Karlsruhe Institute of Technology

Entwicklung eines genetischen Algorithmus zur effizienten Verkabelung von Windfarmen

Bachelorarbeit
von

Ivo Baar

An der Fakultät für Informatik
Institut für Theoretische Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. Peter Sanders
Betreuende Mitarbeiter:	Franziska Wegner

Bearbeitungszeit: 25. Januar 2017 – 24. Mai 2017

Eigenständigkeitserklärung

Ich versichere hiermit, die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Die verwendeten Hilfsmittel und Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, 24. Mai 2017

Abstract

Windfarmen sind ein wichtiger Zweig der erneuerbaren Energien. Es werden zunehmend neue Projekte geplant. Einen großen Teil der Kosten einer Windfarm nimmt die Verkabelung der Turbinen ein. Da das Problem der Minimierung dieser Kosten NP-schwer ist, werden algorithmische Lösungen zur Approximation benötigt. In dieser Arbeit wird daher ein genetischer Algorithmus vorgestellt, der sich diesem Problem widmet. Dafür wird eine Repräsentation einer Verkabelung gewählt. Mehrere dieser Repräsentationen bilden dann eine Population, die durch genetische Operatoren iterativ verändert wird. Diese Operatoren selektieren, mutieren und rekombinieren diese repräsentierten Individuen der Population und minimieren dadurch deren Kosten. Viele Algorithmen, die für dieses Problem vorgestellt wurden, konzentrieren sich auf kleinere Windfarmen. Die Turbinenzahlen steigen jedoch stetig an und immer größere Projekte werden geplant. Daher wird in dieser Arbeit auf einem Benchmark Datenset getestet, das, auch für heutige Verhältnisse, große Windfarmen enthält. Als Referenz wird ein MILP-basierter Algorithmus herangezogen, mit dem die Ergebnisse verglichen werden. Vor allem für Windfarmen mit bis zu 200 Turbinen schneidet der vorgestellte Algorithmus, sogar mit geringerer Laufzeit, besser ab.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Windfarmen sind Systeme, die die kinetische Energie des Windes in elektrische Energie umwandeln. Dies geschieht durch eine oder einen Zusammenschluss mehrerer sogenannter Windturbinen, die diese Umwandlung vornehmen. Es existieren Windfarmen auf dem Festland, die sogenannten *Onshore Windfarmen*. Jedoch ist es auch möglich Windturbinen im Meer zu platzieren. Diese Windfarmen werden *Offshore Windfarmen* genannt. Ein Thema, mit dem sich die Planung von Offshore Windfarmen beschäftigen muss, ist der Transport des Stroms von der Windfarm zum Festland. Dazu werden häufig sogenannte Substations genutzt, die das Erzeugnis mehrerer Turbinen auf ein höheres Spannungsniveau transformieren. Dieser kann dann direkt in das Hochspannungsnetz an der Küste eingespeist werden. Als Beispiel ist hier in Abbildung 1.1 die *Burbo Bank Extension* Windfarm zu betrachten, die sich vor der Küste Englands befindet. Sie besitzt 32 Turbinen, welche an einer Substation angeschlossen sind. Von dort wird der Strom durch das Exportkabel – in der Abbildung rot – an das Festland geleitet.

Ein Problem dieser Offshore Windfarmen ist die Wahl der Verkabelung zwischen Turbinen und Substations. Diese sollte möglichst kostengünstig gewählt werden und trotzdem fähig sein, den erzeugten Strom mit minimalem Verlust weiterleiten zu können. Dabei muss auch entschieden werden, welche Turbinen an welche Substation angeschlossen werden. Dies stellt ein NP-schweres Problem dar. Dadurch ist es nicht möglich (falls $\mathsf{P} \neq \mathsf{NP}$), in

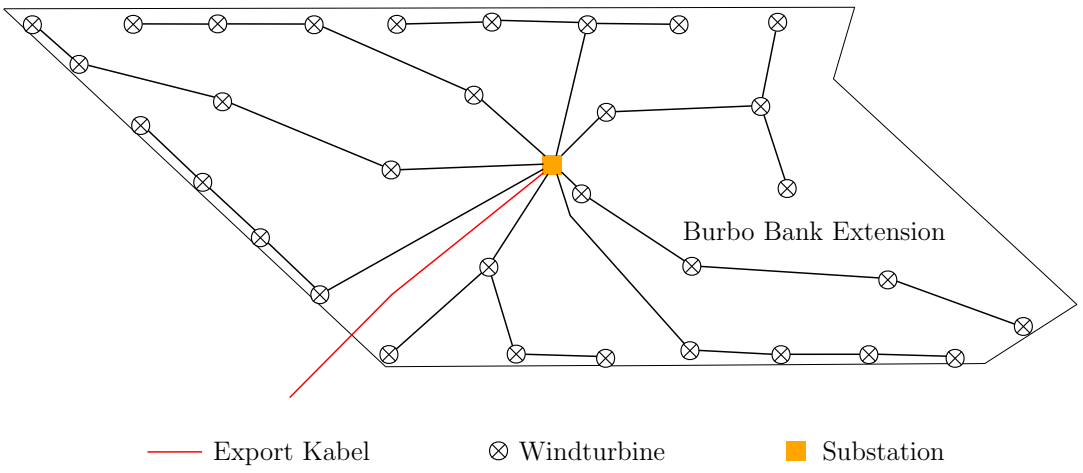


Abbildung 1.1: *Burbo Bank Extension* mit 32 Turbinen und einer Substation [bur].

annehmbarer Zeit die optimale Lösung für eine Verkabelung einer großen Windfarm zu berechnen. Daher ist es wichtig, durch Approximationen den Zeitaufwand zu verringern und dennoch eine Lösung zu finden, die nahe an den Kosten der optimalen Lösung liegt.

Ein möglicher Ansatz zur Berechnung solcher Approximationen ist ein genetischer Algorithmus, welcher aus der Klasse der evolutionären Algorithmen stammt. Diese lassen sich historisch gesehen in vier Klassen unterteilen: Genetische Algorithmen, Evolutionsstrategien, Genetische Programmierung und Evolutionäre Programmierung. Die genannten Klassen verschmelzen heutzutage aber zunehmend und einzelne Algorithmen des Gebiets lassen sich nicht mehr so einfach einer gewissen Art zuordnen. Genetische Algorithmen sind die verbreitetste Version der evolutionären Algorithmen und zeichnen sich in der Regel durch Repräsentation der Lösungen in Bitstrings aus. Dadurch sind standardisierte Mutationen und Rekombinationen möglich [ES03]. Wirklich bekannt wurden genetische Algorithmen vor allem durch die Arbeit von John Holland [Hol92], in der er die Grundsteine für diese Art Algorithmus legt.

Aufgrund der guten Resultate bei großen Lösungsräumen bieten sich genetische Algorithmen für das Windfarm Verkabelungsproblem an. Ein solcher genetischer Algorithmus wird in dieser Thesis vorgestellt. Er beruht auf einer Kodierung einer Lösung in ein sogenanntes Gen. Auf diesen Genen wird dann eine Simulation des Evolutionsvorgangs durchgeführt. Hier werden wiederholt Gene von Lösungen selektiert, gekreuzt und mutiert, um am Ende ein besseres Ergebnis zu erzeugen und sich so weiterzuentwickeln. Dieser Algorithmus wird dann mit anderen Optimierungsalgorithmen für das Problem verglichen und auf Konkurrenzfähigkeit überprüft.

Der Algorithmus ist vor allem für Windfarmen mit bis zu 200 Turbinen geeignet und liefert im Durchschnitt bessere Ergebnisse als der Referenzalgorithmus. Bei größeren Turbinenzahlen wird zunehmend mehr Zeit benötigt, um eine gute Lösung zu entwickeln. Jedoch scheint er im Vergleich zu anderen – vorallem genetischen – Algorithmen die für das Problem entworfen wurden, sehr performant zu sein.

In Kapitel 2 wird der aktuelle Stand der Forschung von verwandten Problemen und Algorithmen näher dargelegt. Grundlagen zu genetischen Algorithmen sind in Kapitel 3 zu finden. Zur genaueren Problemdefinition, wie sie in dieser Arbeit benutzt wird, ist Kapitel 4 zu betrachten. Die Hauptkapitel dieser Arbeit finden sich in den Kapiteln 5 und 6, in denen der entwickelte Algorithmus vorgestellt und evaluiert wird. Am Ende, in Kapitel 7, wird eine Zusammenfassung der Thesis und ein Ausblick auf mögliche Verbesserungen gegeben.

2. Stand der Forschung

Für das Windfarm Verkabelungsproblem existieren derzeit einige Ansätze, die durch bestimmte Kabelkonfigurationen versuchen, ein kostengünstiges Kabellayout zu berechnen. Konfigurationen sind bestimmte festgelegte Kantenstrukturen für *Circuits*. Ein Circuit sind alle Turbinen, die in einem zusammenhängenden Teilgraph an einer Substation angebracht sind. Mögliche Varianten für Konfigurationen sind beispielsweise eine Reihenverkabelung der Turbinen im Circuit. Eine weitere mögliche Konfiguration ist zum Beispiel ein *Sternen-Cluster*, das zuerst eine Turbine an die Substation und dann alle weiteren Turbinen im Circuit an diese Wurzel Turbine anschließt [ZCB09]. Diese Konfigurationen sind sehr einschränkend. Es ist wahrscheinlich, dass das Optimum des Lösungsraums nicht erreicht werden kann, da dieses in den meisten Fällen nicht genau die Strukturen aufweist, die eine solche Konfiguration mit sich bringt. Im Folgenden werden einige Ansätze für das Verkabelungsproblem vorgestellt und mit dem Ansatz dieser Bachelorthesis abgeglichen.

Der Forschungsbeitrag von Berzan et al. [CKMO] spielt in dieser Arbeit zum Verständnis eine Rolle. Zunächst wird das sogenannte *Windfarm Verkabelungsproblem* (engl. *Full Farm Problem*) in Teilprobleme aufgeteilt. Des Weiteren werden zwei Algorithmen vorgestellt, von denen einer annimmt, dass nur ein Kabeltyp benutzt wird, was eine Einschränkung des Problems ist. Hierdurch lässt sich jedoch das *Stromkreislauf Problem* (engl. *Circuit-Problem*) durch einen einfachen minimalen Spannbaum (engl. *Minimal Spanning Tree*; kurz *MST*) lösen und das Substation-Problem wird zum CMST Problem (Capacitated Minimal Spanning Tree Problem). Letzteres ist NP-schwer, jedoch gibt es gute Heuristiken [JR04], die polynomielle Laufzeit aufweisen. Der zweite Algorithmus ist ein Divide-and-Conquer Algorithmus, der das Problem rekursiv zerlegt und Schritt für Schritt berechnet. Ein weiterer interessanter Punkt, der in diesem Paper angesprochen wird, ist eine Modellierung von Terraineigenschaften, welche sich dann auf die Kosten der Verlegung der Kabel auswirken. Dadurch werden die Bedingungen für Windfarmen auf dem Festland realistischer dargestellt.

Basierend auf Dutta und Overbye [DOEE] wird in der vorliegenden Arbeit eine Zusammenstellung von Kabeltypen genutzt, die in der Realität benutzt werden und hier für die Evaluation als Referenzkabeltypen dienen werden. Siehe hierzu Tabelle 2.1.

M. Gonzales-Longatt et al. [GLWRT12] stellen einen genetischen Algorithmus zur Lösung des Verkabelungsproblems vor. Beim Bewerten der Lösungen spielen in dieser Arbeit nicht nur die Kosten der Verkabelung selbst, sondern auch die Kosten von anderen Elementen der Windfarm, beispielsweise Transformatoren, eine Rolle.

Kabeltyp	Kapazität	Kosten pro Längeneinheit
1	5	20
2	8	25
3	12	27
4	15	41

Tabelle 2.1: Kabeltypen, die bei der Evaluation zum Berechnen von Instanzen gewählt wurde. Die Kapazität bezieht sich auf die Anzahl an Windturbinen, von denen der Strom durch das gegebene Kabel transportiert werden muss.

Ein weiterer genetischer Algorithmus wird von Z. Chen et al. [ZCH06] vorgestellt. Hier wird zur Bewertung einer Lösung auch die Zuverlässigkeit der einzelnen Elemente in Betracht gezogen. Dazu gehören beispielsweise auch Turbinen und Transformatoren, die in der Windfarm benutzt werden. Darüber hinaus fällt auf, dass die Anfangspopulation des genetischen Algorithmus nicht zufällig generiert wird, sondern Individuen von jedem wichtigen Gen-Aspekt in der Population enthalten sind. Diese verschiedenen Eigenschaften der Gene sind je nach Problemdefinition und Repräsentation unterschiedlich wählbar. Vermutlich wird die Konvergenz der Population ebenfalls beschleunigt, sofern mit einer gültigen Startpopulation angefangen wird, da die Laufzeit zur Suche nach gültigen Lösungen entfällt. Des Weiteren werden einige Abbruchkriterien benutzt, von denen manche in Kapitel 3 näher erläutert werden. Getestet wurden mehrere Varianten des Algorithmus, die sich vor allem in den Selektionsoperatoren unterscheiden. Die Windfarm, die als Grundlage für einen Test dient, ist die *Burbo Bank Windfarm* mit insgesamt 25 Windturbinen.

F. Blaabjerg et al. [ZCB09] haben einen Ansatz ähnlich dem aus dem vorher erwähnten Paper vorgestellt. In diesem werden drei Kabelkonfigurationen für Circuits genannt. Zum einen wird eine Reihe von hintereinander geschalteten Turbinen betrachtet. Die zweite Konfiguration ist eine Ringverkabelung, bei der ein Kabelkreis betrachtet wird und die letzte Konfiguration ist ein *Sternen-Cluster*. Die Windfarm, an der der Algorithmus getestet wird, ist eine Instanz mit 60 Turbinen und einer Entfernung von 6 Kilometern bis zur Küste. Es wurden 4,8 Stunden zur Berechnung eines Ergebnisses auf einem 1,8 GHz Prozessor benötigt.

Die genetischen Algorithmen, die im Zusammenhang mit der Windfarmverkabelung entwickelt wurden, konzentrieren sich hauptsächlich auf Konfigurationen der Circuits. Dies sind vorgegebene Circuit Layouts, die sich nicht verändern lassen. Die Algorithmen von Berzan et al. [CKMO] sind in der Praxis nicht funktional genug, da die Laufzeit mit der Zahl der Turbinen zu exponentiell ansteigt. Für die Berechnung einer Windfarm mit 14 Turbinen wurde die optimale Lösung in weniger als einer Minute gefunden. Das ist jedoch eine sehr kleine Anzahl von Turbinen, im Vergleich zu aktuellen Projekten. Hier wäre beispielsweise die *Hornsea Project Two Windfarm* [Hal16] [HP2] zu nennen, die mit bis zu 300 Turbinen an der Yorkshire Küste geplant ist. Für diese Windfarm würde der entwickelte MST-basierte Algorithmus eine inakzeptable Laufzeit aufweisen. Daher wird in dieser Bachelorarbeit ein genetischer Algorithmus vorgestellt, der nicht durch die Wahl von Konfigurationen eingeschränkt wird. So können Lösungsmöglichkeiten in Betracht gezogen werden, die anderweitig nicht möglich gewesen wären.

3. Grundlagen

In diesem Kapitel werden Grundlagen dargelegt, die benötigt werden, um Konzepte und Denkansätze in dieser Arbeit zu verstehen. Fortgeschrittene Leser können dieses Kapitel nach Wunsch überspringen.

Zuerst wird die grundlegende Funktionsweise genetischer Algorithmen erläutert, danach wird auf die verschiedenen Aspekte dieser eingegangen. Hierzu werden die verschiedenen Operatoren erklärt, und es wird verdeutlicht, welche Funktionen diese im Konzept eines genetischen Algorithmus erfüllen.

Ein genetischer Algorithmus ist ein Konzept zur Lösung von Optimierungsproblemen, bei denen es aus Zeit- und Kostengründen keinen Sinn macht, eine optimale Lösung zu suchen. Daher werden sie vor allem bei Problemen eingesetzt, die sehr viele mögliche Lösungen haben und daher nach einer Approximation an ein Optimum gesucht wird. Die Grundidee ist es, einen Evolutionsvorgang – mit Lösungen als Individuen – zu simulieren, sodass diese sich nach einigen Generationen zu guten Lösungen weiterentwickeln, die die Voraussetzungen erfüllen. Ein Beispiel einer solchen Voraussetzung ist eine zufriedenstellende Lösungsgüte für das gegebene Problem. Diese kann von Problem zu Problem variieren, im Fall des Windfarm Verkabelungsproblems wäre dies zum Beispiel eine gewisse Kostengrenze, die es zu unterschreiten gilt. Eine Ansammlung von Lösungen zu einem bestimmten Zeitpunkt wird *Population* oder auch *Generation* genannt. Daher wird, wie in der Biologie, zuerst eine Repräsentation der Lösungen in sogenannten *Genen* benötigt, um diese danach kreuzen und mutieren zu können. Diese Repräsentation ist in der Regel eine einfache Datenstruktur, um schnell und effizient Operationen an ihr durchführen zu können. In Abbildung 3.1 ist die grundlegende Struktur eines genetischen Algorithmus zu sehen. Bei dieser wird zuerst die Population initialisiert, danach bewertet und dann mit den genetischen Operatoren verändert. Dies geschieht so lange, bis ein Abbruchkriterium erreicht wird und die Lösung ausgegeben wird. Die einzelnen Schritte werden in den folgenden Abschnitten etwas ausführlicher dargelegt. Grundlagen zur Funktionsweise von genetischen Algorithmen sind in [ES03] zu finden.

Repräsentation von Lösungen

Beim Entwurf der Repräsentation von Lösungen sollte bei einem genetischen Algorithmus sorgfältig vorgegangen werden. In der Realität existieren oft zu viele Faktoren, um alle mit in die Problemstellung einfließen zu lassen. Oftmals müssen einige Eigenschaften der Probleminstanz eingeschränkt werden, um einen Rahmen zu erhalten, der es zulässt, Lösungen

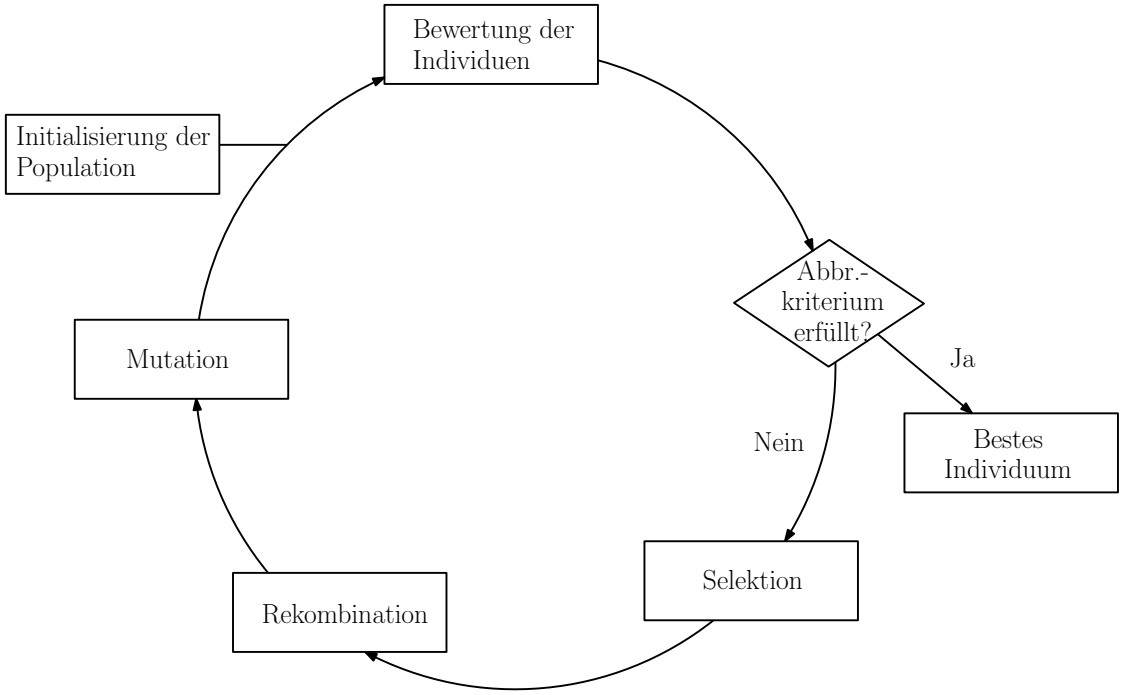


Abbildung 3.1: Aufbau eines genetischen Algorithmus. Anfangs wird die Population initialisiert, dann evaluiert und danach durch die genetischen Operatoren verändert. Dies wird so lange iteriert, bis ein Abbruchkriterium erfüllt wurde. Danach wird die beste gefundene Lösung ausgegeben.

nach den Bedürfnissen zu evaluieren. Der erste Schritt ist es daher, eine Repräsentation zu finden, die das gewünschte Problem genau genug spezifiziert und im Computer effizient bearbeitet werden kann. Lösungsinstanzen in der realen Welt werden auch *Phenotypen* genannt, wohingegen Lösungen, die als Repräsentation im Computer vorliegen, als *Genotypen* bezeichnet werden – angelehnt an die Begrifflichkeit der Biologie. Der genetische Algorithmus agiert nur auf den Genotypen und am Ende wird der beste Genotyp als Phenotyp interpretiert, um eine gute Lösung zu erhalten. An dieser Stelle ist zu beachten, dass um den besten Phenotyp als Lösung zu erhalten, für diesen auch ein Genotyp als Repräsentation existieren muss [ES03].

Fitnessfunktion

Um feststellen zu können, wie hoch der Gütefaktor einer Lösung ist, wird die *Fitnessfunktion* eingeführt. Diese bewertet eine Lösung und weist ihr einen Wert zu, damit mehrere Lösungen miteinander verglichen werden können. Die Fitnessfunktion ist ein zentraler Punkt jedes genetischen Algorithmus, da dieser nur funktioniert, sofern Lösungen richtig bewertet werden. Somit kann die Fitnessfunktion als zu optimierende Funktion betrachtet werden. Sie hat jedoch nicht nur eine bewertende Wirkung. In vielen Fällen wird die Fitnessfunktion auch verwendet, um Lösungen zu bestrafen, die problemspezifische Gültigkeitsanforderungen nicht erfüllen. Daher wird die ungültige Lösung durch eine massive Verschlechterung der Fitness im nächsten Selektionsprozess mit hoher Wahrscheinlichkeit verworfen.

Selektion

Die Fitness einer Lösung wird beispielsweise für die Entscheidung genutzt, welche Lösungen für die Kreuzung in Betracht gezogen werden. Der *Selektionsoperator* übernimmt diese Aufgabe und entscheidet wiederholt in verschiedenen Generationen von Lösungen, welche

Lösungen als Eltern in Frage kommen. Hier ist anzumerken, dass dies nicht unbedingt die besten Individuen sein müssen. Sofern es Lösungen mit schlechterer Fitness erlaubt wird, ein Elternteil zu werden, wird die Diversität der Population leichter aufrechterhalten. Dies folgt aus dem Fakt, dass Versionen der besten Lösungen oft mehrmals in der Population enthalten sind. Daher wird, sofern nur die besten Individuen gewählt werden, eine große Zahl gleicher Individuen gewählt. Unter Diversität wird hier die Anzahl unterschiedlicher Lösungen in der Population betrachtet. Zudem wird nach der Rekombination der Eltern und Bewertung der Kinder mittels Selektion entschieden, welche Lösungen „überleben“ und in die neue Generation übernommen werden. Somit wird garantiert, dass sich die Lösungspopulation im Laufe der Zeit verbessert, da Lösungen mit schlechter Fitness nicht weiter übernommen werden. Es gibt viele verschiedene Ansätze, die Selektion in einem genetischen Algorithmus umzusetzen. Von diesen werden einige im Kapitel 5 vorgestellt.

Rekombination / Crossover

Wie zuvor erwähnt, wird ein *Rekombinations-* oder *Crossover-Operator* benötigt, um den Evolutionsprozess zu simulieren. Dieser übernimmt – im besten Fall – gute Teile von zwei Lösungen und versucht, diese in einer oder mehreren Kindlösungen zu vereinen. Ziel des Crossovers ist es, Lösungen zu erzeugen, die eine bessere Fitness aufweisen als beide Elternteile. Somit steigt die Fitness der Population im Laufe der Zeit stetig an. Bei einigen Problemen sind Crossover-Operatoren recht einfach zu entwerfen, vor allem wenn einfach ersichtlich ist, welcher Teil einer Lösung eine hohe Güte aufweist. Als Beispiel liegen die Lösungen oftmals in Bitstring Repräsentation vor. Eine einfache Lösung für einen Crossover ist für ein solches Problem in Abbildung 3.2 zu sehen. Es wird zuerst ein zufälliger Index gewählt, an dem beide Elternlösungen aufgetrennt werden. Danach wird für die Erzeugung der Kindelemente jeweils einmal der erste Teil jeder Elternlösung mit dem zweiten Teil des entsprechend anderen Elternteils verknüpft [Wei15].

Abbildung 3.2: Veranschaulichung eines Crossover-Operators für Bitstrings. Index, an dem die Trennung stattfindet, ist in den Elternlösungen rot markiert.

Mutation

Um zufällige Anpassung von Individuen eines Evolutionsvorgangs zu simulieren, wird auf den *Mutationsoperator* zurückgegriffen. Dieser nimmt zufällig kleine Änderungen an Lösungen vor. Dieser Operator hat in erster Linie den Zweck, die Diversität von Lösungen in der Population aufrechtzuerhalten [Wei15]. Sofern nur Crossovers benutzt werden würden, würden sich die Lösungen nach gewisser Zeit zu stark ähneln oder sogar gleich sein, was dazu führt, dass keine besseren Lösungen gefunden werden können. Dies folgt dadurch, dass sich die Population in einem lokalen Optimum festläuft und durch den Crossover mit ähnlichen Individuen den Bereich des lokalen Optimums nicht verlassen kann. Mutationen verhindern dies, indem die Lösungen zu anderen aus einem anderen Abschnitt des Lösungsraums mutiert werden. Dadurch besteht die Möglichkeit, trotz fortgeschrittener Population in einem lokalen Optimum, dieses zu verlassen und bessere Lösungen zu finden. Mutationen geschehen zufällig, das heißt, dass nicht jede Lösung in jeder Iteration mutiert wird.

Erstellen einer Anfangspopulation

Zur Bestimmung der Anfangspopulation gibt es verschiedene Ansätze. Einer wäre, die initiale Population rein zufällig zu erstellen. Dies bewirkt, dass mit einer hohen Wahrscheinlichkeit eine hohe Diversität der Population vorliegt. Daher wird der Lösungsraum gut abgedeckt und es ist wahrscheinlicher, dass das globale Optimum gefunden wird, anstatt in einem Lokalen festzustecken. Eine weitere Möglichkeit ist, kontrolliert Startindividuen zu erzeugen, die bestimmte gewünschte Eigenschaften haben (Siehe [ZCH06, ZCB09]). Dadurch kann die Konvergenz der Population in gewisser Weise zu einer gewünschten Lösung geleitet werden. Jedoch ist es oft interessanter, zufällige Lösungen zu untersuchen, da so auch Lösungen erreicht werden können, die vorher eventuell nicht in Betracht gezogen wurden.

Abbruchkriterien

Abbruchkriterien geben an, ab wann die Berechnung abgeschlossen oder zumindest für das momentan betrachtete lokale Optimum abgeschlossen ist. Häufig werden mehrere dieser Kriterien umgesetzt. Meist ist eines der Abbruchkriterien eine maximale Rechenzeit. Diese kann beim Start des Programms festgelegt werden und garantiert, dass innerhalb einer gewissen Zeit ein Ergebnis geliefert wird. Ähnlich dazu ist der Abbruch nach einer gewissen Anzahl an Iterationen des genetischen Vorgangs. Dies ist ebenso ein sehr allgemeines Kriterium, welches oftmals genutzt wird. Mögliche weitere Kriterien wären zum Beispiel der Abbruch nach Erreichen einer gewissen Fitness. Dies ist ein Kriterium, welches jedoch nur eingesetzt werden sollte, sofern die optimale Fitness oder zumindest eine gute Referenzfitness bekannt ist. So kann der Parameter an der Fitness der Referenzlösung orientiert werden. Sofern keine Referenzfitness zur Verfügung steht, was in den meisten Fällen zutrifft, sollte in der Regel vermieden werden durch Fitness abbrechen. Dies folgt daraus, dass der Population durch zu frühen Abbruch die Chance genommen wird, ein besseres lokales Optimum zu finden – sofern ein Solches existiert. Es existieren zudem Abbruchkriterien, die das Programm abbrechen sollen, sofern die Population in einem lokalen Optimum feststeckt. Referenzwerte hierfür sind beispielsweise die Diversität der Lösung oder die fehlende Verbesserung der bisher besten Lösung [ZCH06]. Dies kann beispielsweise umgesetzt werden, indem betrachtet wird, wie lange die beste Lösung bereits als beste Lösung eingestuft wurde. Dann wird dementsprechend abgebrochen, sofern dies zu lange oder nach zu vielen Generationen der Fall war. Diversität kann bei einigen Problemen anhand der Ähnlichkeit der Lösungen in der Population erkannt werden. Dies ist jedoch von Problem zu Problem unterschiedlich zu handhaben.

4. Problemdefinition

Zunächst wird auf die Elemente einer Windfarm in unserem Modell eingegangen. Hier werden Windturbinen, Kabel unterschiedlicher Typen und sogenannte Substations – auch Collector Points genannt – benutzt. Windturbinen sind die stromerzeugenden Entitäten der Windfarm, wohingegen Substations als Sammel- und Weiterleitungspunkt betrachtet werden können. Kabel sind das Medium, durch das der Strom von den Turbinen zu den Substations geleitet wird. Eine Substation und alle Turbinen, die an dieser Substation angeschlossen sind, werden *Kollektorsystem* genannt. Dieses Kollektorsystem kann dann wieder in kleinere Teilblöcke aufgetrennt werden, die sogennanten Circuits. Ein weiteres Element, das in Windfarmen zu finden ist, ist der sogenannte *Netzpunkt* (engl. *Grid Point*, siehe Abbildung 4.1). Dieser ist der Anbindungspunkt an das Stromnetz, der vor allem bei Offshore Windfarmen sehr weit von den eigentlichen Turbinen entfernt sein kann. In dieser Thesis wird dieser jedoch nicht weiter beachtet, da in der Praxis meist jede Substation direkt an den Grid Point angeschlossen wird und damit nur Fixkosten zu den Gesamtkosten addiert werden. Das Hauptaugenmerk liegt also auf der Minimierung der Kosten der Verkabelung zwischen Substations und Windturbinen.

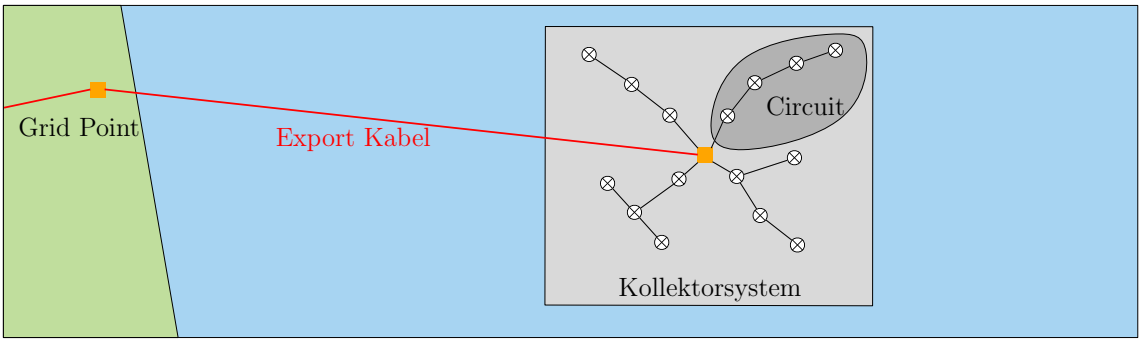


Abbildung 4.1: Darstellung einer Windfarm mit einer Substation und Anbindung an den Grid Point auf dem Festland. Als Kollektorsystem wird eine Substation und alle an dieser angeschlossenen Turbinen bezeichnet.

Seien im Folgenden die Menge V_T die Menge der Windturbinen und V_S die Menge der Substations. Als Menge aller Knoten wird $V = V_S \cup V_T$ benutzt. Als Kanten unseres Graphen $G = (V, E)$ werden alle Verknüpfungen zwischen Windturbinen untereinander und Windturbinen mit Substations in Betracht gezogen. Die Kanten E werden ungerichtet betrachtet und diese ungerichtete Kantenmenge wird als $\underline{E}$ geschrieben. Damit ist gemeint,

dass für $\underline{(u,v)} \in \underline{E}$ gilt :$\underline{(u,v)} = \underline{(v,u)} = \{u,v\}$. Es muss mindestens ein Element aus der Menge der Windturbinen sein, da die Verkabelung zwischen den Substations hier nicht von Belang ist (siehe vorheriger Absatz). Jeder Kante wird durch die Funktion $\kappa\colon \underline{E} \to K$ ein Kabeltyp zugewiesen, wobei K die Menge der möglichen Kabeltypen darstellt. Mit den Funktionen $\mathrm{cap}\colon K \to \mathbb{R}_{\geq 0}$ und $\mathrm{c_{cab}}\colon K \to \mathbb{R}_{\geq 0}$ kann jedem Kabeltyp eine Kapazität (Anzahl der Turbinen, deren Stromproduktion weitergeleitet werden kann) und Kosten pro Längeneinheit zugewiesen werden. Um die Kapazitätsfunktion so betrachten zu können, muss angenommen werden, dass in einer Windfarm nur ein Windturbinentyp vorliegt. Natürlich könnte diese Funktion auch auf unterschiedliche Turbinentypen angepasst werden, um die Flussbedingung zu erfüllen. Diese besagt, dass für jede Kante $\underline{e} \in \underline{E}$ gelten muss, dass die Ungleichung $|V_{\underline{e}}| \leq \mathrm{cap}(\kappa(\underline{e}))$ erfüllt ist, wobei $V_{\underline{e}}$ die Turbinen im Unterbaum enthält, der an der Kante $\underline{e}$ angeknüpft ist. Ein weitere Bedingung die erfüllt sein muss, ist die Kapazitätsbedingung. Für jede Substation existiert eine obere Schranke, wieviele Turbinen an dieser angeschlossen sein dürfen. Die Kapazitätsbedingung besagt, dass bei keiner Substation mehr Turbinen als diese Schranke angebunden sein dürfen. Zur Berechnung der Kosten eines Kabels fehlt noch eine Distanzfunktion $\mathrm{len}\colon \underline{E} \to \mathbb{R}_{\geq 0}$, die einer gegebenen Kante ihre euklidische Distanz zuordnet. Wenn im Folgenden von einer Verkabelung gesprochen wird, dann ist eine Menge E und eine Funktion κ gemeint, mit der die Kosten eindeutig definiert sind. Eine Lösung, die zudem die Fluss- und die Kapazitätsbedingung erfüllt, heißt gültig oder valide. Im Endeffekt ist das Ziel dann, folgende Kostenfunktion zu minimieren:

$$\sum_{\underline{e} \in \underline{E}} \mathrm{c_{cab}}(\kappa(\underline{e})) \cdot \mathrm{len}(\underline{e})$$

Nun können drei verschiedene Problemschichten betrachtet werden, welche zum Verständnis des Problems beitragen.

Circuit-Problem

Hier ist die unterste Problemschicht das sogenannte Circuit-Problem. Dabei geht es darum, für eine Substation und gegebene Windturbinen eine möglichst kostengünstige und gültige Verkabelung zu finden, die alle gegebenen Turbinen als Baum an die Substation anbindet. Dieses Problem wäre, sofern nur ein Kabeltyp betrachtet wird, ein minimales Spannbaumproblem. Da aber verschiedene Kabeltypen in Betracht gezogen werden, die jeweils verschiedene Kapazitäten und verschiedene Kosten haben, ist schon dieses Teilproblem NP-schwer [LRWW17].

Substation-Problem

Als nächste Problemschicht kann das Substation-Problem vorgestellt werden. Hier geht es darum, eine Substation und gegebene Windturbinen in die günstigsten Circuits aufzuteilen und für diese das Circuit-Problem zu lösen.

Full-Farm-Problem

In der letzten Schicht, die dann das sogenannte Full-Farm-Problem löst, werden alle Substations und Windturbinen einer Windfarm betrachtet. Zunächst müssen den verschiedenen Substations daher Windturbinen zugeordnet werden, dann muss für diese das Substation-Problem gelöst werden.

Flexible-Full-Farm-Problem

Eine weitere mögliche Problemschicht, die zur Vollständigkeit auch erwähnt werden sollte, ist das Flexible-Full-Farm-Problem. Dieses Problem optimiert genau nach dem gleichen Prinzip wie das Full Farm Problem, jedoch sind nun die Positionen der Substations nicht als Vorbedingung festgelegt, sondern können aus einer endlichen Menge an Standpunkten gewählt werden.

5. Methodik

In diesem Kapitel werden die Konzepte vorgestellt, die im Algorithmus umgesetzt wurden und Gründe genannt, warum diese gewählt wurden. Zuerst wird am Pseudocode 5.1 die umgesetzte Vorgehensweise verdeutlicht. Genaue Erklärungen der einzelnen Operatoren werden in den folgenden Unterabschnitten dargelegt.

Für die Initialisierung – im Algorithmus Zeile 1 bis 3 – werden anfangs nur zufällige Startpopulationen betrachtet. Dennoch müssen einige problemspezifische Eigenschaften gegeben sein. Zuerst ist zu nennen, dass der Graph zusammenhängend sein muss, da ansonsten die Lösung ungültig wäre. Schließlich wird nach einer Windfarm gesucht, die den Strom von allen beteiligten Turbinen weiterleitet. Am Ende wird die Lösung aber ohne Verkabelung zwischen den Substations ausgegeben, da diese in der Regel direkt von den einzelnen Substations zum Grid Point verläuft. Aufgrund der gewählten Repräsentation ist es aber nützlich, diese mitzubetrachten, um mittels Breitensuche leicht überprüfen zu können, ob die Lösung valide ist oder nicht. Daher ist eine weitere, nicht offentsichtliche Einschränkung, dass Substations nicht über diverse Turbinen verbunden sein dürfen, da diese Turbinen sonst an zwei Substations angeschlossen wären. Dies ist nach der Problemdefinition nicht erwünscht, da eine Turbine eindeutig einer Substation zugewiesen sein soll. Zudem dürfen keine Zyklen in den Anfangslösungen enthalten sein, da auch das eine vorgegebene Eigenschaft der Problemspezifikation ist. Dies garantiert, dass alle Ergebnisse erreicht werden können, da keine Einschränkungen der Lösungen stattfinden. Jedoch sind Verschlechterungen des Konvergenzverhaltens zu erwarten, da Zeit für die Suche nach validen Lösungen verloren geht. Wird die Startpopulation kontrolliert initialisiert, so benötigt dieser Schritt auch dementsprechend mehr Zeit. Da der Algorithmus aber auch schon bei der Suche nach validen Lösungen die Gesamtkosten durch neue Kabeltypen und Kanten optimiert, sollte die Population auch mit zufälliger Initialisierung nicht viel langsamer konvergieren. Dies wird im Evaluationsteil durch Tests geprüft. Zudem wird entschieden, ob eine aufwändige Initialisierung die Güte der Lösung durch Einschränken der Lösungspopulation vermindert.

Die Lösungspopulation wird nach der Initialisierungsphase in Zeile 4 evaluiert, um Fitnesswerte zur Selektion der Eltern zur Verfügung zu haben. Darauf folgt der Eintritt in die Hauptiterationsschleife in Zeile 5, in der der eigentliche evolutionäre Prozess stattfindet. Hier werden zuerst die Eltern für den Crossover-Operator ausgewählt. Dies geschieht im umgesetzten Algorithmus durch die Wahl der n besten Lösungen in der Population. Für den Selektionsschritt gibt es einige mögliche Konzepte. Als Referenz wurde das sogenannte *Restricted Tournament Crowding/Selection* umgesetzt und im Evaluationsteil untersucht.

Algorithm 5.1: Genetischer Algorithmus

```
   Input: Graph G = (V, E), Set of Cabletypes K, Integer PopulationSize, Integer
          Maxtime
   Data: Solution S, SolutionSet Population
   Output: Solution bestSolution

   // Initialization
 1 forall i < PopulationSize do
 2 |  S ← generateRandomSolution(G, K)
 3 |  Population.addSolution(S)
 4 evaluatePopulation()
   // Main loop
 5 while timeSpent < Maxtime ∧ ¬terminationCondition do
 6 |  Parents ← selectParents(Population)
 7 |  Children ← crossoverParents(Parents)
 8 |  evaluateChildren()
 9 |  Population ← selectNewPopulation(Population ∪ Children)
10 |  Children ← mutatePopulation(Population)
11 |  evaluateChildren()
12 |  Population ← selectNewPopulation(Population ∪ Children)
13 |  if getBestSolution(Population).betterThan(bestSolution) then
14 |  |  bestSolution ← getBestSolution(Population)
```

Sofern der Crossover aktiviert ist, folgt in den Zeilen 6 bis 9 die Rekombination der Eltern, die mehrere Kindlösungen erzeugt. Diese werden wiederum evaluiert und dann mit der gesamten ursprünglichen Population vereint, um die neue Population auszuwählen. Auch hier gibt es wiederum verschiedene Möglichkeiten, dies umzusetzen. Nach der Wahl der neuen Population werden in den Zeilen 10 bis 12 Lösungen zufällig mutiert. Hierbei wird in zwei unterschiedlichen Schichten zufällig entschieden, ob, und wenn ja, wie die Lösung mutiert wird. Danach wird, wie beim Verwenden des Rekombinationsoperators, eine Gesamtpopulation aus den ursprünglichen Lösungen und den Kindlösungen gebildet, die der Mutationsoperator erzeugt. Aus dieser wird wiederum die neue Population ausgewählt. Am Ende einer Iteration der Hauptschleife wird die Population erneut evaluiert und die beste Lösung, sofern sie besser ist als die überhaupt beste gefundene Lösung, zwischengespeichert. So wird am Ende in den Zeilen 13 und 14 garantiert, dass wirklich die beste Lösung ausgegeben wird, die während der Berechnung auftrat.

Zudem ist zu erwähnen, dass der Crossover in der Implementierung nur alle 3 Iterationen vollzogen wird. Das reduziert die Laufzeit, da sich in unserem Fall eine Verbesserung der Population hauptsächlich durch Mutationen ergibt. Inwiefern es Sinn macht, den Crossover Operator zu verwenden, wird im Abschnitt 6.2 untersucht.

Zum Einlesen der Ausgangsinstanz und zur Ausgabe der Lösung wurde das *Open Graph Drawing Framework* (kurz OGDF) verwendet [CGJ+14].

5.1 Repräsentation von Lösungen

Wie in Kapitel 3 erwähnt, ist die Wahl der Repräsentation essentiell, da die anderen Operatoren von dieser abhängen. Bei den vielen genetischen Algorithmen wird als Repräsentation ein Bitstring gewählt, was bei diesem Problem jedoch nicht sehr sinnvoll erscheint. Der

Grund hierfür ist die Möglichkeit verschiedene Anzahlen an Circuits zu haben, für die dann wieder Parameter wie die Größe des jeweiligen Circuits zu speichern wären. Das würde bedeuten, dass unterschiedliche Längen der Repräsentation vorkommen können, was die Implementierung der genetischen Operatoren signifikant erschweren würde. Bei einer überschaubaren Anzahl an betrachteten Konfigurationen im Aufbau der Windfarm wäre ein Bitstring vermutlich eine bessere Variante [LYX09]. Da jedoch beliebige Baumstrukturen zugelassen werden sollen, sollte eine Repräsentation gewählt werden, die dies einfach umsetzt. Daher wird für diese Arbeit die Repräsentation mittels der Adjazenzmatrix des Graphen gewählt. Für die Einträge werden Ganzzahlen verwendet, die, sofern ein Eintrag ungleich Null existiert, den Kabeltyp der Kante repräsentiert. Im Endeffekt wird also doch speicherintern eine Bitstring-Repräsentation genutzt, welche aber durch eine Datenstruktur eines höheren Levels für leichtere Benutzung gekapselt wurde. Ein Beispiel eines ähnlichen Ansatzes liefern Overbury und Berthouze [OB15]. Da das Problem nur ungerichtete Kanten betrachtet, wird aus Symmetriegründen oftmals nur die obere Dreiecksmatrix genutzt.

5.2 Fitnessfunktion

Die Fitnessfunktion ist in der Implementierung relativ einfach gehalten. Hier wird zuerst die eingehende Lösung auf Zusammenhang geprüft. Sofern sie nicht zusammenhängend ist, wird eine unzulässige Fitness zurückgeliefert, die dann von anderen Teilen des Programms behandelt werden kann. Dies ist nötig, da manche Operatoren für eine korrekte Funktionsweise zusammenhängende Lösungen benötigen.

Befinden sich Zyklen in der Lösung, so werden diese entfernt, woraufhin die Berechnung der Fitness fortgesetzt werden kann. Hier wird die längste Kante mit dem kleinsten im Zyklus vorkommenden Kabeltyp entfernt, da vor allem in fortgeschrittenen Populationen relativ wenige Kanten mit hohem Kabeltyp vorliegen. Diese sind dann meist Kanten, die einen ganzen Circuit an eine Substation anschließen und sich somit oft positiv auf die Lösungsqualität auswirken.

Die Fitnesswerte sind in vorliegender Problemstellung die Kosten der Verkabelung, welche sich aus der euklidischen Distanz der Kanten und den Kosten für deren Kabeltypen zusammensetzen. Hier ist zu erkennen, dass das Programm die Fitness der Lösung – im Gegensatz zur herkömmlichen Arbeitsweise von genetischen Algorithmen – nicht maximiert, sondern minimiert, was jedoch einfach zum entsprechend anderen Optimierungsproblem umgeformt werden kann. Sofern die Lösung an einer Substation mehr Turbinen angeschlossen hat, als nach der Kapazitätsbedingung möglich wäre, so wird sie von der Fitnessfunktion bestraft. Hier wird zur Fitness eine Konstante addiert, welche mit der jeweiligen Instanz skaliert. So werden größere Instanzen, bei denen die Kosten ohnehin höher sind, auch höher bestraft. Dies ist wichtig, da sonst eine invalide Lösung eventuell eine bessere Fitness als eine valide Lösung erhalten kann, was nicht erwünscht ist.

Eine weitere wichtige Funktion der Fitnessfunktion ist die Aufrechterhaltung der Flussbedingung. Dazu wird vor jeder Berechnung der Fitness für die vorliegenden Kanten berechnet, welche minimalen Kabeltypen eingesetzt werden müssen, um die Flussbedingung zu erfüllen. Anfangs wurden die Kabeltypen durch Mutation angepasst, jedoch ist es wesentlich sinnvoller, diese dynamisch zu berechnen. Mehr dazu in Abschnitt 5.5.1.

5.3 Selektionsoperator

Wie bereits in Kapitel 3 beschrieben, wird bei der Selektion zwischen Elternselektion und Selektion der neuen Population unterschieden. Jedoch können die verschiedenen Konzepte für beide Selektionen eingesetzt und getestet werden.

Als erste Möglichkeit wird *Restricted Tournament Crowding / Selection* [Har95] (kurz RTC/RTS) umgesetzt. Dabei werden iterativ zufällig zwei Individuen aus der Population ausgewählt und verglichen. Die Lösung mit der besseren Fitness wird in die neue Population aufgenommen und aus der Menge der möglichen Lösungen entfernt. Die schlechtere Lösung wird in der nächsten Runde mit den Verbleibenden wieder zur Auswahl freigegeben. Dies wird so lange iteriert, bis die neue Population die gewünschte Größe erreicht hat. Durch dieses Verfahren soll garantiert werden, dass jede Lösung eine Chance hat, zu überleben, jedoch besteht eine höhere Chance, sich gegen andere Lösungen durchzusetzen, sofern die Fitness höher bewertet ist. Dies fördert die Diversität der Population, da auch schlechtere Lösungen, die aber auch positive Eigenschaften haben, in einer weiteren Generation weitergeführt werden können. Der umgesetzte Ansatz des RTC übernimmt, am Ende der Selektionsphase, zusätzlich immer das beste Individuum der letzten Generation in die neue Population.

Ein weiterer Ansatz, der umgesetzt wird, wählt die n Individuen mit der besten Fitness aus der Population. Dabei ist n die gewünschte Populationsgröße. Dies ist ein naheliegender Ansatz, da die besten Lösungen mit hoher Wahrscheinlichkeit auch die besten Kindlösungen erzeugen werden. Dies erfolgt jedoch wiederum auf Kosten der Diversität, die von der Restricted Tournament Selection verstärkt wird. Die Auswirkungen der verschiedenen Selektionsoperatoren auf die finale Lösung sowie auf das Konvergenzverhalten der Population werden im Abschnitt 6.2 genauer untersucht. Die Elternselektion wird Konstant bei den n-besten belassen, da dies die besten Kindelemente verspricht. Mit RTC würden an dieser Stelle zu viele nicht gültige Kindelemente erzeugt, was sowohl Güte, als auch Konvergenzgeschwindigkeit vermindert. Wie die Erzeugung der Kindelemente abläuft, wird im nächsten Abschnitt genauer erläutert.

5.4 Rekombination

Es gibt einige Schwierigkeiten beim Finden eines guten Rekombinationsoperators für das gegebene Problem. Da unzusammenhängende Graphen oder Graphen mit Zyklen als Lösung nicht gültig sind, muss entweder eine Fehlerbehebung implementiert, oder Lösungen mit diesen Eigenschaften verworfen werden. Fehlerbehebung ist laufzeitaufwändig und schränkt somit das Programm erheblich ein. Der bestmögliche Fall wäre, einen Crossover-Operator zu finden, der gute Kindelemente erzeugt, jedoch Fehler schon durch seine Umsetzung vermeidet. Ein solcher Crossover ist nicht leicht zu finden, sofern er überhaupt existiert. In dieser Arbeit wird also ein Rekombinationsoperator umgesetzt, der ohne Vorwissen über die Elternlösungen funktioniert. Hierzu werden aus den Adjazenzmatrizen der beiden Elternlösungen zufällige Abschnitte bestimmt und diese Teile in einer Kindmatrix vereint. Dies kann jedoch auch unzulässige Kindlösungen erzeugen. Daher wird mittels Breitensuche untersucht, ob die erzeugten Lösungen zusammenhängend sind, und falls ja, ob sie Zyklen enthalten. Ist ein Kind unzusammenhängend, so wird es verworfen, da die Fehlerbehebung in diesem Fall sehr aufwändig werden kann. Für Probleme mit Zyklen wurde die Fehlerbehebung implementiert, welche rekursiv Schritt für Schritt Zyklen in Lösungen entfernt. Die Breitensuche inklusive der Fehlerbehebung der Zyklen kommt auch wieder bei den umgesetzten Mutationen zum Einsatz, die im nächsten Kapitel vorgestellt werden.

Eine weitere Idee für einen Rekombinationsoperator, die untersucht, aber nach kurzer Testphase wieder verworfen wurde, war abwechselnd disjunkte Circuits beider Eltern in die Kindlösung zu übernehmen und danach die nicht angebundenen Knoten am Ende noch an naheliegende Knoten anzubringen. Mit diesem Crossover hat sich aber die Population so entwickelt, dass nach einer gewissen Laufzeit die Lösungen der Population nur einen Circuit beinhalteten. Da übrige Knoten mit genannter Heuristik angebunden werden, werden oft

Substation Kapazitäten überschritten, da diese nicht überprüft werden. Da vor allem die Fitnessfunktion die Lösung für jede invalide Substation einmal bestraft, war es von Vorteil, einen Circuit zu entwickeln, da so nur eine Bestrafung stattfinden kann und daher der Fitnesswert besser abschneidet als bei mehreren invaliden Substationkapazitäten. Dies ist offensichtlich keine valide Lösung für das Full-Farm-Problem, da ein Circuit für die gesamte Farm bei Instanzen mit mehreren Substations in keiner Weise Sinn ergibt. Ein weiteres Problem dieser Vorgehensweise ist die erhöhte Laufzeit, die erzielt wird, da die Circuits der Lösungen gefunden und die genannte Fehlerbehebung durchgeführt werden muss.

5.5 Mutationen

5.5.1 Mutation der Kabeltypen

Zu Beginn wurden die Kabeltypen der einzelnen Lösungsindividuen auch durch eine Mutation bestimmt, die einer zufälligen Kante einen zufälligen Kabeltyp zuwies. Das Problem hierbei ist, dass die Lösung oft invalide wurde, sofern eine Kante höheren Kabeltyps durch eine Mutation entfernt wurde oder durch einen Crossover nicht in die Kindlösung übernommen wurde, da die Flussbedingung verletzt war. So wurden oft Individuen als schlecht erachtet, obwohl sie bezüglich des Aufbaus der Kanten sehr gut waren, jedoch aufgrund der falschen Kabeltypen als ungültig angesehen wurde. So konnte sich die Population oft nicht in vielversprechende Gebiete des Lösungsraums vorarbeiten, da die Lösungsfitness bestraft wird und die Lösung durch die Selektion aussortiert wird, sofern die Flussbedingung verletzt ist. Da die Fitnessfunktion nun die Aufgabe der Berechnung der Kabeltypen übernimmt, ist es wesentlich leichter für Populationen, den Lösungsraum zu durchsuchen. Dies erhöht die Güte der Lösungen und senkt zugleich die Zeit, die benötigt wird, um vergleichbar gute Lösungen zu finden.

5.5.2 Mutation der Kanten

Die erste Mutation – Abbildung 5.1 – die in der finalen Version des Algorithmus vorhanden ist, ist eine Kantenmutation. Hierbei wird eine zufällige Turbine ausgewählt und die Kante, die von dieser Turbine in Richtung Substation führt, zu einem beliebigen Knoten umgelegt. Hier werden alle möglichen Kanten zugelassen, was garantiert, dass der Algorithmus jede mögliche Kantenkonstellation erreichen kann. Dies ist offensichtlich von Vorteil, da es dem Algorithmus so auf jeden Fall möglich ist, das globale Maximum zu erreichen. Viele Algorithmen benutzen Heuristiken, um das Problem zu vereinfachen, was dann aber verhindern kann, das globale Optimum zu finden.

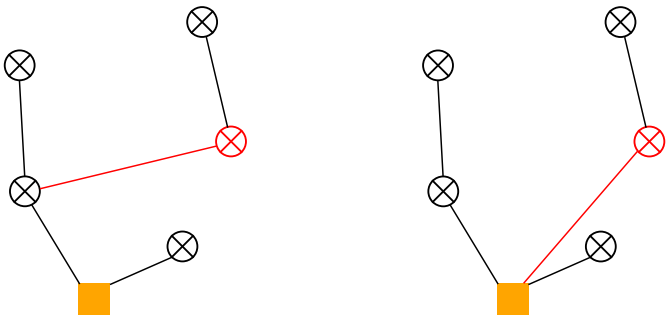


Abbildung 5.1: Zu sehen ist die erste vorgestellte Kantenmutation. Zuerst wird eine zufällige Turbine gewählt, in der Grafik ist das die rot markierte. Von dieser aus wird die rote Kante zum Elternknoten mutiert. Rechts ist das Ergebnis der Mutation zu sehen, bei der ein zufälliger anderer Elternknoten für die gewählte Turbine verknüpft wurde.

Eine zweite Mutation wählt zwei zufällige Knoten aus und tauscht alle Kanten, die an diesen Knoten anliegen, aus. Dies ist vor allem bei zufälligen initialen Populationen sinnvoll, da dadurch schnell große Veränderungen an der Lösung stattfinden können. Dadurch kann schneller in einen Bereich des Lösungsraums vorgedrungen werden, der valide oder bessere Lösungsindividuen enthält. Gegen Ende werden durch diese Mutation jedoch oftmals Lösungen erzeugt, die invalide oder schlechter sind als die Ausgangslösung. Eine vielversprechende Möglichkeit, die aus Zeitgründen nicht umgesetzt wurde, wäre, die Wahrscheinlichkeit dieser Mutation bei fortgeschrittener Berechnung zu verringern oder sogar gar nicht weiter zuzulassen. So stünden mehr Ressourcen für die Ausführung verbessernder Mutationen oder Crossovers zur Verfügung.

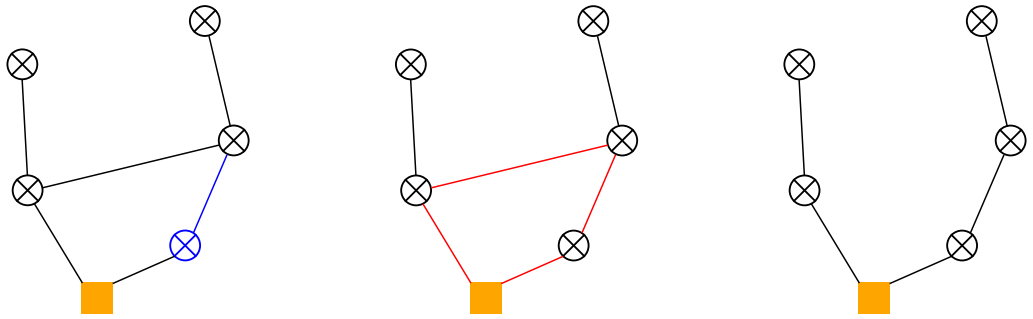


Abbildung 5.2: Dritte vorgestellte Mutation, die zuerst wieder eine Turbine zufällig auswählt. Danach wird eine gute Kante hinzugefügt, was in der Abbildung blau dargestellt ist. Durch diese Kante wird ein Zyklus in der Lösung erzeugt, der danach mittels Fehlerbehebung wieder entfernt wird. Wie zu sehen wird die schlechteste Kante entfernt und das Ergebnis rechts zurückgegeben.

Die letzte umgesetzte Mutation – Abbildung 5.2 – ist allgemein benutzbar, profitiert aber von einer bestimmten Art der Eingabe der zu berechnenden Ausgangsinstanz. Als Voraussetzung dieser Mutation wird in der Initialisierungsphase für jeden Knoten gespeichert, welche adjazenten Kanten in der Ausgangsinstanz enthalten sind. Dies kann offensichtlich durch die Struktur der Eingabe beeinflusst werden. Ein Ansatz, der gute Ergebnisse verspricht, wurde von Sebastian Lehmann [Leh16] vorgeschlagen. Dieser berechnet beim Erzeugen der zu berechnenden Benchmark-Instanzen die vielversprechenden Kanten vor. Dies geschieht, indem die 6 nächsten Nachbarn an jedem Knoten verbunden und danach weitere Kanten, die potentiell profitabel eingestuft werden, hinzugefügt werden. Die Mutation dieses Algorithmus wählt wiederum einen Knoten zufällig aus und fügt an diesen Knoten zufällig eine der vorberechneten Kanten an. Da der Graph in seiner internen Darstellung vorher ein Baum war, wird durch Hinzufügen einer weiteren Kante, die nicht vorher im Graphen war, ein Zyklus erzeugt. Dieser wird dann durch die Fehlerbehebung für Zyklen entfernt. Da die neu hinzugefügte Kante in den meisten Fällen besser ist als manche Kanten, die sich ohnehin schon im Zyklus befinden, wird mit hoher Wahrscheinlichkeit eine gute Kante zur Lösung hinzugefügt und eine schlechte Kante entfernt. Dies beschleunigt die Konvergenz der Lösungspopulation beträchtlich und verbessert zudem die Ergebnislösungen.

5.6 Erstellen der Anfangspopulation

Bevor die Population initialisiert werden kann, muss entschieden werden, welche Populationsgröße benutzt werden soll. Dies wird im Evaluationsabschnitt 6.1 analysiert.

Es wurden zwei mögliche Methoden zur Erzeugung der initialen Population umgesetzt. Eine erzeugt zufällige Lösungsindividuen, welche anfangs sehr schlechte Lösungen sind, da alle Kanten zugelassen werden. Dadurch werden auch Kanten gewählt, die durch ihre

Länge niemals in eine gute Lösung aufgenommen werden würden. Anfangs sind also für fast jede Instanz alle Lösungen invalide, meist wird aber nach wenigen Iterationen eine valide Lösung gefunden. Ein positiver Effekt dieser zufälligen Initialisierung ist jedoch, dass die Population in keiner Weise eingeschränkt ist und sich beliebig entwickeln kann.

Die zweite Möglichkeit ist, die Startpopulation voll mit minimalen Spannbäumen des gegebenen Graphen aufzufüllen. Hierdurch werden zumindest gute Kanten als Start betrachtet, was die Geschwindigkeit für das Finden einer guten Lösung wesentlich beschleunigt. Jedoch kann hierdurch die Diversität der Population stark beschränkt werden, da anfangs nur eine Art Individuum vertreten ist. Es kann für die Mutationen oder den Crossover auch durchaus schwerer sein, bestehende gute Teile einer Lösung weiter zu verbessern. Die Auswirkungen der Startpopulation werden im Evaluationskapitel 6 genauer überprüft und es wird analysiert, welche Startpopulation die bessere Wahl für verschiedene Anwendungszwecke ist.

5.7 Abbruchkriterien

Als Abbruchkriterium kommen mehrere Möglichkeiten in Frage. Eine maximale Berechnungszeit ist ein Abbruchkriterium, das auf jeden Fall umgesetzt werden sollte. Es ist also möglich, eine Zeit zu spezifizieren und den Algorithmus so lange rechnen zu lassen. Nun stellt sich die Frage, ob es Sinn macht, den Algorithmus an einem Punkt weiter rechnen zu lassen, an dem sich die Fitness über lange Zeit nicht verbessert hat. In manchen Fällen, vor allem bei kleineren Instanzen, wäre es wahrscheinlich sinnvoller, die Berechnung abzubrechen und eine weitere komplett neue Population zu berechnen. Dies führt dazu, dass ein unbetrachteter Bereich des Lösungsraums möglicherweise erkundet wird und somit eine weitere gute Lösung gefunden werden kann. Am Ende kann das Ergebnis der besten Berechnung ausgewählt werden. Bei größeren Instanzen kann das jedoch dazu führen, dass eine zweite Population angefangen wird, diese jedoch in der übrigen Rechenzeit noch nicht gegen ihr lokales Optimum konvergiert. Dadurch wird in einem Fall wie diesem die Güte der ersten Population eingeschränkt und eine zweite Population bringt kein sinnvolles Ergebnis. Daher bietet es sich an zu überprüfen, ob im Falle eines konvergenzbedingten Abbruchs die Restzeit größer oder gleich der bereits verstrichenen Rechenzeit ist. Um für Instanzen mit unterschiedlichen Turbinenzahlen besser anwendbar zu sein, sollte der Abbruch nicht zeitlich bedingt, sondern iterationsbedingt stattfinden. Dies ist der Fall, da der Unterschied der bisher besten berechneten Lösungen zur optimalen Lösung bei gleicher Anzahl Iterationen geringer ist als bei gleicher betrachteter Rechenzeit. Dies kommt von der erhöhten Rechenzeit für Operationen bei größeren Instanzen.

6. Evaluation

Die Instanzen, an denen in diesem Abschnitt getestet wird, sind in verschiedene Testsets eingeteilt. Diese Testsets unterscheiden sich in Anzahl der Turbinen und Substations, aber auch in deren Verhältnis und anderen Parametern, wie zum Beispiel der Form der Windfarm oder der Kapazitäten der Substations. Die Eigenschaften der verschiedenen Datensätze sind in Tabelle 6.1 dargestellt. t_{min} und t_{max} beschreiben die möglichen Turbinenzahlen im jeweiligen Testset. Analog dazu bezeichnen s_{min} und s_{max} die minimale und maximale Anzahl an Substations in den betrachteten Instanzen. Eine weitere Größe β, die sich stark auf die Performance des Algorithmus auswirkt, wird in dieser Arbeit *Dichte* (oder engl. *Tightness*) genannt. Diese Variable kann folgendermaßen berechnet werden:

$$\beta = \frac{t}{s \cdot m}$$

Dabei ist t die Anzahl an Turbinen der Instanz, s die Anzahl an Substations und m ist die Kapazität der Substation, also die Anzahl an Turbinen, die maximal pro Substation angeschlossen werden können. Je näher der Wert β an 1 liegt, desto weniger Spielraum ist den Turbinenzahlen an jeder Substation gegeben. Wie später zu sehen sein wird, kommt es bei einer zu hohen Dichte sehr häufig zu schlechten oder gar invaliden Lösungen. Dieses Phämomen wird in Abschnitt 6.2 genauer beschrieben. Daher wird für die Evaluation des Algorithmus die Dichte der Substations reduziert. Ein letzter Parameter einer Instanz sind die Kanten, die als mögliche Kanten zur Lösung in Betracht gezogen werden. Es werden zuerst die spezifizierten nächsten Nachbarn für jeden Knoten mit in die zu lösende Instanz aufgenommen und dann, wie von Lehmann in [Leh16] beschrieben, einige weitere Kanten hinzugefügt, welche das Potential haben, Teil einer guten Lösung zu sein. In dieser Arbeit werden für alle Testsets jeweils die 6 nächsten Nachbarn und die erwähnten zusätzlichen Kanten in Betracht gezogen.

Für die Evaluation ist es wichtig anzumerken, dass Testläufe immer einen Zufallsfaktor mit sich bringen. Das heißt, dass es aufgrund der Natur eines genetischen Algorithmus möglich ist, durch Zufall unterschiedliche Messwerte für eine Instanz bei gleichem Algorithmus und unterschiedlichen Durchläufen zu erhalten. Jedoch wird die Wahrscheinlichkeit für das Auftreten von Ausreißern durch mehrere Messungen des gleichen Tests minimiert. Dementsprechend ist es offensichtlich auch möglich und zudem sehr wahrscheinlich, dass die Ergebnisse nicht dem globalen Optimum entsprechen. Eine gute Referenz zur Bewertung der Ergebnisse des Algorithmus bildet hier die Lösung des Mixed Integer Linear

Testset Bezeichnung	t_{min}	t_{max}	s_{min}	s_{max}	β_{min}	β_{max}
T1	10	80	1	1	0.7	1
T2	10	80	2	7	0.7	1
T3	80	200	4	10	0.5	1
T4	200	500	10	40	0.4	1

Tabelle 6.1: Tabelle der benutzten Instanzgrößen. Wie zu sehen ist, wurden den Testsets T1 – T4 nur eine gewisse Anzahl an Kanten für die letzte vorgestellte Kantenmutation betrachtet.

Programming Ansatz (kurz MILP) – umgesetzt mittels Gurobi Engine – mit der verglichen wird [LRWW17].

In diesem Kapitel werden in erster Linie zwei Arten von Veränderungen am Algorithmus getestet. Diese sind in Abschnitt 6.1 die Populationsgröße und in Abschnitt 6.2 Veränderungen an der Funktionsweise des Algorithmus. Zwei Eigenschaften, auf die sich die Tests konzentrieren, sind sowohl die Kosten der finalen Lösung eines Durchlaufs, als auch die Konvergenzgeschwindigkeit der Population. Diese sagt aus, wie schnell – bezüglich der Laufzeit – der Algorithmus die Kosten der jeweils besten Lösung einer Population minimiert. Die verschiedenen Arten von Algorithmen sind in Tabelle 6.2 dargestellt, werden aber verfeinert im Abschnitt 6.2 erläutert. Die Durchläufe zur Populationsgröße finden auf drei Referenzinstanzen aus T2, T3 und T4 statt. Testset T1 wurde nicht untersucht, da es gleiche Turbinenzahlen wie T2 besitzt und sich nur in der Anzahl der Substations unterscheidet.

Durchläufe finden mit jeweils einem Thread statt, Multithreading kann jedoch per Parameter aktiviert werden. Dies erhöht im Fall dieses Algorithmus nicht die Performance für einen Durchlauf, sondern minimiert lediglich das Risiko, zufällig eine schlechte Lösung durch Konvergieren in ein unvorteilhaftes lokales Optimum zu erhalten. Dies geschieht durch paralleles Berechnen von n Populationen, wobei n die Anzahl der eingegebenen Threads bezeichnet. Nach Berechnung dieser Populationen wird die beste Lösung als Ergebnis ausgewählt.

Bezeichnung	Populationsgröße	Crossover	Anfangspopulation	Selektionsoperator
Pop.Größe	verschiedene Größen	Nein	MST	n-beste Elemente
MST	15	Nein	MST	n-beste Elemente
Crossover	15	Ja	MST	n-beste Elemente
Random	15	Nein	Zufällig	n-beste Elemente
RTC[1]	15	Nein	MST	RTC

Tabelle 6.2: Darstellung der verschiedenen Versionen des Algorithmus und deren Bezeichnungen. Die MST-Version ist die Standardversion, von der die anderen sich immer in einem Aspekt unterscheiden. In Abschnitt 6.1 wird dieser Algorithmus auf Performance bei Veränderung der Populationsgröße untersucht.

6.1 Untersuchung verschiedener Populationsgrößen

In diesem Abschnitt wird die Veränderung der Populationsgröße auf die Performance des Algorithmus untersucht. In diesem Zusammenhang werden vor allem zwei Größen betrachtet: Zum einen die Qualität der Endergebnisse nach 30 Minuten Berechnungszeit

[1] Restricted Tournament Crowding

und zum anderen die Konvergenzgeschwindigkeit, wie schnell die Population gute Lösungen entwickelt. Diese Zeitschranke wurde gewählt, da die Population, für die meisten Instanz- und Populationsgrößen, nach 30 Minuten stark gegen ein lokales Optimum konvergiert.

6.1.1 Endergebnisse der Durchläufe

Es wird an drei Referenzinstanzen getestet, die in Tabelle 6.3 zu sehen sind. Im ersten Teil des Versuchs werden mehrere Durchläufe des Algorithmus mit Populationsgrößen von 5 bis 100 durchgeführt, wobei in jeder Iteration 5 Individuen hinzugefügt werden. Im zweiten Teil werden im niedrigen Bereich feinere Schritte betrachtet. Dort wird die Populationsgröße in Schritten von einem Individuum von 1 bis 20 untersucht.

Instanz	Testset	Turbinen	Substations
2	2	68	6
3	3	179	9
4	4	356	29

Tabelle 6.3: Verschiedene Testinstanzen für die Tests zur Populationsgröße.

Zu erwarten ist, dass ein Abfall der Güte der finalen Lösung stattfindet, sofern die Population zu groß wird, da die Rechenzeit nicht genügt, um ebenbürtige Lösungen zu finden. Außerdem ist zu erwarten, dass bei diesen großen Populationen auch die Konvergenzgeschwindigkeit der Population leidet. Zudem wäre ein Güteabfall bei zu kleinen Populationen wahrscheinlich, da die Diversität der Population eingeschränkt wird. Dies wird vermutet, da in einer Iteration, der Populationsgröße entsprechend skalierend, weniger neu mutierte Lösungen erstellt werden.

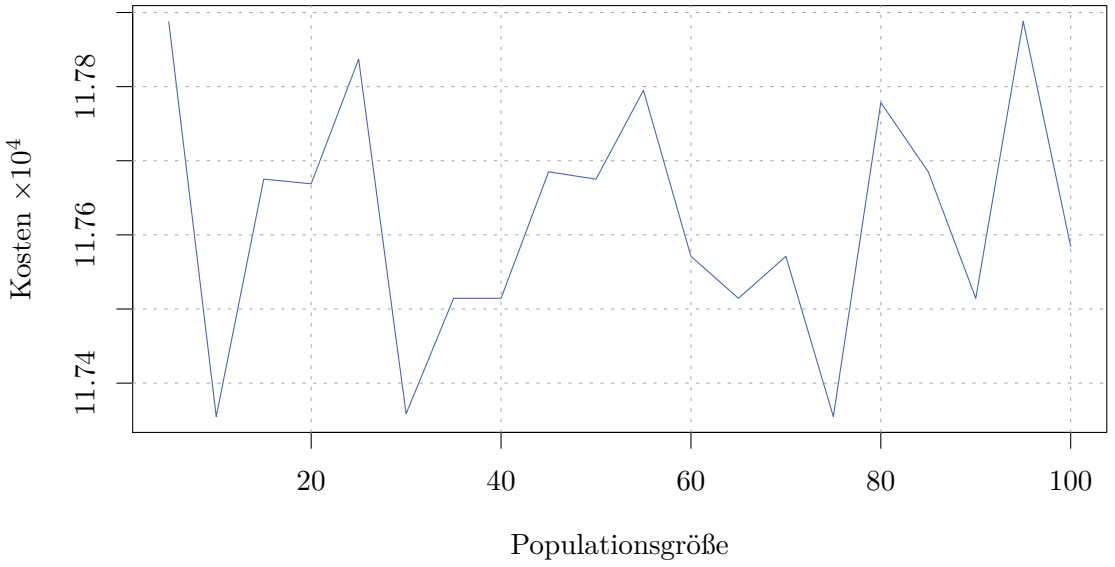


Abbildung 6.1: Endgültige Kosten der Lösungen des Algorithmus bei verschiedenen Populationsgrößen auf Instanz 2.

Abbildung 6.1 lässt den Schluss zu, dass die Güte der Lösungen im Verhältnis zur Populationsgröße auf Testinstanz 2 nur sehr schwach korelliert. Dies folgt sehr wahrscheinlich aus der Tatsache, dass der Algorithmus selbst bei hohen Populationsgrößen für diese Größenordnung von Windfarmen keine Laufzeitprobleme bekommt. Da kleine Populationsgrößen jedoch mehr berechnete Iterationen in der gleichen Zeit zulassen, ist es praktikabler, für diese Größe an Instanzen kleinere Populationsgrößen zu wählen.

Abbildung 6.2: Endgültige Kosten der Lösungen des Algorithmus bei verschiedenen Populationsgrößen auf Instanz 4.

Abbildung 6.2 zeigt die Kosten der Ergebnisse nach einer halben Stunde auf Instanz 4 mit verschiedenen Populationsgrößen. Wie ersichtlich ist, existiert eine positive Korrelation der Kosten und der Populationsgröße. Es gibt Ausreißer, wie zum Beispiel bei 60 Individuen zu sehen ist, aber dennoch ist ein Aufwärtstrend zu erkennen. Dieses Ergebnis folgt in erster Linie aus der erhöhten Rechenzeit, die bei höherer Populationsgröße pro Iteration benötigt wird. Je größer die Anzahl der Individuen, die in einer Iteration bearbeitet werden müssen, desto langsamer konvergiert die Population. Aufgrund dieser Ergebnisse wird ein weiteres Experiment mit Populationsgrößen von 1 bis 20 durchgeführt und untersucht, ob die Kosten mit diesen ebenfalls korreliert sind. Im Anhang lassen sich Abbildungen dieser Ergebnisse finden, welche eine sehr schwache Korrelation aufwiesen. Die finalen Ergebnissen der Berechnung erlauben demnach keinen Schluss auf eine optimale Populationsgröße. Aus diesem Grund wird zusätzlich die Konvergenzgeschwindigkeit betrachtet, welche vor allem in kleinen Bereichen interessante Ergebnisse liefert.

6.1.2 Konvergenzgeschwindigkeit der Durchläufe

Abbildung 6.3 stellt die Konvergenz von zwei Durchläufen des Algorithmus auf Testinstanz 3 mit zwei unterschiedlichen Populationsgrößen dar. Bei Populationsgröße 4 lässt sich bei ungefähr 10 Minuten ein Ausschlag erkennen. Dieser stammt von der Optimierung (siehe Abschnitt 5.7), die den Algorithmus erneut mit neuer Population startet, sofern die

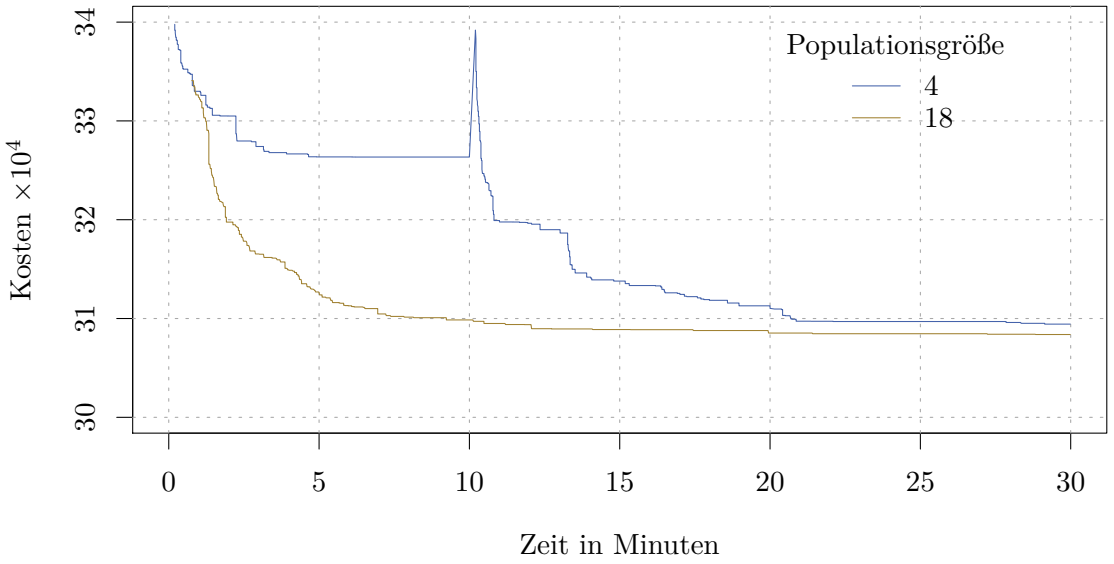


Abbildung 6.3: Kosten im Verlauf der Zeit, bei der Berechnung von Instanz 3 mit verschiedenen Populationsgrößen, siehe Legende.

Population stagniert und sich nicht weiter verbessert. Ähnliche Stagnierungen sind bei Populationsgrößen unter 4 zu beobachten. Diese treten bei kleineren Populationsgrößen vermehrt auf, da dort die einzelnen Iterationen des Algorithmus schneller stattfinden und die Optimierung der Iterationszahl basiert. Die Endergebnisse beider Durchläufe weisen bezüglich der Kosten eine große Ähnlichkeit auf. Dies sollte aber nicht von der Tatsache ablenken, dass sich bei 4 Individuen in der Population ein sehr hohes Plateau zwischen Minute 5 und 10 bildet. Es ist in etwa 6% von der finalen Lösung der Durchläufe entfernt. Falls sich ein weiteres Plateu nach Neustarten der Population bildet, ist es möglich, dass ein wesentlich schlechteres Ergebnis folgt. Daraus lässt sich schließen, dass bei zu kleinen Populationsgrößen, aufgrund der fehlenden Diversität der Population und der Anzahl der Kindelemente pro Iteration, das Risiko, eine schlechte Lösung zu erhalten, wesentlich größer ist.

Wie zu erwarten, ist Abbildung 6.4 zu entnehmen, dass die Konvergenzgeschwindigkeit verringert wird, je höher die Populationsgröße gewählt wird. Zu sehen ist zu jedem gegebenen Zeitpunkt bis mindestens 12 Minuten, dass die Kosten der aktuell besten Lösung nach Populationsgröße sortiert sind. Am Durchlauf mit Populationsgröße 100 ist erkennenbar, dass erst ab circa 5 Minuten das erste Mal eine gültige Lösung gefunden wird. Sofern nur eine geringere Laufzeit zur Verfügung steht, ist also auch zu einer niedrigen Populationsgröße zu raten.

Für die verschiedenen Algorithmenversionen, welche im nächsten Abschnitt untersucht werden, wird aufgrund der Ergebnisse dieses Abschnitts eine Populationsgröße von 15 gewählt. Dies verspricht gute Konvergenzzeiten und weist bessere Robustheit gegen Stagnierungen der Population auf als beispielsweise Populationsgrößen unter 10 Individuen.

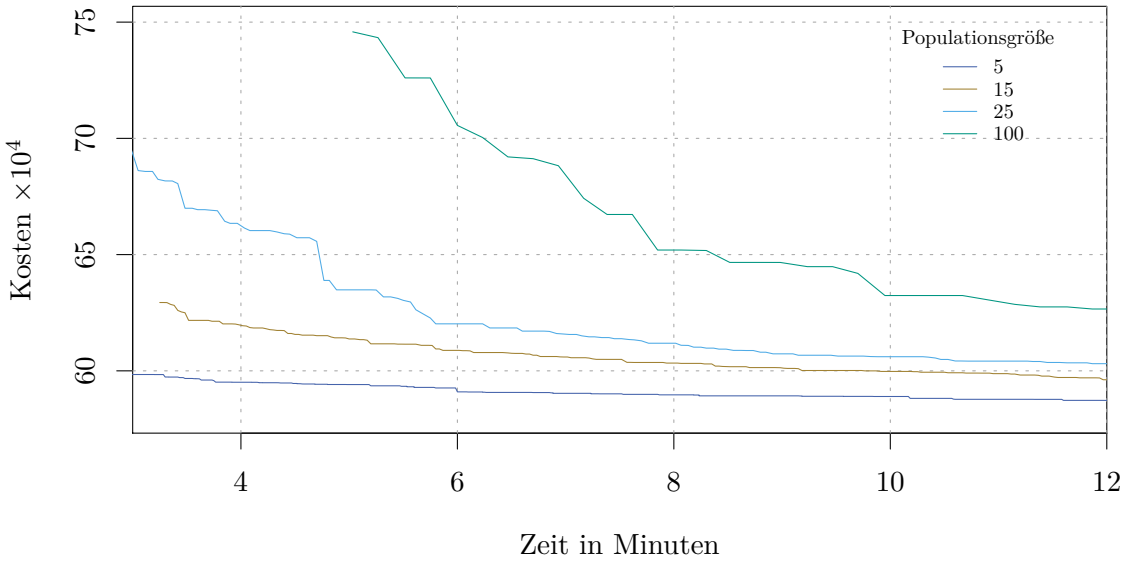
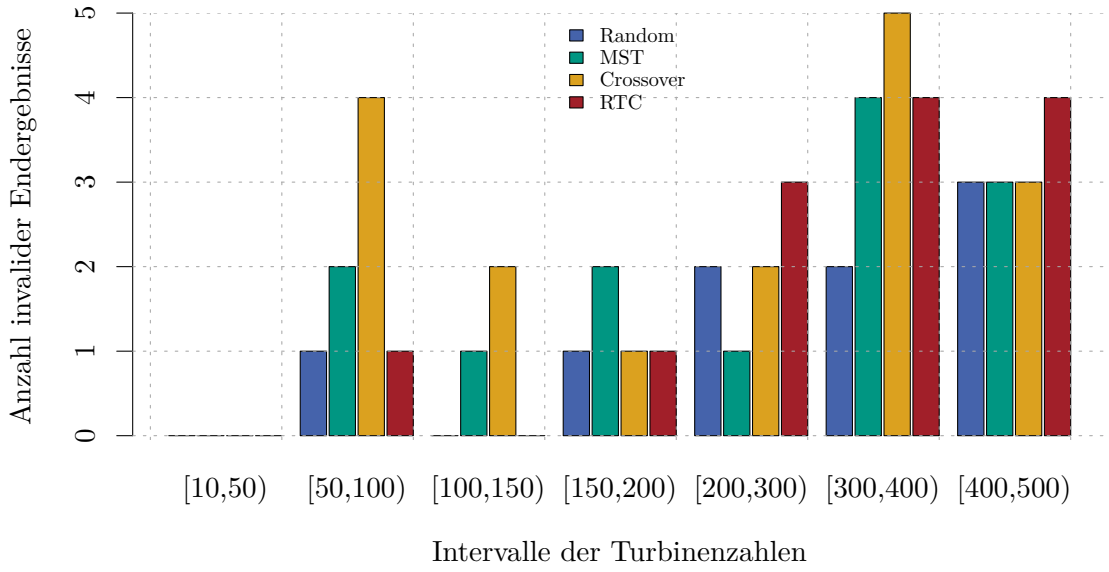


Abbildung 6.4: Test auf Instanz 4. Konvergenzgeschwindigkeit leidet unter erhöhter Populationsgröße. Hieraus folgt die verschlechterte finale Lösung nach 30 Minuten für verschiedene Anzahlen an Individuen.

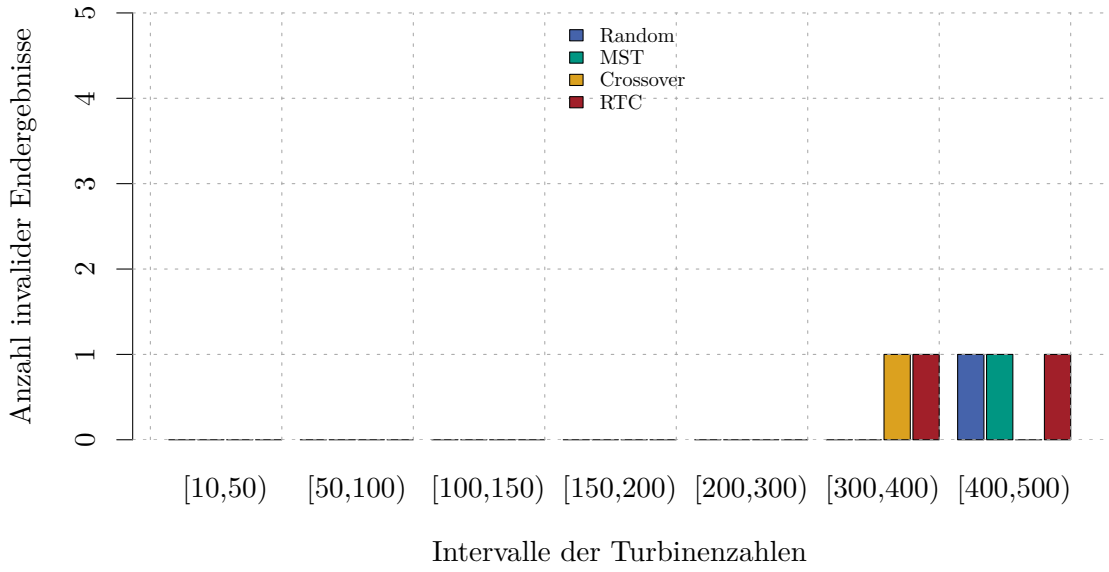
6.2 Änderung der Funktionsweise

Vorab sollten für dieses Kapitel die Metadaten besprochen werden, wie die Tests durchgeführt werden. Der vorgestellte genetische Algorithmus wird zuerst mit einem MILP-Algorithmus verglichen, der auf der Gurobi Engine basiert [Leh16, LRWW17]. Dazu wird diesem pro Instanz aus T3 und T4 eine Stunde Laufzeit gegeben, für Instanzen aus T1 und T2 20 Minuten. Der genetische Algorithmus hat pro Durchlauf von Instanzen aus T3 und T4 immer 30 Minuten Rechenzeit zur Verfügung, für die Instanzen aus T1 und T2 auch wieder die Hälfte des MILP – 10 Minuten. Die Berechnungen werden mit je einem Thread auf einem Server mit zwei Intel Xeon E5430 CPU ausgeführt, die mit 2.66 GHz Taktfrequenz arbeiten. Die verfügbaren 8 Kerne werden jedoch nie alle gleichzeitig zur Berechnung verwendet, sodass keine Berechnung durch Scheduling des Betriebssystems größere Zeiteinbußen hat. Der verfügbare Arbeitsspeicher beträgt 32 Gigabyte, jedoch werden für eine Population der größten Instanzen aus T4 nur circa 100 Megabyte benötigt.

Zum einen wird der vorgeschlagene Standardalgorithmus untersucht, und zudem drei andere Versionen, von denen sich jede in einem Aspekt von der Standardvariante unterscheidet. Die Populationsgröße wird für alle Versionen, aufgrund von Abschnitt 6.1, auf 15 Individuen festgelegt. Als Standardversion wird die in Tabelle 6.2 als *MST* aufgelistete betrachtet. Den Namen erhält sie hauptsächlich aus dem Grund, da sie die initiale Population mittels MST-Methode 5.6 initialisiert. Sie beinhaltet keinen Crossover Operator und benutzt den vorgeschlagenen Selektionsoperator der n-besten Elemente. Die zweite Version ist die *Crossover*-Version, bei der der Crossover alle 3 Iterationen durchgeführt wird und aus den besten 10 Individuen der Population Kindelemente erzeugt. Die dritte Variante verändert die Initialisierung der Population zu 15 zufälligen Bäumen anstatt 15 Individuen der MST-Lösung. Die letzte umgesetzte Version ändert den benutzten Selektionsoperator



(a) Unveränderte Dichte



(b) Angepasste Dichte

Abbildung 6.5: Invalide Endergebnisse in Relation zu den Turbinenzahlen der Instanz in den dargestellten Intervallen. Abbildung (a) ist hierbei mit der vorher spezifizierten Dichte berechnet, bei Abbildung (b) wird die Dichte der Instanz gelockert.

zu Restricted Tournament Crowding, wie in Kapitel 5.3 beschrieben. Leider konnten aus Zeitgründen nicht alle möglichen Kombinationen dieser Eigenschaften auf ihre Performance analysiert werden. Daher wird immer ein Aspekt verändert, so kann am Ende trotzdem eine gute Einschätzung der Veränderung der Performance durch die verschiedenen Aspekte erfolgen.

Instanzen, die betrachtet werden, stammen aus T1-T4, jedoch ist hier zu beachten, dass die Substationkapazitäten erhöht werden, um die Dichte der Instanzen zu verringern. Dies ist für die Performance des genetischen Algorithmus nötig, da der Algorithmus bedeutend schlechter arbeitet, falls die Dichte zu hoch ist. Dies resultiert daraus, dass es vor allem bei dichten Instanzen wesentlich weniger Möglichkeiten gibt, die Turbinen auf die Substations zu verteilen. Daraus folgt dann, dass in vielen Fällen keine gültige Lösung gefunden werden kann und der Algorithmus eine invalide Lösung so gut wie möglich verbessert, jedoch nie in einen validen Bereich des Lösungsraums vordringt. Leicht zu erkennen ist dies anhand von Abbildung 6.5. Dort werden die Anzahlen an invaliden Lösungen im Verhältnis zu Turbinenintervallen der Instanzen veranschaulicht. Die verschiedenen Algorithmen, wie in Tabelle 6.2 abgebildet, sind jeweils nebeneinander in den Intervallen dargestellt. Erkennbar ist, dass bei 11 Testinstanzen pro Testset ohne die Veränderung der Dichte die Anzahl von invaliden Lösungen wesentlich höher ist als nach der Anpassung der Dichte. Ganz besonders deutlich ist, dass nach der Auflockerung lediglich die großen Instanzen einzelne invalide Lösungen mit sich bringen. Die einzelnen Substationkapazitäten werden der Einfachheit halber für jede Instanz nur um 4 Turbinen erhöht. Dadurch spielt die unterschiedliche Dichte der Instanzen immernoch eine Rolle, jedoch ist hierdurch die Performance des genetischen Algorithmus sehr stark angestiegen.

Für die nachfolgenden Tests werden aus jedem der Testsets T1 bis T4 30 verschiedene Instanzen berechnet. Diese weisen zufällige Turbinenanzahlen, Substationanzahlen und Dichten in der Reichweite des jeweiligen Testsets vor.

6.2.1 Vergleich mit MILP-Algorithmus

Wie in Abschnitt 6.1 liegt ebenso im Nachfolgenden das Hauptaugenmerk im ersten Abschnitt auf der Qualität der Endergebnisse und im zweiten auf der Konvergenzgeschwindigkeit, mit der eine gute Lösung gefunden wird.

Vergleich der finalen Lösungen pro Instanz

Zuerst wird die durchschnittliche Abweichung der Ergebnisse der Algorithmenvarianten im Verhältnis zur MILP-Lösung untersucht. Die Instanzen sind nach Turbinenzahlen gruppiert und dann wird der Durchschnitt der Abweichung für jede dieser Gruppen gebildet.

Für die verschiedenen Versionen des Algorithmus können vorweg Hypothesen aufgestellt werden. Zuerst wird erwartet, dass die Laufzeit, die benötigt wird, um eine gute Lösung zu erhalten, mit der Anzahl der Turbinen skaliert. Des Weiteren wird vermutet, dass durch Hinzunahme des Crossover Operators die Berechnung verzögert wird und so im Schnitt schlechtere Ergebnisse folgen. Zuletzt wird von der Version mit Restricted Tournament Crowding als Selektionsoperator eine diversere Population erwartet, was dann zu besseren Ergebnissen am Ende führen kann.

Die Hypothesen können anhand von Abbildung 6.6 untersucht werden. Es werden nur valide Endergebnisse in die Berechnung des Durchschnitts pro Intervall einfließen lassen, da die Bestrafung der Fitnessfunktion für invalide Lösungen den Durchschnitt verfälscht hätte. So wäre die Grafik nicht mehr sinnvoll darstellbar. Auf den ersten Blick lässt sich sofort die erhöhte Abweichung bei hohen Turbinenzahlen erkennen, was die erste Hypothese unterstützt. Wie sich anhand der Konvergenzgeschwindigkeit im nächsten

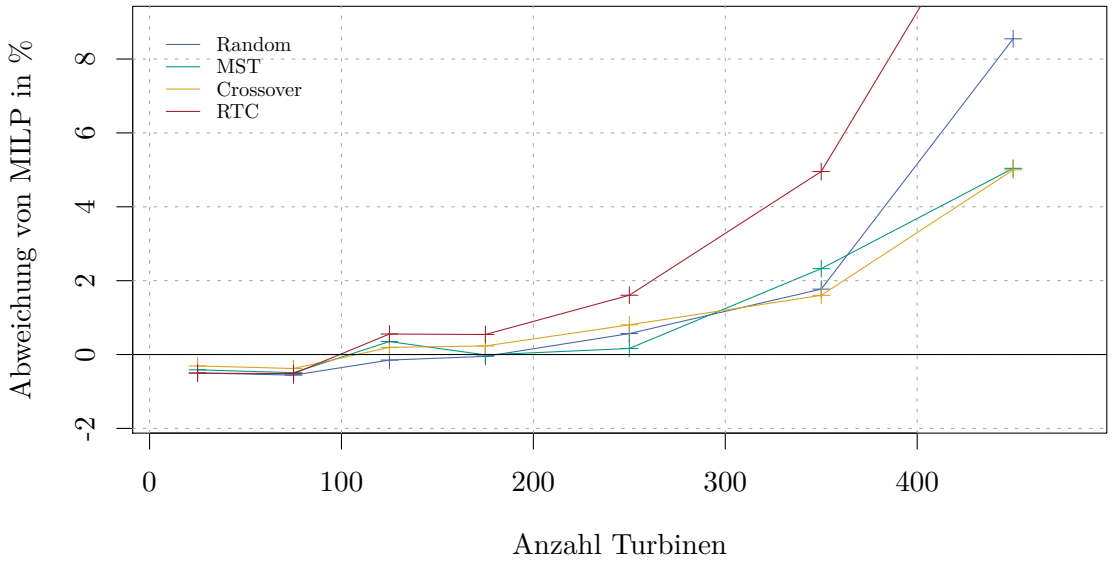


Abbildung 6.6: Dargestellt ist der Mittelwert der Abweichung zur MILP-Lösung bei Tests der Algorithmen. MILP Lösung nach 20/60 Minuten – je nach Testset –, den Algorithmenvarianten wurden 10/30 Minuten Laufzeit pro Instanz gegeben. Bis zu 200 Turbinen wurden Intervalle von 50 Turbinen in einem Mittelwert zusammengefasst, danach wurden jeweils 100-Turbinen-Schritte betrachtet.

Abschnitt vermuten lässt, stammt dies von der zu geringen Laufzeit des Algorithmus für diese Größe von Instanzen.

Ein weiterer auffälliger Aspekt ist, dass ab ca 100 Turbinen der RTC-Ansatz durchweg schlechter abschneidet. Für hohe Anzahlen von Turbinen ist er sogar bedeutend schlechter als die restlichen Varianten. Dies widerlegt die zweite Hypothese und zeigt, dass für den umgesetzten Algorithmus das Restricted Tournament Crowding eine schlechtere Selektionsalternative bildet. Das lässt sich höchst wahrscheinlich auf die verringerte Konvergenzgeschwindigkeit zurückführen. Durch den Selektionsoperator bleiben im Durchschnitt in jeder Iteration schlechtere Individuen erhalten als beim n-Beste Verfahren. Daher wird viel Rechenzeit für die Mutation von schlechteren Lösungen aufgewendet. Diese haben dann offensichtlich eine geringere Chance, eine bessere Lösung zu bilden, als die bisher besten gefundenen Lösungen.

Auch die dritte Hypothese wird durch die Untersuchungen widerlegt. Der Crossover-Ansatz liefert nur bedingt schlechtere Resultate als die anderen. Ab circa 300 Turbinen sind die besten Ergebnisse sogar mit Crossover erreicht worden. Die weitere Kombinations- und Selektionsschicht, welche der Rekombinationsoperator mit sich bringt, wirkt sich anscheinend positiv auf die Konvergenzgeschwindigkeit der Population aus.

Sofern die Ergebnisse nur für die Testsets T1, T2 und T3 genauer untersucht werden, lassen sich Aussagen über die beste Variante für kleinere Turbinenzahlen treffen. Hierzu ist Abbildung 6.7 zu betrachten, welche eine Vergrößerung der Abbildung 6.6 zeigt.

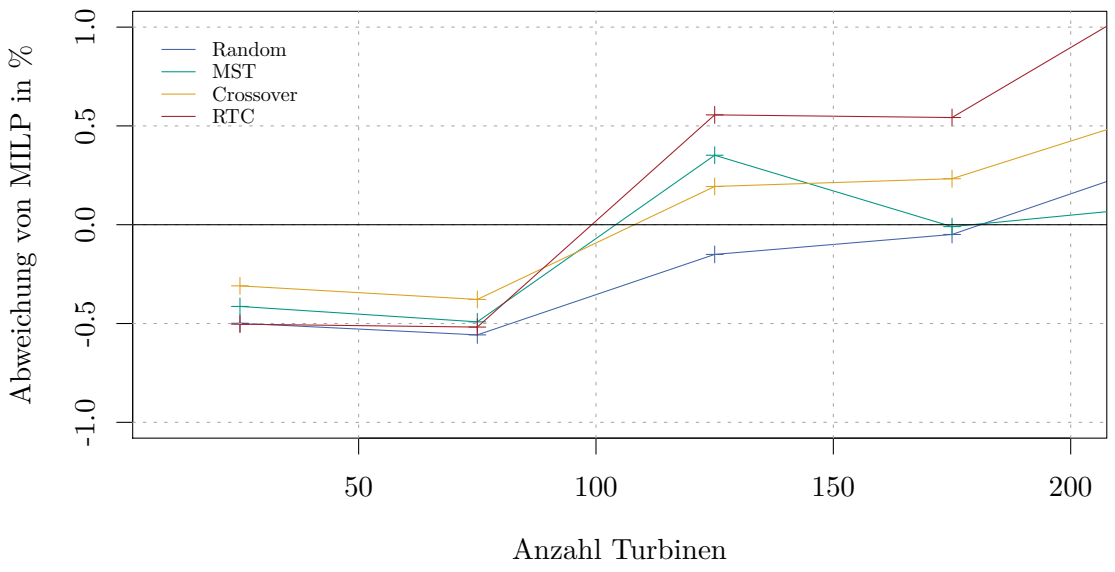


Abbildung 6.7: Vergrößerung von Abbildung 6.6. Es ist deutlich zu sehen, dass für kleinere Turbinenzahlen die Random-Variante gewählt werden sollte.

Die Algorithmenvariante mit zufälliger initialer Population schneidet hier am besten ab. Vor allem im Bereich von 100 bis 150 Turbinen ist sie die einzige, die bessere Ergebnisse als die MILP-Lösung liefert.

Auch die RTC-Variante ist für kleine Turbinenzahlen nicht zu vernachlässigen, lässt aber, wie im vorherigen Abschnitt erwähnt, ab circa 100 Turbinen deutlich nach.

Untersuchung der Konvergenzgeschwindigkeit anhand Referenzinstanz

Zur Betrachtung der Konvergenzgeschwindigkeit wird eine Referenzinstanz aus T4 ausgewählt, da dort ziemlich klar zu sehen ist, wie sich die verschiedenen Algorithmenvarianten auf die Berechnung auswirken.

Abbildung 6.8 zeigt die prozentuale Abweichung der aktuell besten Lösung im Verhältnis zum Endergebnis des MILP-Algorithmus. Wie bei der Untersuchung der Endergebnisse erwähnt, ist der schlechtere Trend der Endergebnisse bei großen Instanzen im Vergleich zum MILP auf die beschränkte Laufzeit zurückzuführen.

Im Beispiel weichen die Ergebnisse der Algorithmenvarianten nicht stark vom MILP ab, dennoch ist zu erkennen, dass die RTC-Variante wesentlich langsamer konvergiert als die restlichen vorgeschlagenen Versionen. Sie ist zudem auch die Letzte, die eine gültige Lösung für die Instanz findet.

Auch beim Random-Ansatz kann die verlangsamte Konvergenz beobachtet werden, jedoch reicht hier die Laufzeit von 30 Minuten aus, um eine gute Lösung zu finden. Interessant ist, dass die beste gefundene Lösung für die Instanz sogar von der Random-Variante stammt. Dies lässt sich wahrscheinlich auf die erhöhte Diversität der Startpopulation zurückführen, welche es der Population erlaubt, sich sehr frei zu entwickeln. Wahrscheinlich wäre eine

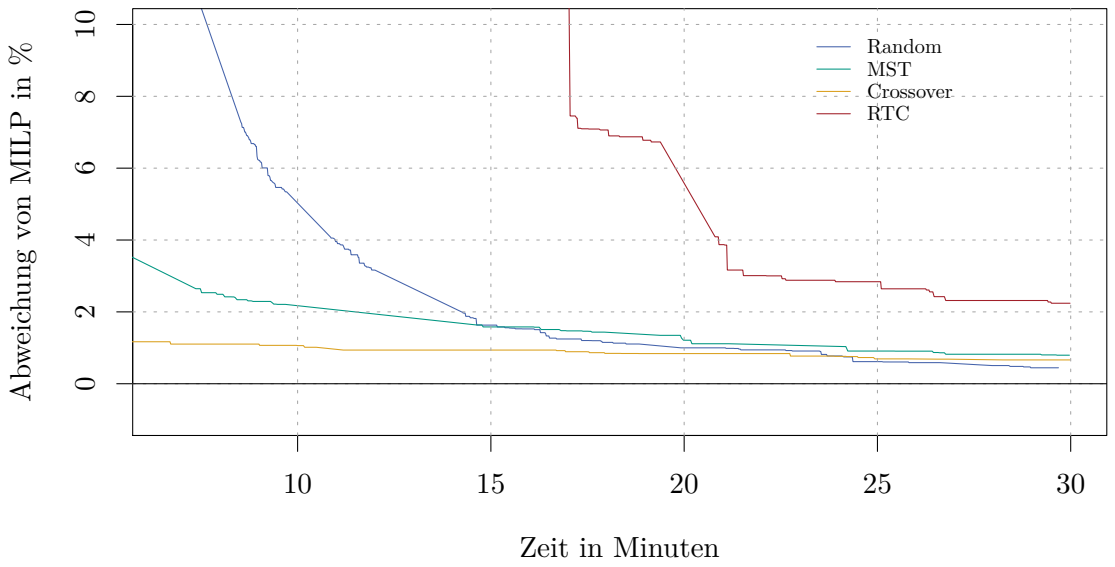


Abbildung 6.8: Abweichung der aktuell besten gefundenen Lösung im Verhältnis zur MILP-Lösung nach einer Stunde.

Version mit einer zufälligen initialen Population bei längeren verfügbaren Laufzeiten im Durchschnitt besser als eine MST-initialisierte.

6.2.2 Vergleich zur MST-Lösung

Um nochmals den Vorteil eines genetischen Algorithmus im Gegensatz zu einem Spannbaumansatz zu unterstreichen, wird in diesem Abschnitt für jede betrachtete Instanz eine Spannbaum Lösung berechnet.

Es wird kein eigener Algorithmus entworfen, sondern vielmehr wird für jede Instanz der minimale Spannbaum berechnet, die Substation Verkabelung entfernt und die Kabeltypen angepasst, um die Flussbedingung zu erfüllen. An dieser Stelle ist anzumerken, dass dadurch in vielen Fällen sogar ungültige Lösungen erzeugt werden, da die Kapazitätsbedingung nicht behandelt wird.

Natürlich ist es nicht als tatsächlicher Vergleich mit entwickelten MST-basierten Algorithmen zu betrachten, aber es gibt einen Eindruck, in welchen Größenordnungen sich die Kostenunterschiede aufhalten.

Hierzu ist Abbildung 6.9 zu betrachten. Ähnlich wie in Abschnitt 6.2 werden hier die durchschnittlichen Abweichungen von der Lösung des Referenzalgorithmus in Prozent angegeben. Gruppiert ist wiederum nach den Turbinenzahlen der Instanzen.

Wie erkennbar ist, ist sogar die RTC-Version nach einer halben Stunde Berechnungszeit besser als die vorgeschlagene MST-Version. Mit fast 15 % niedrigeren Kosten bei 200–300 Turbinen schneidet der genetische Algorithmus gut ab. Dieses Minimum folgt wahrscheinlich aus der Güte der MST-Lösung im Verhältnis zur optimalen Lösung der betrachteten Instanz. In kleinen Instanzen ist die MST-Lösung eine akzeptable Lösung, sofern diese valide ist. In manchen Fällen sind Teile der Verkabelungen dieser Instanzlösungen sehr ähnlich zur Lösung

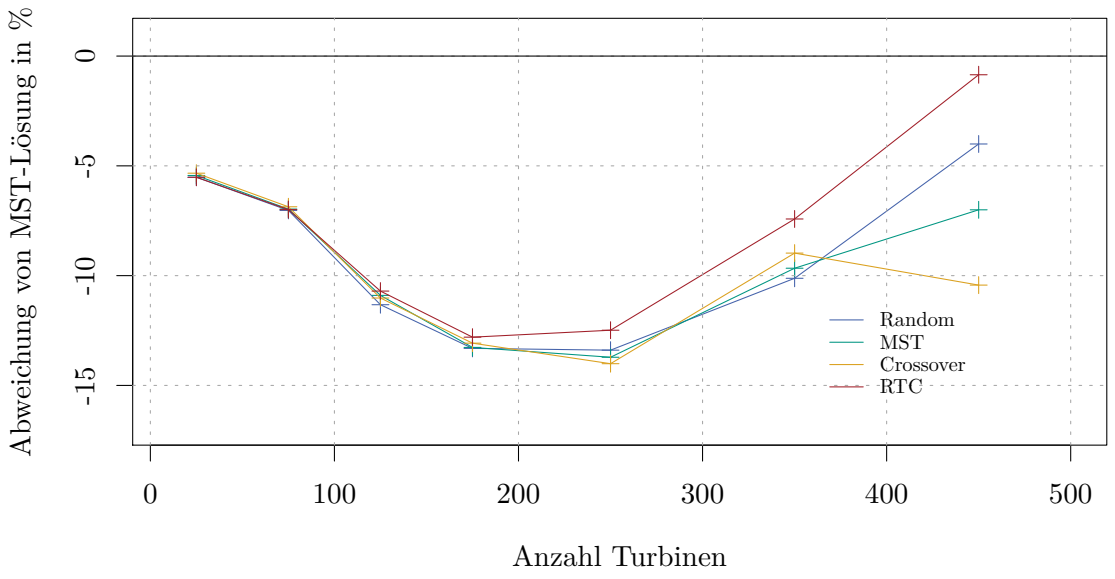


Abbildung 6.9: Mittelwert der Abweichung der Kosten von der MST-Lösung in % im Verhältnis zu Intervallen von Turbinen der Instanzen. Die Abweichungen sind wiederum bei kleineren Instanzen in Intervalle von 50 Turbinen eingeteilt und ab 200 Turbinen in Schritten von 100 zusammengefasst.

des genetischen Algorithmus. Dies ist aber bei steigender Instanzgröße zunehmend weniger der Fall. Da der vorgestellte genetische Algorithmus bei größeren Instanzen schlechter abschneidet, nimmt die relative Abweichung zur MST-Lösung bei höheren Turbinenzahlen wieder ab.

Im nächsten Abschnitt werden die Ergebnisse der Untersuchung der verschiedenen Versionen kurz zusammengefasst.

6.2.3 Zusammenfassung der Ergebnisse

Die Standardversion MST scheint konstant gute Ergebnisse für jede Art von Windfarm zu liefern. Jedoch können bestimmte Veränderungen für bestimmte Arten von Instanzen eine Verbesserung der Performance bewirken. Für kleine Instanzen ist es beispielsweise ratsam, die Anfangspopulation zufällig zu initialisieren. Dadurch wird gewährleistet, dass sich die Population beliebig entwickeln kann. Für sehr große Instanzen schneiden jedoch die Crossover- und die MST-Version am besten ab.

Vielversprechend scheint daher auch eine Kombination aus zufälliger Initialisierung mit Crossover zu sein. Dies könnte in Zukunft noch getestet werden.

Um einen Eindruck der Visualisierung der Ergebnisse des Algorithmus zu bekommen, wird im nächsten Abschnitt eine mögliche Variante vorgestellt.

6.3 Visualisierung einer berechneten Instanz

In diesem Abschnitt wird gezeigt, wie eine visuelle Darstellung der Ausgabe des Programms aussehen kann. Dazu ist in Abbildung 6.10 eine Windfarm aus Testset T3 mit 9 Substations

und 112 Turbinen zu sehen. Die Kantenbeschriftungen stellen die verwendeten Kabeltypen für die verschiedenen Kanten dar. Die Kosten der dargestellten Lösung sind im Verhältnis zur MILP-Lösung um circa 0,5% niedriger. Auffällig ist, dass der mit Abstand am meisten genutzte Kabeltyp der Kleinste ist. Das ist auf die niedrigeren Kosten zurückzuführen. Oft werden die Kanten so gewählt, dass nur das letzte Kabel zur Substation einen größeren Kabeltypen verwendet. Dies ist auch der Grund, warum ein MST-Ansatz nicht optimal für dieses Problem ist. Die kürzesten Kanten sind nicht immer die besten, da die Kabeltypen der anderen Kanten im Circuit von der Platzierung der Kabel abhängen. So kann es durchaus vorkommen, dass eine längere Kante gewählt wird, wodurch der Kabeltyp anderer Kanten verringert und im Endeffekt die Kosten gesenkt werden.

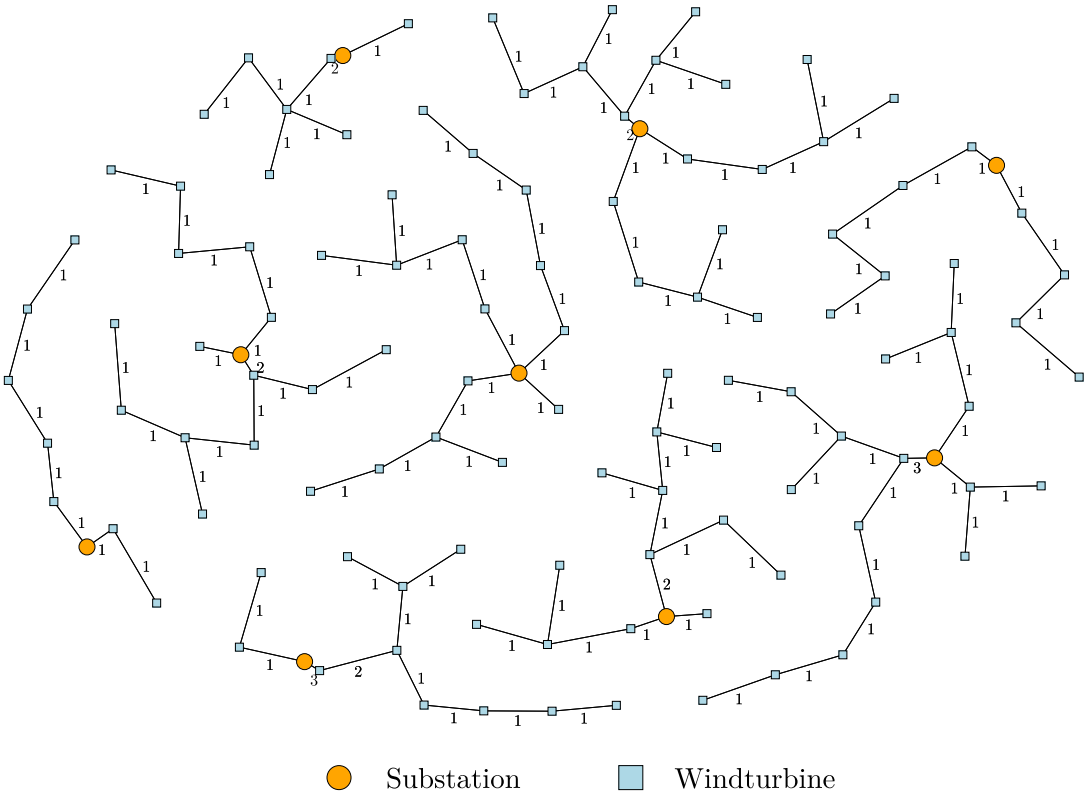


Abbildung 6.10: Visualisiertes Ergebnis der Berechnung des genetischen Algorithmus nach 10 Minuten auf einer Instanz mit 9 Substations und 112 Windturbinen. Die Kantenbeschriftungen bezeichnen den benutzten Kabeltyp wie in Tabelle 2.1 beschrieben.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Algorithmus zur Berechnung eines Kabellayouts mit minimalen Kosten für Windfarmen vorgestellt. Da diese Art von Problem in annehmbarer Zeit nicht optimal gelöst werden kann, wurde auf das Konzept eines genetischen Algorithmus gesetzt. Es wurden mehrere Varianten des Algorithmus auf ihre Performance auf verschiedenen Benchmark Instanzen getestet. Diese Varianten unterscheiden sich in verwendeten Operatoren, sowie verwendeter Initialisierung. Zudem wurde untersucht, welche Größe der Population für Instanzen von 10–500 Windturbinen, die besten Ergebnisse verspricht. Um die Performance bewerten zu können, wurde ein MILP-basierter Algorithmus als Referenz genutzt. Es wurde deutlich, dass verschiedene Versionen des Algorithmus unter gewissen Bedingungen verschieden gut abschneiden.

Für kleinere Instanzen (10-200 Turbinen) liefert eine Version mit zufällig generierter Anfangspopulation und ohne Crossover-Operator die besten Lösungen. Die Ergebnisse übertreffen in dieser Turbinenreichweite, sogar mit weniger verfügbarer Laufzeit, im Durchschnitt die Lösung des Referenzalgorithmus. Ab 300 Turbinen schneidet eine Version mit kontrollierter Initialisierung und eingesetztem Crossover-Operator besser ab. Jedoch können für Instanzen über 200 Turbinen nur vereinzelt Lösungen erzielt werden, die kostengünstiger als die Referenzlösung sind. Dies hat in erster Linie zwei Gründe. Zum einen reicht die Laufzeit bei großen Instanzen oft nicht aus, um die Population vollständig zur Konvergenz zu bringen. Zum anderen wird die Anzahl der lokalen Optima des Lösungsraums mit ansteigender Instanzgröße erhöht. Dies erhöht die Wahrscheinlichkeit zu einem schlechteren lokalen Optimum zu konvergieren und dadurch eine Lösung zu erhalten, die bezüglich der Kosten schlechter abschneidet.

Es wurde nur eine Art von Problemstellung betrachtet. Diese kann für zukünftige Windfarmplanung noch stark erweitert werden. Beispielsweise bestünde die Möglichkeit, variable Substationpositionen zu betrachten, was die Kosten der Verkabelung nochmals senken könnte. Darüber hinaus wäre es möglich, verschiedene Arten von Windturbinen zuzulassen, um für eine gegebene Windfarm auch die optimalen Turbinentypen zu wählen.

Natürlich gibt es Möglichkeiten, den Algorithmus sowie das Programm selbst zu verbessern. Durch gute Programmierkenntnisse kann sich die Performance des Programms sicher erhöhen lassen. Darüber hinaus ist der vorgestellte Rekombinationsoperator sehr naiv gewählt und könnte möglicherweise durch einen clusterbasierten Ansatz verbessert werden. Eine weitere Idee zur Verbesserung wäre, die Frequenz der Mutationen pro Iteration dynamisch, in Relation zur Instanzgröße, anzupassen. So bestünde die Möglichkeit, bei

größeren Instanzen mit wesentlich weniger Iterationen zu einem guten Ergebnis zu kommen. Da genetische Operatoren wie Mutation und Rekombination gut parallelisierbar sind, würde sich eine Umsetzung dieser auf mehreren Kernen der CPU, oder gar GPU, stark auf die Performance auswirken.

Literaturverzeichnis

[bur] *4C Offshore*. http://www.4coffshore.com/offshorewind/, Accessed: 2017-04-23.

[CGJ+14] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein und P. Mutzel.: *The Open Graph Drawing Framework (OGDF)*. Chapter 17 in: R. Tamassia (ed.), Handbook of Graph Drawing and Visualization, CRC Press, 2014. http://www.ogdf.net.

[CKMO] C.Berzan, K.Veeramachaneni, J. McDermott und U. O'Reilly: *Algorithms for Cable Network Design on Large-scale Wind Farms*.

[DOEE] S. Dutta und T. J. Overbye: *A clustering based wind farm collector system cable layout design*. Seiten 1–6, 2011 IEEE.

[ES03] Agoston E. Eiben und J. E. Smith: *Introduction to Evolutionary Computing*. Springer Verlag, 2003, ISBN 3540401849.

[GLWRT12] F. M. Gonzalez-Longatt, P. Wall, P. Regulski und V. Terzija: *Optimal Electric Network Design for a Large Offshore Wind Farm Based on a Modified Genetic Algorithm Approach*. IEEE Systems Journal, 6(1):164–172, March 2012, ISSN 1932-8184.

[Hal16] Josh Halliday: *Second phase of world's biggest offshore windfarm gets go-ahead*, 2016. https://www.theguardian.com/environment/2016/aug/16/hornsea-project-two-windfarm-second-phase-grimsby.

[Har95] Georges Harik: *Finding Multimodal Solutions Using Restricted Tournament Selection*. In: *Proceedings of the Sixth International Conference on Genetic Algorithms*, Seiten 24–31. Morgan Kaufmann, 1995.

[Hol92] John H. Holland: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992, ISBN 0262082136.

[HP2] *Hornsea Project Two Offshore Wind Farm*. http://www.4coffshore.com/windfarms/hornsea-project-two-gb-uk1u.html, Accessed: 2017-04-23.

[JR04] Raja Jothi und Balaji Raghavachari: *Revisiting Esau-Williams' Algorithm: On the Design of Local Access Networks*. In: *IN PROC. 7TH INFORMS TELECOMMUNICATIONS CONF*, Seiten 104–107, 2004.

[Leh16] Sebastian Lehmann: *Simulated Annealing-Based Heuristics for Wind Farm Cabling Problems*. Seiten 19–20, 38–40, May 2016.

[LRWW17] Sebastian Lehmann, Ignaz Rutter, Dorothea Wagner und 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)*, Seiten 203–215. ACM Press, 2017. https://doi.org/10.1145/3077839.3077843.

[LYX09] H. Lingling, F. Yang und G. Xiaoming: *Optimization of electrical connection scheme for large offshore wind farm with genetic algorithm.* In: *International Conference on Sustainable Power Generation and Supply*, Seiten 1–4, 2009.

[OB15] Peter Overbury und Luc Berthouze: *Using novelty-biased GA to sample diversity in graphs satisfying constraints.* In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, Seiten 1445–1446. ACM, 2015.

[Wei15] Karsten Weicker: *Evolutionäre Algorithmen*, 2015, ISBN 978-365-80995-8-9. `http://dx.doi.org/10.1007/978-3-658-09958-9`.

[ZCB09] M. Zhao, Z. Chen und F. Blaabjerg: *Optimisation of electrical system for offshore wind farms via genetic algorithm.* IET Renewable Power Generation, 3(2):205–216, 2009, ISSN 1752-1416.

[ZCH06] M. Zhao, Z. Chen und J. Hjerrild: *Analysis of the Behaviour of Genetic Algorithm Applied in Optimization of Electrical System Design for Offshore Wind Farms.* In: *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, Seiten 2335–2340, 2006.

Anhang

Im Anhang werden einige Abbildungen aufgeführt, die den Evaluationsteil 6 ergänzen. Zuerst werden zur Vollständigkeit die Abbildungen A.1-A.4 zur Untersuchung der Populationsgröße dargestellt. Danach werden die durchschnittlichen Abweichungen zum MILP der verschiedenen Algorithmen, die in Abschnitt 6.2 vorgestellt wurden, betrachtet. Diese werden in Abbildung A.5 in Relation zur Dichte der Windfarm Instanzen gestellt. Zuletzt wird in Abbildung A.6, genau wie in Abbildung 6.8, die Konvergenzgeschwindigkeiten anhand einer Instanz überprüft, jedoch im Verhältnis zur entsprechenden MST-Lösung anstatt der MILP-Lösung.

Populationsgröße

Die Ergebnisse der Tests nach dem Endergebnis mit Populationsgröße in der Reichweite von 1–20 Individuen waren auf jeder der drei Testinstanzen recht unaussagekräftig. Das folgt daraus, dass die Populationsgrößen in diesen Bereichen keine Auswirkung auf die Laufzeitperformance der Instanzen hatte. In Abschnitt 6.1 konnte man sehen, dass bei großen Instanzen und großen Populationsgrößen schlechtere Ergebnisse erzielt wurden, da 30 Minuten zur Berechnung nicht ausreichen, um zu einem guten Ergebnis zu konvergieren. Dies ist bei kleinen Größen nicht der Fall. Auch bei größeren Populationsgrößen auf Instanz 2 und 3 war der Einfluss der Individuenzahl nicht groß genug, um die Ergebnisse zu verschlechtern. Die Schwankung der Kosten, wie sie in den Abbildungen zu sehen ist, ist auf die zufällige Funktionsweise der genetischen Operatoren zurückzuführen.

Abweichung von MILP-Lösung im Bezug auf Dichte der Instanz

Wie in Abbildung A.5 zu sehen, fallen die Ergebnisse bei Dichte 0.45 schlechter aus als bei anderen Dichten. Dies resultiert in erster Linie daraus, dass die einzigen Instanzen, die diese Dichte besitzen, aus Testset T4 stammen. Diese sind also die größten betrachteten Instanzen und es ist, wie im Evaluationsteil besprochen, klar, dass diese aufgrund von Zeitmangel schlechter abschneiden. Zudem ist auffällig, dass die MST-Version des Algorithmus, in Bezug auf Veränderung der Dichte, die konstantesten Ergebnisse liefert. Wie in der Abbildungsbeschreibung erwähnt, ist das Minimum der Graphen im Intervall von 0.8 bis 0.9 zu sehen. Dies stammt vom erhöhten Anteil kleiner Windfarmen in dieser Reichweite und von der konstanten Substationkapazitätserhöhung. In dieser Dichtenreichweite hat die Auflockerung der Dichte am meisten Einfluss, da so andere Zuteilungsmöglichkeiten für Turbinen zu Substations ermöglicht werden.

Abbildung A.1: Ergebnisse der Berechnung mit verschiedenen Populationsgrößen auf Instanz 2. Diese wurden in Schritten von einem Individuum von 1 bis 20 erhöht.

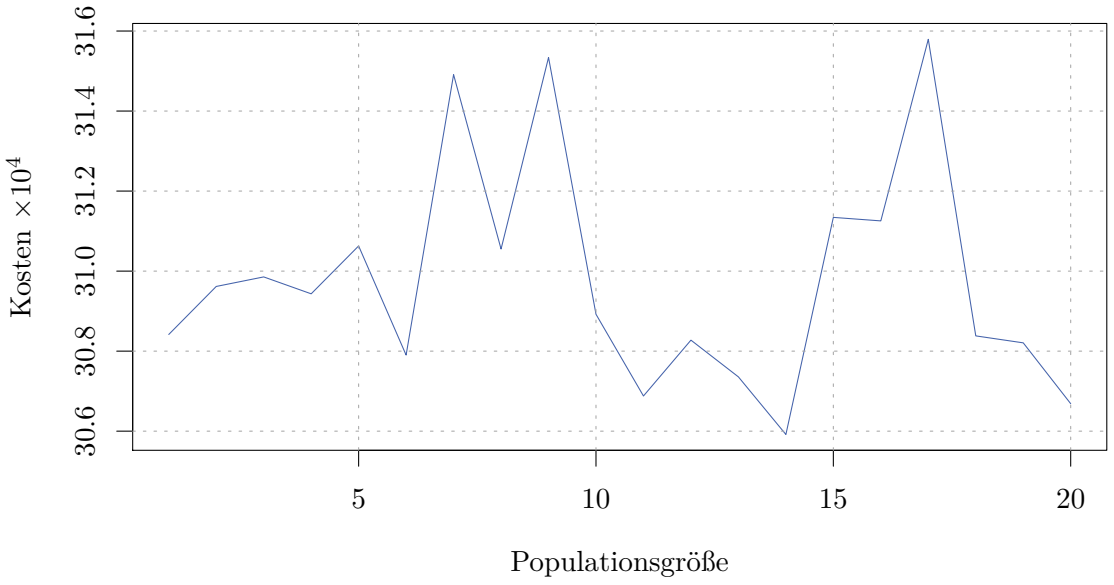


Abbildung A.2: Ergebnisse der Berechnung mit verschiedenen Populationsgrößen auf Instanz 3. Auch in dieser Abbildung wurden Schritte von einem Individuum betrachtet.

Abbildung A.3: Ergebnisse der Gegenüberstellung von Populationsgröße und Kosten der besten Lösung nach 30 Minuten auf Testinstanz 3. Die Durchläufe erfolgten in Schritten von 5 Individuen.

Abbildung A.4: Ergebnisse der Berechnung auf Instanz 4 mit verschiedenen Populationsgrößen. Größenschritte von einem Individuum.

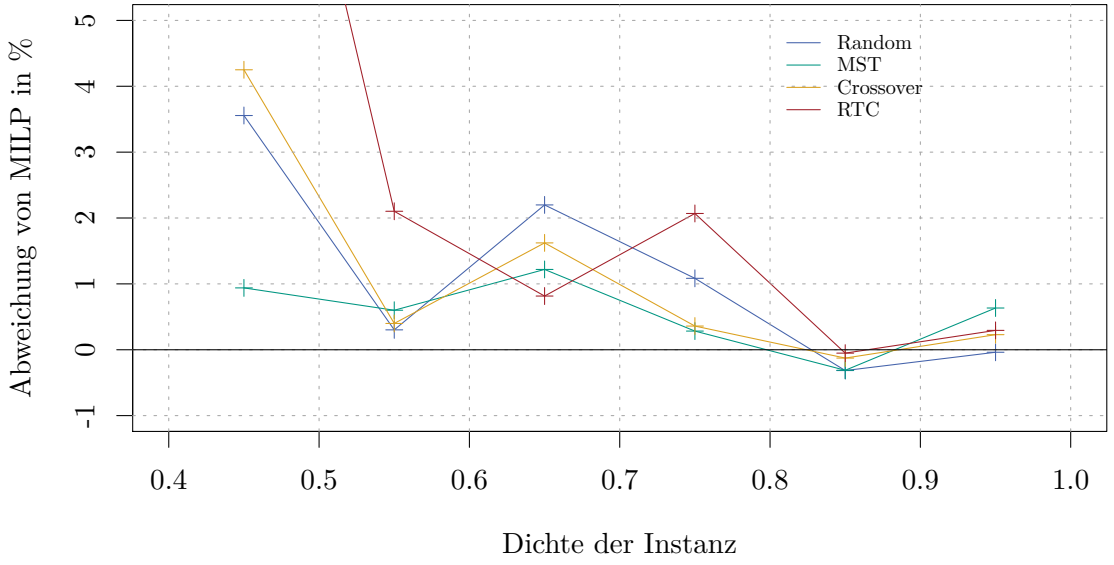


Abbildung A.5: Abweichung der Kosten von der Referenzlösung des MILP in Abhängigkeit von der Tightness der Instanz. Es wurden die Mittelwerte in Intervallen von $0,1$ gebildet. Zu erkennen ist ein Tiefpunkt bei 0.85, welcher wohl daher stammt, dass kleine getestete Instanzen nur eine Dichte von 0.7 oder höher haben können und der Algorithmus bei kleineren Windfarmgrößen besser abschneidet. Der Anteil von kleinen Windfarmen ist also im Breich von $0.7 - 1$ höher als davor.

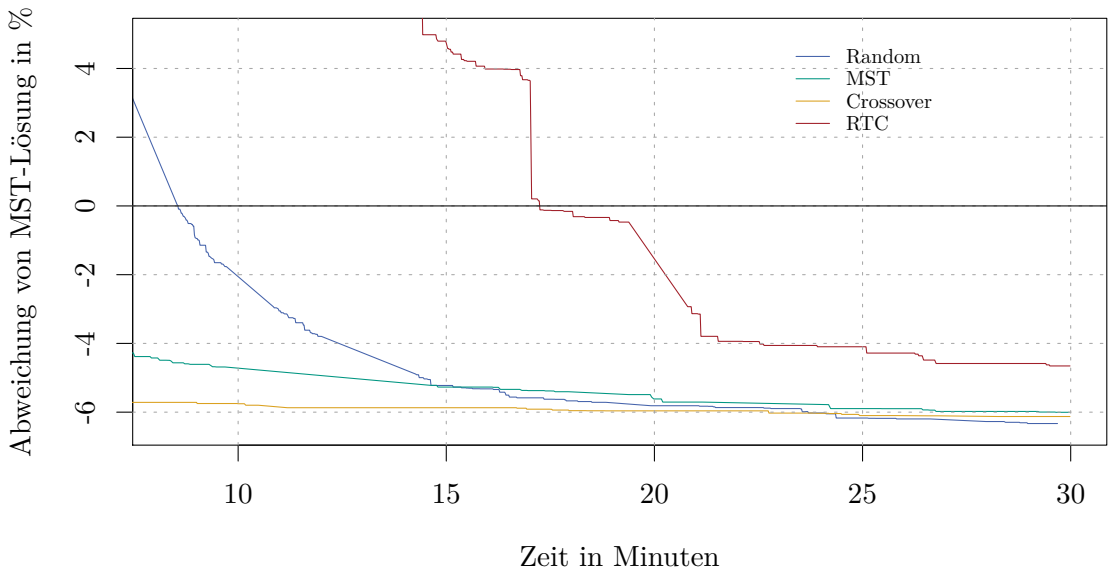


Abbildung A.6: Darstellung der Konvergenzgeschwindigkeit der verschieden Varianten im Verhältnis zur MST-Lösung. Instanz enthält 315 Turbinen und 21 Substations. MST-Lösung ist ohne jeglichen Substationkapazitätsbedingung berechnet, was an sich invalide ist. Zu erkennen ist außerdem die erwartete langsamere Konvergenz von Random und RTC-Variante. Schließen kann man daher, dass RTC bei größeren Instanzen vor allem deswegen schlechter ist, da die Konvergenz verzögert ist und daher 30 Minuten nicht zum Konvergieren reichen.

Abweichung von MST-Lösung in Bezug auf die Berechnungszeit

Abbildung A.6 ist äquivalent zur Abbildung 6.8, die einzige Anpassung ist, dass mit der MST-Lösung statt mit der MILP-Lösung verglichen wird. Wie man sehen kann, wird die MST-Lösung sogar vom RTC-Ansatz, der im Schnitt die schlechtesten Lösungen hervorbringt, nach circa 18 Minuten übertroffen. Wie im Evaluationsteil erwähnt, ist die MST-Lösung im Fall vieler Instanzen nach unseren Kriterien nicht einmal eine valide Lösung, da die maximale Substationkapazität oft überschritten wird. Dennoch liefert der vorgestellte Algorithmus im Durchschnitt für alle Instanzgrößen bessere Ergebnisse.