School of Systems&Enterprises

EM 605

Introduction to Operations Research

Metaheuristics

STEVENS INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY

Topics

- Metaheuristics
  - What is a metaheuristic?
  - Tabu Search
  - Simulated Annealing
  - Genetic Algorithms

Heuristic Methods

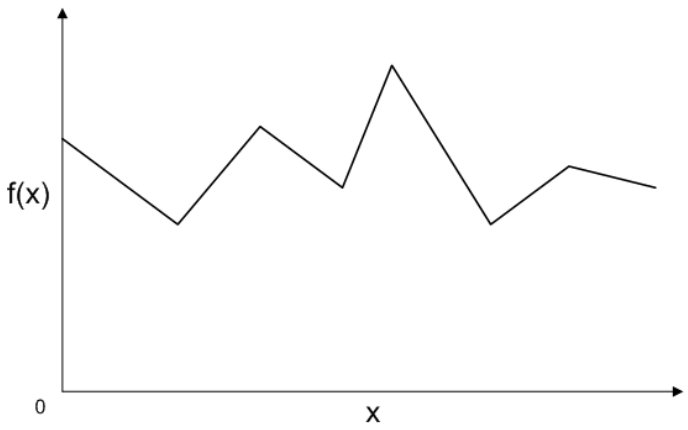
- Some problems are very complicated, and it is not possible to solve for an optimal solution
- Heuristic methods discover a very good feasible solution, but not necessarily an optimal solution
- No guarantee can be given about the quality of a solution – it depends on the design of the heuristic method
- The method is usually a full-fledged iterative algorithm
- Heuristic methods are often based on carefully tailored, simple common-sense ideas – but are often very specific to the problem under investigation

Metaheuristics

- A general solution method that provides general structure and strategy guidelines for developing a specific heuristic method for a specific problem
- This generality means we don’t need to start from scratch to develop a heuristic method

Local Improvement Procedure

- Starts with a trial solution and searches in the neighborhood to find a better solution
- Will usually converge to a local optimum
- This local optimum may not be the global optimum
- Global optimum will be found only if the initial trial solution is in the neighborhood of the global optimum
- The procedure can be repeated by selecting various initial trial solutions

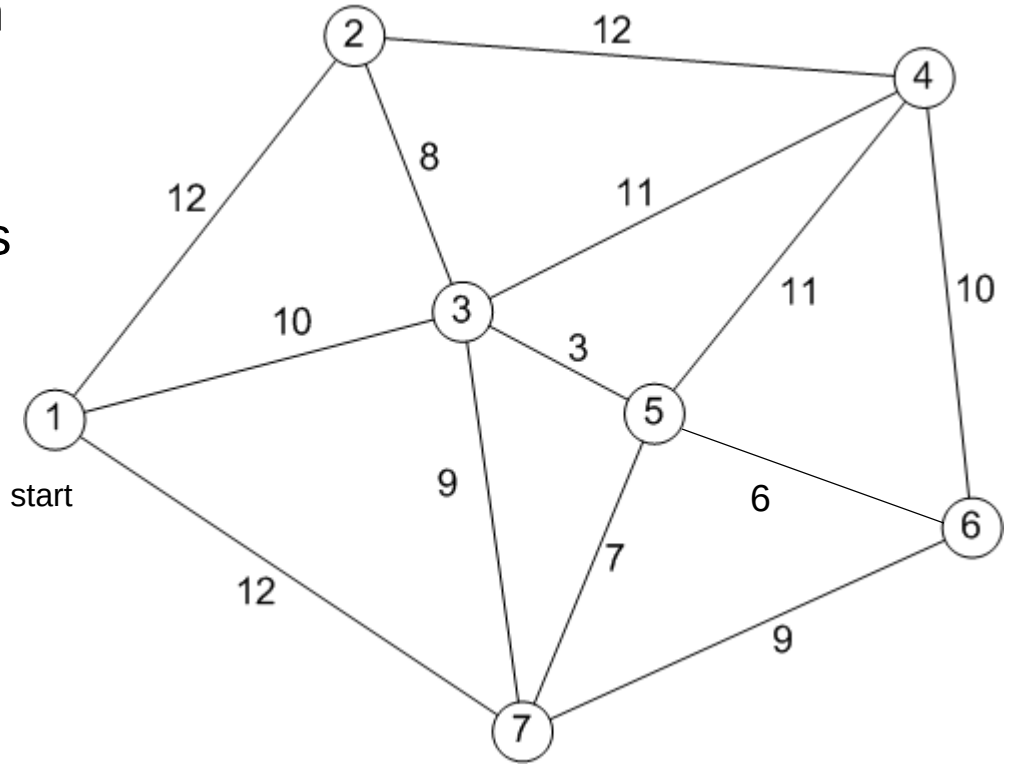


Nature of Metaheuristics

- Orchestrates the interaction between local improvement procedures and higher level strategies to create a process that is capable of escaping from local optima and performing a robust search of the feasible region
- Different metaheuristics execute this escape in different ways
- Trial solutions that immediately follow a local optimum are allowed to be inferior to this local optimum

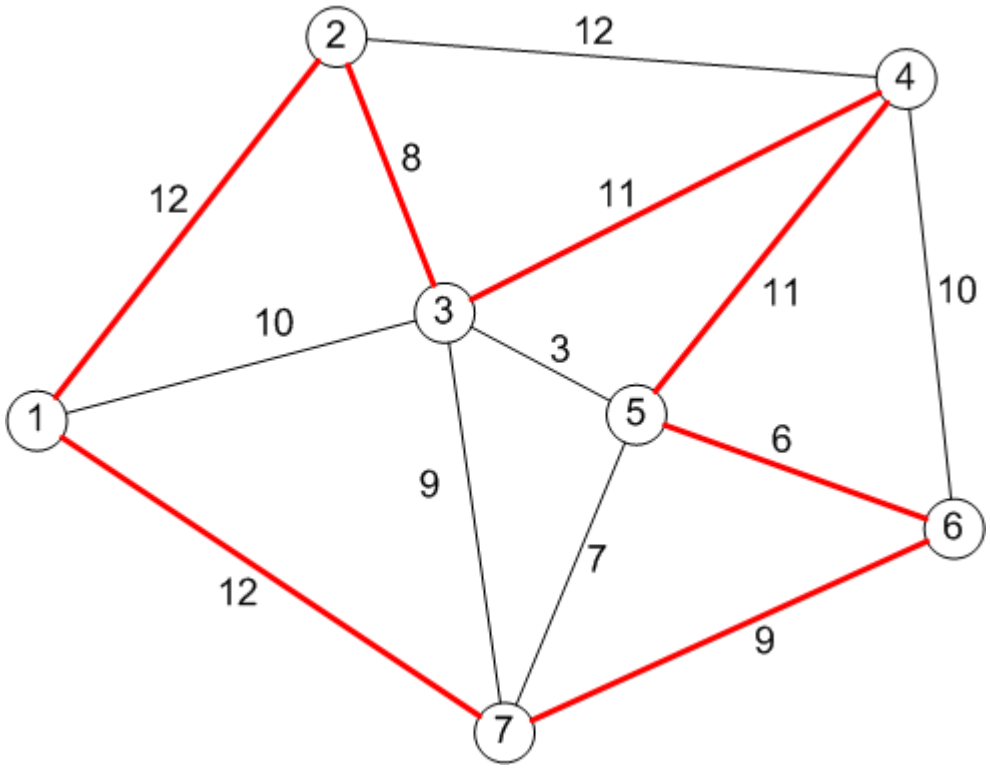
An Example of a Metaheuristic Method The Traveling Salesman Problem

- Starting from city 1, the salesman must travel to all cities once before returning home
- The distance between each city is given, and is assumed to be the same in both directions
- Only the links shown are to be used
- Objective - Minimize the total distance to be travelled



Traveling Salesman Problem

Pick, develop, generate, or devise an initial solution



1-2-3-4-5-6-7-1 is a feasible solution with a total distance of 69

Reversing the entire tour is the same solution 1-7-6-5-4-3-2-1

Sub-Tour Reversal Algorithm

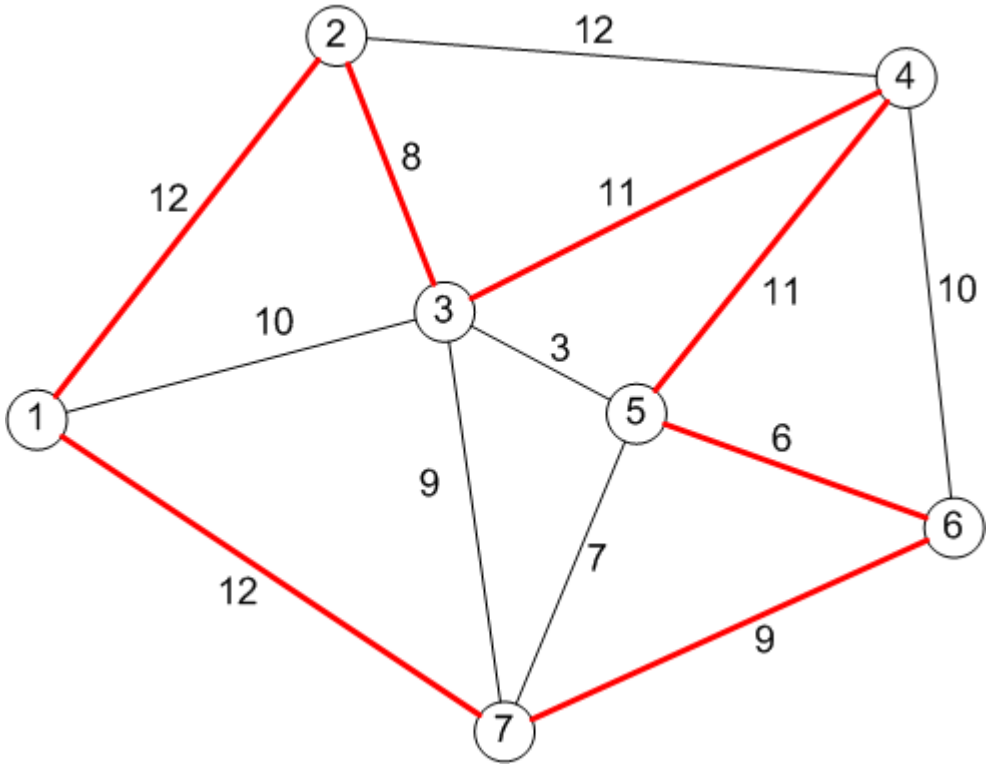
- This method selects a sub-sequence of cities in a trial solution and simple reverses the order of that sub-sequence
- Sub-sequence can consist of two or more cities
- Procedure:
  - Start with any feasible tour
  - Consider all possible ways of performing sub-tour reversal
  - Select the one that provides the largest decrease in distance
  - Stop when no sub-tour reversal will improve the current trial solution

Traveling Salesman Problem

Initial trial solution:

1-2-3-4-5-6-7-1 distance 69

Four sub-tours are possible in this iteration



Traveling Salesman Problem

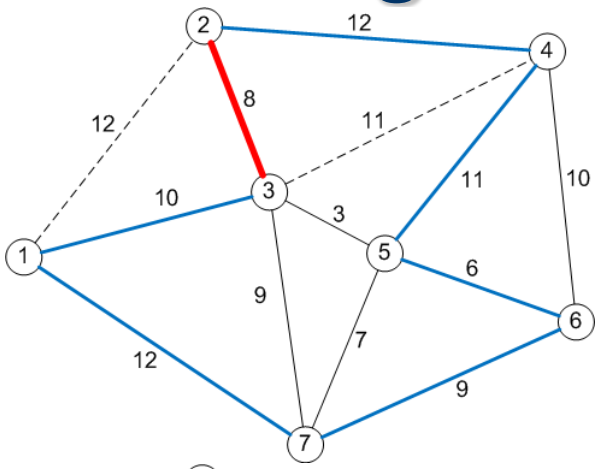
Sub-tour 1

Reverse 2-3

1-**3-2**-4-

5-6-7-1

Distance: **68**



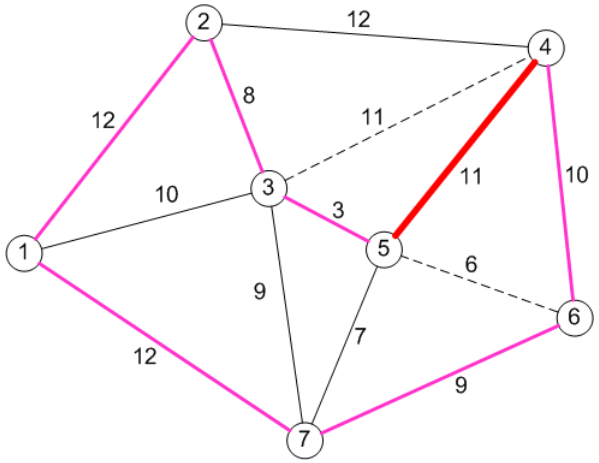
Sub-tour 3

Reverse 4-5

1-2-3-

5-4-6-7-1

Distance: **65**



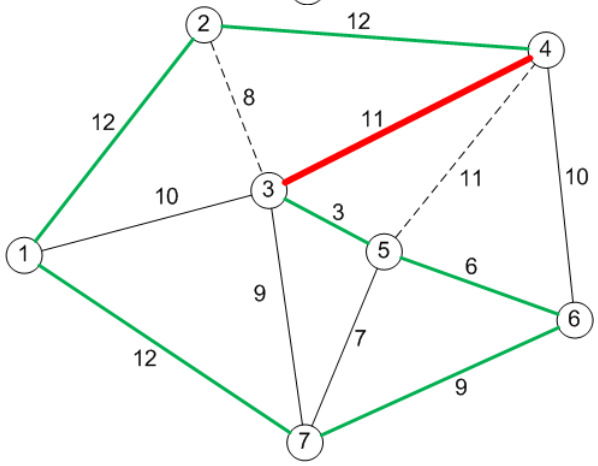
Sub-tour 2

Reverse 3-4

1-2-**4-3**-

5-6-7-1

Distance: **65**



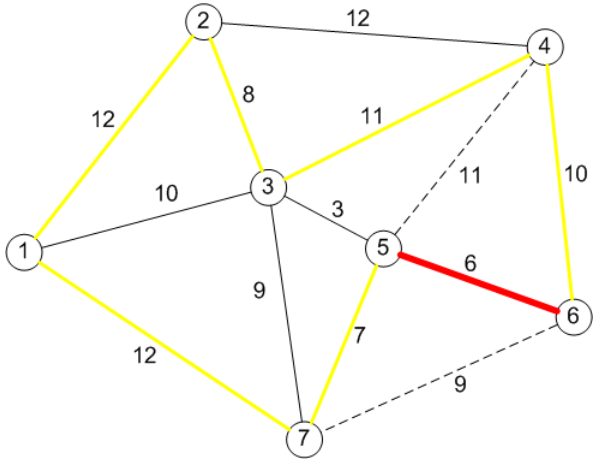
Sub-tour 4

Reverse 5-6

1-2-3-4-

6-5-7-1

Distance: **66**



Sub-tours 2 and 3 give the least distance of 65 - sub-tour 2 is arbitrarily chosen as the next trial solution

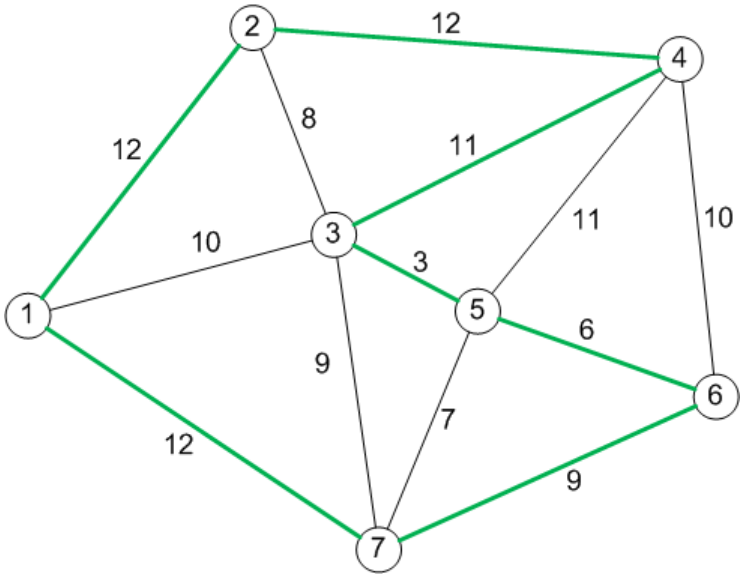
Traveling Salesman Problem

In the next iteration, only one sub-tour is possible

Initial trial solution

1-2-**4-3**-5-6-7-1

Distance: 65

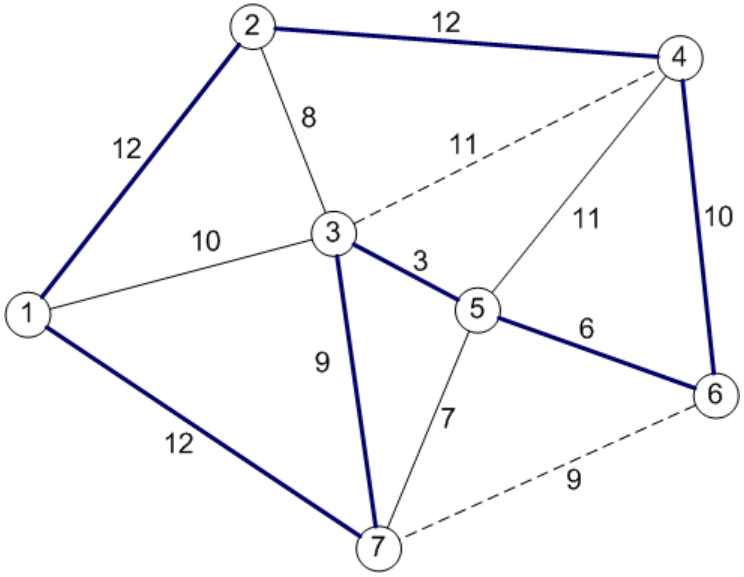


Sub tour

Reverse **3-5-6**

1-2-4-**6-5-3**-7-1

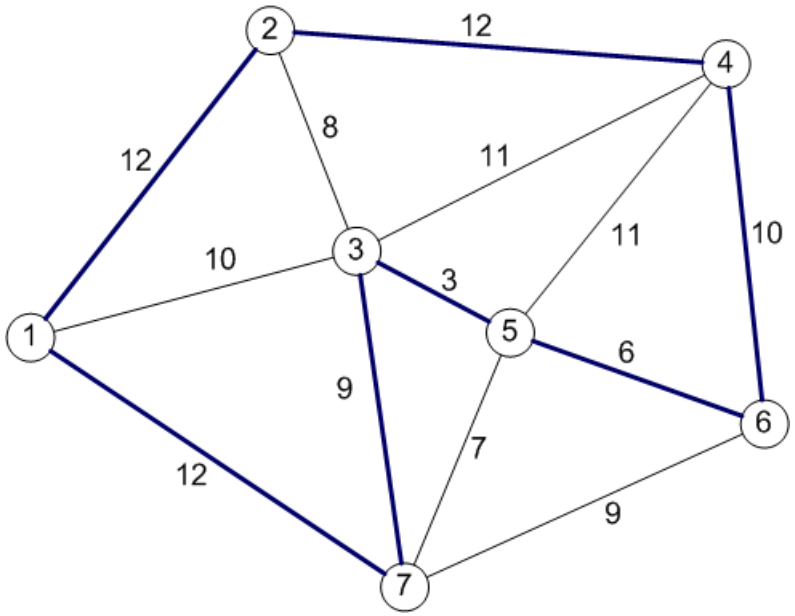
Distance: 64



Traveling Salesman Problem

No further sub-tours are possible

So, this is the final solution of this sub-tour reversal algorithm



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

Distance: 64

But this is not the optimal solution, which is 63

This solution cannot be obtained by this algorithm

Metaheuristic Methods

- **Tabu Search**
- Simulated Annealing
- Genetic Algorithms

Tabu Search

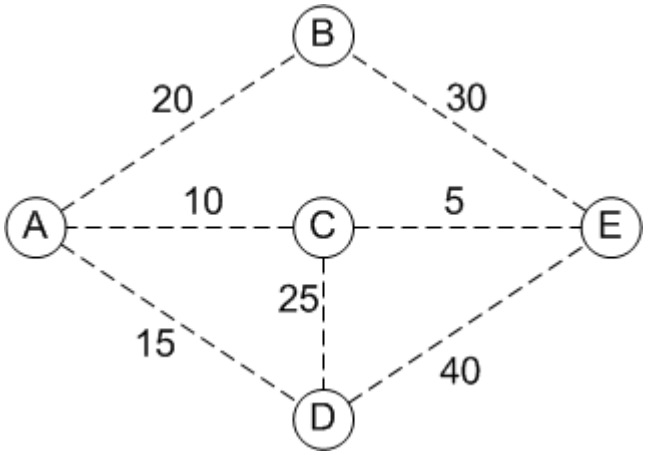
- Includes a “local search procedure” subroutine
  - Begins as a local improvement procedure in the usual way
  - Continues search by allowing non-improving moves
- Sometime referred to as ***steepest ascent/mildest descent*** approach
- The danger is that the process can cycle back to the same optimum
  - This is avoided by maintaining a tabu list which records “tabu moves” that would return to a solution recently visited
- Can incorporate advanced concepts like intensification and diversification

Tabu Search Algorithm

- Initialization
  - Start with a feasible initial trial solution
- Iteration
  - Use an appropriate local search procedure
  - Eliminate any move on the current tabu list except if it would result in a better solution
  - Adopt the next trial solution whether it is better or worse than the current trial solution
  - Update tabu list to forbid cycling back
  - If list is full, delete oldest member to provide flexibility
- Stopping rule
  - Use some stopping criterion like number of iterations, CPU time, or number of consecutive iterations without an improvement
  - Stop at any iteration where there are no feasible moves
  - Accept best solution found in any iteration so far

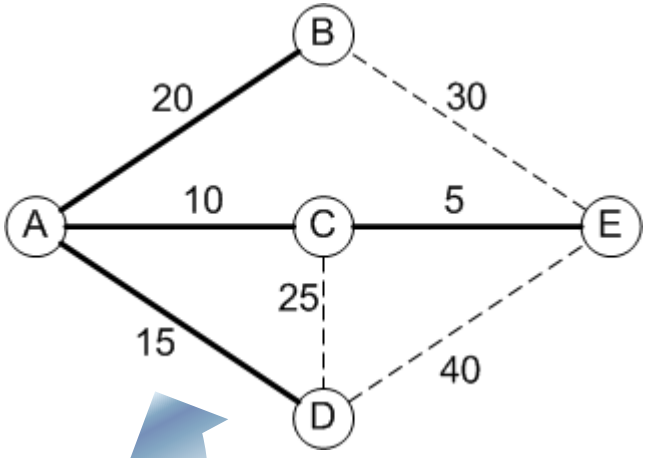
Min. Spanning Tree Problem

Network with potential links and costs



Optimal Solution

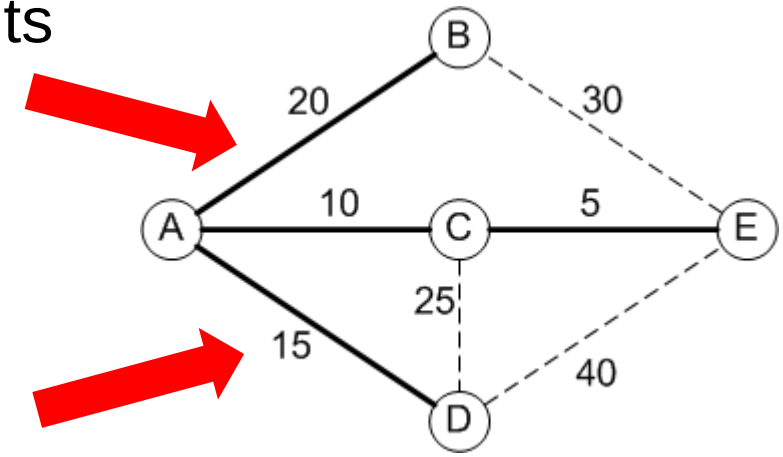
Cost = 50



Can you eyeball this and give a solution?

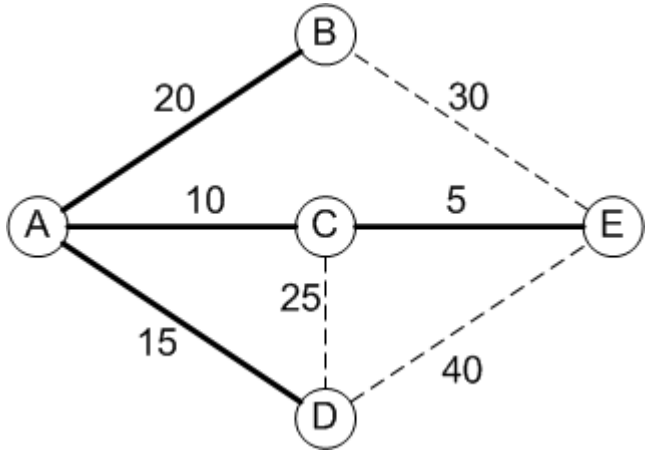
Min. Spanning Tree Problem

- To illustrate a tabu search, two constraints are added
  - Constraint 1: Link AD can be included only if link DE is also included
  - Constraint 2: At most one of the three links AD,CD and AB can be included
- As you can see here, the previous “optimal solution” violates both these constraints



Min. Spanning Tree Problem

- Penalties are added to account for the constraints
  - Charge a penalty of 100 if constraint 1 is violated
    - Constraint #1 - “Link AD can be included only if link DE is also included”
  - Tiered charges apply for constraint 2 violations
    - Constraint #2 – “At most one of the three links AD,CD and AB can be included
      - Charge 100 if two of the three links are included
      - Charge 200 if all three links are included
- With penalties, the cost of the original optimal solution goes from 50 to 250



Specifics on the Tabu search Algorithm

- Local search procedure
  - At each iteration, choose the best immediate neighbor of the current trial solution that is not ruled out by its tabu status
- Neighborhood structure
  - An immediate neighbor of the current trial solution is one that is reached by adding a single link and then deleting one of the other links in the cycle that is formed by the addition of this link
- Form of tabu moves
  - List the links that should not be deleted

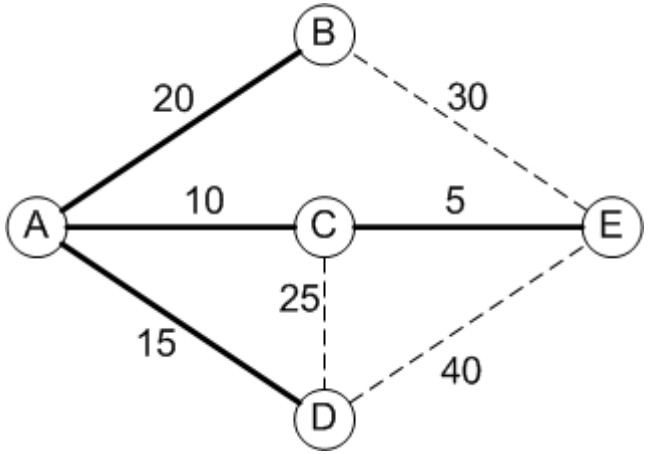
Specifics on the Tabu search Algorithm

- Addition of a tabu move
  - At each iteration, after choosing the link to be added, also add this link to the tabu list
- Maximum size of tabu list
  - Two – whenever a tabu move is added to a full list, delete the older of the two tabu moves already in the list
- Stopping rule
  - Stop after three consecutive iterations without an improvement in the best objective function value
  - Also stop when there are no immediate neighbors not ruled out by their tabu status

Initialization

Previous optimal solution is chosen to be the initial trial solution

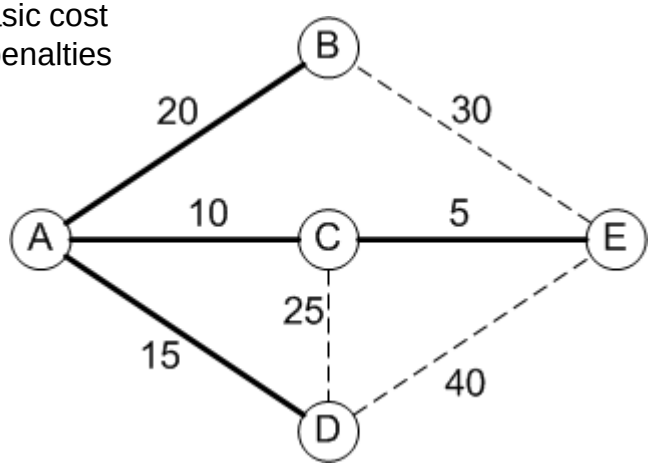
Including penalties for violating the constraints, the cost becomes 250

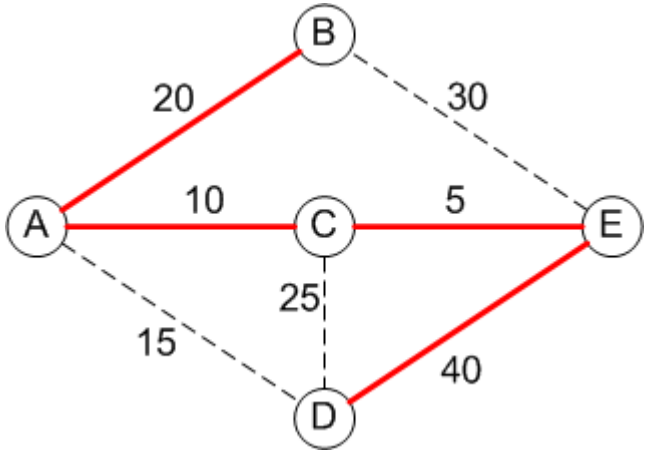


Iteration 1

Link to Add	Link to 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**

75 = basic cost
200 = penalties



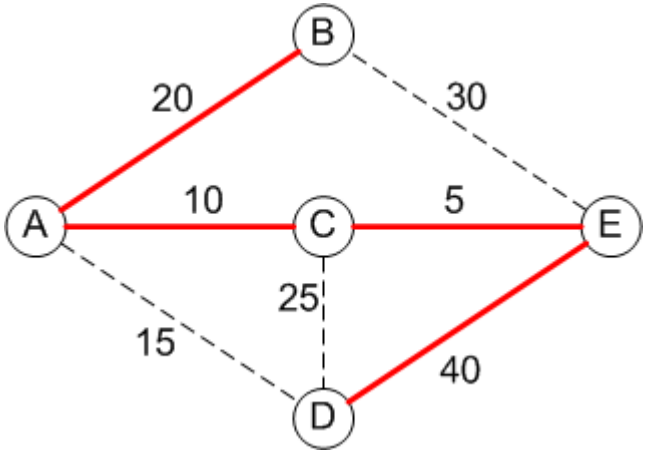


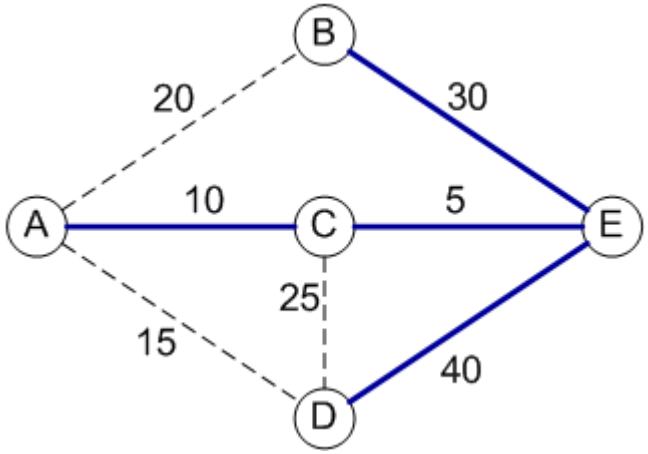
Tabu list:

1. DE
2. --

Iteration 2

Link to Add	Link to 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*	(Tabu move)
CD	CE	95 + 100 = 195



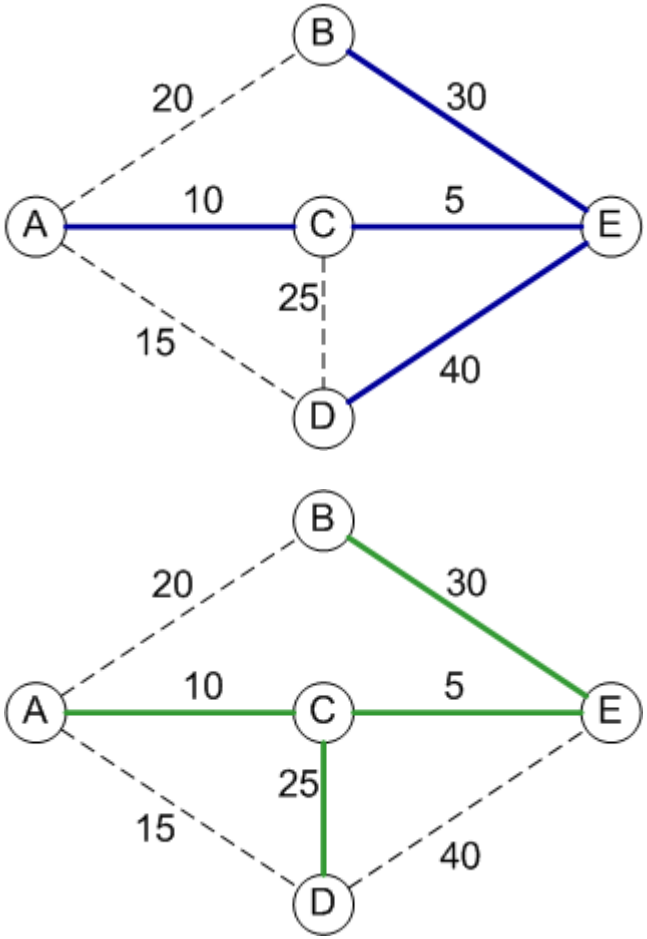


Tabu list:

1. BE
2. DE

Iteration 3

Link to Add	Link to 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



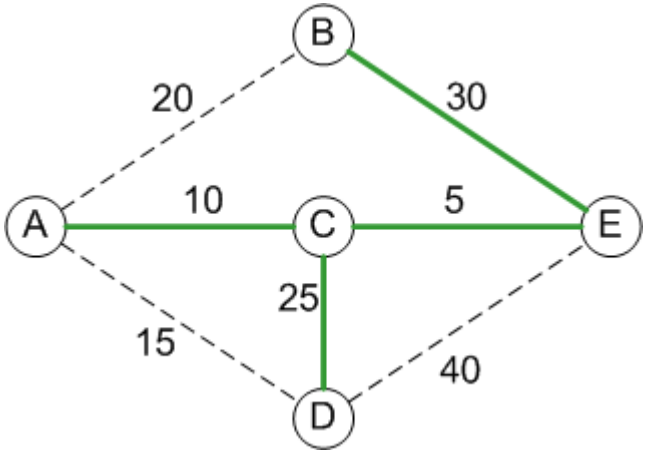
Tabu list:

1. CD
2. BE

* Tabu move is considered because it finds a better solution

Min. Spanning Tree Problem

This solution found in iteration 3 is the optimal solution, but the tabu search algorithm will not know this yet



Tabu Search Algorithm – Traveling Salesman

- Local search procedure
  - At each iteration, choose the best immediate neighbor of the current trial solution that is not ruled out by its tabu status
- Neighborhood structure
  - An immediate neighbor of the current trial solution is one that is reached by making a sub-tour reversal.
  - **Such a reversal requires adding two links and deleting two links**
- Form of tabu moves
  - List the links such that a particular sub-tour reversal would be tabu if **both** links to be deleted in this reversal are on the list

Tabu Search Algorithm – Traveling Salesman

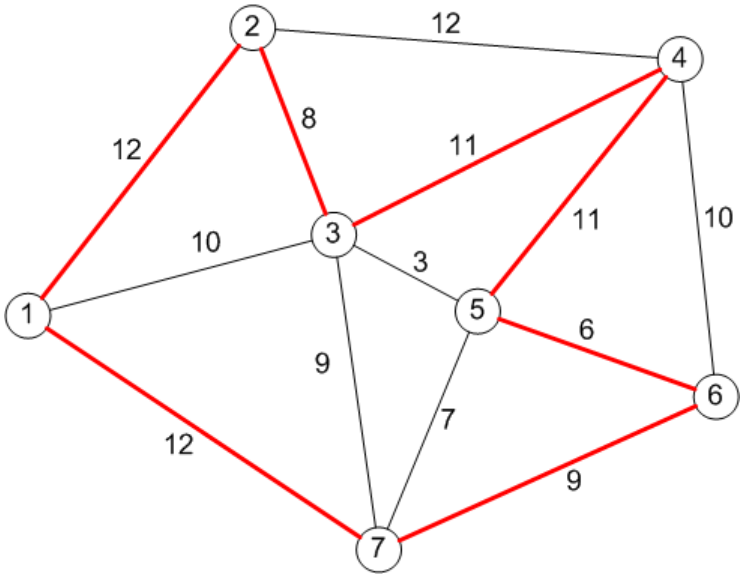
- Addition of a tabu move
  - At each iteration, after choosing the two links to be added, also add these two links to the tabu list
- Maximum size of tabu list
  - Four (two links each from two iterations)– whenever a pair of links is added to a full list, delete the older pair of two links already in the list
- Stopping rule
  - Stop after three consecutive iterations without an improvement in the best objective function value
  - Also stop when there are no immediate neighbors not ruled out by their tabu status

Tabu Search Algorithm – Traveling Salesman

Initial trial solution

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

Distance: 69



Iteration 1

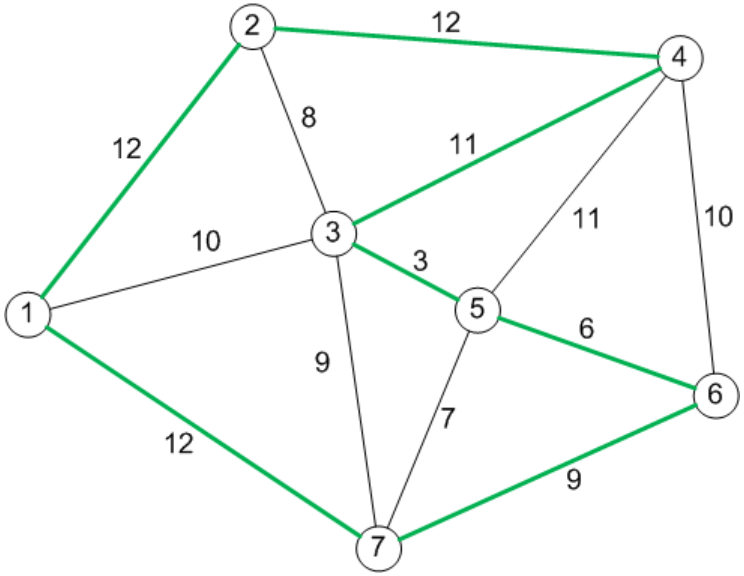
Choose to reverse 3-4

Delete links 2-3 and 4-5

Add links 2-4 and 3-5

New Solution 1-2-4-3-5-6-7-1

Distance: 65



Tabu Search Algorithm – Traveling Salesman

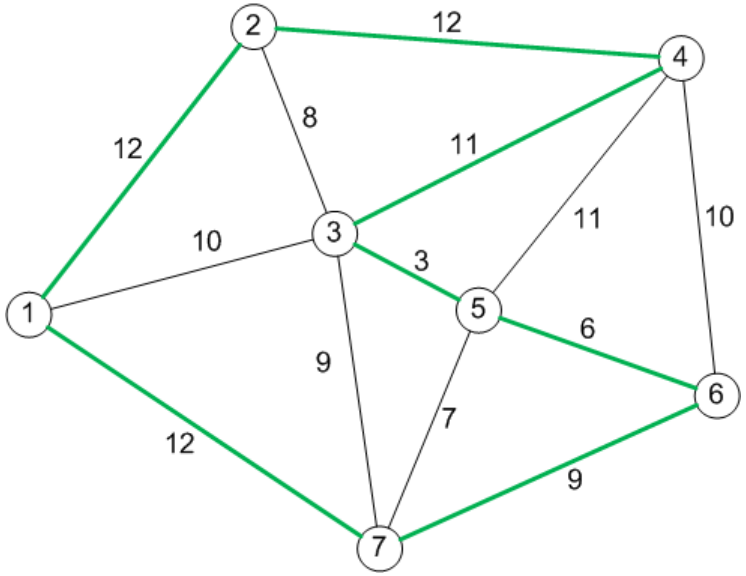
Iteration 1

Solution 1-2-4-3-5-6-7-1

Distance: 65

Tabu List

2-4 and 3-5



Iteration 2

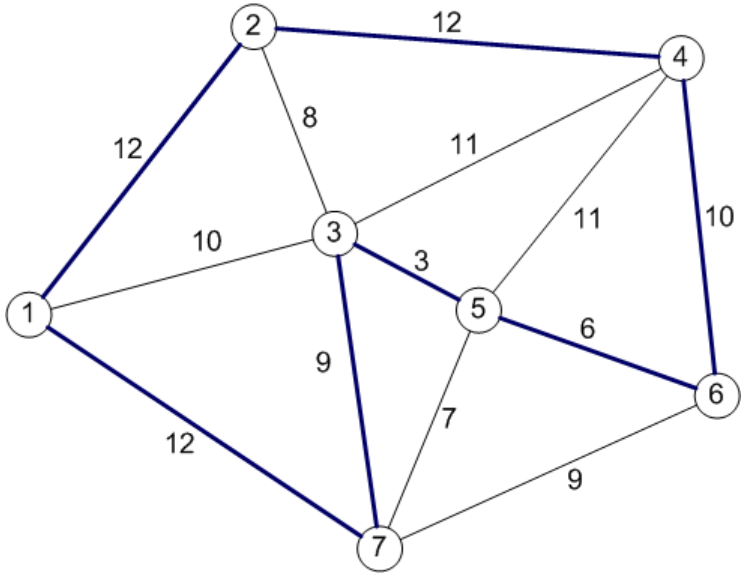
Choose to reverse 3-5-6

Delete links 4-3 and 6-7

Add links 4-6 and 3-7

New Solution 1-2-4-6-5-3-7-1

Distance: 64



Tabu Search Algorithm – Traveling Salesman

Iteration 2

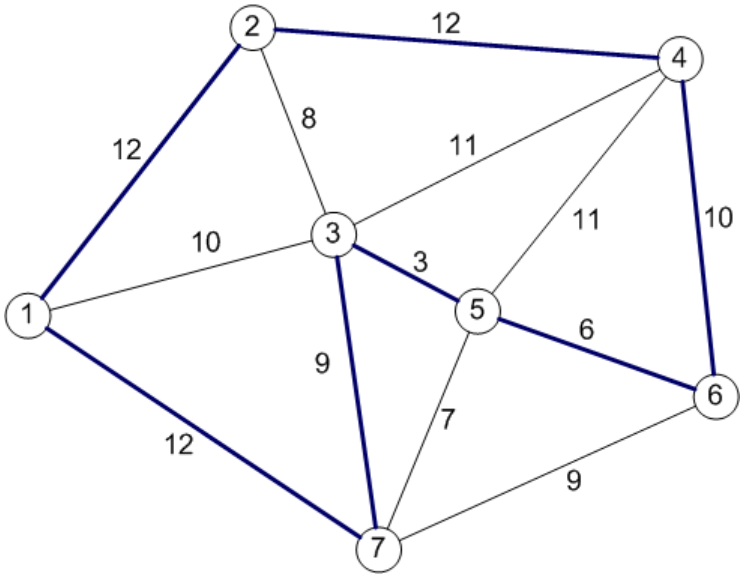
Solution 1-2-4-6-5-3-7-1

Distance: 64

Tabu List

4-6 and 3-7

2-4 and 3-5



Iteration 3

There are only 2 immediate neighbors

1. Reverse 6-5-3 to form
   1-2-4-3-5-6-7-1; Distance 65
2. Reverse 3-7 to form
   1-2-4-6-5-7-3-1; Distance 66

But option 1 uses tabu moves

So, choose to reverse 3-7

Delete links 5-3 and 7-1

Add links 5-7 and 3-1

New Solution 1-2-4-6-5-7-3-1

Distance: 66

Tabu Search Algorithm – Traveling Salesman

Iteration 3

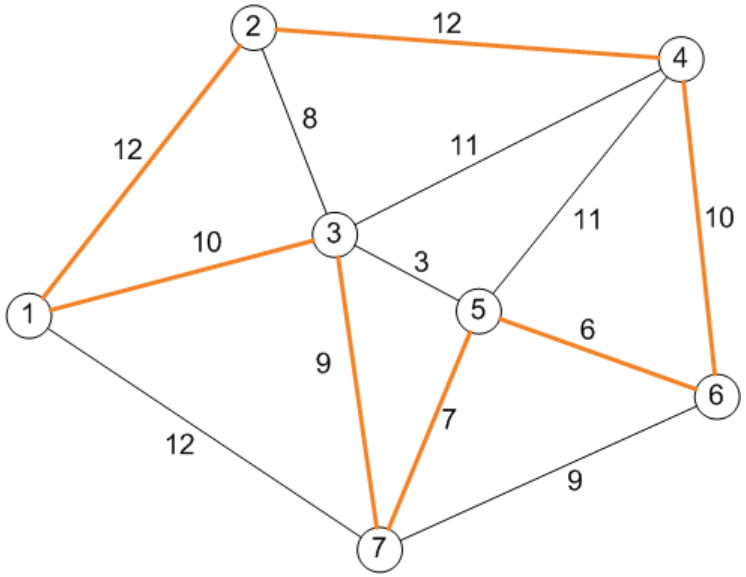
Solution 1-2-4-6-5-7-3-1

Distance: 66

Tabu List

5-7 and 3-1

4-6 and 3-7



Iteration 4

There are 4 immediate neighbors

1. Reverse 2-4-6-5-7 to form
   1-7-5-6-4-2-3-1; Distance 65
2. Reverse 6-5 to form
   1-2-4-5-6-7-3-1; Distance 69
3. Reverse 5-7 to form
   1-2-4-6-7-5-3-1; Distance 63
4. Reverse 7-3 to form
   1-2-4-6-5-3-7-1; Distance 64

Options 2&4 use tabu moves

Option 3 has better distance than option 1

Tabu Search Algorithm – Traveling Salesman

Iteration 4

Choose to reverse 5-7

Delete links 6-5 and 7-3

Add links 6-7 and 5-3

Solution 1-2-4-6-7-5-3-1

Distance: 63

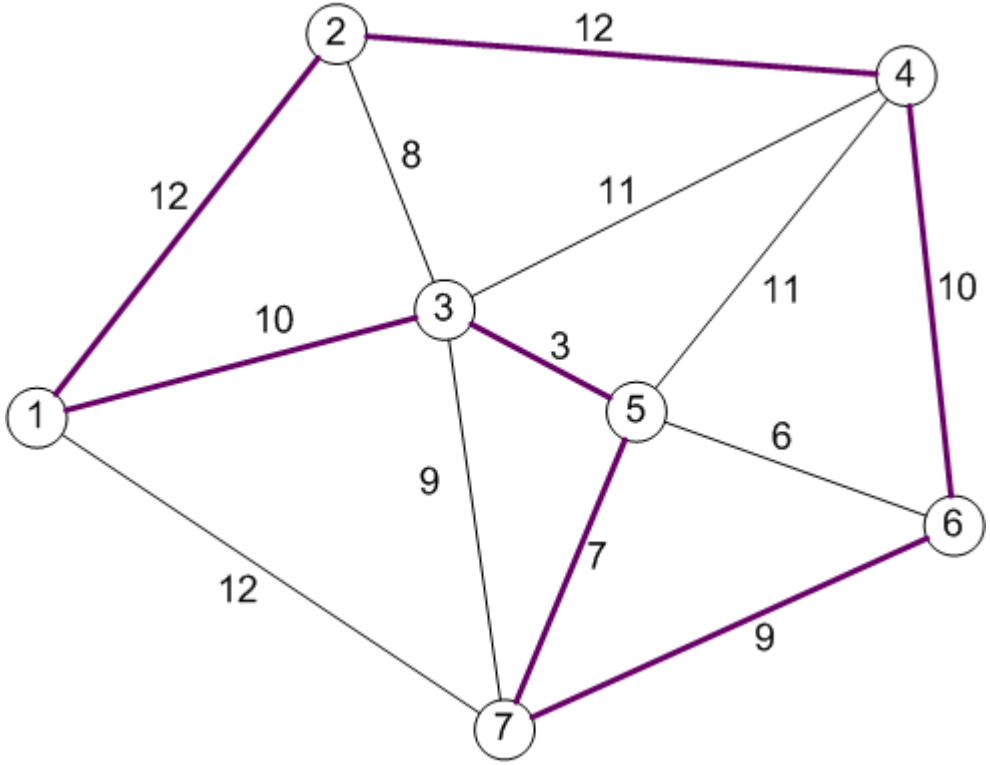
Tabu List

6-7 and 5-3

5-7 and 3-1

This is the optimal solution

The algorithm will terminate here because there are no further immediate neighbors, not knowing that the optimal solution is found



Simulated Annealing (SA)

- Focuses on searching for the “tallest hill”, which can be anywhere in the feasible region
- Early emphasis is on taking steps on random directions, to explore much of the feasible region
- The search will gradually gravitate toward the parts of the feasible region with the tallest hills
- Given enough time, the process often will reach and climb to the top of the tallest hill
- Difference from the tabu search lies in how an immediate neighbor is selected to be the next trial solution

Move Selection Rule

We specify some parameters:

Z_c = objective function value for the current trial solution (c=current)

Zn = objective function value for the current candidate to be the *next* trial solution (n=next)

T = a parameter that measures the tendency to accept the current candidate to be the next trial solution if it is not an improvement on the current trial solution

- Selection rule for maximization:
  - If $\mathbf{Z_n \geq Z_c}$, always accept this candidate
  - If $\mathbf{Z_n < Z_c}$, accept new candidate with probability: P(acceptance) = e^x here, $x=(Z_n - Z_c)/T$
- Selection rule for minimization:
  - If $\mathbf{Z_c \geq Z_n}$, always accept this candidate
  - If $\mathbf{Z_c < Z_n}$, accept new candidate with probability: P(acceptance) = e^x here, $x=(Z_c - Z_n)/T$

Value of ‘T’

- Acceptance of a candidate worse than the current solution depends on the value of T
- A slightly worse candidate has a higher probability of acceptance; there is very low probability of a much worse candidate to be accepted
- Simulated annealing starts with a relatively large value of T and then gradually decreases the value of T as the search continues
- Usually seen as a table of temperature values

Physical Annealing Process

- Simulated annealing (SA) process is based on the analogy to a physical annealing process
- Physical Annealing involves
  - Melting a metal at a high temperature and cooling it slowly until it reaches a low-energy stable state with desirable physical properties
- At any given temperature T in the process, the energy level of the atoms in the substance is fluctuating but tending to decrease
- A key question is to select an appropriate temperature schedule to use for our simulation of the physical process

SA – Traveling Salesman

- Initial trial solution
  - Any feasible solution may be entered; can be even generated randomly; 1-2-3-4-5-6-7-1 is a reasonable choice
- Neighborhood structure
  - An immediate neighbor of the current trial solution is one that is reached by making a sub-tour reversal
- Random selection of an immediate neighbor
  - Selecting a sub-tour to be reversed requires selecting the slot in the current sequence of cities where the sub-tour currently begins and then slot where the sub-tour currently ends
  - Random numbers are used to give equal probabilities to selecting any of the eligible starting and ending slots

SA – Traveling Salesman

- Temperature schedule
  - Five iterations are performed at each of five values of T
    - $T_1 = 0.2Z_c$
    - $T_2 = 0.5T_1$
    - $T_3 = 0.5T_2$
    - $T_4 = 0.5T_3$
    - $T_5 = 0.5T_4$

Notice how the values of T decrease as the number of iterations increases

Say $Z_c = 69$
For iterations 1 - 5
$T_1 = 69 * 0.2 = 13.8$
And for iterations
#6 - 10 $T_2 = 6.9$
#11 - 15 $T_3 = 3.45$
#16 - 20 $T_4 = 1.725$
#21 - 25 $T_5 = 0.8625$

    - Z_c is the objective function value for the initial trial solution
  - This is only an illustrative temperature schedule, but it is one that is typically used, especially for smaller problems
  - More iterations per temperature stage may be required for larger problems

SA – Traveling Salesman

- Initial trial solution
  - $1\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}1;\ Z_c = 69\ ;\ T_1 = 0.2Z_c = 13.8$
- Iteration 1
  - A sub-tour can begin from any city from 2 to 6
  - These 5 slots can be given *equal probability* as indicated below
    - 0.0000 – 0.1999: sub-tour begins in city 2
    - 0.2000 – 0.3999: sub-tour begins in city 3
    - 0.4000 – 0.5999: sub-tour begins in city 4
    - 0.6000 – 0.7999: sub-tour begins in city 5
    - 0.8000 – 0.9999: sub-tour begins in city 6

SA – Traveling Salesman

- Iteration 1
  - Suppose the random number generated for beginning the sub-tour was 0.2434, then the sub-tour would begin from city 3
  - Starting a sub-tour at city 3, there are 4 slots with equal probability where the sub tour can end
    - 0.0000 – 0.2499: sub-tour ends in city 4
    - 0.2500 – 0.4999: sub-tour ends in city 5
    - 0.5000 – 0.7499: sub-tour ends in city 6
    - 0.7500 – 0.9999: sub-tour ends in city 7
  - Suppose the random number generated for ending the sub-tour was 0.1423, then the sub-tour will end in city 4

SA – Traveling Salesman

- Reversing 3-4, the new solution is
  - 1-2-4-3-5-6-7-1; $Z_n = 65$
    - Since this is a feasible solution and $Z_n<Z_c$ this solution is automatically accepted to be the next trial solution (because our aim here is to **minimize** the distance)
  - If a feasible solution is not obtained, then random numbers would be generated again to find another solution
  - If the new solution is feasible but not a better solution
    - Let's say that 1-2-4-6-5-3-7-1 is our current solution ($Z_c=64$)
    - Let's say the next iteration reverses 3 and 7, giving us 1-2-4-6-5-3-7-1
    - So $Z_n= 66$; $Z_c= 64$; $T_1=13.8$
    - $X=(Z_c-Z_n)/T_1=(64-66)/13.8 = -0.1449$, $e^X = e^{-0.1449} = 0.865$
  - If the next random number generated is <0.865, this solution will be accepted, otherwise, it will be rejected

SA – Traveling Salesman

- Similarly, the remaining iterations following the temperature schedule will be performed
- The best trial solution is any iteration will be chosen as the best solution to the problem
- Due to the randomness built into the algorithm, the sequence of trial solutions will be different
  - For large problems, it is typical to reapply the SA algorithm several times to increase the chances of finding the optimal solution

Genetic Algorithms (GA)

- This algorithm is effective in exploring various parts of the feasible region and gradually evolving toward the best feasible region
- GA is based on the theory of evolution formulated by Charles Darwin
- The “survival of the fittest” phenomenon is evident in the process of natural selection

Genetic Algorithms (GA)

- In any species that reproduces, each offspring inherits some of the chromosomes from each of the two parents
- Genes within the chromosomes determine the individual features of the child
  - A child who inherits better features of the parents is more likely to survive, become a parent and pass on some these features to the next generation
  - There is a random low-level mutation rate in the DNA of the chromosomes
    - This changes the features of a chromosome that a child inherits from a parent
    - Some mutations provide desirable improvements

Genetic Algorithms (GA)

- Applying these concepts to optimization problems:
  - Rather than processing a single trial solution at a time, we now work with an entire population of trial solutions
  - GA tries to generate improving populations of trial solutions as it proceeds
  - Mutations occasionally occur so that certain children also acquire features that are not possessed by either parent
  - This helps GA explore new and perhaps better feasible regions
- The biological analogy defines the core of GA – it should be rigidly adhered to

Outline of Basic GA

- **Development of an appropriate encoding scheme is a key part of developing an effective genetic algorithm for any application**
- **Initialization**
  - Start with an initial population of feasible trial solutions, perhaps by generating them randomly
  - Evaluate the fitness for each member of this current population
  - This “fitness” is analogous to the objective function value

Outline of Basic GA

- **Iteration**
  - Use a random process that is biased toward the more fit members of the current population to be selected as parents
  - Pair up the parents randomly and then have each pair generate two children
  - If the child solution is infeasible, repeat the process
  - Retain the children and enough of the best members of the current population to form the new population of the same size for the next iteration – discard other members
  - Evaluate the fitness of each new member in the new population

Outline of Basic GA

- **Stopping rule**
  - Fixed number of iterations
  - Fixed amount of CPU time
  - Fixed number of consecutive iterations without any improvement in the best trial solution found so far
- **Use the best trial solution found on any iteration as the final solution**

Some specifics of using GA

- **Population size**
  - Ten – this is a reasonable size for small problems
- **Selection of parents**
  - From among the five most fit members of the population, select four randomly
  - From among the five least fit members, select two randomly
  - Use these six members as parents, and pair them up randomly to form three couples

More specifics of using GA

- Passage of features from parents to children
  - This is highly problem dependent
- Mutation rate
  - 0.1 – this is the probability that an inherited feature of a child mutates into an opposite feature
  - Smaller mutation rates are commonly used for larger problems
- Stopping rule
  - Stop after five consecutive iterations without any improvement in the best trial solution found so far

GA - Example

- **Problem**
  - Maximize
    $f(x) = 12x^5 - 975x^4 + 28{,}000x^3 - 345{,}000x^2 + 1{,}800{,}000x$
  - Subject to $0 \leq x \leq 31$,
  - Let's say that x must be an integer - *this is a new constraint*
- **So, what are our potential solutions?**
  - Any of the integers between 0 and 31
- **What have we done here?**
  - We've constrained the problem, reducing our search space to a critical region
  - This reduces search time and costs, without sacrificing a lot of accuracy

GA - Example

- **Problem**
  - **Maximize**
    $f(x) = 12x^5 - 975x^4 + 28{,}000x^3 - 345{,}000x^2 + 1{,}800{,}000x$
- **When applying GA, strings of binary digits are often used to represent solutions, which turns out to be convenient**
- **In this problem, only five binary digits are required to write any feasible value (solution set is $0 \leq x \leq 31$)**
  - **Obtained by converting the decimal number to a binary number**
  - **For instance, x = 3 is 00011 in base 2**

    x = 10 is 01010 in base 2

    x = 25 is 11001 in base 2

GA - Example

- Each binary digit is referred to as one of the **genes** of the solution
- When both parents have the same feature, it will passed down to each child
- When the two parents carry opposite features on the same gene, the choice is made at random
  - **If the two parents P1 and P2 are 00011 and 01010**
  - **The two children will be C1: 0x01x and C2: 0x01x, where x can be 0 or 1, and is to be identified randomly**
  - **These are easier to see if you stack the parents**

00011
01010

0x01x

  - **If the random number is less than 0.5, x = 0, otherwise x=1**

GA - Example

- **Original Children**

  0x01x

  Say our rule is that if the random number generated is
  0.0000 – 0.49999 chromosome = 0
  0.5000 – 0.99999 chromosome = 1

- **Then say the next 4 random numbers generated are:**
  0.7265, 0.5190, 0.0402, 0.3639
  The children become:

  C1: 01011 **C2: 00010**

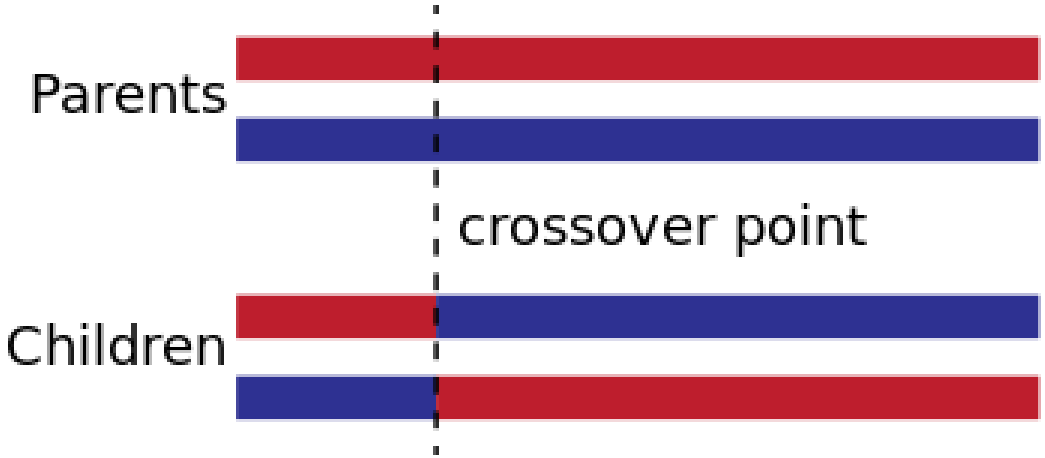
- **This particular method of generating children is called uniform crossover**

Single-point crossover

A single crossover point on both parents' organism strings is selected

All data beyond that point in either organism string is swapped between the two parent organisms

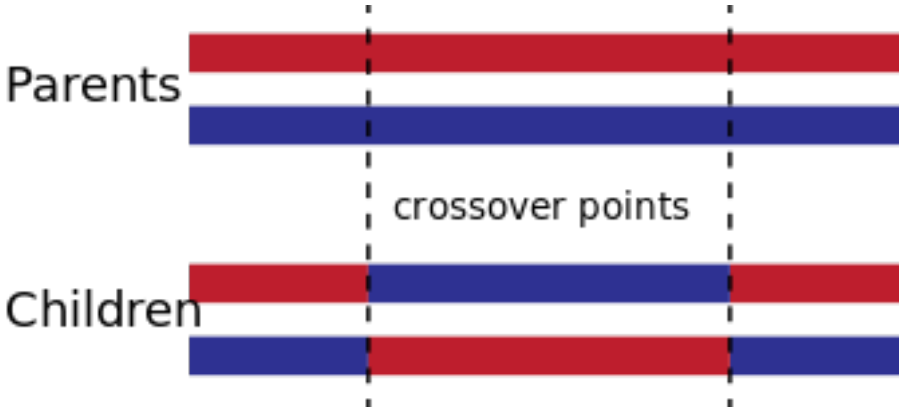
The resulting children are shown below



Two-point crossover

Two-point crossover calls for two points to be selected on the parent organism strings

Everything between the two points is swapped between the parent organisms, with the resulting two child organisms show below



Other crossover strategies

- partially matched crossover (PMX)
- cycle crossover (CX)
- order crossover operator (OX1)
- order-based crossover operator (OX2)
- position-based crossover operator (POS)
- voting recombination crossover operator (VR)
- alternating-position crossover operator (AP)
- sequential constructive crossover operator (SCX)[

Particularly useful in developing a solution to the traveling salesman problem

GA - Example

- **Mutation**
  - Let's set the probability of mutation to 0.1, and therefore, the random numbers
    - 0.0000-0.0999 correspond to a mutation
    - 0.1000-0.9999 correspond to no mutation
  - For the children C1: 01011 and C2: 00010, if the eighth random number generated corresponds to a mutation, C2 will become 00110 while C1 would be unchanged

Before mutating step: C1: 01011 C2: 00010

After mutating step: C1: 01011 C2: 00110

8th random # generated affects 8th position

GA - Example

- **A sample initial population**

Member	Initial population	Value of x	Fitness
1	01111	15	**3,628,125**
2	00100	4	3,234,688
3	01000	8	3.,055,616
4	10111	23	**3,962,091**
5	01010	10	2,950,000
6	01001	9	2,978,613
7	00101	5	**3,303,125**
8	10010	18	**4,239,216**
9	11110	30	1,350,000
10	10101	21	**4,353,187**

5 most fit members

GA - Example

- From the five most fit members, 4 are selected randomly (10,8,4,1)
- From the five least fit members, 2 are selected randomly (2,6)
- These six parents are randomly paired into three couples (10-2, 8-4, 1-6)

Member	Initial population	Value of x	Fitness
1	01111	15	3,628,125
2	00100	4	3,234,688
3	01000	8	3.,055,616
4	10111	23	3,962,091
5	01010	10	2,950,000
6	01001	9	2,978,613
7	00101	5	3,303,125
8	10010	18	4,239,216
9	11110	30	1,350,000
10	10101	21	4,353,187

GA - Example

- From these 6 parents, 6 children are generated

Member	Parents	Children	Value of x	Fitness
10 2	10101 00100	00101 10001	5 17	3,303,125 4,064,259
8 4	10010 10111	10011 10100	19 20	4,357,164 4,400,000
1 6	01111 01001	01011 01111	11 15	2,980,637 3,628,125

Mbr.	Initial Pop.	Value of x	Fitness
1	01111	15	**3,628,125**
4	10111	23	**3,962,091**
8	10010	18	**4,239,216**
10	10101	21	**4,353,187**

- These 6 children along with 4 best fit members of the preceding population form the new population
- Iterations are carried out as scheduled
- The best solution from any iteration is selected

Member	Initial Population	Value of x	Fitness
1	01111	15	**3,628,125**
2	00100	4	3,234,688
3	01000	8	3.,055,616
4	10111	23	**3,962,091**
5	01010	10	2,950,000
6	01001	9	2,978,613
7	00101	5	**3,303,125**
8	10010	18	**4,239,216**
9	11110	30	1,350,000
10	10101	21	**4,353,187**

Mbr	Parents	Children	Value of x	Fitness
10 2	10101 00100	00101 10001	5 17	3,303,125 4,064,259
8 4	10010 10111	10011 10100	19 20	4,357,164 4,400,000
1 6	01111 01001	01011 01111	11 15	2,980,637 3,628,125

Mbr.	Initial Pop.	Value of x	Fitness
1	01111	15	**3,628,125**
4	10111	23	**3,962,091**
8	10010	18	**4,239,216**
10	10101	21	**4,353,187**

GA - Question

- What happens if the original constraint of $0 \leq x \leq 31$ is modified to $0 \leq x \leq 25$?

GA - Question

- We still need 00000 places to represent a solution, but it's possible to now generate a solution that is infeasible
- What happens if we get an infeasible solution?
- An infeasible solution is termed a miscarriage – it is discarded and the entire process of recreating a child is repeated until a feasible solution is generated