proceeding with project j. The integer programming model of the problem is as follows:

$$\text{max} \qquad \sum_{j=1}^{n} c_j x_j \tag{30}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i \in \{1, 2, \cdots, m\} \tag{31}$$

$$x_j \in \{0, 1\}. \tag{32}$$

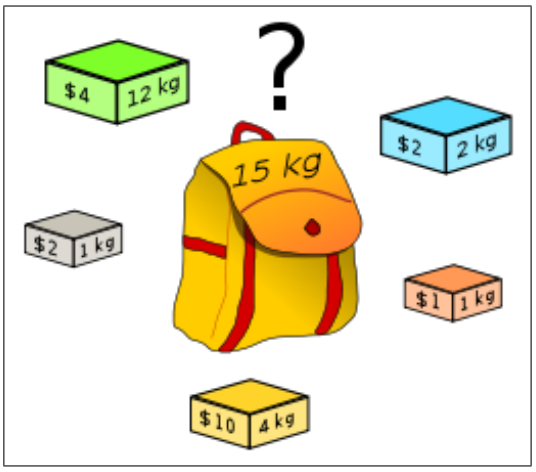

Figure 23: The knapsack problem

Many problems have financial constraints, production capacity constraints, space constraints, etc. Thus, knapsack problems often feature as subproblems of more complex problems.

The quadratic knapsack problem (QKP) is a generalization of the knapsack problem (KP) and it is formulated as:

$$\begin{cases} \max_x & \sum_{j=1}^{n} c_j x_j + \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} d_{kj} x_k x_j, \\ s.t. & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1, \cdots, m, \\ & x \in \{0, 1\}^n, \end{cases}$$

where the coefficients c_j, d_{ij}, a_{ij}, and $b_i \geq 0$; indices i and j are used to denote i constraint and j-th variable, respectively. Without loss of generality, it is assumed that $\sum_{j=1}^{n} a_{ij} > b_i, i = 1, \cdots, m$. Both KP and QKP are NP-hard in the strong sense [5]. QKP has been intensively studied due to its wide applications.

2.2 Traveling Salesman Problem

The traveling salesman problem (TSP) is one of the most widely studied combinatorial optimization problems [4, 6] because of its wide applicability. Here we describe the problem and present some solution methods. The TSP can be easily stated as follows. A salesman wants to visit n distinct cities and then returns to home city. He wants to determine the sequence of the travel so that the overall traveling distance (or travel time) is

minimized while visiting each city not more than once. Hence, the salesman wants to find a tour of minimum length (or least travel time).

More formally, in the traveling salesman problem, a salesman has to perform a tour which involves visiting each of the cities, say $V = \{1, 2, \cdots, n\}$ exactly once and then returning to the starting point. The cost c_{ij} of traveling directly from city i to city j is given, and it is required to find a tour of minimum total cost.

It can be defined using a complete graph (graph-based version), where the cities are given within the framework of a weighted graph $G = (V, E)$ that dictates the connections between cities so that the problem must be solved with constraints on the order that cities can be visited in. The TSP can therefore be modeled as a graph problem by considering a graph and a distance function $c : A \to R^+$. The usual terminology of the TSPs the vertices are called cities and Hamiltonian circuits are called tours. The problem is then to find a Hamiltonian circuit of minimum total length (or travel time) that contains each vertex of V exactly once. A tour is then a cycle in G which touches every node in V exactly once. The set of these cities V together with their distances $D = (c_{ij})$ are known, where the distance (or travel time) between city i and j is c_{ij}. The graph stated above can be a directed graph or an undirected graph. If it is not directed (since the distance between each pair of cities is independent of the direction in which the sales man travels) then the above TSP is a symmetric TSP [8], since $c_{ij} = c_{ji}$ for all $i, j \in V$. This restriction is not always valid and we then obtain in the case of the directed graph what is known as the asymmetric TSP [7].

Although the TSP is conceptually simple, it is difficult to obtain an optimal solution. In an n-city situation, any permutation of n cities yields a possible solution. As a consequence, $(n-1)!$ possible tours ($n!$ if the home city is any city) must be evaluated in the search space in the asymmetric TSP (and $\frac{1}{2}(n-1)!$ for the symmetric TSP with home city being fixed).

Remark 2.1. *The matrix D is said to satisfy the triangle inequality iff $c_{ij} + c_{jk} \leq c_{ik}$ for all $i, j, k \in V$. This occurs, e.g., in Euclidian TSP where V corresponds to a set of points in R^2 and c_{ij} is the straight-line distance between i and j (e.g. in two-dimensional plane). In this form of the problem, distances between cities are calculated using the Euclidean distance formula.*

2.2.1 Mathematical Model of Asymmetric TSP

The asymmetric TSP can be modeled as a graph problem by considering a complete directed graph $G = (V, E)$ with $E = V \times V$, and assigning a cost c_{ij} to every arc $e = (i, j)$. The problem can now be formulated as a linear integer programming problem where the variables are defined as:

$$x_{ij} = \begin{cases} 1 & \text{if saleman goes from city } i \text{ to city } j, \text{or } e = (i, j) \text{ in the tour} \\ 0 & \text{otherwise.} \end{cases} \tag{33}$$

Hence the mathematical model is the following (linear) binary integer program (BIP):

$$\text{min} \quad \sum_{1=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{34}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in V, (i \neq j), \tag{35}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in V, (i \neq j), \tag{36}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad S \subset V, \, 2 \leq |S| \leq n-2, (i \neq j), \tag{37}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in E. \tag{38}$$

Fig. 24 gives the definition of x_{ij} when $e = (i, j)$ is used in a tour (and when not used).

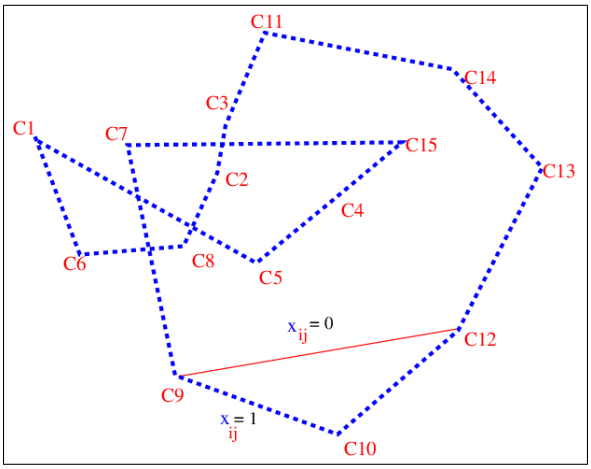

Figure 24: Definition of variable x_{ij}

The constraints specified eqs. (35)-(36) and (38) do not ensure that a binary solution forms indeed a tour (why?). Indeed, the solution without (37) is not a tour but a collection of directed cycles called subtours. Can you create an example?

2.2.2 Mathematical Model of Symmetric TSP

The given complete graph $G = (V, E)$ is undirected, hence it is symmetric - once we obtain a tour, it does not matter which direction we proceed. TSP aims to find the least-cost tour that visits all the vertices. Such a tour (or cycle) visiting all vertices is called a Hamiltonian tour. Defining binary variables x_e for each $e \in E$ and edge costs (or travel times) c_e, symmetric TSP can be formulated as follows:

$$\text{min} \quad \sum_{e \in E} c_e x_e \tag{39}$$

$$\text{subject to} \quad \sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V, \tag{40}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \qquad S \subset V,\ 3 \leq |S| \leq n-3, \tag{41}$$

$$x_e \in \{0,1\}, \qquad (i,j) \in E. \tag{42}$$

Here, $\delta(i)$ refers to all edges connected to vertex i, and $E(S)$ denotes all the edges connecting the vertices of S to S. Constraint (40) ensures that each vertex is visited exactly once, and (41) are the subtour elimination constraints. Even though this is a correct formulation, it has the obvious issue that there are exponentially many subtour elimination constraints. In practice, one can attempt to solve the problem without sub-tour elimination constraints. If the solution contains subtours, add constraints eliminating those subtours, and repeat. In each case, if the integrality constraint is relaxed, the problem is a LP. If the solution of the LP is integral and contains no subtour, that solution is optimal.

The symmetric TSP can also be formulated as follows:

$$\text{min} \quad \sum_{i,j:i<j} c_{ij} x_{ij} \tag{43}$$

$$\text{subject to} \quad \sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \qquad \forall k \in V, \tag{44}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad S \subset V,\ 3 \leq |S| \leq n-3, (i<j), \tag{45}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in E. \tag{46}$$

2.3 Heuristics and Approximate Solution Methods for TSP

Like many combinatorial optimization problems TSP is NP-hard and thus there is very little hope that we will be able to develop efficient algorithms for these problems. Nevertheless, many of these problems are fundamental and solving them is of great importance. There are various approaches to cope with these hardness results. Here, we discuss the following two solution approaches:

- Approximation Algorithms: Approximation algorithms are efficient algorithms that compute suboptimal solutions with a provable approximation guarantee. That is, here we insist on polynomial-time computation but relax the condition that the algorithm has to find an optimal solution by requiring that it computes a feasible solution that is close to optimal (see Figs. 25 & 26).

- Heuristics: Any approach that solves the problem without a formal guarantee on the quality of the solution can be considered as a heuristic for the problem. Some heuristics provide very good solutions in practice. An example of such an approach is local search: Start with an arbitrary solution and perform local improvement steps until no further improvement is possible. Moreover, heuristics are often practically appealing because they are simple and thus easy to implement.

We present one local search heuristic (the 2-Opt Heuristic) and one approximate algorithm (The Christofides Algorithm). Other meta-heuristics such as genetic algorithm and simulated annealing will be presented in a later section.

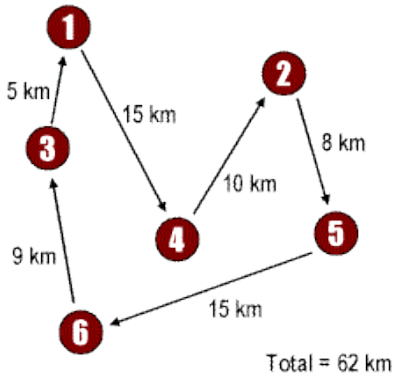

Figure 25: Near optimal solution

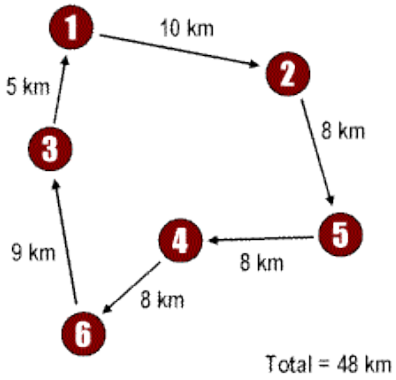

Figure 26: Optimal solution

2.3.1 The Approximation Algorithm

Suppose that P is a minimization problem with set of instances $\mathscr{I}$, and let A be an algorithm that returns for every instance $I \in \mathscr{I}$ a feasible solution $A(I)$ for P. The algorithm A is an α-approximation algorithm with $\alpha \geq 1$ if A finds a feasible solution for all instance I of P in polynomial time such that

$$A(I) \leq \alpha OPT(I), \forall I \in \mathscr{I}, \tag{47}$$

where $OPT(I)$ is the optimal solution of the instance I and $\alpha : \mathscr{I} \rightarrow R^{+}$. For the maximization problem P, the α-approximation can be correspondingly defined with

$$A(I) \geq \frac{1}{\alpha} OPT(I), \forall I \in \mathscr{I}. \tag{48}$$

What would you call the algorithm which find the solution in Fig. 25 in polynomial time? Indeed, the algorithm will be an approximation algorithm and may be called 1.29-approximation algorithm.

2.3.2 The Christofides Algorithm

The Christofides Algorithm is an approximation algorithm for the STSP (symmetric salesman problem). The algorithm guarantees that the distance for the TSP solution found is at maximum $3/2$ times as much as the distance in the optimal solution [26]. Central to

algorithm is fact that every finite undirected graph has an even number of vertices with odd degree (the number of edges touching the vertex). The algorithm is depicted in Figures 27.a - e. and works as follows:

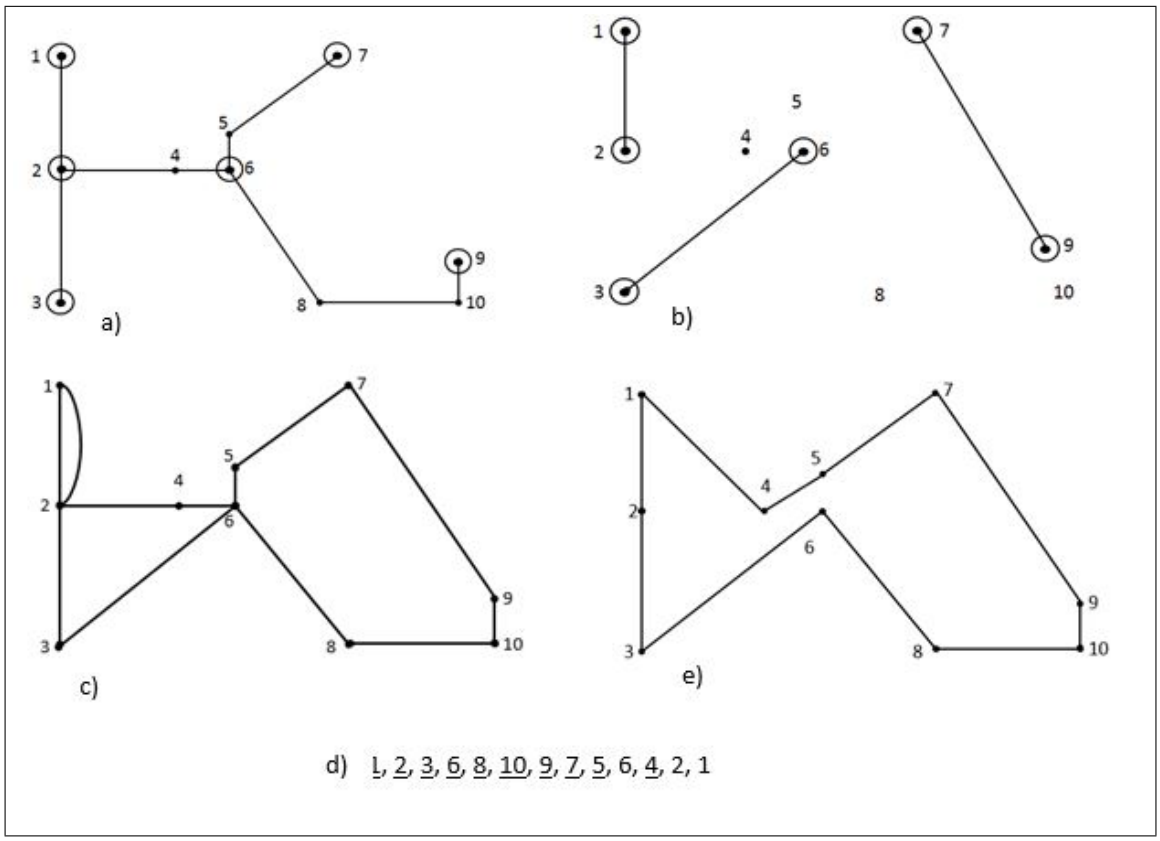

Figure 27: Christofides algorithm [26]

1. Find the minimum spanning tree (MST), the shortest path that will connect all the vertices in the graph G, representing the TSP.

2. Identify all vertices with an uneven number of edges in G, in Figure 27a) the nodes with uneven edges have been circled, these are 1; 2; 3; 6; 7 and 9.

3. Find the shortest set of edges to match all these vertices in pairs. In Figure 27b) edges 1-2; 3-6 and 7-9 were matched.

4. Add these edges to the graph G, where an edge is in both the MST and the matching edges, the edge is doubled. This can be seen in Figure 27c).

5. Construct an Eulerian walk, going through every edge exactly once. Figure 27d) gives the Eulerian walk or cycle for the example.

6. Delete all duplicate vertices in the Eulerian circuit. The remaining sequence of vertices is a Hamiltonian cycle that visits each vertex once and is a solution to the TSP, see Figure 27e).

2.3.3 2-Opt Improvement Heuristic

The 2-Opt Heuristic is probably the most basic local search heuristic for the TSP. 2-Opt starts with an arbitrary initial tour (T) and incrementally improves this tour by making successive improvements that exchange two of the edges in the tour with two other edges. Hence 2-Opt Heuristic generates a random initial tour (city permutation) and iteratively improves it until a local minimum is reached. Given the current tour T (the best tour) the 2-Opt Heuristic searches for a better tour within the neighborhood $N(T)$ of T. Whenever a better tour found within $N(T)$ then current best solution becomes the new best and the process is repeated from the current best T. The current T is considered a local minimum if no neighboring solution is found in $N(T)$ that replaces the current best tour T. $N(T)$ is the set of all two adjacent tours.

What are two adjacent tours? Two TSP tours are called 2-adjacent if one can be obtained from the other by deleting two edges and adding two edges. A TSP tour T is called 2-optimal if there is no 2-adjacent tour to T with lower cost than T.

Basically, the 2-Opt Heuristic looks for a 2-adjacent tour with lower cost than the current tour. If one is found, then it replaces the current tour[6]. This continues until there is a 2-optimal tour. The algorithm terminates in a local optimum in which no further improving step is possible. An example showing how a 2-adjacent tour in $N(T)$ is obtained is shown in Fig. 28. The neighborhood $N(T)$ consists of all such 2-adjacent tours for a given tour T.

In each iteration, the 2-Opt Heuristic applies best possible 2-opt move. This means finding the best pair of edges $(i, i+1)$ and $(j, j+1)$ (not adjacent) such that replacing them with the edges (i, j) and $(i+1, j+1)$ minimizes the tour length[7]. The 2-Opt Heuristic stops when no such tour is found in $N(T)$. Notice that one only needs to check: dist$(i, i+1)$+dist$(j, j+1)$ >dist(i, j)+dist$(i+1, j+1)$ (where dist means distance or edge length i.e. sub tour length), and not the entire tour length in order to determine the neighboring tour is better.

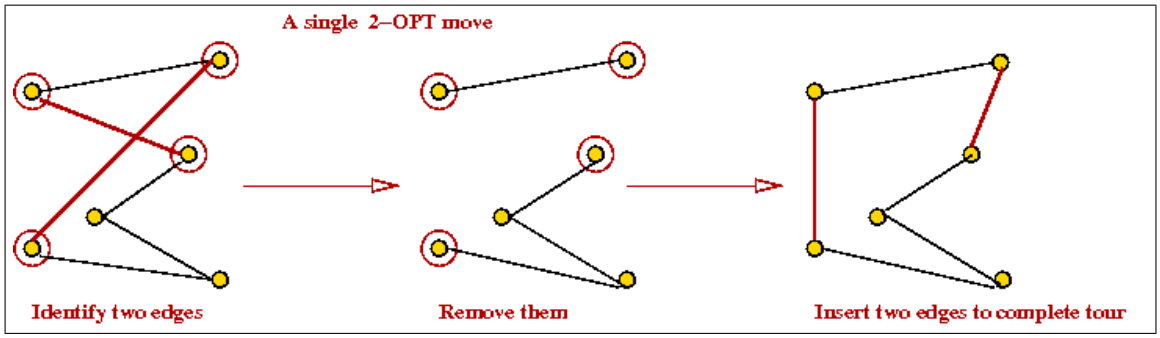

Figure 28: The 2-Opt move

The 2-Opt Heuristic can be written in a general form as follows. Improvement heuristics are based on searching the neighborhood $N(T)$. In this case, the $N(T)$ consists of all tours that can be obtained from T deleting two arcs and inserting two arcs.

Consider the following example of 7 city TSP. Let $T = (c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_1)$ where we assume that the starting city is c_1 which is fixed. For i=1 and j=3,4,5,6 the

[6]This is known as the 2-Opt Move.

[7]This procedure is then iterated until no such pair of edges is found. The resulting tour is called 2-optimal. Note that the global optimum tour is 2-optimal, of course.

2-adjacent neighbors obtained considering all $(i, i+1)$ and $(j, j+1)$ are:

$$\begin{array}{ll} (c_1, c_3, c_2, c_4, c_5, c_6, c_7, c_1), & (c_1, c_4, c_3, c_2, c_5, c_6, c_7, c_1), \\ (c_1, c_5, c_4, c_3, c_2, c_6, c_7, c_1), & (c_1, c_6, c_5, c_4, c_3, c_2, c_7, c_1) \end{array} \tag{49}$$

Similarly for i=2 and j=4,5,6 the 2-adjacent neighbors obtained considering all $(i, i+1)$ and $(j, j+1)$ are:

$$\begin{array}{ll} (c_1, c_2, c_4, c_3, c_5, c_6, c_7, c_1), & (c_1, c_2, c_5, c_4, c_3, c_6, c_7, c_1), \\ (c_1, c_2, c_6, c_5, c_4, c_3, c_7, c_1), & \end{array} \tag{50}$$

For i=3 and j=5,6 the 2-adjacent neighbors obtained considering all $(i, i+1)$ and $(j, j+1)$ are: $(c_1, c_2, c_3, c_5, c_4, c_6, c_7, c_1), (c_1, c_2, c_3, c_6, c_5, c_4, c_7, c_1)$. For i=4 and j=6, the only 2-adjacent neighbor is $(c_1, c_2, c_3, c_4, c_6, c_5, c_7, c_1)$. Hence $N(T)$ (the 2-adjacent neighbors of T) consists of all above 10 tours.

The Basic Steps for the 2-Opt Heuristic

Start with tour T
If there is a tour $\hat{T} \in N(T)$ such that dist($\hat{T}$)<dist(T), then replace T by $\hat{T}$ and repeat
Otherwise, quit with a locally optimal solution.

The Detailed Steps for the 2-Opt Heuristic

Step 1. Create an initial $T = (c_1, c_2, \cdots, c_n, c_1)$
Step 2. Set i=1, D be the length of tour T
Step 3. Set $j = i + 2$;
Step 4. Break th link (c_i, c_{i+1}) and (c_j, c_{j+1}), as shown in Fig. 28, and create new tour $\hat{T}$=$(c_1, c_2, \cdots, c_i, c_j, \cdots, c_{i+1}, c_{j+1}, c_{j+2}.., c_1)$. If cost($\hat{T}$) $< D$ then set $T := \hat{T}$, update D, and go to Step 2. Else go to Step 5.
5. Set $j = j + 1$. If $j < n$ then go to Step 4. In the opposite case increase i by 1 (i.e. $i = i + 1$). If $i < (n - 2)$ then go to Step 3. Otherwise finish

---

The frame-work of the 3-Opt Heuristic can be summarized as follow:

- 3-opt neighborhood: Two TSP tours are called 3-adjacent if one can be obtained from the other by deleting three edges and adding three edges.
- A TSP tour T is called 3-optimal if there is no 3-adjacent tour to T with lower cost than T.
- 3-Opt Heuristic looks for a 3-adjacent tour with lower cost than the current tour. If one is found, then it replaces the current tour. This continues until there is a 3-optimal tour.

Remark 2.2. *In the next chapter, we will present the simulated annealing (SA) algorithm and the genetic algorithm (GA) for solving TSPs. Pseudo-codes of SA and GA will also be provided.*