Ant-based Algorithms for the Wind Farm Cable Layout Problem

Bachelor Thesis of

Miriam Nedlin

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. Peter Sanders
Advisors:	Franziska Wegner

Time Period: 10th January 2017 – 10th May 2017

KIT – University of the State of Baden-Wuerttemberg and National Laboratory of the Helmholtz Association **www.kit.edu**

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 10th May 2017

Abstract

In the design of offshore wind farms the transport of energy though cable connections is an important aspect as the required cables are a high cost factor that needs to be optimized. When planning a wind farm the construction of a cost efficient cabling layout between the turbines and substations is an important step. Such cabling optimization problems are NP-hard. As a consequence we have to rely on heuristic or approximation algorithms for calculating satisfying solutions.

Ant Colony Optimization is a heuristic optimization algorithm inspired by the behavior of real ants. In this thesis we present four different ant-based algorithms for solving the wind farm cabling layout problem. To investigate this problem we use a wind farm model with fixed turbine and substation positions as input. Given a set of cable types with different capacities and costs we want to find an optimal cabling system with minimal costs. We compare our algorithms to a benchmark algorithm using Mixed Integer Linear Programming (MILP) representation. We will see that our ant-based algorithms generate better results than the MILP for some smaller instances. Our best ant-based approach is using a representation similar to Kruskal's algorithm. For larger instances our proposed algorithms perform worse than the MILP, but we give suggestions how to build an improved algorithm based on our results.

Deutsche Zusammenfassung

Bei der Planung von Windpark-Anlagen, insbesondere bei Offshore-Windparks, stellt die Verkabelung einen wichtigen Kostenfaktor dar. Daher ist die Optimierung des Kabelsystems ein bedeutender Bestandteil der Planungsphase von Windenergieanlagen. Diese so genannten Verkabelungsprobleme sind NP-schwer. Aus diesem Grund müssen heuristische Algorithmen benutzt werden, um eine ausreichend gute Lösung zu erzeugen.

Ant Colony Optimization ist ein heuristischer Optimierungsalgorithmus, der vom Verhalten der Ameisen inspiriert wurde. In dieser Arbeit werden vier verschiedene Ant-basierte Algorithmen vorgestellt, mit denen das Windfarm Verkabelungsproblem gelöst werden soll. Um dieses Problem zu untersuchen, verwenden wir ein Windpark-Modell mit fest platzierten Turbinen und Umspannwerken sowie einer Auswahl von Kabeltypen mit unterschiedlichen Leistungskapazitäten und Kosten als Eingangsparameter. Ziel ist es, ein optimales Verkabelungssystem mit minimalen Kosten zu finden. Wir vergleichen die Ergebnisse der vier Algorithmen mit einem Benchmark-Algorithmus, der das Problem durch Mixed Integer Linear Programming (MILP) versucht zu lösen. Wir werden sehen, dass die Ant-basierten Algorithmen bessere Ergebnisse für viele kleinen Instanzen liefern, als der Vergleichsalgorithmus. Bei großen Instanzen erzeugt der MILP-Algorithmus bessere Ergebnisse. Dazu geben wir Hinweise, wie man auf Grundlage unserer Ergebnisse einen besseren Algorithmus entwerfen könnte.

Contents

1. Introduction

Today's energy production relies heavily on conventional methods based on fossil sources like coal and oil or nuclear power energy [onl16b]. Those fossil fuels are non-renewable, that is, their supply is limited and they become more expensive. A long-term change to renewable energy production becomes more profitable and will be finally necessary.

Wind is a very efficient renewable energy source that is free of charge, unlimited and almost everywhere available. Typically, a large number of turbines that produce energy are concentrated in so called wind farms.

Since 1990 several tens of thousands of wind turbines have been installed in Germany. As a result, wind energy makes a significant contribution to German power supply. The AEEStat (Arbeitsgruppe Erneuerbare Energien-Statistik) published statistics about the quantity of sources renewable energy in total energy production in Germany from 2016. The relative amount of renewable energy is for electricity 31.7%, heating 13.4% and traffic 5.1%. The share of wind energy from all renewable energy is about 20% [onl16a].

For a high electrical power yield a high average wind speed as well as a consistent wind speed and direction is required. Conditions like this can be found in coastal or offshore area. [onl16c]

In general the location of wind farms is usually far away from energy consumers. Consequently power transmission is an important issue when planning wind farms.

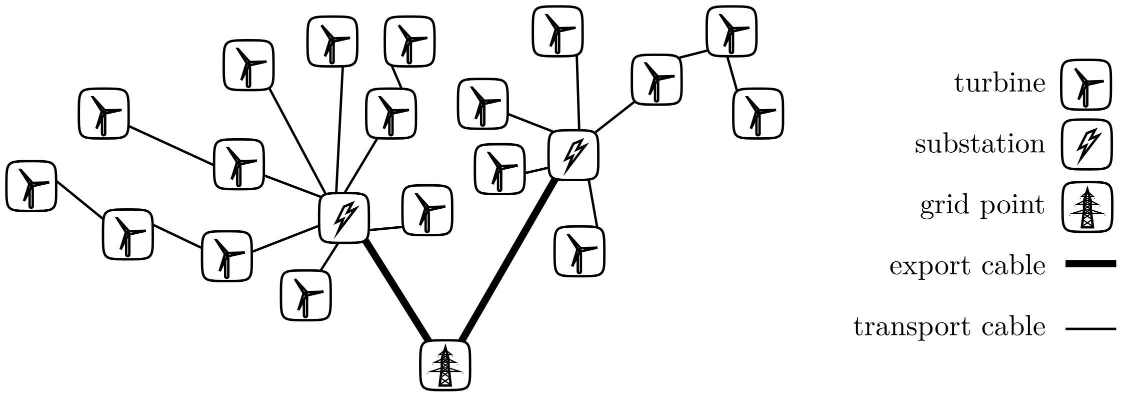


Figure 1.1: Structure of a wind farm

A typical wind farm contains a group of wind turbines, substations and grid points which are connected by a cabling system (see Figure 1.1). The turbines are driven by wind wheels and convert wind's kinetic energy into electrical power. This electrical energy is transferred by the cabling system to a substation collecting the electrical power of a group of turbines. The substation transforms the electrical energy to alternating current with a high voltage level to minimize transport energy loss. All substations are connected to a grid point, which is the entry of the general power grid.

When planning a wind park there is a variety of properties to consider. Most important in view of cost and energy production is positioning of the turbines and substations as well as the electrical infrastructure of the park. Finding an optimal wind farm layout is a very complex task that is like many other optimization problems such as the traveling salesman problem not solvable in polynomial time. Even the substation problem which contains finding a cabling between multiple turbines and a single substation is NP-hard as it is equal to the capacitated minimum spanning tree problem [AAN82]. As a result wind farm projects today are often designed manually.

We limit the wind farm layout problem by considering the turbines and substations as objects with fixed positions. The remaining subproblem consists of finding an optimal cable system. We call this problem the wind farm cabling problem. As reported by Kirby et al. [KXLS02] such electric infrastructure accounts for approximately 15% of the total cost of an offshore wind farm. Different cabling layouts differ in costs and reliability through redundancy.

As this optimization problem cannot be solved exactly, heuristic algorithms are necessary. Especially evolutionary algorithms achieve promising results for similar problems. There are different known algorithms that are suitable to provide a good solution for the wind farm layout problem. The following methods appear to be particularly suitable for our problem:

Genetic algorithm which simulates the process of natural selection by regarding a set of solutions as a population. A new population generation is being created based on results of the previous generation. First the best solutions of last generation are being selected. This process is called selection. Then genetic operations are applied to these solutions. These operations include mutation which performs small random modification on a solution and crossovers which merges two solutions into a single solution.

Simulated annealing which is an optimization algorithm that considers a single solution and applies random changes to this solution. Changes that improve the solution's quality are always accepted whereas worse solutions are only accepted by chance depending on the simulated temperature. At the beginning this temperature is set high which leads to high chance of accepting worse solution. Over time the temperature and chance of accepting solutions that are worse than the current solution declines.

Ant colony optimization is a meta-heuristic optimization method that simulates the behavior of ants in a graph model (see Chapter 3). In every calculation step an ant will be simulated that finds a random solution by choosing a path in the graph. Depending on the quality of this solution a smaller or larger amount of pheromones will be dropped along the chosen path. The pheromone affects the random selection of future ant. An ant will more likely choose a direction where more pheromones where dropped.

Tabu search is a heuristic search algorithm that finds a solution by checking similar solution to the current solution. From this neighborhood the best solution is selected as new current solution. All selected solutions are added to a tabu list and will be ignored in future steps.

Ant colony optimization (ACO) is an algorithm that has been successfully applied to other optimization problems such as the vehicle routing problem [HSLL05] or maximizing energy yield in the wind farm layout problem [ES12] which are both similar to our problem. For many problems such as the multiple traveling salesman problem [CM07] ant-based algorithms achieved better result than other meta-heuristics. In this thesis we want to apply ACO on the wind farm cabling problem.

We use four different representation approaches for the wind farm cabling problem. The first approach is an algorithm that solves a modified traveling salesman problem (TSP). In contrast to the usual TSP this modification allows the solution to have a tree-structure. This solution fits more to a wind farm structure. Our second approach uses a representation that result in an algorithm similar to Kruskal's algorithm for minimum-spanning trees. Our third algorithm is using reversed representation where the ants begin at the turbines and search for a path to the substations. In our last algorithm every turbine node selects a successor and forwards all incoming current to its successor. These four algorithms are being compared to results created by the simulated annealing-approach and mixed integer linear programming implementation by Lehmann et al. [LRWW17].

This thesis is structured as the following: In the beginning Chapter 1 gives a short introduction to the topic. Chapter 2 presents other works considering the wind farm cabling problem. Chapter 3 contains basic notation used in this thesis. In Chapter 4 the cabling problem will be explained in detail. Chapter 5 contains an explanation of our used algorithms and finally in Chapter 6 our results are evaluated.

2. Related Work

The wind farm cabling problem (see Chapter 4) is an NP-hard problem. Consequently non trivial problem instances cannot be solved in feasible time. A solution can be calculated in polynomial time using heuristic algorithms. These solution are not necessary optimal, but often near optimal. We will now give a short summary of works related to our topic.

2.1 Cable Network Design on Large-scale Wind Farms

Berzan et al. [Ber05] considers the wind farm cabling problem (see Chapter 4) with a single and with multiple cable-types. The problem is divided into three problem layers and a solution approach for each individual layer is given.

The first and lowest layer is the circuit layer. In the circuit layer problem a set of turbines has to be connected in a way that from every turbine there has to be a path to the root turbine which is then connected to a substation. When considering only a single cable-type this problem is equal to the minimum spanning tree problem which can be solved in linear time. Finding an optimal solution becomes significantly harder when multiple cable-types are allowed. The author presents multiple approaches for this problem. Eventually a divide-and-conquer algorithm is shown that provides optimal results for circuits with upto 14 turbines in under one minute.

The second layer is called the substation problem layer. In this sub-problem a set of turbines has to be connected to a single substation. For this intermediate and the highest layer only the single cable-type variations of the problems are considered. In case of the substation layer the problem is equivalent to the capacitated minimum spanning tree problem. This problem is a known NP-hard problem with a great number of exact and heuristic algorithms.

The last and highest layer is the full farm problem. In this problem a cabling between a set of turbines and a set of substation has to be found. For the single cable-type case this problem is compared to the local access network design problem. A feasible solution to a problem is found by solving a min-cost flow problem to determinate the assignment from turbines to substation. Then every individual substation problem is solved by using the Esau-Williams heuristic which is explained and discussed by Gavish [Gav91]. These algorithms has been applied to generated problem instances with up to 1000 turbines.

In order to create a more realistic cost model the pairwise costs between turbines and substations are precalculated using a terrain grid.

2.2 Simulated Annealing-Based Heuristics for Wind Farm Cabling Problems

In [LRWW17] a simulated annealing approach to the wind farm cabling problem is presented. As described in Chapter 1 simulated annealing is a heuristic optimization algorithm that starts with an initial solution and randomly modifies its current solution. Changes that result in a better solution are always accepted whereas worse changes are only accepted by a certain chance.

The algorithm works on a graph representation of a wind farm. A solution state is represented by a potential that is assigned to every node and the states of the edges. An edge is either cut edge and not used for this solution or normal edge. A cabling from this representation is then generated by creating an intermediate representation that contains the maximum cable throughput when energy flows from each node to the next connected node with highest potential. The final cabling is then chosen by selecting the cheapest possible cable for every edge that fits the appropriate throughput requirement.

In addition multiple improvements have been implemented:

- Usually the temperature declines at a fixed rate. With a dynamic temperature curve the decrease of temperature is reduced as long as better solution are being found.
- Resetting the temperature to a higher level when no improvement has been found for a long time.
- Generating multiple independent solutions and combining (crossing) them
- Using a two-level approach where turbines are first assigned to substation and every substation problem is solved individually

This algorithm was implemented and compared to a mixed integer linear programming (MILP) representation of the problem. Overall the simulated annealing approach delivered better results than the generic MILP solution for generated instances with up to 450 turbines. For larger instances this simulated annealing approach performed worse than the generic algorithm.

3. Preliminaries

In this this chapter basics and notation for graphs and flow networks are presented.

3.1 Graph Theory

A graph is an abstract representation of objects and relations between these objects. Such a graph G is defined as tuple $G = (V, E)$ with V as a set of nodes and $E \subseteq V \times V$ as a set of edges between theses nodes. We call a graph *directed* if E is a set of ordered tuple or *undirected* if E is a set of unordered tuple.

In G the neighborhood $N(v)$ of a node $v \in V$ is defined as the set of all nodes adjacent to v. We call a pair of nodes (v, w) adjacent if v and w are connected though an edge $e = (v, w)$ or $e = (w, v)$. For directed graph we also use $N^+(v)$ as the subset of neighbors with incoming edges to v and $N^-(v)$ as the subset with outgoing edges to v. Similar $d(v)$ is the set of all edges connected with v. d^+ is the subset of outgoing edges and d^- is the subset of incoming edges. We call all edges in $d(v)$ *incident* to v.

$$N : V \to \mathcal{P}(E)$$
$$N(v) = \bigcup x : (v, x) \in E$$

The degree of a node $d(v)$ is defined as the amount of edges connected to node v. For directed graph we also define in-degree $d^-(v)$ as the amount of incoming edges and out-degree $d^+(v)$ of outgoing edges.

3.2 Flow Network

Given a directed graph $G = (V, E)$, a capacity function $c : E \to \mathbb{R}_{\geq 0}$, a source node $v_{source} \in V$ and a drain node $v_{drain} \in V$ we define a flow network as tuple $(G, c, v_{souce}, v_{drain})$. A flow f for this network is a function that maps a flow value to every edge in graph G. This flow f implies an excess value $ex_f : V \to \mathbb{R}$ for every every node. This excess value is defined by the sum of incoming and outgoing flow of a node:

$$ex_f(v) = \sum_{e \in d^+(v)} f(e) - \sum_{e \in d^-(v)} f(e) \tag{3.1}$$

We call a flow f valid if no flow value exceeds the capacity and the excess for all nodes (except source and drain) is zero.

$$\forall e \in E : f(e) \leq c(e) \tag{3.2}$$

$$\forall v \in V \setminus \{v_{drain}, v_{source}\} : ex_f(v) = 0 \tag{3.3}$$

Note that we can transform an undirected graph into a directed graph by replacing all edges with edges in both directions. We can easily create a flow networks for undirected graphs by using this algorithm mentioned before.

4. Wind Farm Optimization Problems

In this section the wind farm optimization problem will be presented. A modeling of the problem is presented in three layers (sub-problems), following Lehman et al. [LRWW17]. The problem will be described using a graph model.

We will now give a bottom-up overview of an actual wind farm. An example farm is shown in Figure 4.1. Power is being generated at several *turbines* within the farm. In order to minimize power loss through transmission we need to step up voltage. This transformation is done at *substations* placed all over the wind farm. The electric current is transmitted from turbines to substations through *transport cables*. From there energy has to be delivered to *grid points* though *export cables*. These grid points are connected to the electric power distribution network.

4.1 Flow problem representation

We consider wind farms modeled as graph $G = (V, E)$ with just two node types: Turbines and Substations. We want to find a valid cabling between these turbines and substations. A cabling is a set of connections between nodes with a certain cable for each connection. A problem instance has the following parameters:

- Set of turbine and substation nodes $V = V_S \cup V_T$
- Set of cable types K
- Capacity function $\mathrm{cap} : K \to \mathbb{R}_{\geq 0} \cup \{\infty\}$
- Cost function $\mathrm{c_{cab}} : K \to \mathbb{R}_{\geq 0} \cup \{\infty\}$

A cabling solution is a tuple (κ, f) consisting of flow f on G which contains all connections between nodes and mapping $\kappa : E \to K$ which maps a cable type to every edge. We call a solution (κ, f) valid if the following constrains are fulfilled:

1. There is a path from every turbine to a substation
2. Even with all turbines running at maximum power rating electric current has to stay below cable capacity.
3. In the same situation as in 2 the maximum power rating for substations must not be exceeded.

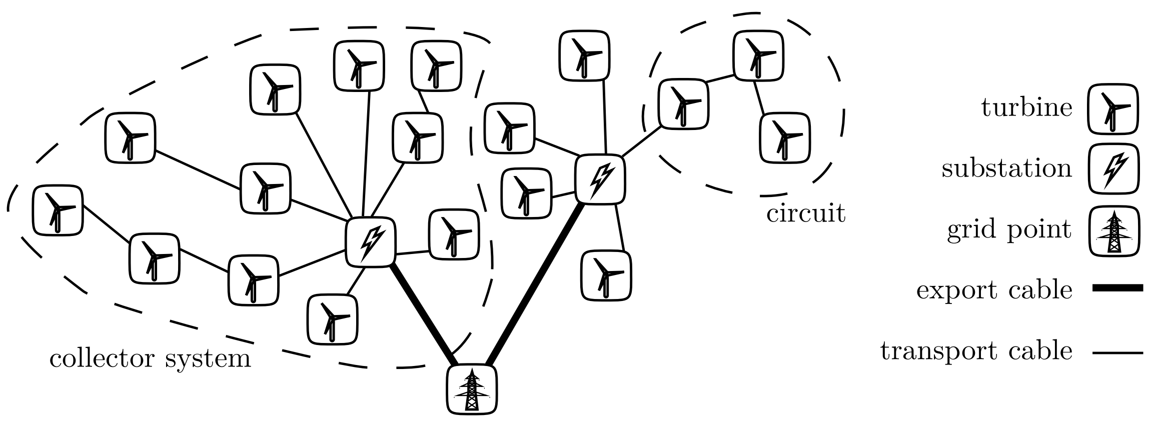


Figure 4.1: Example for a wind farm structure. Every turbines is connected to a substation through transport cables. The substation are connected to a grid point though export cables. A single substation with all connected turbines is called collector system. A set of interconnected turbines with one turbine connected to a substation is called a circuit.

We verify these constrains by constructing a flow problem. (See Chapter 3)

Let $V_T \subset V$ a subset containing all turbines, $V_S \subset V$ containing all substations. We define a modified graph for our flow problem $G' = (V', E')$ with two added nodes v_{source} and v_{source} as

$$V' = V \cup \{v_{\text{source}}, v_{\text{drain}}\} \tag{4.1}$$

$$E' = E \cup E'_T \cup E'_S,\ E'_T = \left\{ \bigcup_{v \in V_T} (v, v_{\text{source}}) \right\},\ E'_S = \left\{ \bigcup_{v \in V_S} (v, v_{\text{drain}}) \right\} \tag{4.2}$$

and edge capacity function $c : E' \to \mathbb{N}$ defined as

$$c(e) = \begin{cases} C_{\text{turbine}} & \text{when } (e \in E'_T) \\ C_{\text{substation}} & \text{when } (e \in E'_S) \\ \text{cap}(\kappa(e)) & \text{otherwise} \end{cases} \tag{4.3}$$

where C_{turbine} is maximum power rating of a turbine and $C_{\text{substation}}$ is maximum power rating of a substation.

Let $f_{\max} = |V_T| C_{\text{substation}}$ the maximum capacity all turbines produce together. If a flow with capacity $f \geq f_{\max}$ exists, we call our solution valid as all three constrains for a valid solution are fulfilled.

For the sake of simplicity we assume that all turbines have the same maximum power rating as they are usually not placed far from each other and have the same type.

In our case the cables connecting the substations to the grid points are given and we do not consider these connections further. Therefore condition 2 is automatically fulfilled. We also consider the locations of all substations and turbines as fixed.

We use a simplified cable capacity metric where the capacity represents the maximum number of turbines connected through a single cable that is sufficient to fulfill the third constrain. Therefore all capacities are integers. A similar metric is also used for substation power rating. The capacity of a substation is given by the maximum number of turbines that does not overload the substation.

4.2 Circuit Problem

The circuit problem is first and most simple sub-problem. A circuit is a set of turbines connected though cables with a single turbine v_t connected to a substation. In the circuit problem we are given a set of turbines and cable-types. We want to find the cheapest possible cabling between all turbines.

If we apply this to problem to our flow network representation from section 4.1. The set of substations V_S contains only a single substation v_s. In our graph $G = (V, E)$ there is only one edge between the substation and the turbines. This edge $\{v_s, v_t\}$ is always part of the solution. The costs c of a valid solution (κ, f) are defined by $c(\kappa, f) = \sum_{e \in E} \mathrm{c}_{\mathrm{cab}}(e) len(e)$.

When we limit the problem to one single cable type, this problem is equal to the capacitated minimum spanning tree problem which is a known NP-hard problem [AAN82].

4.3 Substation Problem

The next larger problem is the substation layout problem. In this problem we want to connect a set of turbines to a single substation with multiple circuits.

For this problem we are given a set of cable types K and a complete graph $G = (V, E)$ where $E = V \times V$. In this graph $v_s \in V$ represents a substation and all other vertices represent turbines. We are also given a cost function $\mathrm{c}_{\mathrm{cab}} : K \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ and a capacity function $\mathrm{cap} : K \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ that specifies every cable type. Note that K also contains an extra cable type k_0 that represents no cable between two points. For this cable type $\mathrm{cap}(k_0) = 0$ and $\mathrm{c}_{\mathrm{cab}}(k_0) = 0$ is always true.

As described in section 4.1 we can now form a flow problem that contains the constraints necessary for a valid cabling between the turbines and substation. The solution of this problem (κ, f) assigns a cable type $\kappa(e)$ to every edge e in graph G and provides a flow f that describes the energy transfer in the cable network. We want to find a solution, that minimizes the total cost across all edges: $\min \sum_{e \in E} \mathrm{c}_{\mathrm{cab}}(\kappa(e)) len(e)$.

4.4 Full Farm Problem

The previous problem 4.3 is a simplified problem with only one single substation. In most cases wind farms have more than one substation. This leads to the substation assignment problem where a set of turbines have to be connected to a set of substations.

Input for this problem is the same as in the substation problem except we extend the graph G in the following way: G now contains a subset $S \subset V$ that represents all substations in our graph. All remaining vertices $V \setminus S$ are turbines. Cable cost and capacity function remain the same.

Again we use a constructed flow problem representation as described in Section 4.1 to describe this problem. Every turbine is connected to a dummy source with an edge e that has capacity C_{turbine} (maximum power rating of a turbine). Likewise every substation is connected to dummy sink through an edge with capacity $C_{\text{substation}}$ (maximum power rating of a substation).

A solution candidate is given through a tuple (κ, f). Again, we want to find a valid solution, that minimizes the total cost across all edges: $\min \sum_{e \in E} \mathrm{c}_{\mathrm{cab}}(\kappa(e)) len(e)$

5. Documentation

When we limit our wind farm optimization problem to a single substation and single cable type, we can solve this limited problem by finding a minimum spanning tree that fulfills the capacity constrain described in Chapter 4. This problem is known as capacitated minimum spanning tree problem and is NP-hard. As a subproblem of our wind farm optimization is a NP-hard problem the problem itself is NP-hard too and we need a heuristic algorithm in order to find a solution in feasible runtime. In this section we introduce the ant colony optimization algorithm. We first describe a standard ant algorithm, then two modified algorithms that we used specifically for our problem.

5.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a heuristic optimization algorithm. The idea for this algorithm is based upon ant behavior observed in nature: If an ant finds a source of food, it will transport a piece of food back to its anthill. On the way back it will drop so called food trail pheromones, which mark the path for other ants. They can detect these pheromones and follow the path to the source of food found. The more pheromones on a path the higher the chance that an ant will follow this path. As the pheromones will decay over time, shorter paths to a food source get a higher pheromone concentration than longer paths.

Algorithm 5.1: Ant Colony Optimization

```
   Input: Graph $G = (V, E)$, cost function cost : $\mathcal{P}(E) \to \mathbb{R}$
   Data: Pheromone Values $P_E : E \to \mathbb{R}$, Current solution $\mathsf{C} \subseteq E$
   Output: $S \subseteq E$
1  Initialize pheromone values
2  repeat
3      $\mathsf{C} \leftarrow$ findPath($G$, $P_E$)
4      $P_E \leftarrow$ updatePheromones($\mathsf{C}$, $P_E$)
5      if cost($\mathsf{C}$) $<$ cost($\mathsf{S}$) then
6          $\mathsf{S} \leftarrow \mathsf{C}$
7  until time limit reached
8  return $\mathsf{S}$
```

Algorithm 5.2: Ant Colony Optimization: FindPath

```
Input: Graph $G = (V, E)$, Pheromone Values $P_E$
Data: Current Node c, Choice List L, Node marking $M : V \to \{0, 1\}$
Output: $T \subseteq E$
1  forall $v \in V$ do
2      $M(v) \leftarrow 0$
3  c ← Select random node from $V$
4  repeat
5      L.clear()
       // Select all valid neighbors
6      forall $\{c, v\} \in E$ do
7          if $M(v) = 0$ then
8              L.insert($c$, $v$)
9      if L is not empty then
10         Select a random edge $e = (c, v)$ from L using $P_E$ as weight
11         $M(v) \leftarrow 1$
12         $T \leftarrow T \bigcup \{e\}$
13         $c \leftarrow v$
14 until L is empty
```

The first implementation of ACO is called ant system [DMC96]. Ant system finds a solution to the traveling salesmen problem. The calculated solution is not necessarily optimal as ACO is a probabilistic algorithm. The algorithm takes a graph as input and searches for the shortest round-trip connecting all vertices.

The algorithm is shown in Algorithm 5.1. Our input parameters consists of a graph $G = (V, E)$ and a cost function which calculates the cost of every valid solution cost : $\mathcal{P}(E) \to \mathbb{R}$. The algorithm uses a mapping $P_E : E \to \mathbb{R}$ that represents the pheromone value for every edge. The result of this algorithm is a subset $S \subseteq E$ which is also a spanning tree for G. At the beginning the input graph will be initialized by setting the pheromone value to 1 for all edges $e \in E$ (code line 1). Then the algorithm performs two steps: `FindPath` (line 3) and `UpdatePheromones` (line 4) until an exit condition is met. After each iteration of the algorithm the best found solution will be updated (line 6). The exit condition is usually a predefined execution time limit (line 7).

The method `FindPath` generates a new solution by simulating an ant walking through the graph. The algorithm for this subroutine is shown in Algorithm 5.2. The method uses graph $G = (V, E)$ and the current pheromone values $P_E : E \to \mathbb{R}$. The ant's position is represented as current node $c \in V$. Additionally the algorithm use a mapping $M : V \to \{0, 1\}$ which assigns either a marked (1) or an unmarked (0) state to every node. At the beginning all nodes are unmarked (line 1–2) and c is assigned to a random node (line 3). Then the algorithm check every outgoing edge from c (line 6). Each edge connected to an unvisited (unmarked) node will be added to a list of edges L (line 8) containing all possible next step for the ant. Then a random edge from list L will be selected (line 10). Considering an edge $e \in E$ that is connected to current node c. The chance $P_e \in [0, 1]$ for selecting edge $e \in E$ from list L is given by

$$P_e = \eta(e)\tau(e)P, P = \sum_{\substack{x=\{c,v\}\in E \\ M(v)=0}} \eta(x)\tau(x) \tag{5.1}$$

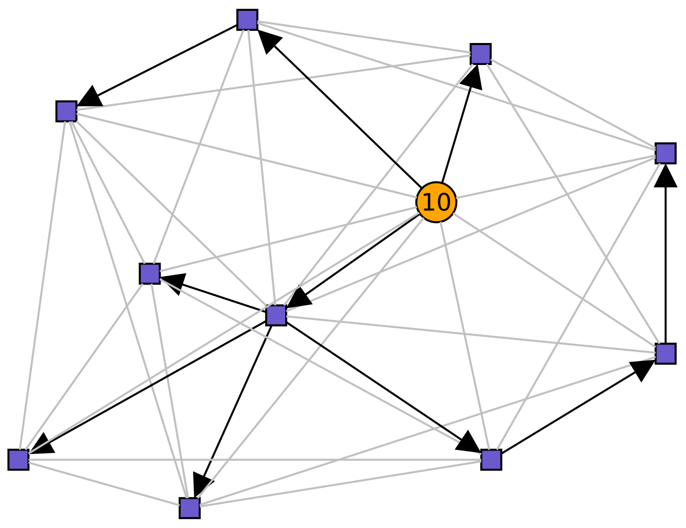


Figure 5.1: A flow generated in an iteration of Algorithm 5.3.

where

- P is the total weight of all incident edges to c that are connected unvisited nodes
- $\eta : E \to \mathbb{R}$ is a heuristic value which is typically (length of e)$^{-1}$
- $\tau :\to \mathbb{R}$ A pheromone value that holds information from previous ants

After choosing a random edge $e = \{c, v\}$, c will be updated to v and the edge selection will be repeated (line 11–13). When L is empty, the algorithms returns a set of all chosen edges.

After a round trip was found, the algorithm performs an update to all pheromone values. This is done by the `UpdatePheromones` subroutine. All τ-values will be decrease by a fixed percentage. Then all edges within the generated path will increase its pheromone values depending on the total cost of the solution. A better solution results in a larger increase. Let Solution $S \subseteq E$ a set that contains all edges chosen by the ant. Updated pheromone values τ' are calculated by

$$\tau'(e) \leftarrow \begin{cases} \tau(e) \cdot (1-p) & , e \in S \\ \tau(e) \cdot (1-p) + \frac{C}{\text{cost}(S)} & , \text{otherwise} \end{cases} \tag{5.2}$$

where $p \in [0, 1]$ and $C \in \mathbb{R}^+$ are fixed parameters.

5.2 Traveling Salesman Approach

The traveling salesman approach is our first approach that is a modified version of the ACO algorithm for the TSP presented in Section 5.1.

Considering the wind farm problem we can directly use ACO algorithm from Section 5.1 to find a solution for our problem. But the optimal solution for a wind farm-problem is not necessarily a single straight path and we want to allow branches in the resulting spanning tree. We can achieve this with a slightly modified algorithm.

We use a different `FindPath` approach as shown in Algorithm 5.3. Instead of a current node c this algorithm uses a queue of node elements Q. Every node in Q denotes a branch

Algorithm 5.3: FindPath in Traveling Salesman Approach

```
    Input: Graph $G = (V, E)$, Substation nodes $V_S \subset V$, Pheromone Values $P_E, P_V$
    Data: Queue Q, Current Node c, Choice List L, Node marking $M : V \to \{0, 1\}$
    Output: $T \subseteq E$
1   $T \leftarrow \emptyset$
2   forall $u \in V$ do
3       if $u \in V_S$ then
4           Q.push($u$)
5           M($u$) $\leftarrow 1$
6       else
7           M($u$) $\leftarrow 0$
8   while Q is not empty do
9       c $\leftarrow$ Q.pop()
10      L.clear()
        // Select all valid neighbors
11      forall $\{c, u\} \in E$ do
12          if M($u$) $= 0$ then
13              L.insert($c$, $u$)
14      if L is not empty then
15          Select a random edge $e = (c, u)$ from L using $P_E$ as weight
16          Select a random amount of branches $n$ for target node $u$ using $P_V(u)$ as
              weight
17          M($u$) $\leftarrow 1$
18          $T \leftarrow e$
19          forall $i \in (1..n)$ do
20              Q.push($u$)
```

from this node. This algorithm also uses a subset of nodes $V_S \subset V$ which contains all substation nodes as input. At the beginning Q will be initialized with every substation node (line 2–4). In every step the first queue element will be removed from Q and used as current node c (line 5). Then the next edge will be selected the same way as in Algorithm 5.2 (line 6–12).

After selecting an edge $e = (c, v)$ node v will be queued n-times in Q, where n is a random integer between 0 and a program parameter $N_{\max} \in \mathbb{N}$. Random distribution of n is like edge selection based upon heuristic and pheromone values and uses Equation 5.1. But instead of using the heuristic value $\eta(e)$ the distribution for n is calculated from the node degree of node v in a precalculated minimum spanning tree. An example for a resulting spanning tree is shown in Figure 5.1.

Even with the final calculated spanning tree between turbines and substations the actual cabling is ambiguous. We use a breadth-first search to determinate the final cable types. Starting at every substation we visit all turbines and increase the minimum capacity on the path to this turbine by 1. Cables are selected by choosing the cheapest cable type that fulfills the minimum capacity for every edge. In this step the algorithm also verifies that every turbine has been connect to a substation with sufficient capacity. In the case that an invalid solution has been generated that pheromone update step will be skipped.

Algorithm 5.4: FindPath in Kruskal Approach

```
Input: Graph $G = (E, V)$, Pheromone Values $P_E$
Data: Current Node c, Choice List L
Output: $T \subseteq E$
1 repeat
2     L.clear()
      // Select all valid neighbors
3     forall $(c, v) \in E \setminus \{e \in E \mid T \bigcup \{e\}$ contains a circle $\}$ do
4         if $c$ or $v$ are not connected to a substation in $(V, T)$ then
5             L.insert($c$, $v$)
6     if L is not empty then
7         Select a random edge $e = \{c, v\}$ from L using $P_E$ as weight
8         $T \leftarrow T \bigcup \{e\}$
9 until L is empty
```

5.3 Kruskal Approach

Our second presented ant-based algorithm is similar to Kruskal's algorithm for minimum spanning trees.

As a basis we use again the common ACO algorithm 5.1 but with a modified FindPath function. In this approach we apply a different neighborhood definition when choosing the next edge of the solution. The neighborhood contains all edges from the graph that would not create a cycle together with the edges already selected from the ant. This way in each iteration any edge can be selected that connects two connectivity component in graph $G' = (V, T)$ where $T \subseteq E$ is the so far calculated solution set.

The modified ant algorithm for building a single solution is shown in Algorithm 5.2. In every step a list L of selectable edges is formed (line 2–4). As long as list L contains an element a random edge will be selected based on pheromone values (line 6) and added to solution (line 7).

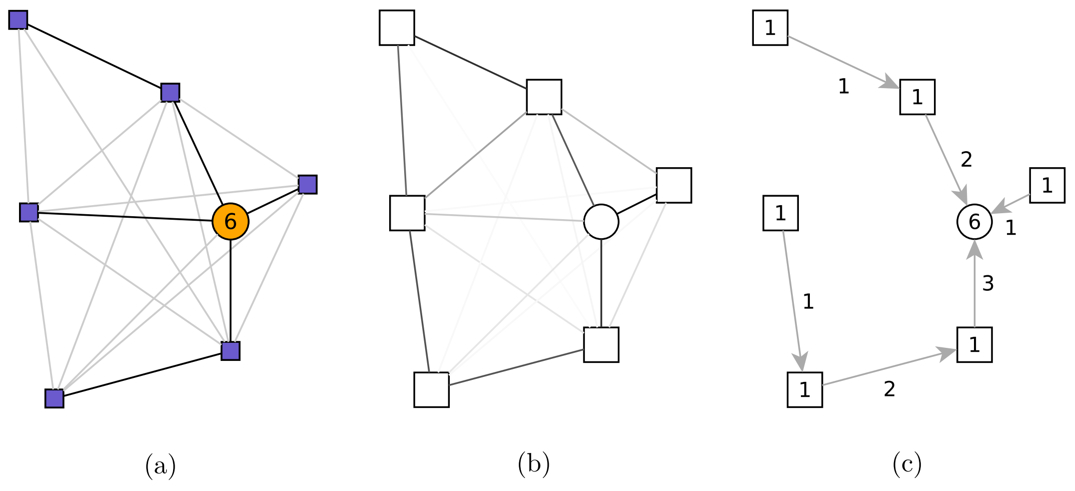


Figure 5.2: Visualization of the Kruskal approach. Figure (a) shows a spanning tree selected by a single ant. Figure (b) contains the pheromone values after 10 000 iterations. Edges with higher values are displayed darker. The last figure (c) shows the generated flow network of the best solution found.

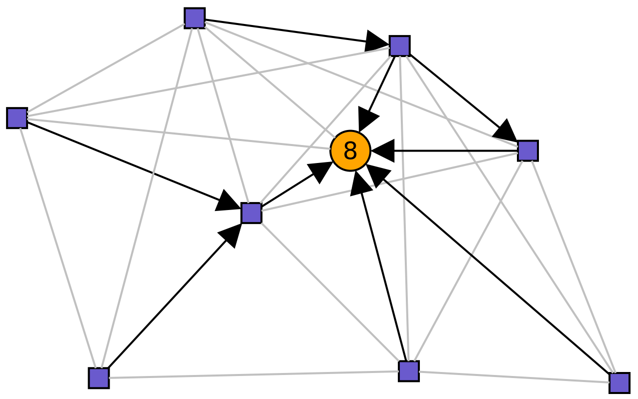


Figure 5.3: Flow generated in a single iteration of Algorithm 5.5. A separate path to the substation was build from each turbine.

In the first frame of Figure 5.2 a random spanning tree created by FindPath is displayed as an example. After 10 000 iterations the pheromone values are distributed on edges according to their value for low-cost solutions displayed in the second frame. The final solution is shown in the last frame. The flow is generated by a breadth-first search start at the substation nodes V_S.

5.4 Turbine Representation

We proposed another algorithm with a different representation for the cabling problem. Instead of using the substations as origin we place an ant on every turbine at the beginning. We call this turbine representation. From this start position each ant is choosing its own path to any substation. In this representation every turbine has its own pheromone data which will be used by the ant starting at that turbine.

In Algorithm 5.5 first all pheromone values will be initialized to 1 (see line 1). Then the main loop begins: In every iteration we begin by resetting our current flow to zero (line

Algorithm 5.5: Turbine Representation

```
Input: Graph $G = (V, E)$, cost function cost : $\mathcal{P}(E) \to \mathbb{R}$
Data: Best Solution S, Flow F, Pheromone Values $P_E(v)$ for each node $v$
Output: $T \subseteq E$
1  Initialize pheromone values
2  repeat
3      Reset flow F to zero
4      forall $v \in V$ do
5          F ← F + findPath($G$, $P_E(v)$)
6      C ← calculateSolution(F)
7      forall $v \in V$ do
8          $P_E(v)$ ← updatePheromones(C, $P_E(v)$)
9      if cost(C) < cost(S) then
10         S ← C
11 until time limit reached
12 return S
```

3). Then we visit every node and generate a path starting at the node to any substation with free capacity (line 4–5). The flow along every path will be increased by 1. After all turbines have been processed the required cable types and thus the final solution is being calculated (line 6). The pheromones values for every node will be updated as in the base ACO-algorithm 5.2 depending on the solution's costs (line 7–8).

5.5 Successor Approach

Algorithm 5.6: SUCCESSOR APPROACH: FINDPATH

```
   Input: Graph $G = (V, E)$, Pheromone Values $P_E$
   Data: Choice List L
   Output: Subset $T \subseteq E$
1  forall $v \in V$ do
2      L.clear()
3      forall $\{v, w\} \in E$ do
4          L.insert($v$, $w$)
5      if L is not empty then
6          Select a random edge $e$ from L using $P_E$ as weight
7          $T \leftarrow T \bigcup \{e\}$
```

At last we present another algorithm with a representation similar to the Kruskal approach in Algorithm 5.3. But instead of allow every edge to be selected in every step of the `FindPath` algorithm we loop through every node and select a single edge from all adjacent edges. We call this successor approach as for every node a successor node is select. The modified `FindPath` is shown in Algorithm 5.6. The algorithm selects every node in the graph (line 1) and forms a list of all adjacent edges (line 2–4). Then a random edge will be selected and added to the solution (line 6–7). Otherwise the pheromone update and calculation of the actual flow though breadth-first search uses the same algorithm as in the Kruskal approach 5.3.

5.6 Parameters and Improvements

In this section we explain different parameters and other modifications used to improve the result created by our algorithms.

Minimum Pheromone Values

The chance of selecting an edge is proportional to the associated pheromone value. Furthermore the pheromone value of an edge can only increase as long as the edge is actually selected. This behavior is useful under most circumstances as it increases the probability for a good solution being generated. But this can also lead to a problem while a single solution gains a much higher chance of being selected than all other solutions together. This situation happens when all nearby solutions of the current best solution are worse than the current solution and can leads to the algorithm creating the same result over and over again.

In order to address this issue we introduce a minimum threshold for the pheromone values. No value can be reduced below this value.

Recover

Another method of tackling the problem mentioned above is recovery from a bad local minimum by restarting the algorithm. After a specific number of iteration without any improvements to the best known we expect the algorithm to be trapped in a local minimum and restart the algorithm by resetting all edge weights to 1.

Ant per Iteration

A single iteration of the ant colony algorithm can result in solution with high costs especially at beginning when the pheromone values are still near its initial values. In order to speed up the algorithm we can run build multiple ant paths per iteration. Only the best ant result per iteration will be selected for updating the pheromone values.

Partitioning

As an advantage of ant colony optimization, we can use pheromone values of similar problems as initial values for other problem instances. This allows us to partition the graph in multiple parts and calculate solutions for these subproblems.

6. Evaluation

In this chapter we will evaluate our algorithm by comparing test result to existing algorithms. Primarily we compare our ant based approaches to an existing MILP-algorithm presented by Lehmann et al. [LRWW17]. We will explain our test environment and discuss the result of our algorithm for different parameters.

6.1 Test Parameters

As a basis for our investigated tests a large number of wind farm setups were generated using an existing implementation by Lehmann et al. [LRWW17]. The proposed algorithm generates a problem instances by placing turbines and substation at randomly chosen positions inside a defined circle or ecliptic shape. The idea of the algorithm is that a turbine or substation can only be placed if the distance to other turbines or substations is larger than a minimum distance. In the presented evaluation the minimum distance between turbines is 1 and between substations $\sqrt{t/s}$. In the case that the algorithm selects a position that does not fulfill the minimum distance condition, the whole generated farm will be scaled up by factor slightly greater than 1. This ensures that the generation process will always terminate.

These generated problem instances are defined by their generation parameters. A full list of all test-cases can be found in Table 6.1. The set of possible cable types is obtained from real cable costs as given by Berzan et al. [Ber05] and is listed in Table A.1.

Table 6.1: Number of turbines and substations listed for all problem instances

Instance	Number of Turbines	Number of Substations
1	22	1
2	57	1
3	22	2
4	27	2
5	148	8
6	125	9
7	287	11
8	431	31

We compare the results from our ant algorithms with result from the MILP approach as a benchmark. In our evaluation, the main aspect of our comparison was the costs of the cabling solutions from different algorithms, but we also looked into the structure of the solution to find improvements for the algorithms.

All tests were run on a computer with two Dual-Core Intel Xeon E5-430 processors. The program was written in C++ and compiled using the g++ compiler version 5.4.0 with compiler-flags `-std=c++14 -O2 -fopenmp -g -D_GLIBCXX_USE_CXX11_ABI=0`. We used the Open Graph Drawing Framework [CGJ$^+$] (Snapshot 2017-02-16) for graph representations. We also used different features of the Qt library version 5.5.1 [Com] and gurobi library version 6.5.2 [GO16] for the MILP algorithm. In order to create fair test results every run was started in single-threaded mode unless otherwise mentioned. Every test instance was running for 30 minutes when using ant algorithms and 60 minutes when using the MILP-algorithm.

We tested each of our four different ant approaches explained in Chapter 5. In our default configuration the pheromone decay per iteration was set to 0.0001. Also after 100 000 iteration without any improvement the algorithm will be restarted as explained in Section 5.6. Resulting data of our test runs are listed in Table 6.2. The table shows the cost of the best found solution for our four ant-based algorithms as well as the cost of the MILP solution. Later we improved the results significantly by using improved parameters.

6.2 Result for small and medium instances

The MILP algorithm is capable of finding sufficient solutions for small problem instances with less than 80 turbines. While solving a problem gurobi maintains an upper and lower bound. This upper bound is the value of the best known solution yet while the lower bound is the highest threshold the algorithm has verified that no better solution is possible. As gurobi is a heavily optimized library a good solution was usually found within a few seconds and the upper bound decreases accordingly. The program then needs a lot more time to increase the lower bound and calculating the actual optimal solution.

Although the MILP algorithm already provides very good results for small problems we want to test the performance of the ant algorithms for these smaller test-cases since the problem are still too large to be solved by a complete search algorithm. Our algorithms were able to generate solution with costs close to the MILP solution, although no algorithm was able to actually find a better solution without any parameter optimization. As shown in Table 6.2 we reached lower costs using MILP solution compared to any ant algorithm for all small instances 1–4. Our four ant approaches differ significantly in terms of the

Table 6.2: Cabling costs of the best solution after 30 minutes (ant approach) or 60 minutes (MILP approach) computation time.

Instance	MILP	TSP	Kruskal	Reversed	Successor
1	42800.7	46439.2	42947.8	63861.8	43038.1
2	113807.0	136936.8	119628.6	264868.4	123777.8
3	39114.2	39663.7	39114.2	41435.4	39626.9
4	49513.6	49725.7	49297.2	65160.6	49629.7
5	261030.0	–	304427.9	1455422.7	385058.1
6	211486.0	290930.5	232854.4	1008381.9	290646.8
7	503880.0	–	653741.3	3072073.2	838286.0
8	707550.0	–	1079445.5	3958828.0	–

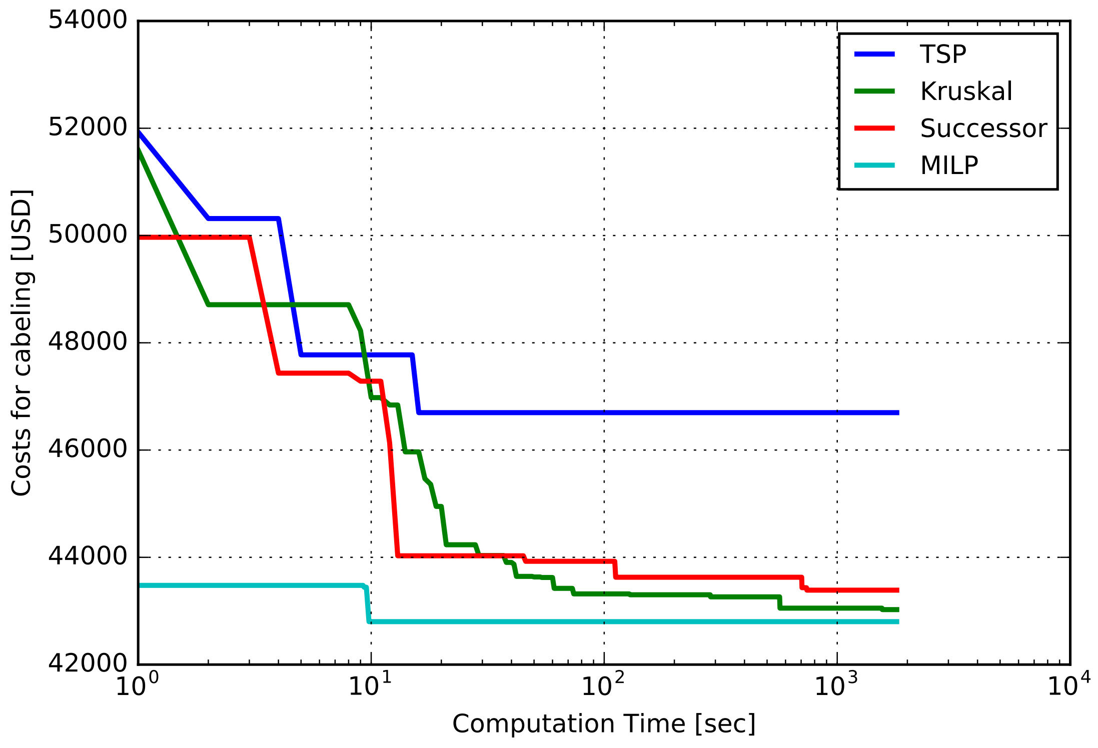


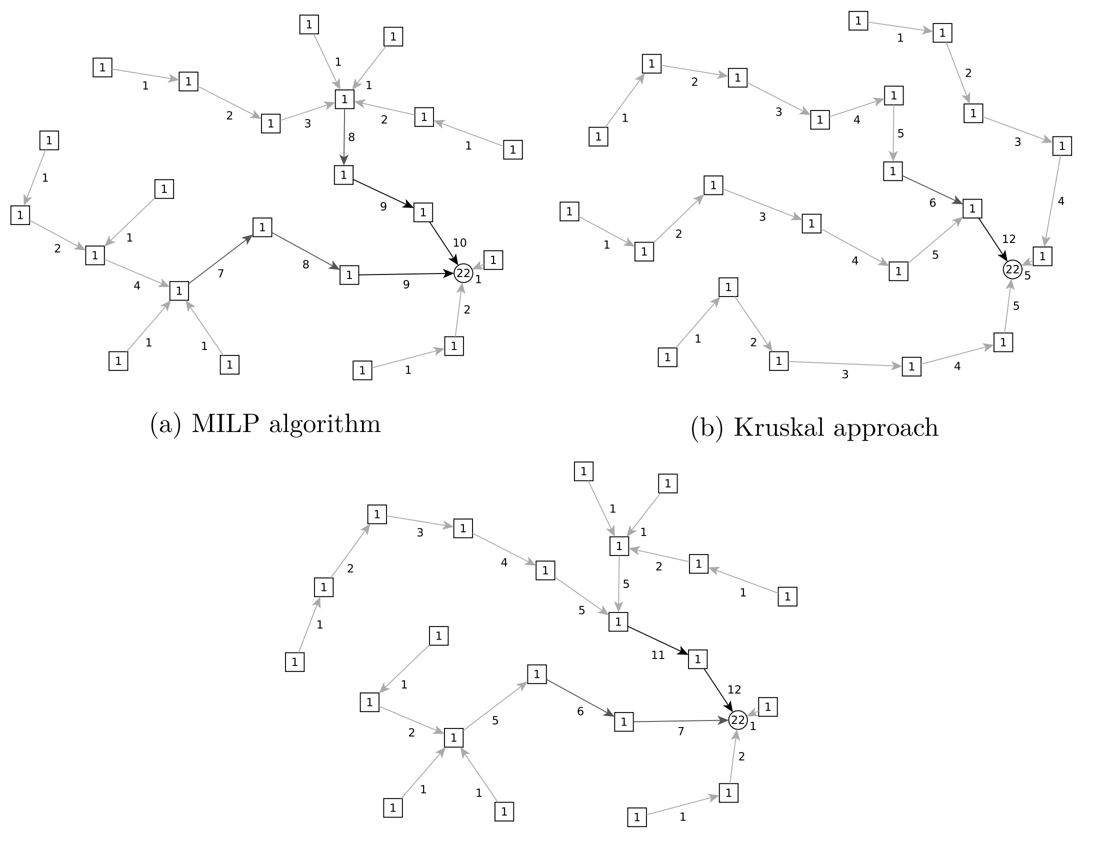
Figure 6.1: Costs of the best cabling solution found for the investigated algorithms as a function of computation time (logarithmic scale).

solution quality. The Kruskal and successor approach found solution which cost less than one percent more than the MILP solution while the TSP and turbine approaches were not able to generate with costs comparable to other algorithms. The results from these algorithms compared to the MILP solution are displayed in Figure 6.2. When looking at the plot in Figure 6.1 the final solution was found within 4 minutes. Then the algorithms were trapped in a local maximum and did not improve further.

As a consequence we tested different parameters for the two most successful ant algorithms. For small problem instance the results were notably improved by increasing the pheromone decay and at the same time decreasing the amount of iteration until the algorithm started all over again. When using these modified parameters the Kruskal and Successor algorithms were able to find a better solution than the MILP approach. Data of these improved results are shown in Table 6.3.

Table 6.3: Comparison of our algorithm results with improved parameters after 30 minutes computation time with the MILP approach after 60 minutes computation time.

Instance	MILP	TSP	Kruskal	Reversed	Successor
1	42800.7	46152.2	42779.8	61017.8	43184.8
2	113807.0	137728.6	119104.6	225155.1	122681.0
3	39114.2	39663.7	39114.2	43145.4	39799.1
4	49513.6	53662.2	49531.4	65252.0	50078.5
5	261030.0	–	288484.3	592255.6	385058.1
6	211486.0	290930.5	226239.5	410900.3	231334.6
7	503880.0	–	592159.7	1287240.5	838286.0
8	707550.0	–	827105.0	1588219.5	–



(a) MILP algorithm

(b) Kruskal approach

(c) successor approach

Figure 6.2: Side-by-side comparison of algorithm results for problem instance 1 with 22 turbines and 1 substation

The algorithm with turbine representation performed worse than all other algorithms. We assumed this algorithm has two major flaws. First the heuristic edge information is based purely on distance between the connected nodes and contains no information whether or not an edge is suitable in a path to a substation. On the other hand all paths from turbines are calculated separately from each other. This leads to results with many underutilized cables.

6.3 Large Instances

For larger problem instances the results of our four ant approaches were also quite different. In every presented ant algorithm an invalid solution may be generated except for the turbine representation. Since only valid solutions increase pheromone values generating other valid solutions becomes more likely. The TSP approach had problems with these large instances as it was the only algorithm which was unable to find any valid solution for some instance. Even in the case that valid solutions were generated the proportion of valid solutions to invalid ones was less than one percent. This problem also applies to the successor approach even though this algorithm is less affected.

As listed in Table 6.2 in our initial test the solution generated by the MILP Solver outperformed the solutions generated by our ant algorithms with default parameters for all large instances 5–8. Figure 6.3 shows the costs of our algorithms divided by the costs of the MILP solution for different instance sizes. The relative performance is notably worse for problem with more than 100 turbines. A typical test run for the algorithm is displayed in

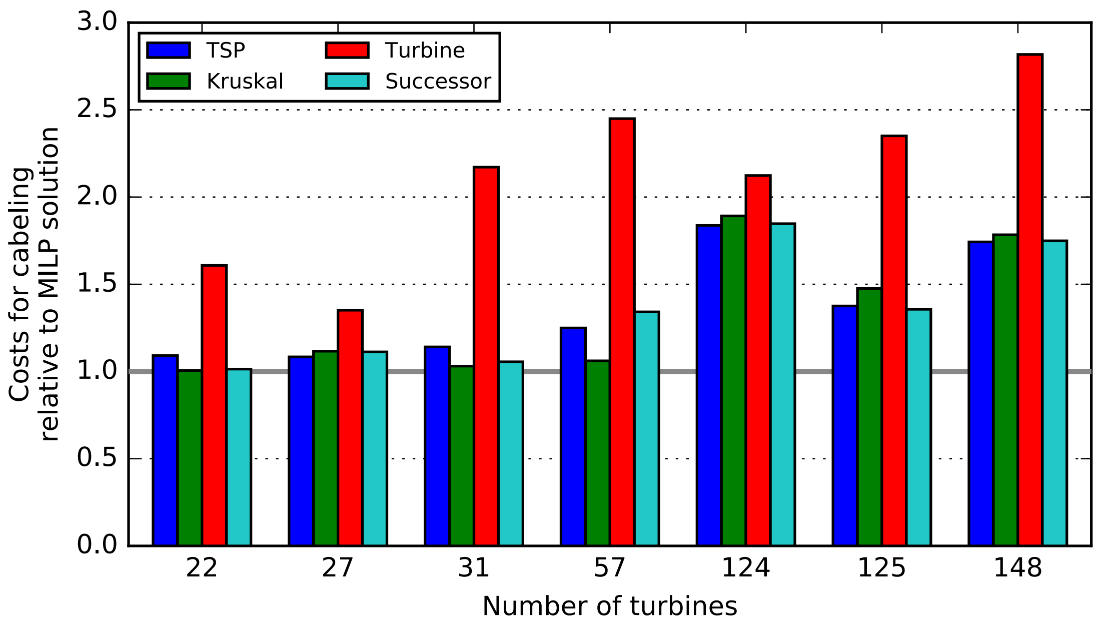


Figure 6.3: Cost comparison of our algorithm without optimized parameters relative to the MILP solution for different instance sizes

Figure 6.4. The algorithm is progressing slowly and even after 30 minutes of computation time the best solution steadily improved.

We realized that our default rate for pheromone-decay per iteration was set too low even for large instances. We rerun the test with our most promising Kruskal approach and different algorithm parameters. The results for different decay parameters are shown in Figure 6.5. As seen the results from the test improved noticeably for higher parameters, but were still worse than our benchmark MILP values. A solution generated by our best ant algorithm and the related MILP solution are shown in A.1 and A.2. When looking at the results we noticed that our algorithm had problem finding a good turbine to substation assignment. We assume that a reason for this is that in our representation the distance between the local optimum of a assignment and the local optimum of a different assignment is too large. This could be solved by a better representation of the problem or a two-level approach where the substation assignment is chosen first and the actual cabling is calculated separately.

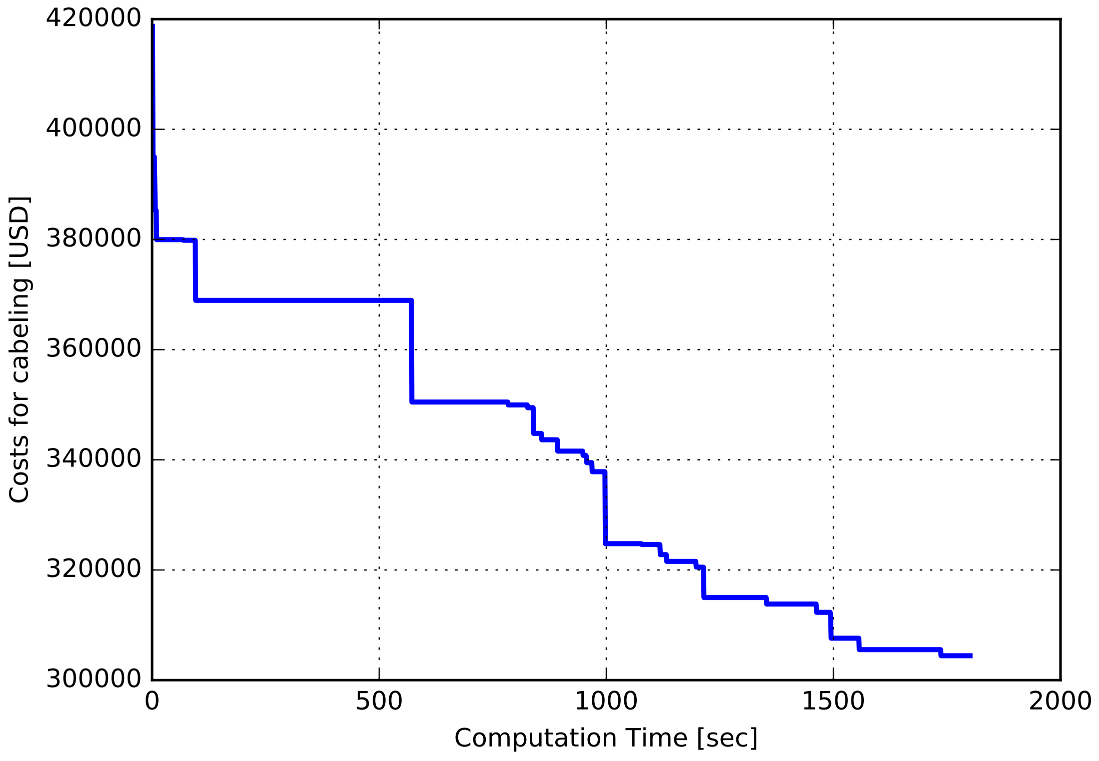


Figure 6.4: Costs of the best solution found while running ant algorithm with Kruskal approach with default parameters

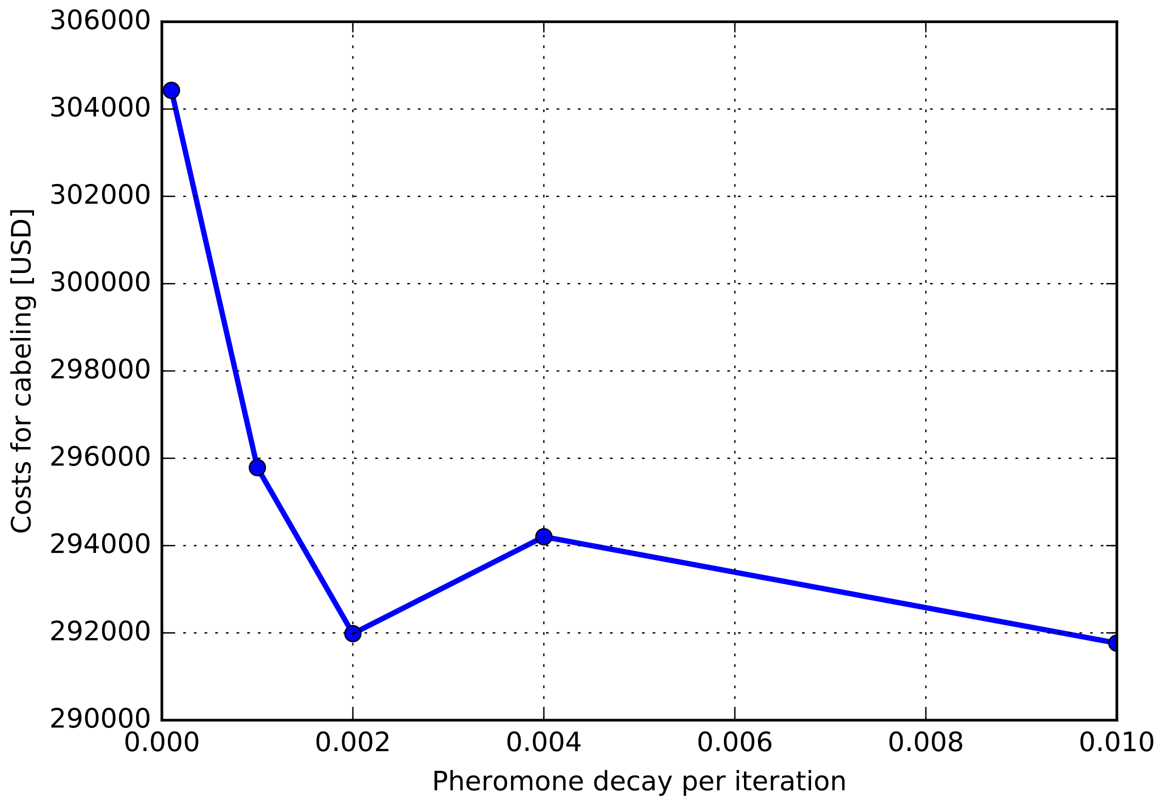


Figure 6.5: Results from the Kruskal approach from instance number 5 with different pheromone decay parameters

7. Conclusion

In this thesis we present four different algorithms for solving the wind farm cabling problem based on ant colony optimization. We run tests to compare these algorithms against each other and with a MILP. The latter is used as a baseline. The aim of the investigation was to minimize the cost for the wind farm cabling system. We simulated different parameter for the algorithm and sizes of wind farms. Our algorithms were compared regarding computation time to find an optimal solution to minimize costs for a wind farm cabling system. The comparison of the tests revealed that the most successful algorithm among the ant-algorithm is the one using a representation similar to Kruskal's algorithm for minimum spanning trees.

In our tests we recognized that the parameter for pheromone decay per iteration has to be set higher than our original estimation. With this optimized parameter our best algorithm performed better than the reference MILP results in some of the smaller instances. With the proposed optimization the Kruskal approach delivered better results than our MILP benchmark up to 31 turbines. The simulation results show that ant-colony optimization is a reasonable choice for these kinds of problems. However not all ant-approaches achieved successful results. Our algorithm with multiple ants starting at turbines performed worse than the MILP benchmark result. Perhaps this is because most generated solutions contain a lot of cables with free capacity.

None of our algorithms were capable of solving larger instances better than the reference solution. A major problem for our representations was that it allows the generation of invalid solutions which either overload a substation or miss a connection from a turbine to a substation. For small instances the number of invalid generated solutions decreased over time since the ant-based approaches only increases the weight of valid solution. This does not work for larger instance when invalid results outnumber valid results. Two of our algorithms even had problems to find valid solutions at all.

As our algorithm using the Kruskal approach only generates valid solutions it is not affected by the problem that increasing numbers of invalid solutions are correlating with the instance size. By analyzing the results from this algorithm we realized that the assignment from turbines to substations was not optimal.

We think a different representation for the ant approach could improve the solution quality significantly. First this representation should only generate valid instances like our Kruskal approach. And it should focus on finding good assignments as it has a big influence on the costs of large instances. Such a representation could also use a two-level approach where the substation assignment is selected separately from the actual cabling. Additionally partitioning of larger problems as described in Section 5.6 could improve the final results from the ant algorithms.

Bibliography

[AAN82] V. Aggarwal, Y.P. Aneja, and K.P.K. Nair. Minimal spanning tree subject to a side constraint. *Computers & Operations Research*, 9(4):287 – 296, 1982.

[Ber05] Constantin Berzan. Algorithms for cable network design on large-scale wind farms. 2005.

[CGJ+] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The Open Graph Drawing Framework (OGDF). `http://www.ogdf.net/`. Accessed: 2017-05-09.

[CM07] Chi-Bin Cheng and Chun-Pin Mao. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling*, 46(9–10):1225 – 1235, 2007.

[Com] The Qt Company. Qt framework. `https://www.qt.io`. Accessed: 2017-05-07.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, Feb 1996.

[ES12] Yunus Eroğlu and Serap Ulusam Seçkiner. Design of wind farm layout using ant colony algorithm. *Renewable Energy*, 44:53 – 62, 2012.

[Gav91] Bezalel Gavish. Topological design of telecommunication networks-local access design methods. *Annals of Operations Research*, 33(1):17–71, 1991.

[GO16] Inc. Gurobi Optimization. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2016.

[HSLL05] Vilson L. Dalle Molle Heitor S. Lopes and Carlos R. Erig Lima. An ant colony optimization system for the capacitated vehicle routing problem. *Proceedings of the XXVI Iberian Latin-American Congress on Computational Methods in Engineering*, pages 1–7, 2005.

[KXLS02] N. M. Kirby, Lie Xu, M. Luckett, and W. Siepmann. Hvdc transmission for large offshore wind farms. *Power Engineering Journal*, 16(3):135–141, 2002.

[LRWW17] Sebastian Lehmann, Ignaz Rutter, Dorothea Wagner, and Franziska Wegner. A simulated-annealing-based approach for wind farm cabling. In *Proceedings of the 8th ACM e-Energy International Conference on Future Energy Systems (ACM eEnergy'17)*. ACM Press, 2017. To appear.

[onl16a] Erneuerbare energien in zahlen. `http://www.umweltbundesamt.de/themen/klima-energie/erneuerbare-energien/erneuerbare-energien-in-zahlen#strom`, 2016. Accessed: 2017-03-26.

[onl16b] Iea - statistics search. `http://www.iea.org/statistics/statisticssearch/`, 2016. Accessed: 2017-05-07.

[onl16c] Where the wind blows – onshore vs. offshore wind energy. `http://futureofenergy.web.unc.edu/2016/04/06/where-the-wind-blows-onshore-vs-offshore-wind-energy/`, 2016. Accessed: 2017-03-26.

Appendix

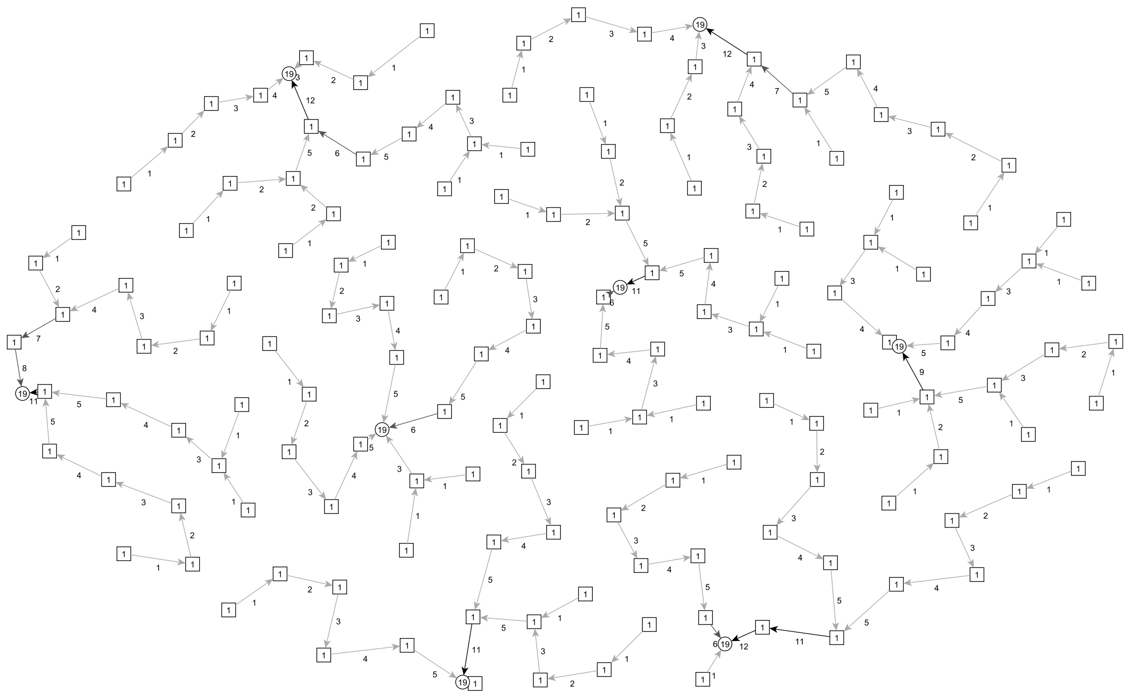


Figure A.1: Result for instance 5 generate by the MILP approach

Table A.1: Selectable cable types

Nr.	Costs	Capacity
1	20	5
2	25	8
3	27	12
4	41	15
5	50	16
6	54	24
7	82	30

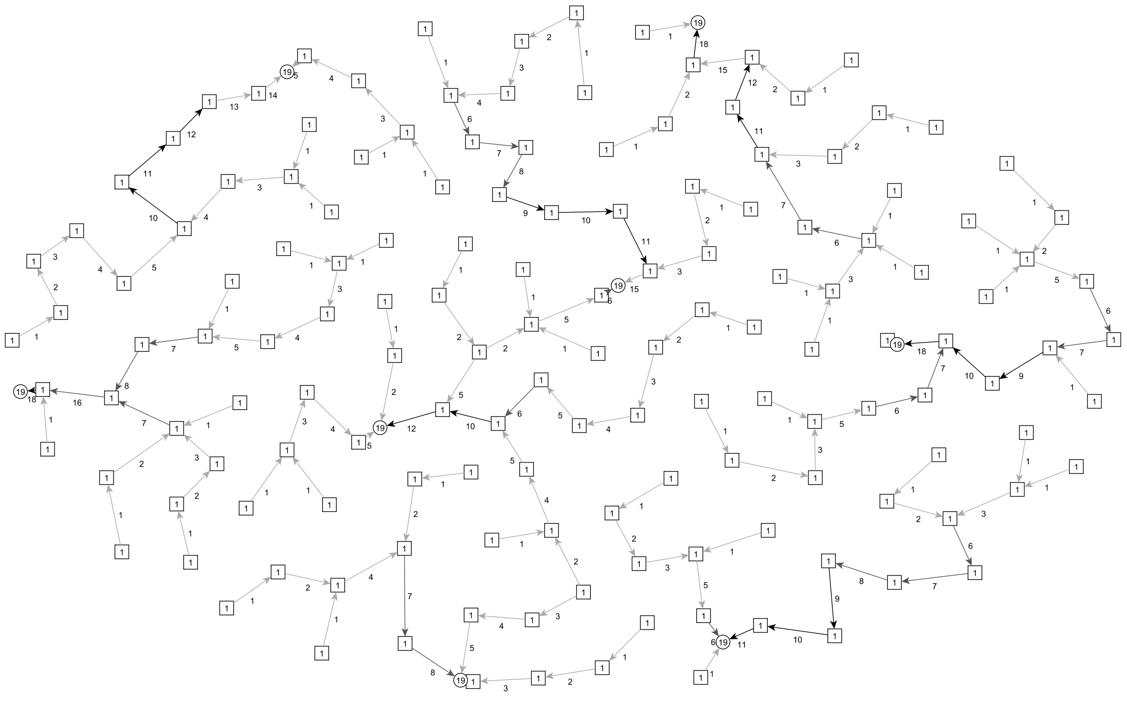


Figure A.2: Result for instance 5 generate by the Kruskal approach