Local search

Celso C. Ribeiro (celso@ic.uff.br)

University of Vienna

Metaheuristics – 2017-11-08

Overview of talk

- Local search
  - Solution representation
    - Steiner tree problem in graphs
    - Traveling salesman problem
    - Alternative representations
  - Neighborhoods
  - Search space graph
  - TSP: Neighborhood and search space graph
  - Local optimality
- Local search (cont'd)
  - Local search as traversal of search space graph
  - Neighborhood search
    - First improving
    - Best improving
    - Examples of neighborhood search
  - Cost function update
    - TSP cost function update
- Concluding remarks

Local search

Local search methods start from any feasible solution and visit other solutions, until a feasible solution that cannot be further improved is found.

Local improvements are evaluated with respect to neighboring solutions that can be obtained by slight modifications applied to a solution being visited.

Local search: *Solution representation*

A solution S of a combinatorial optimization problem is defined by a subset of the elements of the ground set E, i.e. $S \subseteq E$.

Local search: *Solution representation*

A solution S of a combinatorial optimization problem is defined by a subset of the elements of the ground set E, i.e. $S \subseteq E$.

- A feasible solution is one that satisfies all constraints of the problem.
- The objective function value of any (feasible or infeasible) solution S is given by

$$f(S) = \sum_{i \in S} c_i,$$

where c_i denotes the contribution to the objective function value of the ground set element $i \in E$.

Solution representation: *Steiner tree problem in graphs*

Let $G = (V, U)$ be a graph, where the node set is $V = \{1, \ldots, n\}$ and the edge set is U.

We are also given a subset $T \subseteq V$ of terminal nodes that have to be connected.

Solution representation: *Steiner tree problem in graphs*

Let $G = (V, U)$ be a graph, where the node set is $V = \{1, \ldots, n\}$ and the edge set is U.

We are also given a subset $T \subseteq V$ of terminal nodes that have to be connected.

A Steiner tree $S = (V', U')$ of G is a subtree of G that connects all nodes in T. Note that $T \subseteq V' \subseteq V$.

Solution representation: *Steiner tree problem in graphs*

Let $G = (V, U)$ be a graph, where the node set is $V = \{1, \ldots, n\}$ and the edge set is U.

We are also given a subset $T \subseteq V$ of terminal nodes that have to be connected.

A Steiner tree $S = (V', U')$ of G is a subtree of G that connects all nodes in T. Note that $T \subseteq V' \subseteq V$.

- Given any subset V' of nodes such that $T \subseteq V' \subseteq V$, note that any spanning tree of the graph induced in G by V' is also a Steiner tree of G connecting all terminal nodes in T.

Solution representation: *Steiner tree problem in graphs*

Let $G = (V, U)$ be a graph, where the node set is $V = \{1, \ldots, n\}$ and the edge set is U.

We are also given a subset $T \subseteq V$ of terminal nodes that have to be connected.

A Steiner tree $S = (V', U')$ of G is a subtree of G that connects all nodes in T. Note that $T \subseteq V' \subseteq V$.

- Given any subset V' of nodes such that $T \subseteq V' \subseteq V$, note that any spanning tree of the graph induced in G by V' is also a Steiner tree of G connecting all terminal nodes in T.
- Therefore, any Steiner tree of G connecting the terminal nodes can be constructed by
  - selecting a subset of optional nodes $W \subseteq V \setminus T$
  - computing a minimum spanning tree of the graph $G(W \cup T)$ induced in G by $V' = W \cup T$.

Solution representation: *Steiner tree problem in graphs*

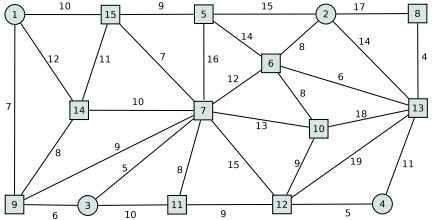
Let $G = (V, U)$ be a graph, where the node set is $V = \{1, \ldots, n\}$ and the edge set is U.

We are also given a subset $T \subseteq V$ of terminal nodes that have to be connected.

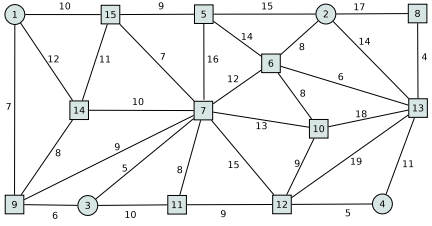
A Steiner tree $S = (V', U')$ of G is a subtree of G that connects all nodes in T. Note that $T \subseteq V' \subseteq V$.

- Given any subset V' of nodes such that $T \subseteq V' \subseteq V$, note that any spanning tree of the graph induced in G by V' is also a Steiner tree of G connecting all terminal nodes in T.
- Therefore, any Steiner tree of G connecting the terminal nodes can be constructed by
  - selecting a subset of optional nodes $W \subseteq V \setminus T$
  - computing a minimum spanning tree of the graph $G(W \cup T)$ induced in G by $V' = W \cup T$.
- As a consequence, every solution S of the Steiner tree problem can be represented by a binary vector $(x_1, \ldots, x_n)$, in which $x_i = 1$ if node $i \in V' = W \cup T$; $x_j = 0$ otherwise, for every $i = 1, \ldots, n$.

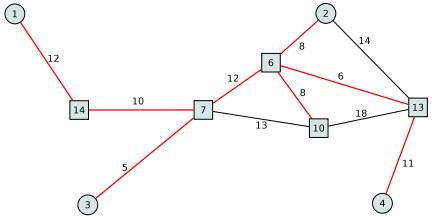
Solution representation: *Steiner tree problem in graphs*



Solution representation: *Steiner tree problem in graphs*

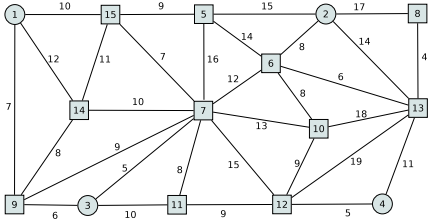


$W = \{6, 7, 10, 13, 14\}$ with weight 72.

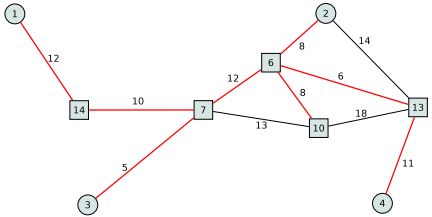


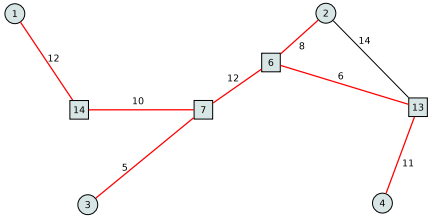
Minimum weight spanning tree induced by W is red.

Solution representation: *Steiner tree problem in graphs*



$W = \{6, 7, 10, 13, 14\}$ with weight 72.

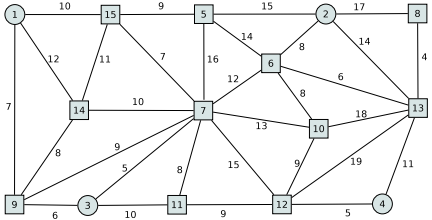




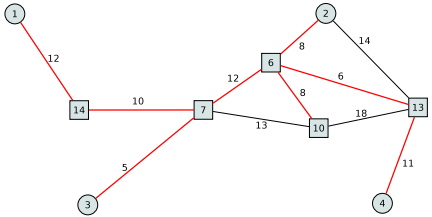
$W = \{6, 7, 13, 14\}$ with weight 64.

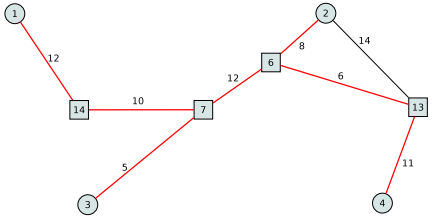
Minimum weight spanning tree induced by W is red.

Solution representation: *Steiner tree problem in graphs*

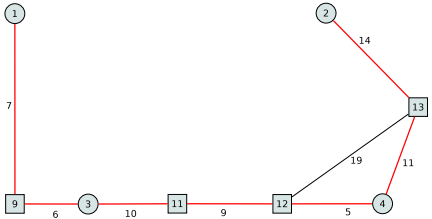


$W = \{6, 7, 10, 13, 14\}$ with weight 72.





$W = \{6, 7, 13, 14\}$ with weight 64.



$W = \{9, 11, 12, 13\}$ with weight 62.

Minimum weight spanning tree induced by W is red.

Solution representation: *Traveling salesman problem*

Let $V = \{1, \ldots, n\}$ be the set of cities a traveling salesman has to visit, with non-negative lengths d_{ij} associated with each pair of cities $i, j \in V$.

- Any tour visiting each of the n cities exactly once corresponds to a feasible solution.

Solution representation: *Traveling salesman problem*

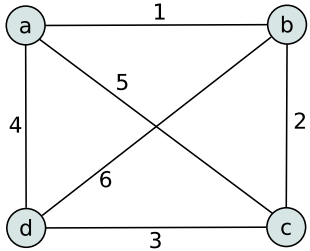
Let $V = \{1, \ldots, n\}$ be the set of cities a traveling salesman has to visit, with non-negative lengths d_{ij} associated with each pair of cities $i, j \in V$.

- Any tour visiting each of the n cities exactly once corresponds to a feasible solution.
- Every feasible solution S can be represented by a binary vector $(x_1, \ldots, x_m)$, where $m = n(n-1)/2$ and $x_k = 1$ if the edge indexed by k belongs to the corresponding tour, $x_k = 0$ otherwise, for every $k = 1, \ldots, m$. However, this representation applies to any edge subset, regardless if it corresponds to a tour or not.
- Therefore, the edge subset $\{k = 1, \ldots, m : x_k = 1\}$ must define a tour for this solution to be feasible.

Solution representation: *Traveling salesman problem*

- The figure illustrates a complete graph with four nodes.
- Numbers on the six edges represent their indices.
- Every solution can be represented by a binary vector

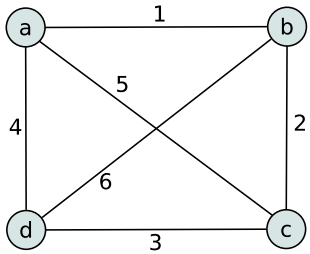
$$(x_1, x_2, x_3, x_4, x_5, x_6).$$



Solution representation: *Traveling salesman problem*

- The figure illustrates a complete graph with four nodes.
- Numbers on the six edges represent their indices.
- Every solution can be represented by a binary vector

$$(x_1, x_2, x_3, x_4, x_5, x_6).$$

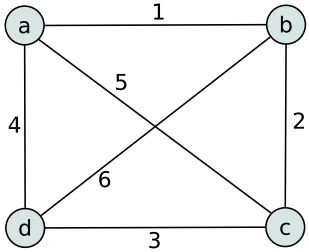


There are three different tours, corresponding to the incidence vectors:

Solution representation: *Traveling salesman problem*

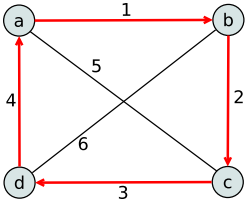
- The figure illustrates a complete graph with four nodes.
- Numbers on the six edges represent their indices.
- Every solution can be represented by a binary vector

$$(x_1, x_2, x_3, x_4, x_5, x_6).$$

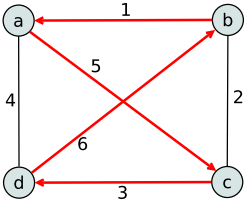


There are three different tours, corresponding to the incidence vectors:

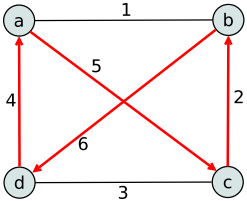
$(1, 1, 1, 1, 0, 0)$



$(1, 0, 1, 0, 1, 1)$



$(0, 1, 0, 1, 1, 1)$



Solution representation: *Traveling salesman problem*

- Any solution to the traveling salesman problem can alternatively be represented by a circular permutation $(\pi_1, \ldots, \pi_n)$ of the n cities, with $\pi_i \in V$ for every $i = 1, \ldots, n$ and $\pi_i \neq \pi_j$ for every $i, j = 1, \ldots, n : i \neq j$.
- This permutation is associated with the tour defined by the edges (π_1, π_2), (π_2, π_3), $\ldots$, (π_{n-1}, π_n), and (π_n, π_1).

Solution representation: *Traveling salesman problem*

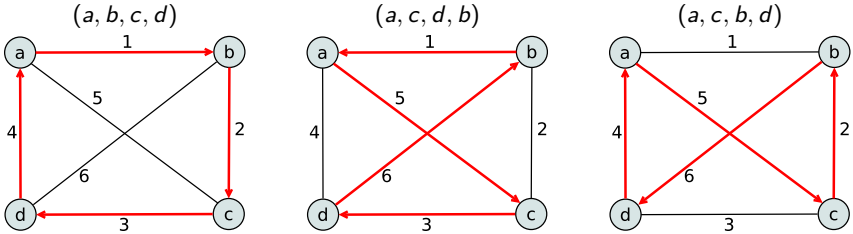
- Any solution to the traveling salesman problem can alternatively be represented by a circular permutation $(\pi_1, \ldots, \pi_n)$ of the n cities, with $\pi_i \in V$ for every $i = 1, \ldots, n$ and $\pi_i \neq \pi_j$ for every $i, j = 1, \ldots, n : i \neq j$.
- This permutation is associated with the tour defined by the edges (π_1, π_2), (π_2, π_3), $\ldots$, (π_{n-1}, π_n), and (π_n, π_1).

The three tours represented by the incidence vectors $(1, 1, 1, 1, 0, 0)$, $(1, 0, 1, 0, 1, 1)$, and $(0, 1, 0, 1, 1, 1)$ correspond, respectively, to the circular permutations

Solution representation: *Traveling salesman problem*

- Any solution to the traveling salesman problem can alternatively be represented by a circular permutation $(\pi_1, \ldots, \pi_n)$ of the n cities, with $\pi_i \in V$ for every $i = 1, \ldots, n$ and $\pi_i \neq \pi_j$ for every $i, j = 1, \ldots, n : i \neq j$.
- This permutation is associated with the tour defined by the edges (π_1, π_2), (π_2, π_3), $\ldots$, (π_{n-1}, π_n), and (π_n, π_1).

The three tours represented by the incidence vectors $(1, 1, 1, 1, 0, 0)$, $(1, 0, 1, 0, 1, 1)$, and $(0, 1, 0, 1, 1, 1)$ correspond, respectively, to the circular permutations



Solution representation: *Alternative representations*

The choice of one representation over another can lead to simpler implementations or faster algorithms.

It can also be helpful to work simultaneously with two different representations, since one can be more effective than the other for the implementation of some specific operation and vice-versa for some other operation.

Solution representation: *Alternative representations*

The choice of one representation over another can lead to simpler implementations or faster algorithms.

It can also be helpful to work simultaneously with two different representations, since one can be more effective than the other for the implementation of some specific operation and vice-versa for some other operation.

Some of the most frequently used solution representations:

- *0-1 incidence vector*: Typically used when the ground set is partitioned into two subsets, one corresponding to the elements that belong to the solution, while the others do not.
- *Generalized incidence vector*: Often used when the ground set has to be partitioned into a number of subsets, each of them with a different interpretation. Examples: graph coloring problem, vehicle routing, and scheduling problems.
- *Permutation*: Typically applies to scheduling, assignment, and routing problems in which one is interested in establishing an optimal order.

Neighborhoods

- A *neighborhood* of a solution $S \in F$ can be defined by any subset of F.
- More formally, a neighborhood is a mapping that associates each feasible solution $S \in F$ with a subset $N(S) = \{S_1, \ldots, S_p\}$ of feasible solutions also in F.

Neighborhoods

- A *neighborhood* of a solution $S \in F$ can be defined by any subset of F.
- More formally, a neighborhood is a mapping that associates each feasible solution $S \in F$ with a subset $N(S) = \{S_1, \ldots, S_p\}$ of feasible solutions also in F.
- Each solution $S' \in N(S)$ can be reached from S by an operator called *move*.
- Normally, two neighboring solutions S and $S' \in N(S)$ differ only by a few elements and a move from a solution S consists simply in changing one or more elements in S.

Neighborhoods

- A *neighborhood* of a solution $S \in F$ can be defined by any subset of F.
- More formally, a neighborhood is a mapping that associates each feasible solution $S \in F$ with a subset $N(S) = \{S_1, \ldots, S_p\}$ of feasible solutions also in F.
- Each solution $S' \in N(S)$ can be reached from S by an operator called *move*.
- Normally, two neighboring solutions S and $S' \in N(S)$ differ only by a few elements and a move from a solution S consists simply in changing one or more elements in S.
- Usually, $S \in N(S')$ whenever $S' \in N(S)$.

Search space graph

Define the *search space graph* $\mathcal{G} = (F, M)$ to be such that:

- Its node set corresponds to the set F of feasible solutions.
- Its edge set M is such that there is an edge $(S, S') \in M$ between two solutions $S, S' \in F$ if and only if $S' \in N(S)$ and $S \in N(S')$.

Search space graph

Define the *search space graph* $\mathcal{G} = (F, M)$ to be such that:

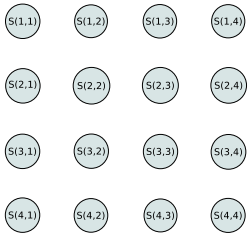
- Its node set corresponds to the set F of feasible solutions.
- Its edge set M is such that there is an edge $(S, S') \in M$ between two solutions $S, S' \in F$ if and only if $S' \in N(S)$ and $S \in N(S')$.
- An extended search space graph may be similarly defined, encompassing not only the set of feasible solutions F but, instead, the whole set $\hat{F} = 2^E$ formed by all subsets of elements of the ground set E.

Search space graph

Define the *search space graph* $\mathcal{G} = (F, M)$ to be such that:

- Its node set corresponds to the set F of feasible solutions.
- Its edge set M is such that there is an edge $(S, S') \in M$ between two solutions $S, S' \in F$ if and only if $S' \in N(S)$ and $S \in N(S')$.
- An extended search space graph may be similarly defined, encompassing not only the set of feasible solutions F but, instead, the whole set $\hat{F} = 2^E$ formed by all subsets of elements of the ground set E.

The figure shows an instance of a combinatorial problem in which the set F is formed by 16 feasible solutions depicted in a square grid and represented by $S(i, j)$, for $i, j = 1, \ldots, 4$.



Search space graph

A search space graph associated with the above problem instance can be created by imposing a neighborhood definition on the node set F.

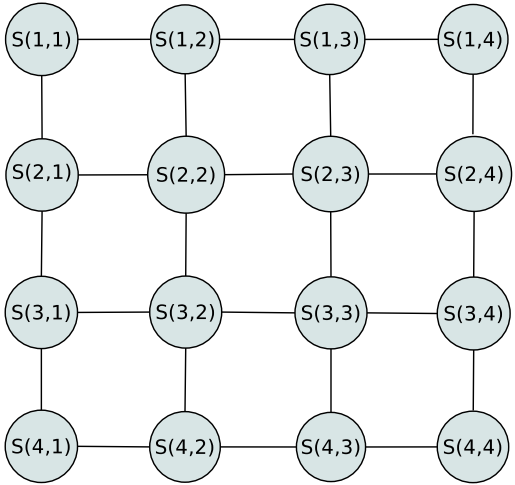
Search space graph

A search space graph associated with the above problem instance can be created by imposing a neighborhood definition on the node set F.

Neighborhood N_1 is defined such that any solution $S(i,j)$ has neighbors

- $S(i+1,j)$
- $S(i-1,j)$
- $S(i,j+1)$
- $S(i,j-1)$

whenever they exist.



Neighborhood N_1

Search space graph

Other different neighborhoods can be defined and imposed on the same set of feasible solutions.

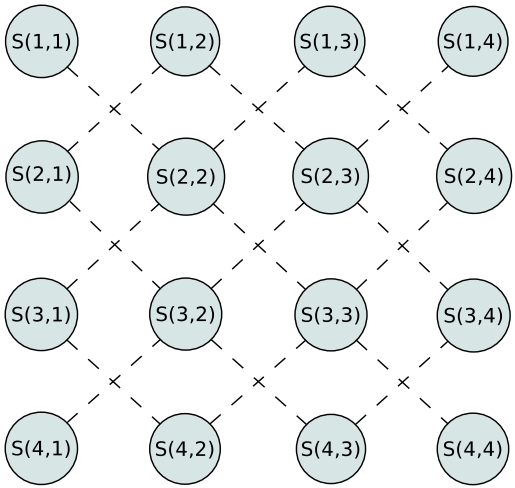
Search space graph

Other different neighborhoods can be defined and imposed on the same set of feasible solutions.

Another neighborhood N_2 can be defined, such that any solution $S(i, j)$ has neighbors

- $S(i+1, j+1)$
- $S(i+1, j-1)$
- $S(i-1, j+1)$
- $S(i-1, j-1)$

whenever they exist.

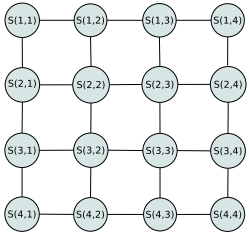


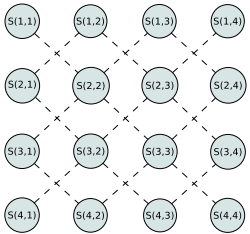
Neighborhood N_2

Search space graph

- Some pairs of solutions are closer within N_1 or N_2.

Neighborhood N_1



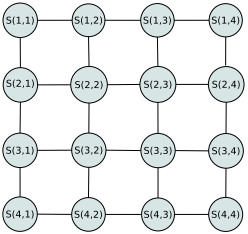


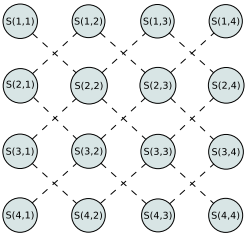
Neighborhood N_2

Search space graph

- Some pairs of solutions are closer within N_1 or N_2.
  - Six moves are necessary to traverse the N_1 search space graph from $S(1,1)$ to $S(4,4)$.
  - Only three moves are necessary if neighborhood N_2 is used.

Neighborhood N_1



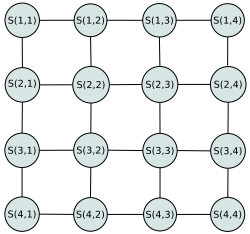


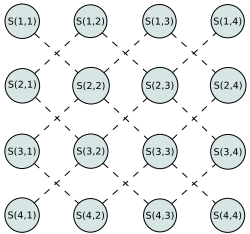
Neighborhood N_2

Search space graph

- Some pairs of solutions are closer within N_1 or N_2.
  - Six moves are necessary to traverse the N_1 search space graph from $S(1,1)$ to $S(4,4)$.
  - Only three moves are necessary if neighborhood N_2 is used.
- While any feasible solution can be reached when neighborhood N_1 is used, N_2 is not connected.

Neighborhood N_1



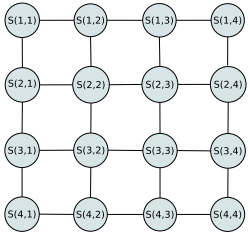


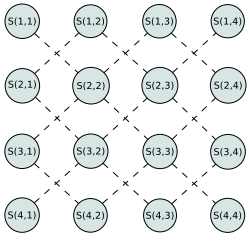
Neighborhood N_2

Search space graph

- Some pairs of solutions are closer within N_1 or N_2.
  - Six moves are necessary to traverse the N_1 search space graph from $S(1,1)$ to $S(4,4)$.
  - Only three moves are necessary if neighborhood N_2 is used.
- While any feasible solution can be reached when neighborhood N_1 is used, N_2 is not connected.
  - Only half of the solutions in N_2 are reachable from any given solution.
  - This can lead to implementation difficulties or even make it impossible to find good solutions.

Neighborhood N_1





Neighborhood N_2

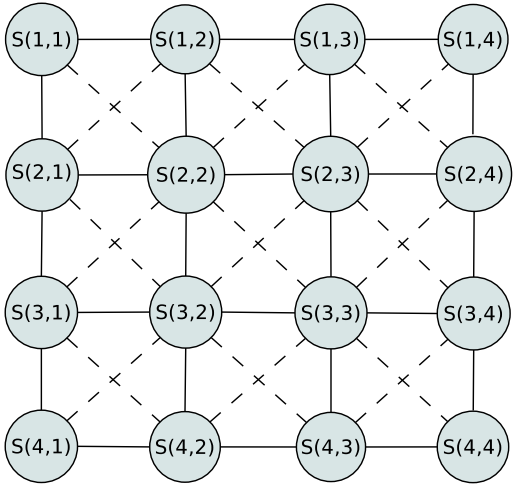
Search space graph

A natural idea is to combine neighborhoods N_1 and N_2 into a single neighborhood.

Search space graph

A natural idea is to combine neighborhoods N_1 and N_2 into a single neighborhood.

Neighborhood N_3 can be defined as the union of N_1 and N_2.



Neighborhood N_3

Search space graph

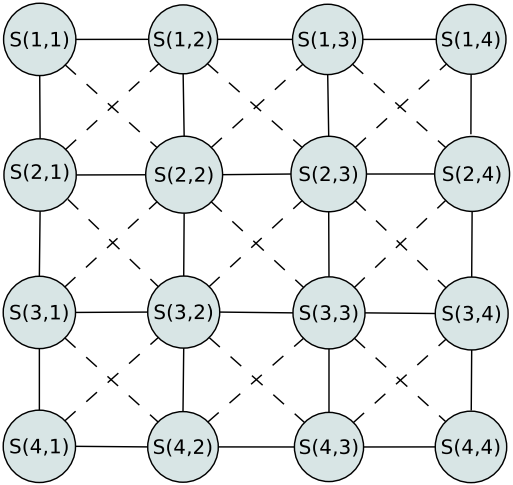
A natural idea is to combine neighborhoods N_1 and N_2 into a single neighborhood.

Neighborhood N_3 can be defined as the union of N_1 and N_2.

Within this new neighborhood, any feasible solution $S(i,j)$ has up to eight neighbors:

- $S(i+1,j)$
- $S(i-1,j)$
- $S(i,j+1)$
- $S(i,j-1)$
- $S(i+1,j+1)$
- $S(i+1,j-1)$
- $S(i-1,j+1)$
- $S(i-1,j-1)$

whenever they exist.

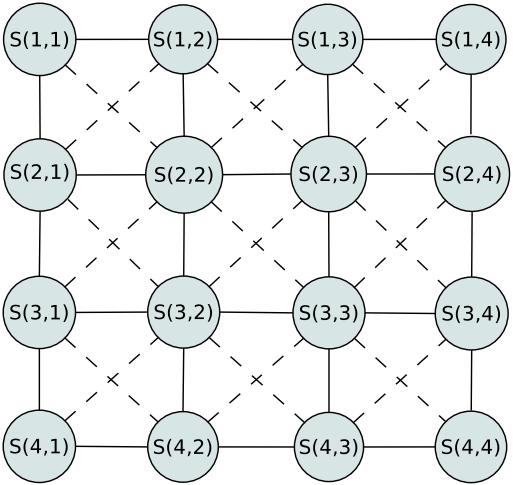


Neighborhood N_3

Search space graph

Different neighborhoods can be defined and used in the implementation of a local search method.

- The larger the neighborhood, the denser will be the search space graph and the shorter will be the paths connecting any two solutions.
- Large neighborhoods require the evaluation of more neighboring solutions $\Longrightarrow$ larger computation times during the investigation of the current solution.



Neighborhood N_3

Search space graph: infeasible solutions

- Search space need not to be formed exclusively by feasible solutions in F.
- It can contain any subset of the set $\hat{F} = 2^E$ formed by all solutions, either feasible or infeasible.
- In this situation, the search can visit feasible and infeasible solutions, but must terminate at a feasible solution.
- Working with more complex search space graphs, which include infeasible solutions, can be essential in some cases to ensure connectivity between any pair of feasible solutions.
- Finding an appropriate neighborhood and the best way to explore it is a crucial step towards the implementation of effective and efficient local search methods.

Knapsack problem: *Neighborhood and search space graph*

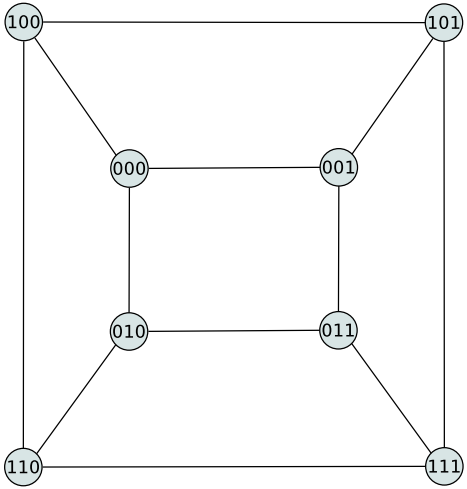
- Knapsack problem is defined over a set $I = \{1, \ldots, n\}$ of items to be packed.
- Every solution can be represented by a 0-1 binary vector $(x_1, \ldots, x_n)$, in which $x_i = 1$ if item i is packed, $x_i = 0$ otherwise, for every $i = 1, \ldots, n$.

Knapsack problem: *Neighborhood and search space graph*

- Knapsack problem is defined over a set $I = \{1, \ldots, n\}$ of items to be packed.
- Every solution can be represented by a 0-1 binary vector $(x_1, \ldots, x_n)$, in which $x_i = 1$ if item i is packed, $x_i = 0$ otherwise, for every $i = 1, \ldots, n$.
- A move from any solution amounts to complementing the value of some variable among $x_1, \ldots, x_n$, while keeping the others fixed: if an item is in the knapsack, then remove it; otherwise, pack it.

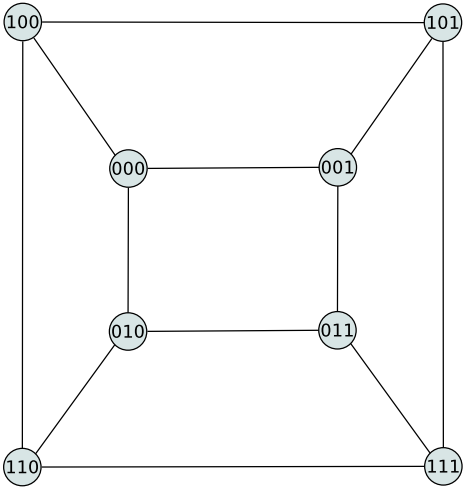
Knapsack problem: *Neighborhood and search space graph*

- Knapsack problem is defined over a set $I = \{1, \ldots, n\}$ of items to be packed.
- Every solution can be represented by a 0-1 binary vector $(x_1, \ldots, x_n)$, in which $x_i = 1$ if item i is packed, $x_i = 0$ otherwise, for every $i = 1, \ldots, n$.
- A move from any solution amounts to complementing the value of some variable among $x_1, \ldots, x_n$, while keeping the others fixed: if an item is in the knapsack, then remove it; otherwise, pack it.
- Problem with $n = 3$ items: each solution has exactly $n = 3$ neighbors.
- Neighbors of (1,0,1) are:



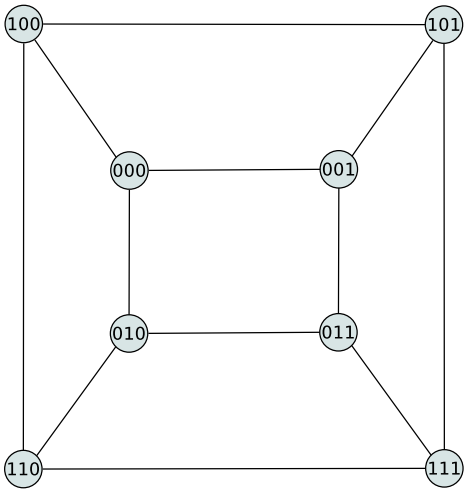
Knapsack problem: *Neighborhood and search space graph*

- Knapsack problem is defined over a set $I = \{1, \ldots, n\}$ of items to be packed.
- Every solution can be represented by a 0-1 binary vector $(x_1, \ldots, x_n)$, in which $x_i = 1$ if item i is packed, $x_i = 0$ otherwise, for every $i = 1, \ldots, n$.
- A move from any solution amounts to complementing the value of some variable among $x_1, \ldots, x_n$, while keeping the others fixed: if an item is in the knapsack, then remove it; otherwise, pack it.
- Problem with $n = 3$ items: each solution has exactly $n = 3$ neighbors.
- Neighbors of (1,0,1) are:
- (0,0,1)



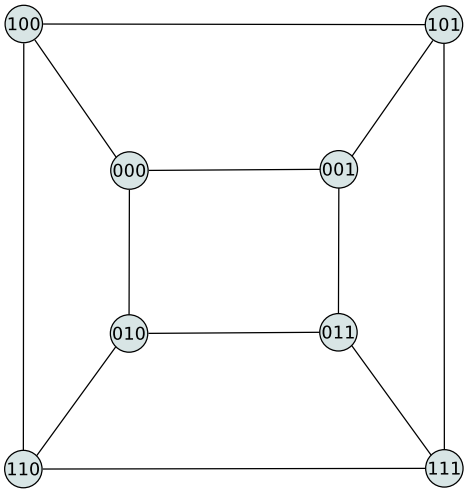
Knapsack problem: *Neighborhood and search space graph*

- Knapsack problem is defined over a set $I = \{1, \ldots, n\}$ of items to be packed.
- Every solution can be represented by a 0-1 binary vector $(x_1, \ldots, x_n)$, in which $x_i = 1$ if item i is packed, $x_i = 0$ otherwise, for every $i = 1, \ldots, n$.
- A move from any solution amounts to complementing the value of some variable among $x_1, \ldots, x_n$, while keeping the others fixed: if an item is in the knapsack, then remove it; otherwise, pack it.
- Problem with $n = 3$ items: each solution has exactly $n = 3$ neighbors.
- Neighbors of (1,0,1) are:
- (0,0,1)
- (1,1,1)



Knapsack problem: *Neighborhood and search space graph*

- Knapsack problem is defined over a set $I = \{1, \ldots, n\}$ of items to be packed.
- Every solution can be represented by a 0-1 binary vector $(x_1, \ldots, x_n)$, in which $x_i = 1$ if item i is packed, $x_i = 0$ otherwise, for every $i = 1, \ldots, n$.
- A move from any solution amounts to complementing the value of some variable among $x_1, \ldots, x_n$, while keeping the others fixed: if an item is in the knapsack, then remove it; otherwise, pack it.
- Problem with $n = 3$ items: each solution has exactly $n = 3$ neighbors.
- Neighbors of (1,0,1) are:
- (0,0,1)
- (1,1,1)
- (1,0,0)



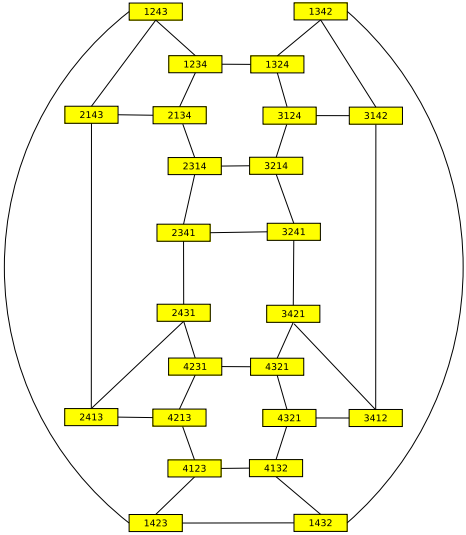
Traveling salesman problem: *Neighborhood and search space graph*

- Traveling salesman problem is defined over a set $V = \{1, \ldots, n\}$ of cities that have to be visited exactly once.
- A feasible solution can be represented by a circular permutation $(\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ of the n cities, with $\pi_i \in V$ for every $i = 1, \ldots, n$ and $\pi_i \neq \pi_j$ for every $i, j = 1, \ldots, n : i \neq j$.
- This circular permutation is equivalent to any of the n linear permutations $(\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$, $(\pi_2, \pi_3, \ldots, \pi_n, \pi_1)$, $\ldots$, and $(\pi_n, \pi_1, \ldots, \pi_{n-2}, \pi_{n-1})$, each originating at a different city.
- All of them correspond to the same tour (π_1, π_2), (π_2, π_3), $\ldots$, (π_{n-1}, π_n), (π_n, π_1).
- The search space graph has exactly $n!$ nodes, each of them corresponding to a permutation of the n cities to be visited.

Traveling salesman problem: *Neighborhood and search space graph*

Neighborhood N_1:

- Defined by all permutations that can be obtained by exchanging the positions of two consecutive cities of the current permutation.
- Any solution $(\pi_1, \ldots, \pi_{i-1}, \pi_i, \ldots, \pi_n)$ has exactly $n-1$ neighbors, each defined by a different permutation $(\pi_1, \ldots, \pi_i, \pi_{i-1}, \ldots, \pi_n)$ characterized by the swap of cities π_{i-1} and π_i, for $i = 2, \ldots, n$.
- The figure illustrates the search space graph corresponding to this neighborhood for a symmetric traveling salesman problem with four cities. Every solution has exactly three neighbors.



Traveling salesman problem: *Neighborhood and search space graph*

Neighborhood N_2:

- Neighborhood N_2 is defined by associating a solution $(\pi_1, \ldots, \pi_i, \ldots, \pi_j, \ldots, \pi_n)$ with all $n(n-1)/2$ neighbors $(\pi_1, \ldots, \pi_j, \ldots, \pi_i, \ldots, \pi_n)$ that can be obtained by exchanging the positions of any two cities π_i and π_j, for $i, j = 1, \ldots, n : i \neq j$.
- Considering the same example of a symmetric traveling salesman problem with four cities, every solution has exactly six neighbors. In particular, the same solution $(1, 2, 3, 4)$ has now $(2, 1, 3, 4)$, $(1, 3, 2, 4)$, $(1, 2, 4, 3)$, $(3, 2, 1, 4)$, $(1, 4, 3, 2)$, and $(4, 2, 3, 1)$ as its neighbors.

Traveling salesman problem: *Neighborhood and search space graph*

Neighborhood N_2:

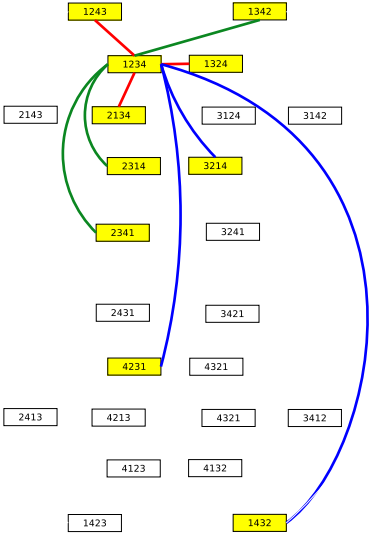
- Neighborhood N_2 is defined by associating a solution $(\pi_1, \ldots, \pi_i, \ldots, \pi_j, \ldots, \pi_n)$ with all $n(n-1)/2$ neighbors $(\pi_1, \ldots, \pi_j, \ldots, \pi_i, \ldots, \pi_n)$ that can be obtained by exchanging the positions of any two cities π_i and π_j, for $i, j = 1, \ldots, n : i \neq j$.
- Considering the same example of a symmetric traveling salesman problem with four cities, every solution has exactly six neighbors. In particular, the same solution $(1, 2, 3, 4)$ has now $(2, 1, 3, 4)$, $(1, 3, 2, 4)$, $(1, 2, 4, 3)$, $(3, 2, 1, 4)$, $(1, 4, 3, 2)$, and $(4, 2, 3, 1)$ as its neighbors.

Neighborhood N_3:

- Neighborhood N_3 is defined by associating a solution $(\pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_j, \ldots, \pi_n)$ with all $n(n-1)/2$ neighbors $(\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_i, \pi_j, \ldots, \pi_n)$ that can be obtained by moving city π_i to position j, with $1 \leq i < j \leq n$, and shifting by one position to the left all cities between positions $i + 1$ and j.
- Every solution has also exactly six neighbors. For example, solution $(1, 2, 3, 4)$ has $(2, 1, 3, 4)$, $(2, 3, 1, 4)$, $(2, 3, 4, 1)$, $(1, 3, 2, 4)$, $(1, 3, 4, 2)$, and $(1, 2, 4, 3)$ as its neighbors.

Traveling salesman problem: *Neighborhood and search space graph*

- The figure superimposes neighborhoods N_1, N_2, and N_3, illustrating the neighbors of solution $(1, 2, 3, 4)$ within each of the three neighborhoods.
- Nodes connected by red edges belong to the three neighborhoods. Nodes connected by blue edges are those within neighborhood N_2, while those connected by green edges belong to neighborhood N_3.



Local optimality

A solution S^+ is said to be a local optimum for a minimization problem with respect to neighborhood N if and only if

$$f(S^+) \leq f(S),\ \forall S \in N(S^+).$$

Local optimality

A solution S^+ is said to be a local optimum for a minimization problem with respect to neighborhood N if and only if

$$f(S^+) \leq f(S), \ \forall S \in N(S^+).$$

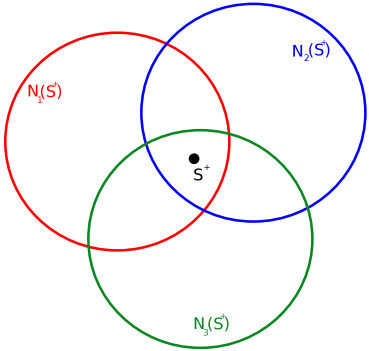
- A global optimum is also locally optimal with respect to any neighborhood, while a local optimum is not necessarily a global optimum.

Local optimality

A solution S^+ is said to be a local optimum for a minimization problem with respect to neighborhood N if and only if

$$f(S^+) \leq f(S), \ \forall S \in N(S^+).$$

- A global optimum is also locally optimal with respect to any neighborhood, while a local optimum is not necessarily a global optimum.

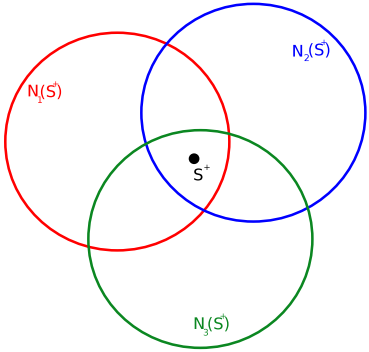


Local optimality

A solution S^+ is said to be a local optimum for a minimization problem with respect to neighborhood N if and only if

$$f(S^+) \leq f(S),\ \forall S \in N(S^+).$$

- A global optimum is also locally optimal with respect to any neighborhood, while a local optimum is not necessarily a global optimum.



Let

- $f(S^+) \leq f(S),\ \forall S \in N_1(S^+)$
- $f(S^+) \leq f(S),\ \forall S \in N_2(S^+)$
- $f(S^+) \leq f(S),\ \forall S \in N_3(S^+)$

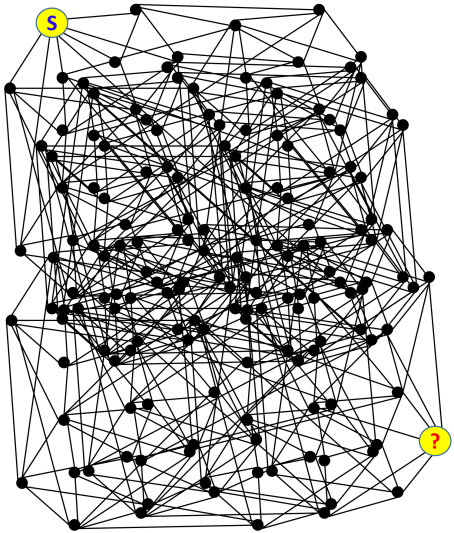
S^+ may not be a global optimum!

Local search as traversal of search space graph

- Local search methods can be viewed as a traversal of the search space graph starting from any given solution and stopping whenever some optimality condition is met.

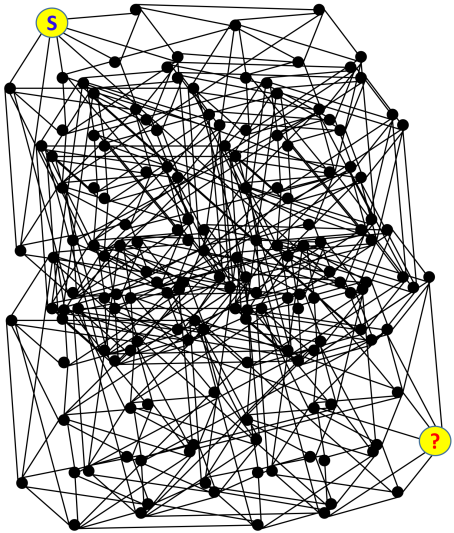
Local search as traversal of search space graph

- Local search methods can be viewed as a traversal of the search space graph starting from any given solution and stopping whenever some optimality condition is met.



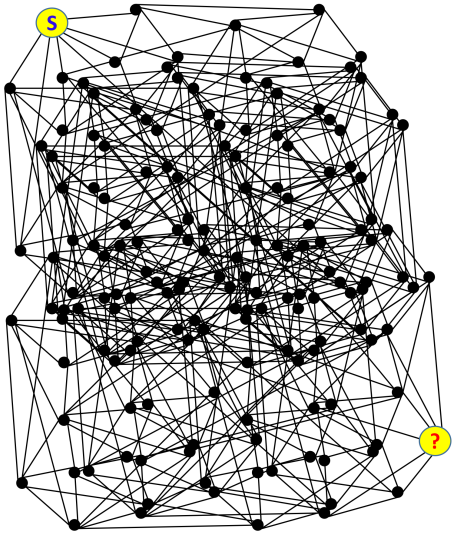
Local search as traversal of search space graph

- Local search methods can be viewed as a traversal of the search space graph starting from any given solution and stopping whenever some optimality condition is met.
- In most cases, a local search procedure is made to stop after a locally optimal solution is encountered.



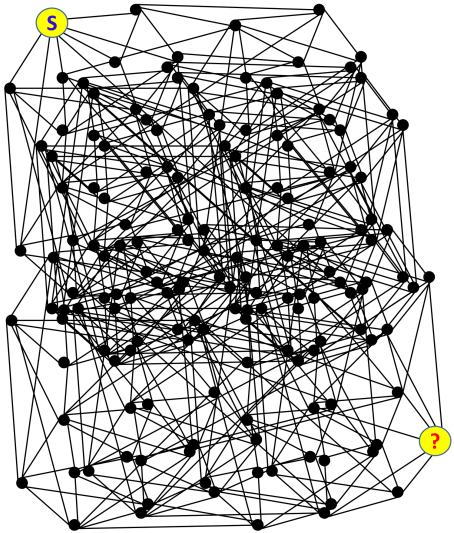
Local search as traversal of search space graph

- Local search methods can be viewed as a traversal of the search space graph starting from any given solution and stopping whenever some optimality condition is met.
- In most cases, a local search procedure is made to stop after a locally optimal solution is encountered.
- Metaheuristics such as tabu search, iterated local search, and GRASP extend the search beyond the first local optimum found, offering different escape mechanisms.



Local search as traversal of search space graph

- Local search methods can be viewed as a traversal of the search space graph starting from any given solution and stopping whenever some optimality condition is met.
- In most cases, a local search procedure is made to stop after a locally optimal solution is encountered.
- Metaheuristics such as tabu search, iterated local search, and GRASP extend the search beyond the first local optimum found, offering different escape mechanisms.
- The effectiveness and efficiency of a local search method depend on several factors, such as the starting solution, the neighborhood structure, and the objective function being optimized.



Local search as traversal of search space graph

- Local search methods can be viewed as a traversal of the search space graph starting from any given solution and stopping whenever some optimality condition is met.
- In most cases, a local search procedure is made to stop after a locally optimal solution is encountered.
- Metaheuristics such as tabu search, iterated local search, and GRASP extend the search beyond the first local optimum found, offering different escape mechanisms.
- The effectiveness and efficiency of a local search method depend on several factors, such as the starting solution, the neighborhood structure, and the objective function being optimized.

The main components or phases of a local search method are:

Local search as traversal of search space graph

- Local search methods can be viewed as a traversal of the search space graph starting from any given solution and stopping whenever some optimality condition is met.
- In most cases, a local search procedure is made to stop after a locally optimal solution is encountered.
- Metaheuristics such as tabu search, iterated local search, and GRASP extend the search beyond the first local optimum found, offering different escape mechanisms.
- The effectiveness and efficiency of a local search method depend on several factors, such as the starting solution, the neighborhood structure, and the objective function being optimized.

The main components or phases of a local search method are:

- **Start:** Construction of the initial solution, from where the search starts. Methods are needed to build an initial solution.
- **Neighborhood search:** Application of a subordinate heuristic or search strategy to find an improving solution in the neighborhood of the current solution.
- **Stop:** Interruption of the search by a stopping criterion, which in most cases consists in the verification that a locally optimal solution has been found. Stopping criteria for the neighborhood search are needed.

Neighborhood search: *First improving*

- At any iteration of an iterative improvement or first-improving neighborhood search strategy, the algorithm moves from the current solution to any neighbor with a better objective function value.
- In general, the new solution is the first improving solution identified along the neighborhood search.
- The pseudo-code describes a local search procedure based on a first-improving strategy for a minimization problem.

```
begin FIRST-IMPROVING(S);
1  improvement ← .TRUE.;
2  while improvement = .TRUE. do
3      improvement ← .FALSE.;
4      forall S' ∈ N(S) while improvement = .FALSE. do
5          if f(S') < f(S) then
6              S ← S';
7              improvement ← .TRUE.;
8          end-if;
9      end-forall;
10 end-while;
11 return S;
end FIRST-IMPROVING.
```

Pseudo-code of a first-improving local search procedure for a minimization problem.

Neighborhood search: *Best improving*

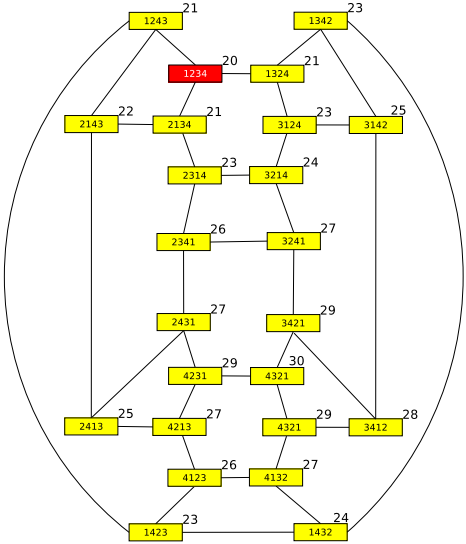
- At any iteration of a best-improving local search strategy, the algorithm moves from the current solution to the best of its neighbors, whenever this neighbor improves upon the current solution.
- The pseudo-code on the right describes a local search procedure based on a best-improving strategy for a minimization problem.

```
begin BEST-IMPROVING(S);
1   improvement ← .TRUE.;
2   while improvement = .TRUE. do
3       improvement ← .FALSE.;
4       f_best ← ∞;
5       forall S' ∈ N(S) do
6           if f(S') < f_best then
7               S_best ← S';
8               f_best ← f(S');
9           end-if;
10      end-forall;
11      if f_best < f(S) then
12          S ← S_best;
13          improvement ← .TRUE.;
14      end-if;
15  end-while;
16  return S;
end BEST-IMPROVING.
```

Pseudo-code of a best-improving local search procedure for a minimization problem.

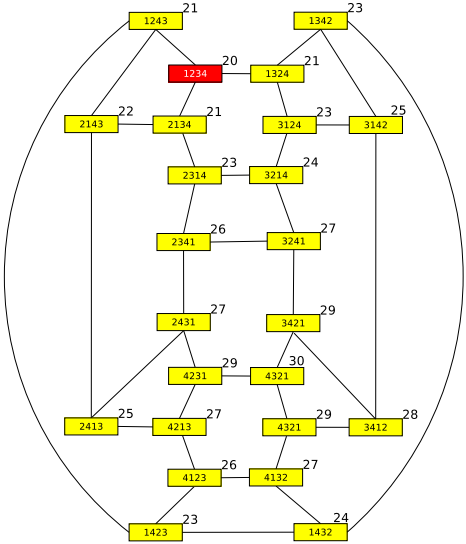
Optimization problem: *Single global optimum*

- Consider the permutation representaton with $n = 4$ whose search space graph is shown in the figure.
- The values of the objective function are shown next to each solution in the figure.



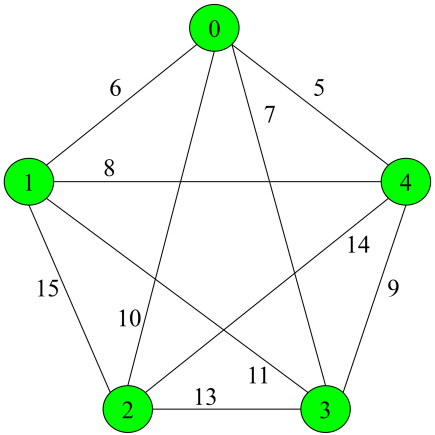
Optimization problem: *Single global optimum*

- Consider the permutation representaton with $n = 4$ whose search space graph is shown in the figure.
- The values of the objective function are shown next to each solution in the figure.
- For this minimization problem, there is only one local minimum (colored red), which is also a global optimum.
- Independent of the starting solution and of the neighborhood search strategy, the local search always stops at the global optimum.



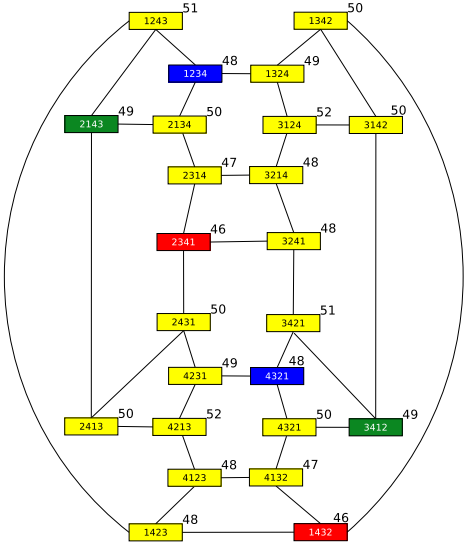
Traveling salesman problem: *Symmetric example*

- Consider the following symmetric traveling salesman problem with five cities shown in the figure.
- Suppose the tour starts from node 0.



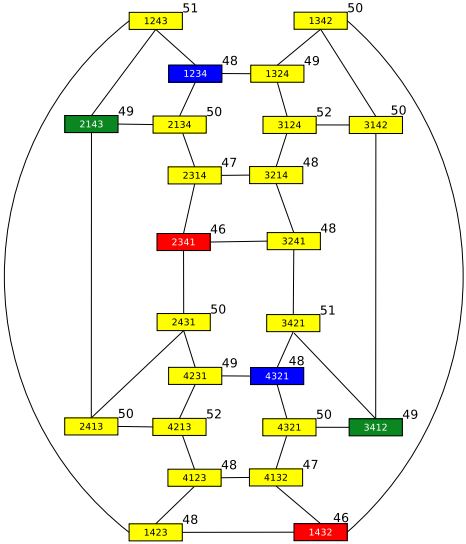
Optimization problem: *Multiple local optima*

- The edge costs are different and the new solution costs are shown in the figure.



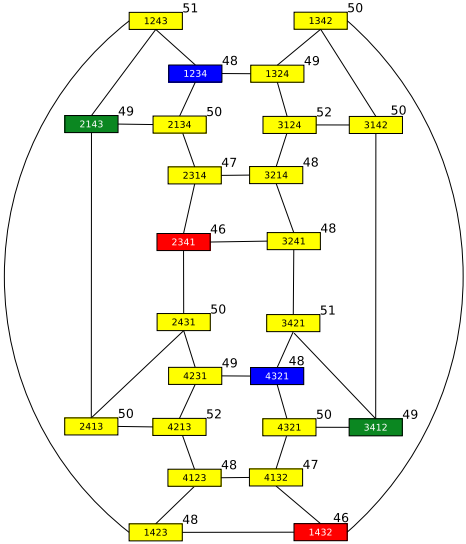
Optimization problem: *Multiple local optima*

- The edge costs are different and the new solution costs are shown in the figure.
- Six nodes of the search space graph correspond to locally optimal solutions: Two green nodes have objective function values equal to 49, two blue nodes have objective function values equal to 48, and the two red nodes have objective function values equal to 46.
- Note that only the red nodes are globally optimal solutions.



Optimization problem: *Multiple local optima*

- The edge costs are different and the new solution costs are shown in the figure.
- Six nodes of the search space graph correspond to locally optimal solutions: Two green nodes have objective function values equal to 49, two blue nodes have objective function values equal to 48, and the two red nodes have objective function values equal to 46.
- Note that only the red nodes are globally optimal solutions.
- The solution obtained by local search varies, depending on both the starting solution and the neighborhood search strategy.



Cost function update

The complexity of each neighborhood search iteration depends on:

- The number of neighbors of each visited solution.
- The efficiency of the computation of the cost function value for each neighbor.

Cost function update

The complexity of each neighborhood search iteration depends on:

- The number of neighbors of each visited solution.
- The efficiency of the computation of the cost function value for each neighbor.

Efficient implementations of neighborhood search usually compute the cost of each neighbor S'

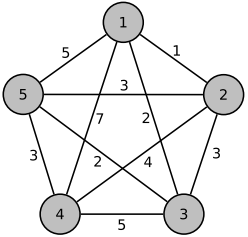
- by updating the cost of the current solution S,
- instead of calculating it from scratch, avoiding repetitive and unnecessary calculations.

Traveling salesman problem: *Cost function update*

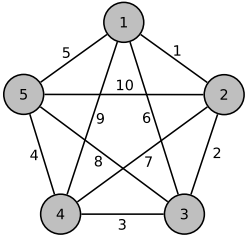
- Recall that every solution S for an instance of the TSP can be represented by an incidence binary vector $(x_1, \ldots, x_m)$, where $m = n(n-1)/2$ and $x_j = 1$ if the edge indexed by j belongs to the corresponding tour, $x_j = 0$ otherwise, for $j = 1, \ldots, m$.

Traveling salesman problem: *Cost function update*

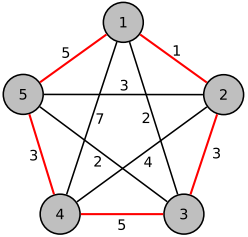
- Recall that every solution S for an instance of the TSP can be represented by an incidence binary vector $(x_1, \ldots, x_m)$, where $m = n(n-1)/2$ and $x_j = 1$ if the edge indexed by j belongs to the corresponding tour, $x_j = 0$ otherwise, for $j = 1, \ldots, m$.
- The figures depicts an example involving a 5-vertex weighted graph.
- The initial solution S corresponding to the incidence vector $(1, 1, 1, 1, 1, 0, 0, 0, 0, 0)$, has a cost of 17.



Edge lengths



Edge indices



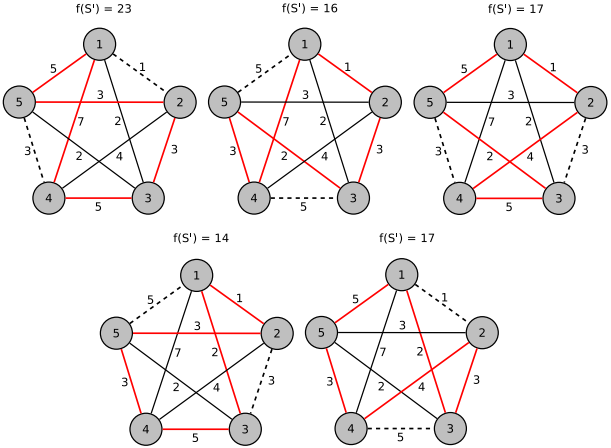
Initial solution: $S =$ $(1, 1, 1, 1, 1, 0, 0, 0, 0, 0)$

Traveling salesman problem: *Cost function update*

- The 2-opt neighborhood for the TSP is defined by replacing any pair of nonadjacent edges of solution S by the unique pair of edges that recreates a Hamiltonian cycle.

Traveling salesman problem: *Cost function update*

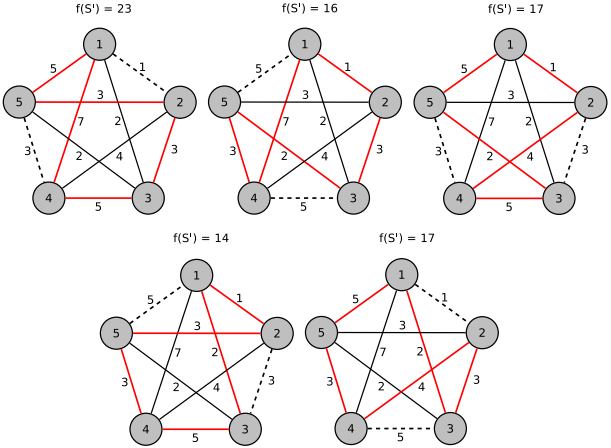
- The 2-opt neighborhood for the TSP is defined by replacing any pair of nonadjacent edges of solution S by the unique pair of edges that recreates a Hamiltonian cycle.
- The figure displays the five 2-opt neighbors of S (whose cost is 17).



First local search iteration

Traveling salesman problem: *Cost function update*

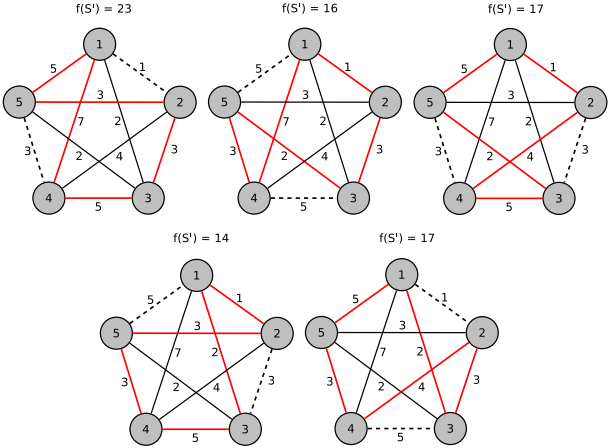
- The 2-opt neighborhood for the TSP is defined by replacing any pair of nonadjacent edges of solution S by the unique pair of edges that recreates a Hamiltonian cycle.
- The figure displays the five 2-opt neighbors of S (whose cost is 17).
- A first-improving neighborhood search strategy returns the second generated neighbor (with cost 16) as the improving solution.



First local search iteration

Traveling salesman problem: *Cost function update*

- The 2-opt neighborhood for the TSP is defined by replacing any pair of nonadjacent edges of solution S by the unique pair of edges that recreates a Hamiltonian cycle.
- The figure displays the five 2-opt neighbors of S (whose cost is 17).
- A first-improving neighborhood search strategy returns the second generated neighbor (with cost 16) as the improving solution.
- A best-improving strategy returns the fourth neighbor (with cost 14).



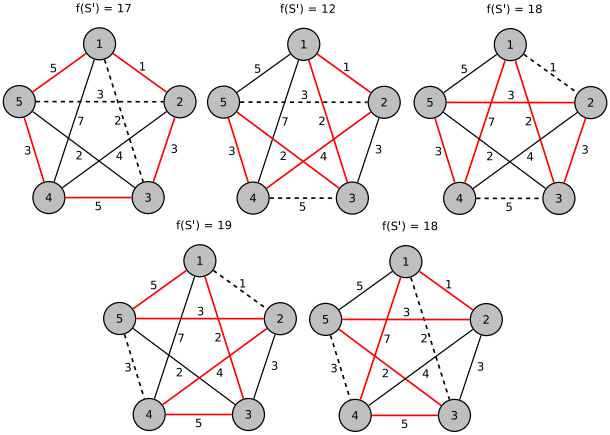
First local search iteration

Traveling salesman problem: *Cost function update*

- Assume that this fourth neighbor is selected and becomes the new current solution $S = (1, 0, 1, 1, 0, 1, 0, 0, 0, 1)$.

Traveling salesman problem: *Cost function update*

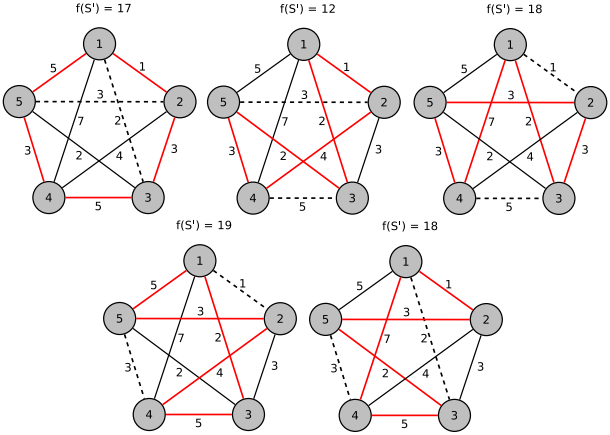
- Assume that this fourth neighbor is selected and becomes the new current solution $S = (1, 0, 1, 1, 0, 1, 0, 0, 0, 1)$.
- The figure displays its five neighbors.



Second iteration: 2nd neighbor is local optimum

Traveling salesman problem: *Cost function update*

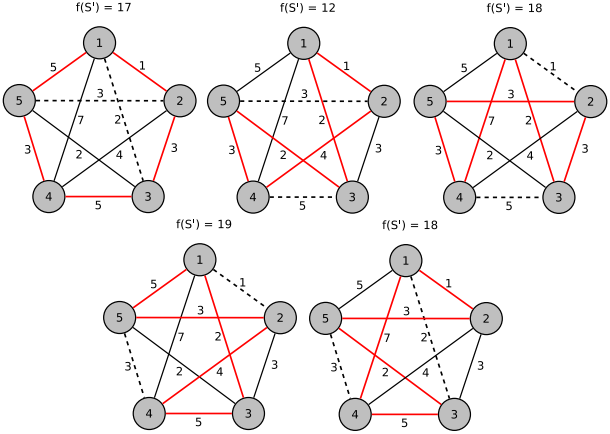
- Assume that this fourth neighbor is selected and becomes the new current solution $S = (1, 0, 1, 1, 0, 1, 0, 0, 0, 1)$.
- The figure displays its five neighbors.
- The first- and best-improving neighbor is the second from left to right (with cost 12).



Second iteration: 2nd neighbor is local optimum

Traveling salesman problem: *Cost function update*

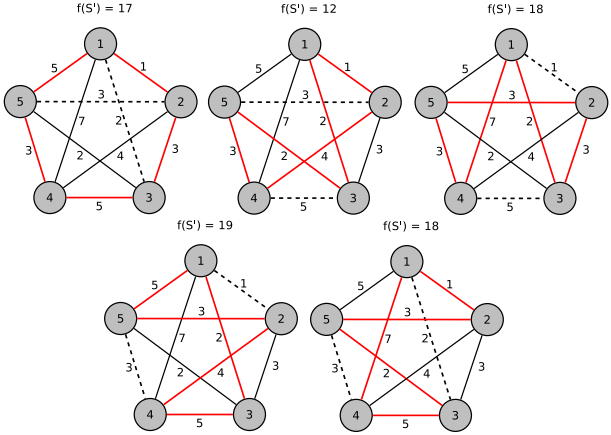
- Assume that this fourth neighbor is selected and becomes the new current solution $S = (1, 0, 1, 1, 0, 1, 0, 0, 0, 1)$.
- The figure displays its five neighbors.
- The first- and best-improving neighbor is the second from left to right (with cost 12).
- Since this solution cannot be improved by any of its neighbors, then it is a local optimum and the search is interrupted.



Second iteration: 2nd neighbor is local optimum

Traveling salesman problem: *Cost function update*

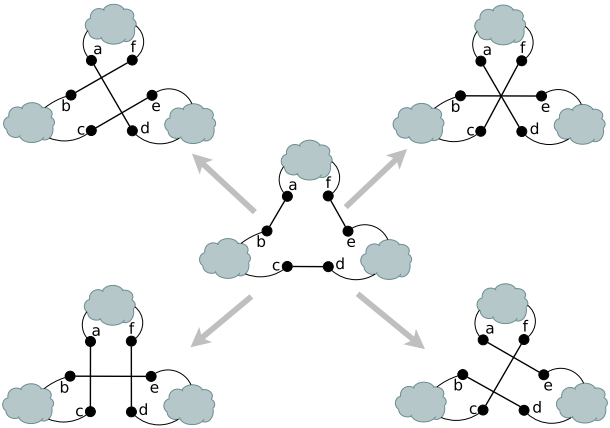
- Assume that this fourth neighbor is selected and becomes the new current solution $S = (1, 0, 1, 1, 0, 1, 0, 0, 0, 1)$.
- The figure displays its five neighbors.
- The first- and best-improving neighbor is the second from left to right (with cost 12).
- Since this solution cannot be improved by any of its neighbors, then it is a local optimum and the search is interrupted.
- The cost of each neighbor S' can be recomputed in $O(1)$ time from the cost of solution S by simply:
  - taking the cost $f(S)$
  - subtracting the lengths of the two removed edges
  - adding the lengths of the two edges that replaced them.



Second iteration: 2nd neighbor is local optimum

Traveling salesman problem: *Cost function update*

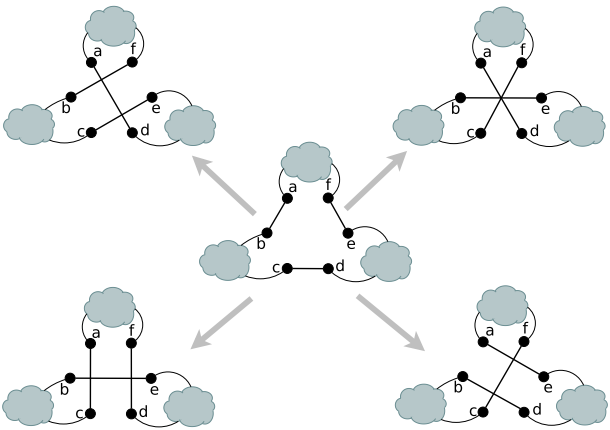
- The 3-opt neighborhood for the TSP can be defined by taking three nonadjacent edges of the current solution and replacing them with any of the four possible combinations of three edges that recreate a tour.
- The figure shows the 3-opt neighborhood.



3-opt neighborhood for the TSP

Traveling salesman problem: *Cost function update*

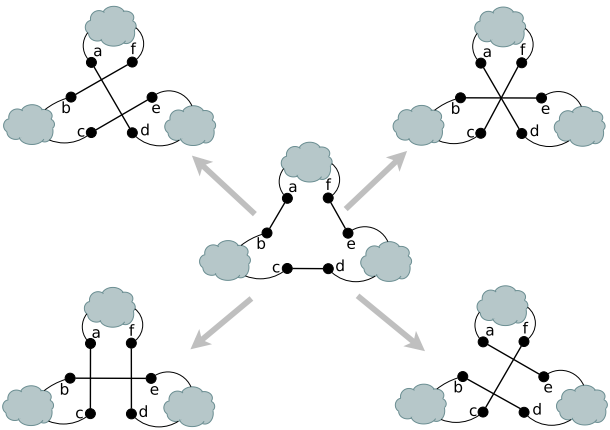
- The 3-opt neighborhood for the TSP can be defined by taking three nonadjacent edges of the current solution and replacing them with any of the four possible combinations of three edges that recreate a tour.
- The figure shows the 3-opt neighborhood.
- Number of neighbors increases to $O(n^3)$:
  - Search becomes slower.
  - More solutions can be investigated and better neighbors might be found.



3-opt neighborhood for the TSP

Traveling salesman problem: *Cost function update*

- The 3-opt neighborhood for the TSP can be defined by taking three nonadjacent edges of the current solution and replacing them with any of the four possible combinations of three edges that recreate a tour.
- The figure shows the 3-opt neighborhood.
- Number of neighbors increases to $O(n^3)$:
  - Search becomes slower.
  - More solutions can be investigated and better neighbors might be found.
- Generalization: k-opt is formed by all solutions that can be obtained by replacing k edges from the current solution by k edges not in the tour, so as to create a new tour.



3-opt neighborhood for the TSP

Concluding remarks

The material in this talk is taken from

- Chapter 4 – Local search

of our book, *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures* (Resende & Ribeiro, Springer. 2016).