  3. **Personal Best Position** $pBest_i$
  4. **Global Best Position** $gBest$

Mathematical Formulation of PSO

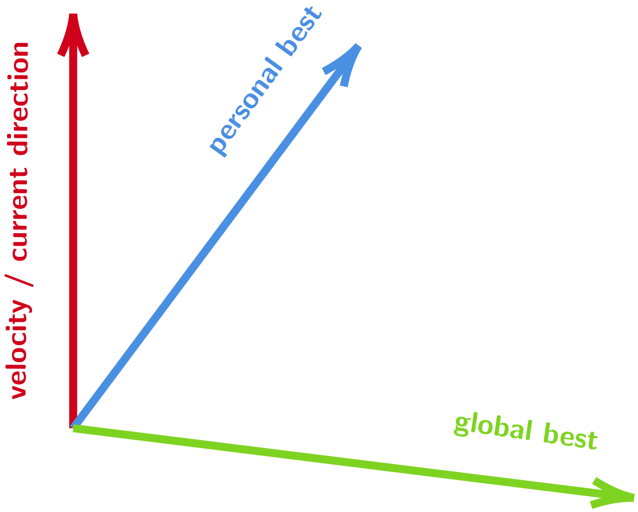
- **Swarm** of particles indicate population of **candidate solutions**.
- Every particle is a candidate solution.
- Every i^{th} particle (at a particular time-step) stores the following :
  1. **Position** x_i
  2. **Velocity** v_i
  3. **Personal Best Position** $pBest_i$
  4. **Global Best Position** $gBest$
- On **every** iteration, position and velocity gets updated.

Position and Velocity Updates

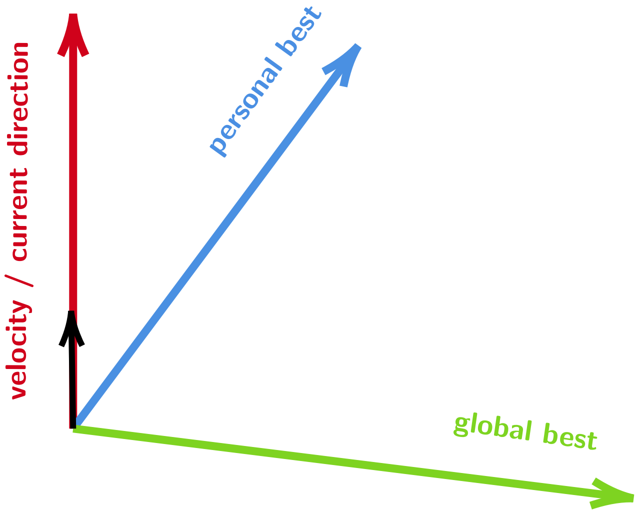
Position and Velocity Updates



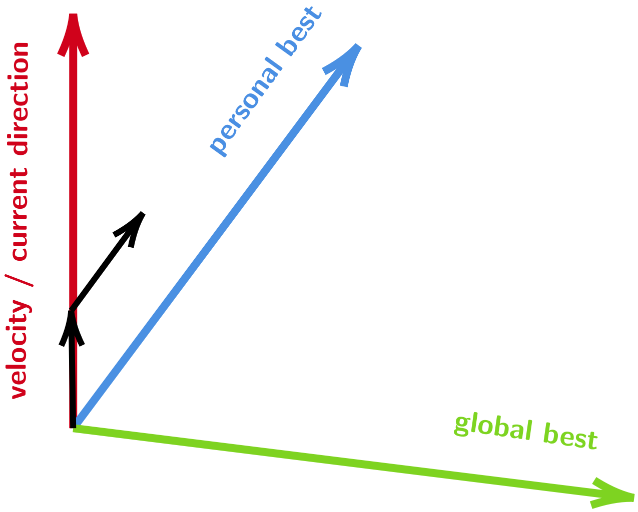
Position and Velocity Updates



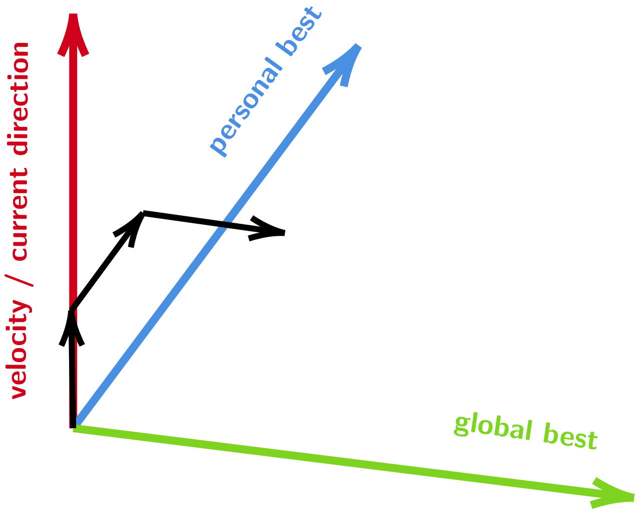
Position and Velocity Updates



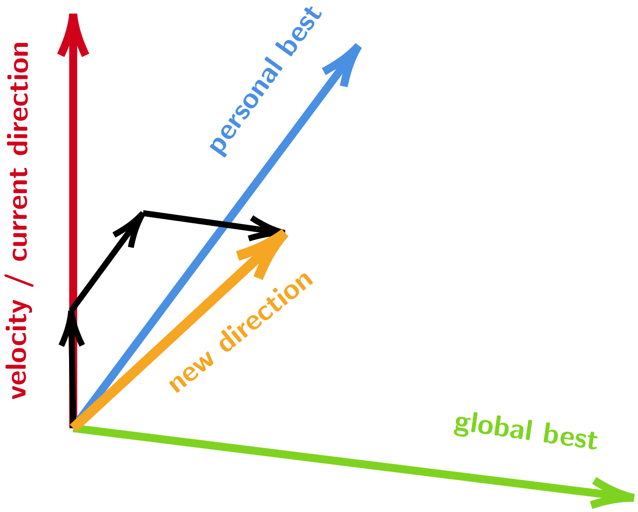
Position and Velocity Updates



Position and Velocity Updates



Position and Velocity Updates



Update Equations

- For each i^{th} particle, first update velocity, then update position.

Update Equations

- For each i^{th} particle, first update velocity, then update position.
- Informally, $v_i^{new} = v_i^{old} + r_1 c_1 (pBest_i^{old} - x_i^{old}) + r_2 c_2 (gBest_i^{old} - x_i^{old})$
- Formally, $v_i^{t+1} = v_i^t + r_1 c_1 (pBest_i^t - x_i^t) + r_2 c_2 (gBest_i^t - x_i^t)$

Update Equations

- For each i^{th} particle, first update velocity, then update position.
- Informally, $v_i^{new} = v_i^{old} + r_1 c_1 (pBest_i^{old} - x_i^{old}) + r_2 c_2 (gBest_i^{old} - x_i^{old})$
- Formally, $v_i^{t+1} = v_i^{t} + r_1 c_1 (pBest_i^{t} - x_i^{t}) + r_2 c_2 (gBest_i^{t} - x_i^{t})$
- c_1 and c_2 are positive constants whereas r_1, $r_2 \in \mathcal{U}(0,1)$

Update Equations

- For each i^{th} particle, first update velocity, then update position.
- Informally, $v_i^{new} = v_i^{old} + r_1 c_1 (pBest_i^{old} - x_i^{old}) + r_2 c_2 (gBest_i^{old} - x_i^{old})$
- Formally, $v_i^{t+1} = v_i^t + r_1 c_1 (pBest_i^t - x_i^t) + r_2 c_2 (gBest_i^t - x_i^t)$
- c_1 and c_2 are positive constants whereas r_1, $r_2 \in \mathcal{U}(0,1)$
- Inertia Weights proposed by **Y. Shi and R. Eberhart (1998.)** [1].
- $v_i^{t+1} = w v_i^t + r_1 c_1 (pBest_i^t - x_i^t) + r_2 c_2 (gBest_i^t - x_i^t)$

Update Equations

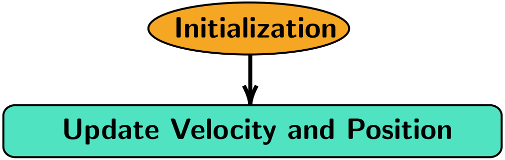
- For each i^{th} particle, first update velocity, then update position.
- Informally, $v_i^{new} = v_i^{old} + r_1 c_1 (pBest_i^{old} - x_i^{old}) + r_2 c_2 (gBest_i^{old} - x_i^{old})$
- Formally, $v_i^{t+1} = v_i^t + r_1 c_1 (pBest_i^t - x_i^t) + r_2 c_2 (gBest_i^t - x_i^t)$
- c_1 and c_2 are positive constants whereas r_1, $r_2 \in \mathcal{U}(0,1)$
- Inertia Weights proposed by **Y. Shi and R. Eberhart (1998.)** [1].
- $v_i^{t+1} = w v_i^t + r_1 c_1 (pBest_i^t - x_i^t) + r_2 c_2 (gBest_i^t - x_i^t)$
- w can also be a positive linear/non-linear function of time.

Update Equations

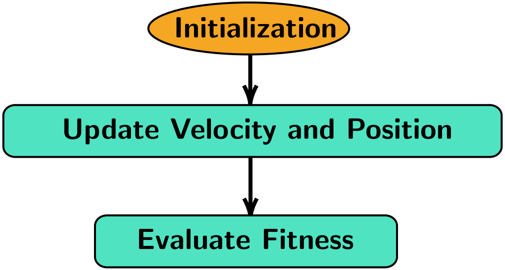
- For each i^{th} particle, first update velocity, then update position.
- Informally, $v_i^{new} = v_i^{old} + r_1 c_1 (pBest_i^{old} - x_i^{old}) + r_2 c_2 (gBest_i^{old} - x_i^{old})$
- Formally, $v_i^{t+1} = v_i^t + r_1 c_1 (pBest_i^t - x_i^t) + r_2 c_2 (gBest_i^t - x_i^t)$
- c_1 and c_2 are positive constants whereas r_1, $r_2 \in \mathcal{U}(0,1)$
- Inertia Weights proposed by **Y. Shi and R. Eberhart (1998.)** [1].
- $v_i^{t+1} = w v_i^t + r_1 c_1 (pBest_i^t - x_i^t) + r_2 c_2 (gBest_i^t - x_i^t)$
- w can also be a positive linear/non-linear function of time.
- Update position, $x_i^{t+1} = x_i^t + v_i^{t+1}$

Flow Chart

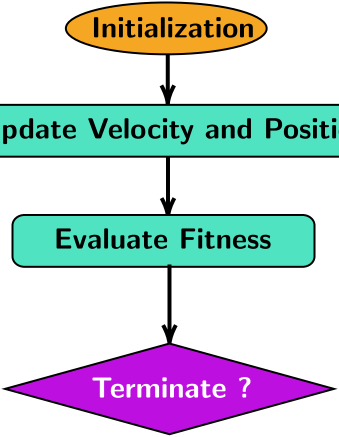
Flow Chart



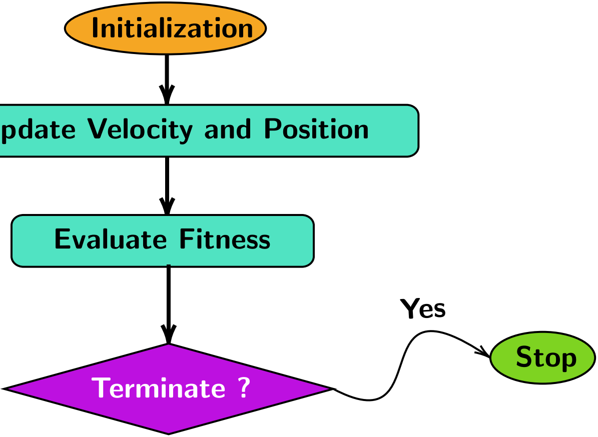
Flow Chart


Initialization
Update Velocity and Position
Evaluate Fitness

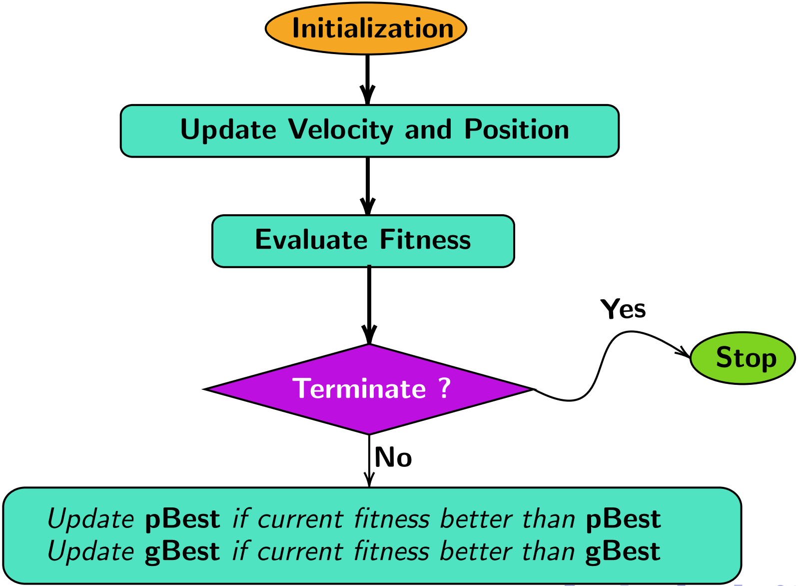
Flow Chart


Initialization
Update Velocity and Position
Evaluate Fitness
Terminate ?

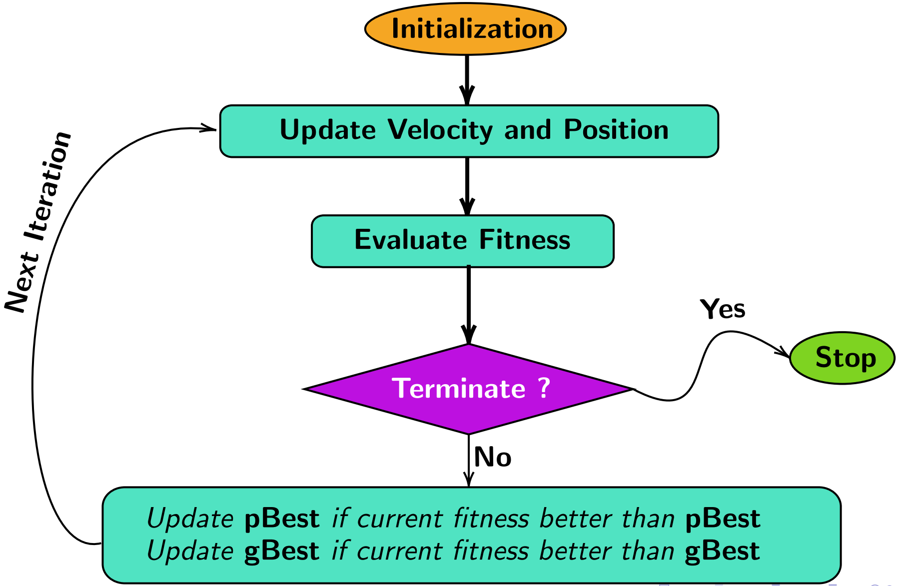
Flow Chart


Initialization
Update Velocity and Position
Evaluate Fitness
Terminate ?
Yes
Stop

Flow Chart


Initialization
Update Velocity and Position
Evaluate Fitness
Terminate ?
Yes
Stop
No
Update **pBest** *if current fitness better than* **pBest**
Update **gBest** *if current fitness better than* **gBest**

Flow Chart


Initialization
Update Velocity and Position
Evaluate Fitness
Terminate ?
Yes
Stop
No
Update **pBest** *if current fitness better than* **pBest**
Update **gBest** *if current fitness better than* **gBest**
Next Iteration

Fitting PSO to Longest Path

- We will take inspiration from the way PSO is *applied* for Shortest Path, proposed by **Mohemmad et al. (2008)** [2] and for TSP, proposed by **Wang et al. (2003)** [3].

Fitting PSO to Longest Path

- We will take inspiration from the way PSO is *applied* for Shortest Path, proposed by **Mohemmad et al. (2008)** [2] and for TSP, proposed by **Wang et al. (2003)** [3].
- Two strategies to encode **path** as **particles**.
  1. **Direct** Encoding
     Considers an array of node IDs as each particle.

Node Idx	1	2	3	4	5

  2. **Indirect** (Priority-based) Encoding
     Considers a dynamic priority array indexed by node IDs.

Node Idx	1	2	3	4	5
Priority	574	651	670	1000	517

Fitness Function

- We will use the inverse of the fitness function used for the Shortest Path Problem.

Fitness Function

- We will use the inverse of the fitness function used for the Shortest Path Problem.
- Let ID_i be the array of node indices for the i^{th} *particle*.

Fitness Function

- We will use the inverse of the fitness function used for the Shortest Path Problem.
- Let ID_i be the array of node indices for the i^{th} *particle*.
- Fitness function for the i^{th} particle will be
  $$F_i = \sum_{j=1}^{|ID_i|-1} w(ID_i(j), ID_i(j+1))$$
  where $w(i_1, i_2)$ represents the weight of the edge between two nodes with node indices i_1 and i_2.

Fitness Function

- We will use the inverse of the fitness function used for the Shortest Path Problem.
- Let ID_i be the array of node indices for the i^{th} *particle*.
- Fitness function for the i^{th} particle will be
  $$F_i = \sum_{j=1}^{|ID_i|-1} w(ID_i(j), ID_i(j+1))$$
  where $w(i_1, i_2)$ represents the weight of the edge between two nodes with node indices i_1 and i_2.
- If there are edges in candidate solution which **does not** exist in actual graph/network, then apply **penalty** by using a large negative number (to simulate $-\infty$) so that this candidate solution is not used.

Fitness Function

- We will use the inverse of the fitness function used for the Shortest Path Problem.
- Let ID_i be the array of node indices for the i^{th} *particle*.
- Fitness function for the i^{th} particle will be

  $$F_i = \sum_{j=1}^{|ID_i|-1} w(ID_i(j), ID_i(j+1))$$

  where $w(i_1, i_2)$ represents the weight of the edge between two nodes with node indices i_1 and i_2.
- If there are edges in candidate solution which **does not** exist in actual graph/network, then apply **penalty** by using a large negative number (to simulate $-\infty$) so that this candidate solution is not used.
- The higher the value of this fitness function, the better candidate solution is proposed.

Approach using Direct Encoding

- Direct Encoding is pretty straightforward.

Approach using Direct Encoding

- Direct Encoding is pretty straightforward.
- Randomly select K number of *particles* i.e. K **paths** by using array of node indices.

Approach using Direct Encoding

- Direct Encoding is pretty straightforward.
- Randomly select K number of *particles* i.e. K **paths** by using array of node indices.
- For each *particle*, find $gBest$ and $pBest$.

Approach using Direct Encoding

- Direct Encoding is pretty straightforward.
- Randomly select K number of *particles* i.e. K **paths** by using array of node indices.
- For each *particle*, find $gBest$ and $pBest$.
- Update each *particle's* velocity and position by updating the array of node indices.

Approach using Direct Encoding

- Direct Encoding is pretty straightforward.
- Randomly select K number of *particles* i.e. K **paths** by using array of node indices.
- For each *particle*, find $gBest$ and $pBest$.
- Update each *particle's* velocity and position by updating the array of node indices.
- Repeat until termination criterion is satisfied.
    1. Maximum number of iterations
    2. Minimum value of Longest Path to obtain
    3. Other custom conditions

Approach using Priority-Based Encoding (Indirect Encoding)

- For convenience, we will assume a starting node and ending node is given eg. **start** at node index **4**, **end** at node index **1**.

Approach using Priority-Based Encoding (Indirect Encoding)

- For convenience, we will assume a starting node and ending node is given eg. **start** at node index **4**, **end** at node index **1**.
- Initially, random values are assigned as priorities for each **particle** i.e. array of node indices with priority values.

Node Idx	1	2	3	4	5
Priority	574	651	670	1000	517

Approach using Priority-Based Encoding (Indirect Encoding)

- For convenience, we will assume a starting node and ending node is given eg. **start** at node index **4**, **end** at node index **1**.
- Initially, random values are assigned as priorities for each **particle** i.e. array of node indices with priority values.

Node Idx	1	2	3	4	5
Priority	574	651	670	1000	517

- Maintain a descending order for priority.

Node Idx	**1**	2	3	**4**	5
Priority	574	651	670	1000	517
Descending order	4	3	2	1	5

Indirect Encoding - Per Iteration Work

- Keep the candidate solution as the path starting from **4** and ending at **1** by using the descending order of priority values. This can be kept in dynamic list to save memory.

Node Idx in Path	4	3	2	1

Indirect Encoding - Per Iteration Work

- Keep the candidate solution as the path starting from **4** and ending at **1** by using the descending order of priority values. This can be kept in dynamic list to save memory.

Node Idx in Path	4	3	2	1

- Calculate F_i i.e. the **fitness value** for this candidate solution path.

Indirect Encoding - Per Iteration Work

- Keep the candidate solution as the path starting from **4** and ending at **1** by using the descending order of priority values. This can be kept in dynamic list to save memory.

Node Idx in Path	4	3	2	1

- Calculate F_i i.e. the **fitness value** for this candidate solution path.
- **If** $F_i > Fitness(pBest_i)$ **then** update $pBest_i = F_i$.
- **If** $F_i > Fitness(gBest)$ **then** update $gBest = F_i$.

Indirect Encoding - Exploration Step

- For exploration, pseudo-random numbers are added to the priority array values for all nodes **except** for the source node.
  Maximum limit of variation << max(priority array)

Indirect Encoding - Exploration Step

- For exploration, pseudo-random numbers are added to the priority array values for all nodes **except** for the source node.
  Maximum limit of variation << max(priority array)
- Example:

Node Idx	**1**	2	3	**4**	5
Priority^{old}	574	651	670	1000	517
Variation	+50	+80	-30	+0	+155
Priority^{new}	+624	+731	+640	1000	+672
Descending order	5	2	4	1	3
New possible path idx	**4**	2	5	3	**1**

Indirect Encoding - Swap with Personal Best

- To swap with personal best (for updating with respect to $pBest$), find common nodes between $pBest$ and current path to update the current path.

Indirect Encoding - Swap with Personal Best

- To swap with personal best (for updating with respect to $pBest$), find common nodes between $pBest$ and current path to update the current path.
- Suppose $pBest$ is as follows:

Node Idx	4	3	1	7	2

Indirect Encoding - Swap with Personal Best

- To swap with personal best (for updating with respect to $pBest$), find common nodes between $pBest$ and current path to update the current path.
- Suppose $pBest$ is as follows:

Node Idx	4	3	1	7	2

- Current path (before swapping):

Node Idx	4	3	7	9	1	5	8	2

Indirect Encoding - Swap with Personal Best

- To swap with personal best (for updating with respect to $pBest$), find common nodes between $pBest$ and current path to update the current path.
- Suppose $pBest$ is as follows:

Node Idx	4	3	1	7	2

- Current path (before swapping):

Node Idx	4	3	7	9	1	5	8	2

- Move **7** one step towards target index.
  Target index is $^{4}/_{5}{}^{th}$ from the initial position.
- Current path (after swapping):

Before Swap	4	3	7	9	1	5	8	2
After Swap	4	3	**9**	**7**	1	5	8	2

Indirect Encoding - Replacing with Global Best

- To replace with global best (for updating with respect to $gBest$), we first segment and then replace.
- For segmenting, we keep lower half of current path, and replace upper half of $gBest$.
- For replacing, we choose a node in $gBest$ but **not** in current path, and proportionally replace that node.

Indirect Encoding - Replacing with Global Best

- To replace with global best (for updating with respect to $gBest$), we first segment and then replace.
- For segmenting, we keep lower half of current path, and replace upper half of $gBest$.
- For replacing, we choose a node in $gBest$ but **not** in current path, and proportionally replace that node.
- Suppose $gBest$ is as follows:

Node Idx	4	10	6	3	2

Indirect Encoding - Replacing with Global Best

- To replace with global best (for updating with respect to $gBest$), we first segment and then replace.
- For segmenting, we keep lower half of current path, and replace upper half of $gBest$.
- For replacing, we choose a node in $gBest$ but **not** in current path, and proportionally replace that node.
- Suppose $gBest$ is as follows:

Node Idx	4	10	6	3	2

- Segmenting with $gBest$:

Before Swap	4	8	9	5	1	6	7	2
After Swap	4	8	9	5	1	6	**3**	**2**

Indirect Encoding - Replacing with Global Best

- To replace with global best (for updating with respect to $gBest$), we first segment and then replace.
- For segmenting, we keep lower half of current path, and replace upper half of $gBest$.
- For replacing, we choose a node in $gBest$ but **not** in current path, and proportionally replace that node.
- Suppose $gBest$ is as follows:

Node Idx	4	10	6	3	2

- Segmenting with $gBest$:

Before Swap	4	8	9	5	1	6	7	2
After Swap	4	8	9	5	1	6	**3**	**2**

- Replacing with $gBest$ (10 in $gBest$ is ${}^{2}/_{5}{}^{th}$ away from the initial position)

Before Replacement	4	8	9	5	1	6	3	2
After Replacement	4	8	**10**	5	1	6	3	2

References I

David Portugal, Carlos Henggeler Antunes, and Rui Rocha.
A study of genetic algorithms for approximating the longest path in generic graphs.
In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 2539–2544. IEEE, 2010.

References II

J. Kennedy and R. Eberhart.
Particle swarm optimization.
In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

References III

Y. Shi and R. Eberhart.
A modified particle swarm optimizer.
In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73, 1998.

Ammar W. Mohemmed, Nirod Chandra Sahoo, and Tan Kim Geok.
Solving shortest path problem using particle swarm optimization.
Applied Soft Computing, 8(4):1643 – 1653, 2008.
Soft Computing for Dynamic Data Mining.

Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, and Wei Pang.
Particle swarm optimization for traveling salesman problem.
In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, volume 3, pages 1583–1585 Vol.3, 2003.