FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Metaheuristics for the Regenerator Location Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Softwareengineering und Internet Computing

eingereicht von

Peter Jahrmann
Matrikelnummer 0425523

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: o.Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl

Wien, 20.03.2013

(Unterschrift Verfasserin) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Peter Jahrmann
Bukovicsgasse 33, 1220 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Danksagung

Ich möchte mich an dieser Stelle bei meinem Betreuer Univ.-Prof. Dr. Günther Raidl, der mich stets durch konstruktive Rückmeldungen und wertvolle Hinweise bei der Bearbeitung des Themas unterstützt hat, für die gute Zusammenarbeit recht herzlich bedanken.

Ebenso danke ich meinen Eltern Karl und Monika und besonders meinem Bruder Klemens, sowie meiner Freundin Sophie und all meinen Freunden für den großen Rückhalt und ständige Motivation, die sie mir in der Zeit der Erstellung dieser Arbeit gegeben haben.

Besonderer Dank gilt auch meinen Tennis- und Tanzsporttrainern Eva und Dieter, sowie Ingrid und Ludwig, die mir seit vielen Jahren zur Seite stehen und mich sowohl sportlich, als auch durch ihre Vorbildwirkung in zahlreichen anderen Belangen ständig weiterentwickelt haben.

Abstract

During the past years optical networks have proven to be the best technology in terms of high-speed data transmission. They offer high transmission rates together with a high bandwidth. Moreover optical fiber is more robust against interferences than copper cable. Nevertheless the quality of an optical signal deteriorates as it travels through the network depending on the covered distance due to impairments in the fiber. Therefore regenerators are installed at certain nodes in the network for a periodical signal regeneration, ensuring that all nodes can communicate with each other flawlessly. Since these regenerators are usually considered to be expensive the regenerator location problem (RLP) demands for a minimum cardinality subset of regenerators to achieve the required condition. A typical instance of the RLP is given by an arbitrary network graph $G = (V, E, d)$ and a parameter $d_{\max}$ indicating the maximum distance a signal can travel without loss of quality. Since it is an NP-hard problem the utilization of heuristical optimization methods is reasonable to obtain solutions for problems with arbitrary sizes. In literature a branch-and-cut algorithm was presented to obtain exact solutions for the RLP. Another work describes the application of a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic with a randomized construction heuristic and a 2x1 local search procedure. In the course of this thesis advanced selection strategies for regenerator placement have been developed and utilized in new construction and local search based procedures. For this purpose special node sets like cliques and independent sets exploit the structure of the underlying network graph more efficiently. All developed heuristics use the *communication graph* $M = (V, E')$, which is the result of a graph transformation procedure applied to the original graph G. In M the information about the actual need of regenerators is presented in an easier way, which correlates with the existence of not directly connected node pairs $\overline{E'}$. The idea behind the usage of cliques is to combine them by the introduction of regenerator nodes until there is only one clique left corresponding to the whole graph. In this case the chosen regenerators form a feasible solution. An independent set represents the information of several not connected node pairs. Therefore regenerators are placed in the graph trying to neighbor as many nodes of different independent sets as possible until there are no independent sets left.

The heuristic strategies were employed in a GRASP framework as well as in a Variable Neighborhood Search (VNS). They were tested on two different test sets and the results were compared to those of the GRASP heuristic from the literature. The developed GRASP heuristics using clique and independent set strategies obtained most of the best known solutions. Also the VNS heuristics performed quite good on the test sets and reached similar results as the GRASP from the previous literature.

Kurzfassung

In modernen Telekommunikationsnetzwerken haben sich im Laufe der letzten Jahre vor allem Glasfaserverbindungen durch ihre hohe Bandbreite und Übertragungsgeschwindigkeit als Grundlage für eine schnelle Informationsübertragung etabliert. Obwohl sie im Vergleich zu den Kupferleitungen weniger anfällig für Störeinflüsse sind, treten trotzdem mit zunehmender Kabellänge Qualitätsverluste bei der Signalübertragung auf. Daher ist es notwendig an speziellen Knoten eines optischen Netzwerks so genannte Regeneratoren zu installieren, damit durch Signalregenerierung eine zuverlässige Kommunikation zwischen allen Knotenpaaren sichergestellt ist. Das Regenerator Location Problem (RLP) fordert dabei die Platzierung von so wenigen Regeneratoren wie nur möglich, um diese Bedingung zu erfüllen. Eine typische Instanz ist in Form eines Netzwerkgraphen $G = (V, E, d)$ und eines Parameters $d_{\max}$ gegeben, der die maximale Distanz, die ein optisches Signal zurücklegen kann beschreibt. Beim RLP handelt es sich um ein NP-schwieriges Problem, das den Einsatz von heuristischen Verfahren nahelegt, um auch für große Instanzen Lösungen in annehmbarer Zeit zu berechnen. In der Literatur wurden bisher exakte Verfahren in Form eines Branch-and-Cut Ansatzes betrachtet, sowie die Anwendung einer Greedy Randomized Adaptive Search Procedure (GRASP) Metaheuristik mit randomisierter Konstruktionsheuristik und so genannter 2x1 Move Nachbarschaftsstruktur vorgestellt. Im Zuge dieser Arbeit werden weitere Mechanismen vor allem im Zusammenhang mit speziellen Knotenmengen im Graphen wie z.B. Cliques oder Independent Sets erforscht und in Konstruktionsheuristiken, sowie Nachbarschaftsstrukturen für die lokale Suche eingesetzt. Die entwickelten Heuristiken nutzen dabei den *Communication Graph* $M = (V, E')$, der den Bedarf an tatsächlich benötigten Regeneratoren in einfacherer Form darstellt. Einerseits ist die Idee bestehende Cliques in M durch die Einführung von Regeneratoren immer weiter zusammenzufassen, bis nur noch eine einzige übrig bleibt, die dem gesamten Graphen entspricht, wobei die gefundenen Regeneratoren eine gültige Lösung formen. Andererseits werden Independent Sets als fehlende Kommunikationsmöglichkeit zwischen mehreren Knotenpaaren identifiziert und diese durch Regeneratoren, die möglichst viele Knoten unterschiedlicher Independent Sets benachbarn immer weiter reduziert, bis ebenfalls eine gültige Lösung entstanden ist.

Die entwickelten Heuristiken wurden sowohl in GRASP, als auch in Variable Neighborhood Search (VNS) Heuristiken eingebettet und im Zuge der Arbeit mit dem in der Literatur beschriebenen GRASP Ansatz verglichen. Es stellt sich heraus, dass die entwickelten GRASP Heuristiken durch die vergleichsweise kurze Laufzeit die besten Ergebnisse erzielen und die VNS Heuristiken ebenso in der Lage sind mit der in der Literatur vorgeschlagenen Lösung mitzuhalten.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

In den letzten Jahren hat die Entwicklung im Bereich der digitalen Kommunikation und die Vielfalt an web-basierten Hilfsleistungen zur Erleichterung des Alltags neue Höhen erreicht. Eine Fülle von neuen bzw. weiterentwickelten Services wie beispielsweise Video-Sharing, TV on Demand, soziale Netzwerke, Realtime Online Gaming und mobiles Internet in allen Facetten haben auch neue Herausforderungen an die Telekom Service Provider gestellt, um den Bedarf an Datenübertragungsrate und Übertragungsgeschwindigkeit zu decken. Um mit dieser Entwicklung Schritt zu halten, werden in modernen Telekommunikationsnetzwerken heutzutage vor allem aus Lichtleitern bestehende Glasfaserkabel zur schnellen Informationsübertragung eingesetzt. Beispielsweise besitzt der OC-192 Standard eine Übertragungsrate von 10 Gb/s, hingegen die weiterentwickelte OC-768 Übertragung bereits 40 Gb/s, oder die bisher letzte Errungenschaft in der Wavelength Division Multiplexing (WDM) Technik OC-3072 schon 160 Gb/s. Eine detaillierte Beschreibung dieses Standards ist in [31] zu finden. Neuere Entwicklungen auf dem Gebiet der Dense Wavelength Division Multiplex (DWDM) Technik versprechen die Datenraten sogar bis in den Terabit Bereich zu heben. Dies geschieht dadurch, dass die im Lichtleiter übertragenen Wellenlängen sehr dicht beeinander liegen, d.h. der Frequenzabstand so gering wie möglich gehalten wird und so mehr Information gleichzeitig durch die Glasfaser geschickt werden kann. Allerdings müssen auch leistungsstarke Filter und zusätzliche qualitätserhaltende Maßnahmen eingesetzt werden, um mit den geringen Frequenzabständen fehlerlos umzugehen. Ein Ein- und Ausblick in die DWDM Technik und den damit verbundenen Möglichkeiten wird in [25] und [37] gegeben.

Die optische Übertragung stellt heutzutage den State of The Art für leistungsfähige Telekommunikationsnetzwerke dar und ist der früheren elektrischen Übertragung mit Kupferleitungen in zahlreichen Punkten überlegen. Diese schließen die Übertragungsrate, Übertragungsreichweite, geringere Anfälligkeit gegenüber von Störeinflüssen, sowie niedrigere Kosten für den Netzwerkbetreiber mit ein. Trotzdem treten auch bei Lichtwellenleitern mit zunehmender Kabellänge Dämpfungen und Qualitätsverluste auf, sodass ein Signal nur über eine gewisse Streckenlänge problemlos übetragen werden kann. Für die darüber hinausgehende Übertragung muss das Signal mit Hilfe eines Regenerators neu aufbereitet werden. Regneratoren unterteilen sich dabei

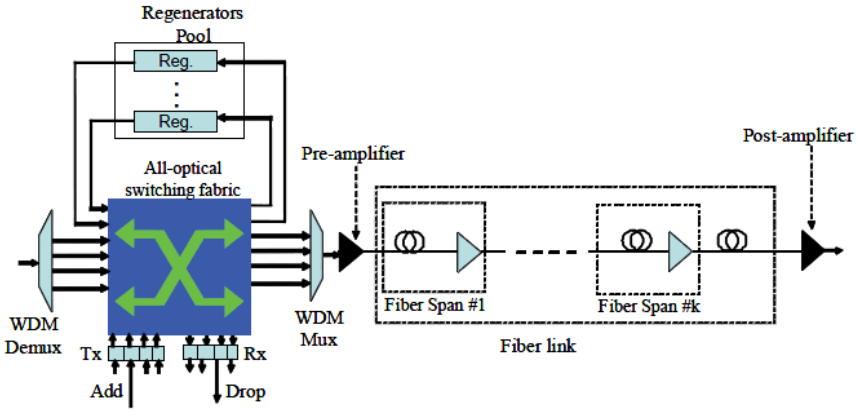


Abbildung 1.1: Aufbau einer Komponente in einem optischen Netzwerk (aus [28])

in drei Hauptklassen, wobei sich diese Unterteilung aus ihrer Funktionsweise ergibt. Während in Klasse 1 lediglich eine Wieder-Verstärkung des optischen Signals stattfindet, führen Regeneratoren der Klasse 2 sowohl eine Wiederverstärkung als auch eine Formwiederherstellung durch. In Klasse 3 werden alle Wiederherstellungsmaßnahmen aus Klasse 2 durchgeführt und zusätzlich eine zeitliche Neusynchronisation vorgenommen. Durch die technische Komplexität von Regeneratoren der Klassen 2 und 3 sind solche Geräte in der Anschaffung und Installation mit höheren Kosten verbunden. Um eine reibungslose Kommunikation im Netzwerk zu gewährleisten, muss ein beliebiger Knoten jederzeit in der Lage sein ein Signal unverfälscht und ohne nennenswerte Qualitätsverluste zu jedem anderen Knoten im Netzwerk zu schicken. Daher muss für jedes Knotenpaar im Netzwerk ein Pfad existieren, der mit genügend Regeneratoren (aufgrund der hohen Quality of Service Anforderungen üblicherweise der Klasse 3) ausgestattet ist, um eine solche Übertragung zu gewährleisten.

Durch den Umstand, dass Regeneratoren im Allgemeinen teuer sind, ergibt sich die Forderung nach der Platzierung von so wenigen Komponenten wie nur möglich, um das gesamte Telekommunikationsnetzwerk zu versorgen. Diese auf den ersten Blick als einfach eingestufte Aufgabe wird in der Praxis bereits bei kleinen Instanzen von lediglich 10 Knoten zu einer Herausforderung. Die Schwierigkeit erwächst vor allem aus dem unterschiedlichen Vernetzungsgrad von Knotenverbänden innerhalb eines realen Netzwerks, der sich aus geographischen und wirtschaftlichen Gegebenheiten ergibt. Dadurch sieht man sich bei der Platzierung von Regeneratoren stets einem zusammenhängenden, ungerichteten Graph gegenüber, bei dem es im Allgemeinen kein symmetrisches Muster an Knotenverbindungen gibt. Es bedarf daher der Formulierung geeigneter Hilfsmittel, die in der Lage sind mit unterschiedlichsten Graphen, seien es nun dichte oder dünne Graphen, umzugehen und in jedem Fall die Anzahl an benötigten Regeneratoren zur Versorgung des Netzwerks zu minimieren.

1.1 Netzwerkdesign

Das moderne Netzwerkdesign ist mit wachsenden Anforderungen hinsichtlich Übertragungsgeschwindigkeit und Durchsatzrate bei gleichzeitigem hohem Maß an Quality of Service konfrontiert. Es ist daher von entscheidender Wichtigkeit bereits beim Entwurf des Netzwerks entsprechende Maßnahmen zu berücksichtigen, um Fehler und Störungen bei der Übertragung so gering wie möglich zu halten. Durch Übetragungsbeeinträchtigungen, wie Dämpfung, Streuung und Überlagerung von nahe beeinanderliegenden Lichtleitern, kann ein optisches Signal nur über eine gewisse Streckenlänge $d_{\max}$ fehlerfrei übetragen werden. Für die darüber hinausgehende Übertragung ist die Verstärkung und Neukalibrierung mittels eines Regenerators erforderlich. Grundlegendes Ziel ist es dabei immer jedem Knoten aus dem Netzwerk einen fehlerfreien Informationsaustausch mit allen übrigen Knoten zu ermöglichen. Die Herangehensweise dieses Ziel zu erreichen, hat sich in mehrere Zweige, die jeweils unterschiedliche Ansätze verfolgen, aufgespalten.

So haben Mohan et al. in [30] bei gegebener Infrastruktur Routing-Algorithmen entwickelt, die einem Informationsrequest entweder statisch oder dynamisch den optimalen Kommunikationspfad mit der geringsten Fehlerquote zuweisen. Darüber hinaus stellen sie eine neue Multiplexing-Methode vor, um die Wellenlängen der Lichtleiter besser zu nutzen und Engpässe im Netzwerk gering zu halten. Auf diese Art und Weise wird versucht durch geschicktes Routing aufbauend auf ein gegebenes Netzwerk die verschiedenen Requests so zu organisieren, dass eine hohe Übertragungsqualität und barrierefreie Kommunikation gewährleistet wird.

Betrachtet man rein die Architektur von Telekommonikationsnetzwerken bei gegebenem, oder vernachlässigbarem Routing, wird versucht die Anzahl der Regeneratoren im Netzwerk bzw. die Anzahl der Regeneratorknoten zu minimieren. Diese Unterscheidung zwischen der Anzahl der Knoten und der Gesamtanzahl von Regeneratoren ist deswegen wichtig, da sich in der Literatur zwei verschiedene Ansätze entwickelt haben, die in ihrem Kern sehr ähnlich sind, aber jeweils ein anderes Optimierungsziel verfolgen. Die zugrundeliegenden Probleme sind das Regenerator Location Problem (im Weiteren als RLP bezeichnet) und das Regenerator Placement Problem (im Weiteren als RPP bezeichnet). Im RLP werden Netzwerkknoten als diskrete Lokalisierungspunkte von Regeneratoren betrachtet und als gültige Lösung für das Problem eine Menge an Knoten akzeptiert. Dabei wird ein Regenerator als "Regenerator pro Knotenverbindung" interpretiert und anhand eines einzigen Kriteriums – der Maximaldistanz einer fehlerlosen Übertragung ($d_{\max}$) – platziert. Anders ist dies beim RPP. Dort wird ein Regenerator zunächst als "Regenerator pro Lichtleiter" betrachtet und bei seiner Platzierung nicht nur ein einziges Kriterium miteinbezogen, sondern durchaus auch mehrere, wie z.B. bestehende Requests, Kapazitäten, etc. In [28] beschreiben Kuipers et al. diese Betrachtungsweise als dem in der Praxis bestehenden Szenario am ähnlichsten. Sie führen einen breiten RPP Begriff ein, bei dem die Gesamtzahl an verwendeten Regeneratoren im Netzwerk und nicht die Anzahl an Regeneratorknoten minimiert werden soll. Sie zeigen anhand eines einfachen Beispiels, dass die Minimierung der Regeneratorknoten in der Praxis nicht automatisch zu insgesamt weniger verwendeten Regeneratoren führt.

Im Beispiel in *Abbildung 1.2* (aus [28]) sind drei Requests $(s1, d1, \Delta)$, $(s2, d2, \Delta)$ und $(s3, d3, \Delta)$ für ein bestehendes Netzwerk gegeben. Dabei besitzt jede vorkommende Knoten-

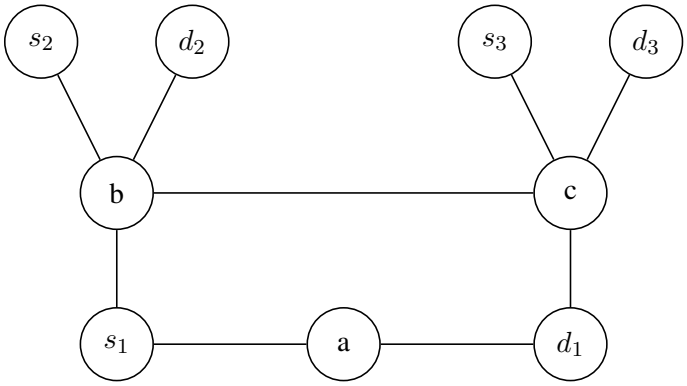


Abbildung 1.2: Netzwerk bei dem die Minimierung der Regeneratorknoten nicht zur minimalen Anzahl an Regeneratoren im Netzwerk führt (aus [28])

verbindung die Länge $\Delta = d_{\max}$. Minimiert man nun die Anzahl der Regeneratorknoten, so müssten in den Knoten b und c Regeneratoren platziert werden. Dafür würden im Gesamten vier Regeneratoren benötigt, nämlich für $(s1, d1, \Delta)$ und $(s2, d2, \Delta)$ jeweils ein Regenerator in b, sowie für $(s1, d1, \Delta)$ und $(s3, d3, \Delta)$ jeweils ein Regenerator in c. Minimiert man hingegen die Gesamtzahl an Regeneratoren im Netzwerk in Hinblick auf existierende Requests, so werden lediglich drei Regeneratoren benötigt. Diese sind einer in a für $(s1, d1, \Delta)$, einer in b für $(s2, d2, \Delta)$ und einer in c für $(s3, d3, \Delta)$.

Anhand dieses Beispiels ist die unterschiedliche Sichtweise auf die Platzierung von Regeneratoren zwischen dem RLP und RPP sehr anschaulich erklärt. Das RPP bietet durch die Miteinbeziehung von vordefinierten Requests eine viel granularere Sichtweise auf die spezielle Umsetzung eines Telekommunikationsnetzwerks. Weitere Arbeiten, in denen verschiedene Herangehensweisen an das RPP beschrieben werden, sowie verschiedene Optimierungskriterien neben der Streckenlänge beleuchtet werden, sind in [26], [36] und [4] zu finden. Das RLP hingegen betrachtet den Netzwerkdesignprozess viel allgemeiner und geht davon aus, dass jeder Knoten im Graphen sowohl als Regeneratorknoten verwendet werden kann, als auch in der Lage sein muss mit allen anderen Knoten zu kommunizieren. Dies führt dazu, dass nur noch ein einziges Kriterium über die optimale Versorgung des Netzwerks entscheidet, nämlich die maximale Distanz $d_{\max}$, die ein Signal ohne Qualitätsverluste zurücklegen kann. Die Erweiterung des RLP, damit es auch mit speziellen Problemfällen aus der Praxis umgehen kann, wird in *Kapitel 1.4* betrachtet.

1.2 Regenerator Location Problem

Das RLP wurde 2007 erstmals von Chen und Raghavan [7] vorgestellt, sowie später von Chen, Ljubic und Raghavan in [6] im Detail beschrieben. Es handelt sich dabei um ein kombinatorisches Optimierungsproblem. Eine Instanz des RLP ist durch einen ungerichteten Graph $G = \{V, E, D\}$ charakterisiert, wobei V die Menge der Knoten, E die Menge der Kanten und D die zugehörige Distanzmatrix bezeichnen. Die Werte innerhalb der Distanzmatrix D lassen

sich als positive Entfernungen

$$D_{v_i,v_j} \geq 0, \forall v_i, v_j \in V \times V \tag{1.1}$$

angeben. Darüber hinaus ist ein Parameter $d_{\max}$ gegeben, der die maximal zurücklegbare Distanz eines optischen Signals im Netzwerk bestimmt, bevor es wieder regeneriert werden muss. Aufgabe ist es nun eine Regeneratorknotenmenge R mit minimaler Kardinalität zu bestimmen, sodass zwischen jedem Knotenpaar $v_i, v_j \in V \times V$ ein Pfad existiert, dessen Länge entweder kleiner als $d_{\max}$ ist, oder der entsprechend viele Regeneratorknoten enthält. Da in einer Lösung R ausschließlich Knoten aus V vorkommen, gilt

$$R \subseteq V \tag{1.2}$$

und die Menge aller möglichen Lösungen W für eine Instanz des RLP lässt sich als

$$W = \mathcal{P}(V), \text{ mit } |W| = 2^{|V|} \tag{1.3}$$

definieren. Um eine bestimmte Lösung R als gültige Lösung zu klassifizieren, werden für alle Knotenpaare $v_i, v_j \in V \times V$ alle bestehenden Pfade

$$p_{n_{(v_i,v_j)}} = \{(v_i, v_k), (v_k, v_{k+1}), \ldots, (v_l, v_j)\} \tag{1.4}$$

mit $n = 1, \ldots, |E|$ und Zwischenknoten $v_k, \ldots, v_l \in V$ im Graphen betrachtet. Damit die Bedingung für eine gültige Lösung erfüllt ist, muss im trivialen Fall zwischen jedem Knotenpaar $v_i, v_j \in V \times V$ zumindest ein Pfad mit Länge $\leq d_{\max}$ existieren, also für die Summe der Distanzen

$$D_{v_i,v_k} + D_{v_k,v_{k+1}} + \ldots + D_{v_l,v_j} \leq d_{\max} \tag{1.5}$$

gelten. Falls kein solcher Pfad existiert, muss es zwischen jedem Knotenpaar $v_i, v_j \in V \times V$ zumindest einen Pfad p_n mit Zwischenknoten $r_x \in R, x = 1, \ldots, |R|$ geben, bei dem weder die Distanz zwischen dem Anfangs- und Endknoten und einem Regeneratorknoten D_{v_i,r_x}, D_{v_j,r_x}, noch die Distanz zwischen zwei aufeinanderfolgende Regeneratorknoten im Pfad D_{r_x,r_y} größer als $d_{\max}$ ist. Es muss für diesen Pfad daher die Bedingung

$$\underbrace{\{(v_i, v_{i+1}), \ldots, (v_{k-1}, v_k)\}}_{(D_{v_i,v_{i+1}} + \ldots + D_{v_{k-1},v_k}) \leq d_{\max}} \cup \underbrace{\{r_x\} \cup \ \ldots \ \cup \{r_y\}}_{D_{r_x,r_y} \leq d_{\max}} \cup \underbrace{\{(v_l, v_{l+1}), \ldots, (v_{j-1}, v_j)\}}_{(D_{v_l,v_{l+1}} + \ldots + D_{v_{j-1},v_j}) \leq d_{\max}} \tag{1.6}$$

erfüllt sein. Generell werden all jene Knotenpaare $v_i, v_j \in V \times V$ zwischen denen kein trivialer Pfad mit Länge $\leq d_{\max}$ besteht und für die die Einführung von Regeneratorknoten notwendig ist als NDC-Knotenpaare bezeichnet (NDC bedeutet „not directly connected“). Um festzustellen welchen Wert eine gültige Lösung R im Zuge der Optimierung besitzt, wird die Zielfunktion

$$f(R) = |R| \tag{1.7}$$

ausgewertet. Eine optimale Lösung R^* für eine bestimmte Instanz des RLP erfüllt daher immer die Bedingung

$$f(R^*) \leq f(R) \tag{1.8}$$

für alle zulässigen Lösungen R.

Flamini et al. haben in [15] bewiesen, was auch in [6] beschrieben wird, dass sowohl das RLP, als auch das RPP NP-schwierige Probleme sind. Das bedeutet es gibt vermutlich kein exaktes Verfahren, das für alle Instanzen in polynomieller Zeit in Bezug zur Problemgröße die optimale Lösung berechnet.

1.3 *Communication Graph*

Ist eine Instanz des RLP als Graph $G = \{V, E, D\}$ gegeben, so ist es oft aufwändig die tatsächlich benötigten Kantenverbindungen zu bestimmen. Daher haben Chen et al. [6] mit der Einführung des *Communication Graphen* eine wichtige Abstraktionsstufe geschaffen, die viele relevante Informationen von G kompakter abbildet. Dabei stellt der *Communication Graph* M einen ungerichteten Graph

$$M = \{V, E'\} \tag{1.9}$$

dar. Um den Originalgraph in den Communciation Graph umzuwandeln, kann eine schrittweise Transformationsprozedur angewendet werden. Diese sieht folgendermaßen aus:

1. Initialisierung eines leeren Graphen M mit Knotenmenge V und Kantenmenge E'
2. Die Menge der Knoten V aus G wird unverändert in M übernommen, also $V_M = V_G$
3. Anwendung eines All-Pair-Shortest-Path Algorithmus in G, um für jedes Knotenpaar $v_i, v_j \in V \times V$ die tatsächlich kürzeste Distanz $D^*_{v_i,v_j}$ zwischen den beiden Knoten zu berechnen
4. Überprüfe alle minimalen Distanzen D^* und füge eine Kante (v_i, v_j) der Kantenmenge E' hinzu, falls $D^*_{v_i,v_j} \leq d_{\max}$

Als Ergebnis des Transformationsprozesses wird der von G abgeleitete *Communication Graph* M gewonnen, der zwar die gleiche Knotenmenge V besitzt, sich jedoch meistens in der Menge von Kanten E' von G unterscheidet und ohne eine Distanzmatrix D auskommt. Durch die Eliminierung aller Kanten, die länger als $d_{\max}$ sind, bzw. durch die Einführung von Kanten zwischen Knotenpaaren deren kürzester Verbindungspfad eine Länge kleiner oder gleich $d_{\max}$ hat, wird die tatsächlich zugeordnete Distanz zu einer Kante überflüssig. Es verbleibt daher lediglich die Information, ob zwei Knoten bereits in der Lage sind ohne Beeinträchtigungen miteinander zu kommunizieren, oder ob sie der Menge der NDC-Knotenpaare zuzurechnen sind. Anhand des *Communication Graphen* ist es dadurch sehr einfach die Menge der NDC-Knotenpaare zu bestimmen. Betrachtet man nämlich den Komplementärgraph $\overline{M} = \{V, \overline{E'}\}$ von M, so entspricht die Menge der Kanten $\overline{E'}$ genau der Menge der NDC-Knotenpaare. Darüber hinaus können Heuristiken durch kontinuierliche Aktualisierung der Kantenmenge E' nach der Einführung eines Regenerators diese Information auch dazu benutzen eine bestimmte Lösung als gültig zu bewerten, indem überprüft wird ob M zu einem bestimmten Zeitpunkt bereits ein vollständiger Graph ist, also $\overline{E'} = \emptyset$ gilt.

Um diesen Konstruktionsschritt zu veranschaulichen, stellt man sich einen Graph bestehend aus drei Knoten A, B, C und D vor. Diese Knoten sind durch die Kanten (A, B), (B, C), (C, D)

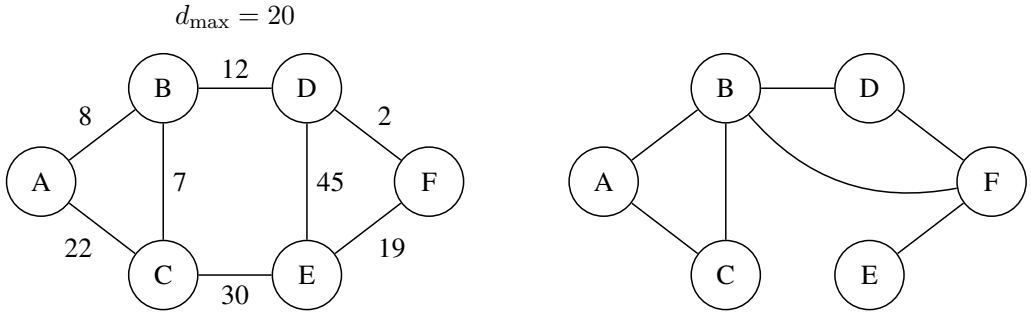


Abbildung 1.3: Transformation des Originalgraphen zum *Communication Graph* mit $d_{\max} = 20$

und (A, D) verbunden und ihre Entfernung aus der Distanzmatrix beträgt jeweils $D_{A,B} = 3$, $D_{B,C} = 3$, $D_{C,D} = 5$, $D_{A,D} = 12$. Ist ferner die maximale störungsfreie Signalreichweite $d_{\max} = 7$ gegeben, so kann automatisch eine direkte Übertragung zwischen A und C angenommen werden, da die kürzeste Verbindung $(A, C) = 6 \leq d_{\max}$ ist. Darüber hinaus wird die Kante (A, D) nicht weiter im *Communication Graph* verwendet, da sie eine längere Distanz, als der erlaubte Schwellwert aufweist und es auch keinen Pfad $\{(A, v_i), \ldots, (v_j, D)\}$ mit entsprechend kürzerer Länge gibt. Daher würde der *Communication Graph* in diesem Beispiel als $M = \{V, E'\}$ mit $E' = \{(A, B), (A, C), (B, C), (C, D)\}$ gegeben sein. Eine Illustration des Konstruktionsschritts in Form einer Gegenüberstellung des Originalgraphen zu seinem entsprechenden *Communication Graph* ist in *Abbildung 1.3* zu finden.

Preprocessing im *Communication Graph*

Im zweiten Schritt der Transformationsprozedur zum *Communication Graph* kann ein optionales Preprocessing durchgeführt werden. Chen et al. haben in [6] gezeigt, dass es eine einfache Möglichkeit gibt manche notwendigen Platzierungsorte für Regeneratoren bereits frühzeitig zu erkennen. Dabei wird ausgenutzt, dass Knoten im *Communication Graph* mit Knotengrad gleich eins auf jeden Fall einen Regenerator im direkt benachbarten Knoten benötigen, um am Ende eine zulässige Lösung zu erhalten. Im Preprocessing wird nun die Menge aller Knoten mit Knotengrad eins im Graphen bestimmt und jeweils einer davon als aktuell bearbeiteter Knoten ausgewählt. Anschließend wird sein einziger direkter Nachbar der Lösungsmenge an Regeneratorknoten R hinzugefügt und er selbst gänzlich aus dem *Communication Graph* entfernt. Durch das Hinzufügen des Nachbarn zur Lösung ist automatisch sichergestellt, dass der aktuelle Knoten später auch mit allen anderen Knoten verbunden ist, sofern der Nachbarknoten mit allen anderen verbunden ist, was es problemlos ermöglicht den aktuellen Knoten gänzlich aus der weiteren Betrachtung auszuschließen. Der Knotengrad des Nachbarknotens wird um eins verringert und dieser falls zutreffend der Menge an Knoten mit Knotengrad eins hinzugefügt. Danach wird der nächste Knoten mit Knotengrad gleich eins ausgewählt und die zuvor beschriebenen

input : *Communication Graph* $M = (V, E')$
output : Menge fixer Regeneratorknoten L

```
1  Initialisierung: R = ∅;
   // Grad(n) == Knotengrad von n in M
2  while (∃ n1 ∈ V mit Grad(n1) == 1) do
3  |  n2 ← Nachbar von n1;
4  |  M ← M \ n1;                                  // Entferne n1 aus M
5  |  if (n2 ∈ R) then
6  |  |  continue;
7  |  end
8  |  if (Grad(n2) == 0) then
9  |  |  return R als vollständige Lösung;
10 |  end
11 |  if Grad(n2) ≥ 1 then
12 |  |  R ← R ∪ {n2};
13 |  end
14 end
```

Algorithmus 1.1: Preprocessing: Eliminierung von Knoten mit Grad eins (aus [6])

Schritte wiederholt, solange bis keine solchen Knoten mehr vorhanden sind. Als Resultat des Preprocessingschritts wird einerseits eine Menge von Regeneratorknoten identifiziert, die auf alle Fälle einen Regenerator besitzen müssen und andererseits eine Menge von Knoten aus dem Optimierungsprozess ausgeschlossen, die für den weiteren Verlauf irrelevant sind. Die bestimmten Regeneratorknoten stellen einen fixen Teil der Lösung dar, der in allen gültigen Lösungen für diese Instanz vorkommen muss. Der verwendete Algorithmus zur Eliminierung der Knoten mit Knotengrad eins ist in *Algorithmus 1.1* skizziert.

Eine weitere Möglichkeit im Preprocessing bereits frühzeitig Regeneratorknoten zu fixieren, ist die Bestimmung von Gelenkpunkten (vgl. engl. Cut Vertice oder Articulation Point, siehe *Kapitel 1.5*) und Brücken im Graphen (vgl. engl. Bridge, siehe *Kapitel 1.5*). Da Brücken ohnehin die Verbindung zwischen zwei Gelenkpunkten darstellen, beschränkt sich das Preprocessing auf das Auffinden aller Gelenkpunkte ohne explizite Bestimmung der möglichen Brücken im Graph. Die Methode dies zu bewerkstelligen prüft den Graph zunächst auf 2-Zusammenhang. Ist dieser nicht gegeben, können einerseits die identifizierten Gelenkpunkte sofort zur Lösung der Regeneratorknoten hinzugefügt werden und andererseits das Problem in mehrere Subprobleme aufgespalten werden. Die Anzahl an neu abgeleiteten Subproblemen entspricht genau der Anzahl an entstehenden 2-Zusammenhangskomponenten. Der Vorteil der Aufspaltung liegt auf der Hand, da dadurch die Probleme jeweils unabhängig und separat voneinander bearbeitet werden können und damit die Komplexität reduziert wird bzw. eine parallele Verarbeitung möglich wird.

Theorem: Im Preprocessing für das RLP können in allen Gelenkpunkten eines nicht 2-zusammenhängenden Graphen automatisch Regeneratoren platziert werden, da sie sicher Teil der minimalen Lösung sind.

Beweis: Sei $G = (V, E')$ ein beliebiger zusammenhängender, jedoch nicht 2-zusammenhängender, ungerichteter Graph, der in einen *Communication Graph* umgewandelt wurde. Da aus der Angabe 2-Zusammenhängigkeit nicht gegeben ist, gibt es eine nichtleere Menge $A \subset V$ an Knoten, die Gelenkpunkte für G darstellen. Eine gültige Lösung für das RLP fordert die Verbindung aller Knotenpaare durch eine direkte Kante oder in Form eines Pfads mit mehreren Regeneratorknoten. Betrachtet man einen beliebigen Gelenkpunkt $c \in A$ so veranlässt er durch seine spezielle Eigenschaft den Graph dazu bei Ausschluss in zwei maximal zusammenhängende Komponenten $C_1, C_2 \subset V$ zu zerfallen, die jeweils mindestens einen Knoten beinhalten. Die beiden Zusammenhangskomponenten sind nicht mehr verbunden, wodurch G unzusammenhängend ist. Für eine gültige Lösung des RLP muss allerdings mindestens ein Knoten aus C_1 über einen Pfad aus Regeneratoren mit einem Knoten aus C_2 verbunden werden, was aus der Bedingung, dass C_1 und C_2 mindestens einen Knoten beinhalten müssen folgt. Daher muss jeder Gelenkpunkt automatisch in der optimalen Lösung einer Instanz vorhanden sein, um die Gültigkeit der Lösung zu gewährleisten.

Zur Bestimmung des 2-Zusammenhangs wird ein Algorithmus verwendet, der die notwendige Bedingung im Graphen überprüft und im Falle der Nichterfüllung sofort sämtliche Gelenkpunkte und Zusammenhangskomponenten identifiziert. Ein detaillierter Pseudocode zur Illustration der Vorgangsweise ist in *Algorithmus 1.2* gegeben. Dabei werden die Überlegungen von Hopcroft und Karp benutzt, die in [21] eine Methode vorgestellt haben, wie Zusammenhangskomponenten bestimmt werden können. Der Algorithmus durchwandert den Graph mit Hilfe einer rekursiven Tiefensuche und versieht dabei jeden Knoten mit einen Label, der die Tiefe des Knotens in der Suche angibt. Wird ein Nachbarknoten y zum ersten Mal extrahiert, wird für den aktuellen Knoten x zunächst nur die Anzahl seiner Kinder aktualisiert und dann sofort mit dem rekursiven Aufruf des Knoten y fortgefahren. Gibt es in einem bestimmten Knoten keine weiteren unbesuchten Nachbarknoten mehr, so stoppt zunächst die Rekursion und arbeitet die gesammelten Informationen auf. Die Variable low des aktuellen Knotens wird dabei mit dem Minimum aus dem low-Wert des Nachbarknotens und dem aktuellen Wert belegt. Sollte in Folge dessen dieser Wert größer oder gleich der eigenen Suchtiefe und der aktuelle Knoten nicht der Startknoten der Suche sein, dann handelt es sich beim aktuellen Knoten um einen Gelenkpunkt. Dies ist dadurch zu erklären, dass in der Variable low jeweils der in der Suchtiefe am höchsten liegende, erreichbare Knoten gespeichert wird und bei Erfüllung der Bedingung alle nachfolgend extrahierten Knoten zu keinem Knoten verbunden sind, der in der Suchereihenfolge vor dem aktuellen Knoten extrahiert wurde. Dadurch muss einerseits der aktuelle Knoten ein Gelenkpunkt sein und andererseits die nachfolgend extrahierten Knoten eine Zusammenhangskomponente bilden.

Da der Algorithmus jeden Knoten zumindest einmal vollkommen extrahieren und jede Kante im Graphen auch mindestens einmal betrachten muss, ist die Laufzeit des Algorithmus auf alle Fälle mit $O(n + m)$ anzugeben, wobei n der Anzahl der Knoten und m der Anzahl der Kanten im Graphen entspricht. Durch den rekursiven Aufruf und die Iteration über alle Nachbarknoten kann es vorkommen, dass manche Kanten mehrfach untersucht werden. Die Anzahl solcher Untersuchungen hängt vom Vernetzungsgrad des Graphen ab. Im schlechtesten Fall bei einem vollständigen Graph wären die zusätzlichen Untersuchungsschritte mit $O((n-1) \cdot (n-1) - m)$ begrenzt. Beim RLP wird jedoch in der Regel nie der worst case eines vollständigen Graphen

input : *Communication Graph* $M = (V, E')$
output : Menge der Gelenkpunkte A und HashMap aller gefundenen Zusammenhangskomponenten Z

```
1  Initialisierung: ∀ v ∈ V setze v.label = −1, A = ∅, Z = ∅, dfsZähler = 0, bccNumb = 0;
2  ArticulationPointDFS(Knoten v ∈ V, Gelenkpunkte A, HashMap Z) {
3      v.label ← dfsZähler++; v.low ← v.label;
4      foreach (Nachbarknoten x ∈ N(v)) do
5          if (x.label == −1) then                          // x ist noch unbesucht
6              x.dfsLevel ← v.dfsLevel++;
7              v.numKind ← v.numKind++;
8              stack.push(Edge (v,x));
9              call ArticulationPointDFS(x);                // Rekursionsaufruf
10             v.low ← Min(v.low, x.low);
11             if (v.label == 1) then
12                 if (v.numKind ≥ 2) then A ← A ∪ {v} ;        // Fall: Wurzel
13                 while (stack.top ≠ (v, x)) do H ← H ∪{stack.pop};
14                 Z[bccNumb++] ← {H};
15             else
16                 if (x.low ≥ v.label) then        // v ist sicher Gelenkspunkt
17                     A ← A ∪ {v};
18                     while (stack.top ≠ (v, x)) do H ← H ∪{stack.pop};
19                     Z[bccNumb++] ← {H};
20                 end
21             end
22         else if (x.dfsLevel < v.dfsLevel − 1) then
23             v.low ← Min(v.low, x.label);
24             stack.push(Edge (v,x));
25         end
26     end
27 }
```

Algorithmus 1.2: Preprocessing: Bestimmung des 2-Zusammenhangs, Zusammenhangskomponenten und eventuellen Gelenkpunkten mittels Tiefensuche. Siehe auch [22]

betrachtet, da dieser ohnehin schon ohne zusätzliche Regeneratoren eine gültige Lösung darstellt. Die Gesamtlaufzeit ist daher im durchschnittlichen Fall $O(n + m)$ und im worst case als $O(n + (n - 1) \cdot (n - 1)) = O(n^2)$ anzugeben.

Es stellt sich nun die Frage, ob das Preprocessing noch besser funktionieren würde, wenn der Graph auf 3-Zusammenhang bzw. n-Zusammenhang mit $n > 2$ geprüft wird. Dabei ist das Ziel jeweils durch den Nachweis des nicht bestehenden n-Zusammenhangs Gelenkpunkte zu identifizieren, die den Graph in seine Zusammenhangskomponenten zerfallen lassen und diese sofort mit Regeneratoren zu versorgen. Leider stellt sich heraus, dass identifizierte Gelenkpunkte im Zuge dieser Vorgehensweise bei $n > 2$ nicht mehr zwingend Teil der optimalen Lösung des

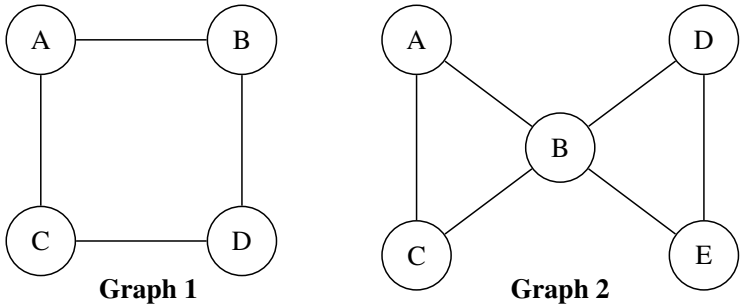


Abbildung 1.4: Beispiel für einen nicht 3-zusammenhängenden (Graph 1), bzw. einen nicht 2-zusammenhängenden Graph (Graph 2)

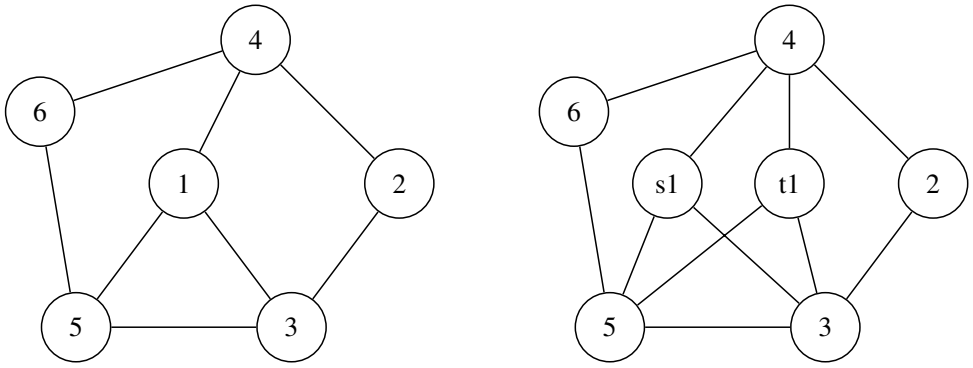
Graphen sind und daher zu keiner zielführenden Problemverkleinerung führen.

Ein einfaches Beispiel, warum die Gelenkpunkte eines 2-zusammenhängenden, aber nicht 3-zusammenhängenden Graphen nicht automatisch Teil der optimalen Lösung sind ist ebenfalls leicht in *Abbildung 1.4*, Graph 1 zu sehen. Die optimale Lösung für diese Instanz wären die Regeneratorknoten $\{A, B\}$ oder $\{A, C\}$ oder $\{B, D\}$ oder $\{C, D\}$. Die Paare an Gelenkpunkten jedoch $\{A, D\}$ und $\{B, C\}$. Nun wird eine Möglichkeit der Gelenkpunktpaare selektiert, z.B. $\{A, D\}$ und in diesen Knoten sofort Regeneratoren platziert. Das führt dazu, dass noch keine gültige Lösung zustande gekommen ist und das Knotenpaar AD noch über einen weiteren Regenerator entweder in B oder in C verbunden werden muss. Analoges passiert bei Selektion des anderen Gelenkpunktpaares. Daher würde dies zu einer Lösung von drei Regeneratoren führen, wobei die optimale Lösung dieser Instanz mit lediglich zwei Regeneratoren auskommt.

Durch diese Beobachtung ist der Test auf 2-Zusammenhang die einzig zuverlässige Methode um mit der Identifikation von Gelenkpunkten das Problem im Preprocessingschritt zu vereinfachen. Wie im *Beweis auf Seite 9* gezeigt, sind alle auf diese Art gefundenen Gelenkpunkte auf jeden Fall Teil der optimalen Lösung. Das Aufspalten des RLP in mehrere Subprobleme hat im Allgemeinen vor allem bei dünnen Instanzgraphen, also Graphen mit einem geringen Vernetzungsgrad, eine große Bedeutung und Auswirkung auf den gesamten Optimierungsprozess.

1.4 Generalized Regenerator Location Problem

Das Generalized Regenerator Problem (GRLP) wurde von Chen, Ljubic und Raghavan in [5] auf der INOC 2009 vorgestellt. Dabei wird die in [28] vorgebrachte Kritik am RLP berücksichtigt und ein allgemeines Regenerator Location Problem definiert, wo eine Unterscheidung zwischen tatsächlich miteinander kommunizierenden Knoten und Knoten in denen Regeneratoren platziert werden können vorgenommen wird. Eine Instanz des GRLP ist als Graph $G = (V = \{S \cup T\}, E, D)$, sowie dem Parameter $d_{\max}$, der wie beim RLP die maximale Reichweite eines Signals ohne Qualitätsverluste angibt, definiert. Dabei ist V die Menge der Knoten, die sich in die Untermengen S und T unterteilt. Die Menge S repräsentiert all jene Knoten, in denen Regeneratoren platziert werden können. Die Menge T enthält alle sogenann-



Graph G mit $S = \{1, 3, 6\}$ und $T = \{1, 2, 4, 5\}$

Graph M mit $M_S = \{s1, 3, 6\}$ und $M_T = \{t1, 2, 4, 5\}$

Abbildung 1.5: Beispiel für den Originalgraph G und den umgewandelten *Communication Graph* M einer Instanz des GRLP

ten Terminal-Knoten, die jeweils miteinander kommunizieren müssen. Der Parameter E in der Problemdefinition stellt die Menge der Kanten im Graphen dar, D die zugehörige Distanzmatrix der Abstände zwischen den Knoten. Ziel ist es nun eine minimale Menge $R \subseteq S$ zu finden, die es ermöglicht für alle Knotenpaare aus T einen Pfad zu finden, bei dem kein Teilstück größer als $d_{\max}$ ist, ohne dass dieses mit einem Regenerator versorgt ist. Das in *Kapitel 1.2* beschriebene RLP stellt demnach einen Spezialfall des GRLP dar, bei dem $S = T$ ist.

Zur Lösung des GRLP wird der Instanzgraph ähnlich wie beim RLP zunächst in einen leichter handhabbaren *Communication Graph* M transformiert. Dabei wird ein All-Pair-Shortest-Path Algorithmus verwendet, um alle Knotenverbindungen, die innherhalb von $d_{\max}$ erreicht werden können, zu identifizieren und sie anschließend als direkte Kanten einzufügen. Die Sub-Graphen M_S und M_T enthalten jeweils die Knoten aus S bzw. T und alle zugehörigen Kanten, bei denen beide Endpunkte jeweils in S oder in T liegen. Ist ein Knoten sowohl in S, als auch in T, werden zwei exakte Kopien des Knotens erstellt und diese Knoten auf die zwei Mengen aufgeteilt, sodass S und T disjunkt sind. Ein Beispiel für diesen Transformationsprozess ist in *Abbildung 1.5* zu finden. Die Knoten des Sub-Graphen M_S werden zum Abschluss noch zu Superknoten zusammengefasst, die den einzelnen zusammenhängenden Komponenten im Sub-Graph entsprechen. Daher würde es im Beispiel in *Abbildung 1.5* zwei S-Komponenten $S1, S2$ geben, die sich aus $S1 = \{s1, 3\}$ und $S2 = \{6\}$ zusammensetzen. Diese Zusammenfassung zu Superknoten ermöglicht es für ein NDC-Knotenpaar (n, m) aus M_T durch Auffinden eines gemeinsamen Superknotens zu dem n und m jeweils adjazent sind das Knotenpaar sofort aus der Menge der NDC-Paare zu eliminieren, da durch Regeneratoren im Superknoten eine ungestörte Kommunikation möglich ist. Die optimale Lösung für das Beispiel ist daher $L = \{s1, 3\}$, damit alle Knoten aus T ohne Einschränkungen miteinander kommunizieren können.

Eine Konstruktionsheuristik für das GRLP versucht zunächst aus der Terminalknotenmenge T jenen Knoten zu selektieren, der den kleinsten Knotengrad im Subgraph M_T besitzt. Dieser

Subgraph beinhaltet die Knotenmenge T und die Kantenmenge E', bei der nur jene Kanten aus G übernommen werden, bei denen beide Endpunkte in T liegen. Anschließend werden alle Superknoten in S durchsucht und jeweils jene Knoten aus S zur Lösung hinzugefügt, die die meisten NDC-Paare der t-Knoten eliminieren. Dies wird solange wiederholt, bis alle Terminalknoten miteinander verbunden sind. Neben der heuristischen Methoden wird in [5] auch die Möglichkeit einer exakten Lösungsmethode vorgestellt. Dabei wird das GRLP zuerst in das Directed Steiner Forest Problem transformiert, anschließend als Multi-Commodity Flow Modell formuliert und schließlich mit gemischt-ganzzahliger linearer Programmierung gelöst.

Insgesamt stellt das GRLP durch die Unterscheidung zwischen Regenerator- und Terminalknoten eine breite Fülle an möglichen Anwendungsszenarien zur Verfügung. Beispielsweise können dadurch Einschränkungen aus dem geographischen Umfeld, oder aus speziellen Anforderungen heraus besser in den Optimierungsprozess miteinbezogen werden.

1.5 Graphentheorie

Graph

Ein Graph stellt eine abstrakte Struktur zur Verfügung, die eine Menge von Objekten und ihren Zusammenhang darstellt. Dabei ist ein Graph allgemein durch das Tupel $G = (V, E)$ definiert. Wobei V die Menge der Knoten (die Objekte) und E die Menge der Kanten (den Zusammenhang der Objekte) bezeichnen. Im Zuge des RLP kann eine Probleminstanz in einem ungerichteten, gewichteten Graphen abgebildet werden. Dabei bedeutet ungerichtet, dass eine Kante zwischen zwei Knoten a, b in beide Richtungen gilt, d.h. im Speziellen die Adjazenzmatrix von G symmetrisch ist. Die Eigenschaft gewichtet gibt an, dass allen Kanten ein bestimmter Wert, nämlich die Distanz zwischen ihren Endpunkten, zugewiesen werden. Der abgeleitete *Communication Graph* des RLP stellt einen ungerichteten und ungewichteten Graph dar. Hier sind alle Kanten gleichrangig, wodurch es keine Unterschiede in der prinzipiellen Bewertung einer Kante gibt.

Komplementärgraph

Der Komplementärgraph $\overline{G}$ von G lässt sich durch das Tupel $\overline{G} = (V, E')$ beschreiben. Dabei enthält der Komplementärgraph genau jene Knotenverbindungen, die im eigentlichen Graph G nicht existieren. Das heißt die Menge E' lässt sich durch $E' = \{\forall(a, b) \in V \times V \mid (a, b) \notin E\}$ beschreiben. Der Komplementärgraph stellt ein wichtiges Hilfsmittel zur Verfügung, um spezielle Knotenmengen, wie z.B. Independent Set oder Vertex Cover, im Graphen besser zu bestimmen.

Zusammenhang und Zusammenhangskomponente

Ein ungerichteter Graph $G = (V, E)$ heißt zusammenhängend, wenn zwischen je zwei beliebigen Knoten $u, v \in V$ ein Kantenzug existiert, der sie verbindet und dies für alle Knoten in V gilt. Eine Zusammenhangskomponente eines Graphen beschreibt einen Subgraph $U \subset V$, der maximal zusammenhängend ist. Ein zusammhängender Graph besitzt daher genau eine Zusammenhangskomponente.

k-Zusammenhang, Gelenkpunkt und Brücke

Man spricht vom k-Knoten bzw. k-Kantenzusammenhang eines Graphen, wenn es keine (k-1) elementige Knoten- oder Kantenmenge gibt, durch deren Entfernung der zusammenhängende Graph G unzusammenhängend wäre. Dabei bezeichnet ein Gelenkpunkt einen (im 2-zusammenhängenden Fall) oder eine Menge solcher Knoten (im allgemeinen Fall), durch deren Entfernung der Graph in seine Zusammenhangskomponenten zerfallen würde. Auf der anderen Seite stellt eine Brücke eine Kante (im 2-zusammenhängenden Fall) oder eine Menge von Kanten (im allgemeinen Fall) dar, deren Entfernung den Graph unzusammenhängend machen würde.

Clique

Eine Clique C ist eine Teilmenge $C \subseteq V$, sodass der Subgraph $G = (C, E')$ vollständig ist. Im speziellen bedeutet das, dass alle Paare von Knoten in C benachbart sind, d.h. $\forall$ Knoten a, b eine Kante $(a, b) \in E'$ existiert, wobei $E' \subseteq E$, mit $E' = \{\forall (a, b) \in E \mid a \in C \wedge b \in C\}$ gilt.

Der Grad einer Clique ist durch die Kardinalität von C bestimmt. Allgemein können Cliques für ein beliebiges k, mit $Grad(C) = k$, $2 \leq k \leq |V|$, im Graphen bestimmt werden. Ob für jedes k eine entsprechende Clique existiert, wird im Maximum Clique Problem addressiert (*siehe Kapitel 2.4*).

Independent Set

Ein Independent Set oder stabile Menge I ist eine Teilmenge $I \subseteq V$, sodass kein Paar von Knoten aus I benachbart ist. Das heißt für alle Knotenpaare $\forall (a, b) \in I$ gilt, dass keine Kante in der Kantenmenge E existiert, also $\nexists (a, b) \in E$ gilt. Dabei gilt der Zusammenhang, dass die Menge I eine Clique im Komplementärgraph $\overline{G}$ bildet.

Der Grad einer stabilen Menge ist durch die Kardinalität von I bestimmt. Abhängig von der Anzahl der Knoten können stabile Mengen für einen Graph mit $|V| \geq 3$ und $Grad(I) \geq 2$ bestimmt werden. Ob die größte stabile Menge für ein bestimmtes k gefunden werden kann wird im Maximum Independent Set Problem behandelt (*siehe Kapitel 2.4*).

Vertex Cover

Ein Vertex Cover oder Knotenüberdeckung C ist eine Teilmenge $C \subseteq V$, sodass für jede Kante $(a, b) \in E$ entweder $a \in C$ oder $b \in C$ gilt. Der Zusammenhang zwischen Knotenüberdeckung und stabiler Menge ist dadurch gegeben, dass für eine stabile Menge I in G immer eine zugehörige Knotenüberdeckung $C = V \setminus I$ existiert.

Den Grad der Knotenüberdeckung beschreibt die Kardinalität der Menge C. Es ist möglich eine Knotenüberdeckung für ein k mit $1 \leq k \leq |V|$ zu finden. Das Minimum Vertex Cover Problem versucht die kleinst mögliche Knotenüberdeckung zu finden.

Dominating Set und Connected Dominating Set

Ein Dominating Set oder eine dominierende Menge S ist eine Teilmenge $S \subseteq V$, sodass jeder Knoten in $V \backslash S$ entweder in S enthalten ist, oder zumindest einen adjazenten Knoten in S besitzt. Die Knotenmenge S muss im allgemeinen Fall nicht notwendigerweise zusammenhängend sein.

Ein Connected Dominating Set oder eine zusammenhängende dominierende Menge fordert darüber hinaus, dass die Knotenmenge S zusammenhängend ist, d.h. für jeden Knoten $v \in S$ gilt, dass er jeden anderen Knoten $u \in S$ über einen Pfad innerhalb von S erreichen kann. Das Problem die kleinste zusammenhängende dominierende Menge zu finden wird im Minimum Connected Dominating Set Problem (siehe *Kapitel 2.3*) addressiert.

1.6 Metaheuristiken

In [1] geben Blum und Roli einen Überblick über Metaheuristiken im Allgemeinen und den damit verbundenen Konzepten, die sich im Laufe der Zeit durch die verschiedenen Arten entwickelt haben. Eine Metaheuristik stellt typischerweise einen Algorithmus zur Lösung eines Optimierungsproblems P dar. Allgemein kann P durch

$$P \ : \ min \ f(x) \tag{1.10}$$

und

$$x \in X \tag{1.11}$$

beschrieben werden, wobei $f(x)$ die zu optimierende Zielfunktion und X die Menge aller gültigen Lösungen für eine Instanz von P ist. Eine Lösung $x^* \in X$ ist optimal, wenn

$$f(x^*) \leq f(x), \forall x \in X. \tag{1.12}$$

Die Idee einer Metaheuristik beruht darauf eine abstrakte Abfolge von Schritten zu definieren, um für eine Vielzahl von Optimierungsproblemen $P_1, \ldots, P_i$ nach Durchlaufen aller Berechnungsschritte die optimale Lösung $x^*_{P_n}$ zu finden. Dabei werden innerhalb der Schritte „kleinere“ Heuristiken zur Berechnung von Zwischenergebnissen kombiniert und in einen Gesamtprozess eingegliedert. Viele Metaheuristiken funktionieren nach dem Prinzip der lokalen Suche, sodass eine gefundene Ausgangslösung iterativ verbessert wird, bis ein lokales Optimum erreicht ist. Allerdings zeigen Wolpert und Macready in [35] auf, dass alle Metaheuristiken über alle denkbaren Optimierungsprobleme P_i gleich gut sind und haben dies anhand mehrerer No Free Lunch Theoreme untermauert. Das heißt eine Metaheuristik H ist immer nur im Bezug auf eine Auswahl spezieller Probleme als bestes heuristisches Lösungsverfahren geeignet, nicht aber generell für alle denkbaren Probleme.

Greedy Randomized Adaptive Search Procedure (GRASP)

Die von Feo und Resende entwickelte Greedy Randomized Adaptive Search Procedure (im Weiteren als GRASP bezeichnet) ist eine auf den Prinzipien der Randomisierung von Konstruktionsheuristiken und der lokalen Suche basierende Metaheuristik und in [12], sowie [13] detailliert

beschrieben. Idee dabei ist es in mehreren Iterationen jeweils zunächst durch eine randomisierte Konstruktionsheuristik möglichst unterschiedliche Ausgangslösungen zu generieren und diese zunehmend mit Hilfe von lokaler Suche zu verbessern. Am Ende jeder Iteration wird die bisher beste gefundene Lösung mit der aktuellen Lösung verglichen und im Fall einer Verbesserung aktualisiert. Je vielfältiger die Lösungen der Konstruktionsheuristik sind, desto großflächiger wird bei einer entsprechend hohen Iterationsanzahl der Raum an möglichen Lösungen durchsucht. Dadurch wird die Wahrscheinlichkeit in lokalen Optima festzusitzen reduziert. Die Vorgehensweise der GRASP Metaheuristik ist folgende:

1. Vorbereitung der Probleminstanz für die Bearbeitung
2. Solange das Abbruchkriterium (z.B. Iterationsanzahl) nicht erfüllt ist:
   a) Erstellung der Ausgangslösung durch eine randomisierte Konstruktionsheuristik
   b) Anwendung der lokalen Suche auf die Ausgangslösung
   c) Aktualisierung der besten Lösung, falls die aktuelle Lösung besser ist als diese

Variable Neighborhood Search (VNS)

Die Variable Neighborhood Search (im Weiteren als VNS bezeichnet) ist eine von Mladenovic und Hansen [29] eingeführte Metaheuristik. Ihr Hauptcharakteristikum ist der Einsatz von mehreren Nachbarschaften in Kombination mit der lokalen Suche. Dabei werden die Nachbarschaftsstrukturen während der Suche systematisch gewechselt, um eine optimale Lösung zu finden. Die Idee dahinter ist, dass das lokale Optimum $x \leftarrow x'$ einer Nachbarschaftsstruktur $N_1(x)$ nicht gezwungenermaßen auch das lokale Optimum $x \leftarrow x''$ für $N_2(x)$ einer anderen ist, hingegen das globale Optimum einer Probleminstanz $x \leftarrow x^*$ immer das Optimum in allen denkbaren Nachbarschaftsstrukturen $N_1(x), N_2(x), \ldots, N_m(x)$ bildet. Ziel des Vorgehens soll daher die kontinuierliche Verbesserung einer Ausgangslösung durch die systematische Anwendung von zuvor definierten Nachbarschaftsstrukturen $N_1(x), \ldots, N_l(x)$ sein, solange bis die Lösung ein lokales Optimum in allen Nachbarschaftsstrukturen darstellt. Der schematische Ablauf sieht folgendermaßen aus:

1. Initialisierung der Liste der Nachbarschaftsstrukturen $\{N_1(x), \ldots, N_l(x)\}$, sowie des Parameters $k = 1$ und Berechnung einer Ausgangslösung x
2. Solange das Abbruchkriterium (z.B. Iterationsanzahl) nicht erfüllt ist:
   a) „Shaking“, bestimme eine zufällige Nachbarlösung x' von x aus der k-ten Nachbarschaft $N_k(x)$, d.h. $x' \in N_k(x)$
   b) Wende eine lokalen Suche auf x' an und berechne das lokale Optimum x''
   c) Falls x'' besser als x ist, setze $x \leftarrow x''$ und $k \leftarrow 1$, ansonsten setze $k \leftarrow k + 1$

Das Prinzip der VNS wurde in der Literatur oftmals erweitert und auf spezielle Situationen zugeschnitten. Daraus haben sich mehrere Ausprägungsformen der VNS entwickelt. Dazu zählt

beispielsweise der *Variable Neighborhood Descent (VND)*, bei dem Nachbarschaftsstrukturen hintereinander systematisch gewechselt werden. Üblicherweise wird dafür eine best oder next improvement Schrittfunktion gewählt und durch die Verschachtelung der eingesetzten Nachbarschaften $N_1, \ldots, N_x$ schlussendlich eine Lösung erreicht, die in allen diesen Nachbarschaften lokal optimal ist.

Eine weitere Form der VNS ist die *Reduced Variable Neighborhood Search (RVNS)*, die sich vor allem gut dazu eignet lokalen Optima zu entkommen. Idee ist es Nachbarlösungen zufällig aus immer größer werdenden Nachbarschaften auszuwählen. Dabei werden die eingesetzten Nachbarschaftsstrukturen $N_1, \ldots, N_x$ zum Beispiel nach ihrer Größe sortiert und durch ein sogenanntes Shaking eine zufällige Lösung aus der Nachbarschaft ausgewählt. Falls die gefundene Lösung besser als die vorherige war, wird wieder mit der Anwendung der ersten Nachbarschaftsstruktur begonnen, ansonsten mit der nächsten fortgefahren.

Die *General Variable Neighborhood Search (GVNS)* kombiniert die beiden Überlegungen der RVNS und des VND. Dabei wird eine RVNS mit Shaking-Nachbarschaften durchgeführt und anschließend nach jeder zufällig bestimmten Lösung aus der Nachbarschaft ein VND als Verbesserungsstrategie eingesetzt. Zu beachten ist, dass die Nachbarschaftsstrukturen der RVNS und des VND nicht ident sein dürfen.

Ein Überblick über die verschiedenen Formen der VNS ist in [18] und [19] zu finden. Anwendungsgebiete in denen die VNS bereits in vielen Arbeiten eingesetzt wurde sind vor allem das Traveling Salesperson Problem, das p-Median Problem, das Graph Coloring Problem und das Minimum Sum of Squares Partitioning Problem.

KAPITEL 2

Verwandte Arbeiten

2.1 Grundlegende Heuristiken für das RLP

In [6] haben Chen, Ljubic und Raghavan insgesamt drei Konstruktionsheuristiken für das RLP vorgestellt, die jeweils verschiedene Strategien für die Generierung von Lösungen verfolgen. Diese bauen auf der in *Kapitel 1.3* beschriebenen Transformation des vorliegenden Netzwerks in den *Communication Graph* inklusive optionalem Preprocessing auf. Wichtig ist in diesem Zusammenhang die in [6] getroffene und bewiesene Überlegung, dass zu einer gültigen Lösung des RLP immer ein Spannbaum existiert, bei dem die Regeneratorknoten jeweils die inneren Knoten des Spannbaums darstellen.

Algorithmus Greedy aus „The Regenerator Location Problem“ [6]

Ein einfacher greedy Algorithmus für das RLP versucht in jeder Iteration den Knoten der Lösung hinzuzufügen, der die größte Anzahl an NDC-Paare eliminieren würde, wenn ein Regenerator an seinem Platz installiert wird. Dies geschieht anhand einer Evaluierungsfunktion χ, die durch Mengenoperationen die Nachbarschaft jedes Knotens so durchsucht, dass alle NDC-Paare gefunden werden, die durch eine Regeneratorinstallation miteinander verbunden wären. Die genaue Implementierung der Evaluierungsfunktion χ ist in *Algorithmus 2.1* zu finden. Anschließend wird der Knoten mit den meisten eliminierten NDC-Paaren ausgewählt, die Lösung um diesen Knoten erweitert, der *Communication Graph* mit den neu entstandenen direkten Kanten ausgestattet und die betroffenen NDC-Paare entfernt. Diese Schritte werden so oft wiederholt, solange es noch NDC-Paare im *Communication Graph* gibt.

Chen et al. haben in [6] die Laufzeit des Greedy Algorithmus analysiert. In jeder Iteration wird zunächst die Anzahl an eliminierbaren NDC-Paaren für jeden noch nicht mit einem Regenerator versorgten Knoten berechnet. Der Aufwand, um dies für einen einzelnen Knoten zu berechnen beträgt $O(|V|^2 - |E|)$. Für alle Kandidatenknoten im *Communication Graph* ergibt sich daher eine Laufzeit von $O(|V|^3 - |V| \cdot |E|)$. Darüber hinaus muss es $O(|V|)$ Vergleiche

input : *Communication Graph* $M = (V, E')$, beliebiger Knoten $v \in V$, Regeneratorknoten R
output : Menge von NDC-Knotenpaaren

```
1  χ(v) ← ∅;
2  A ← N(v) = {v ∈ V : (u, v) ∈ E'};
3  for (u ∈ A ∩ L) do
4  |  A ← A ∪ N(u);
5  end
6  for ((i, j) ∈ A × A) do
7  |  if ((i, j) ∉ E') then
8  |  |  χ(v) ← χ(v) ∪ {(i, j)};
9  |  end
10 end
11 return χ(v);
```

Algorithmus 2.1: Evaluierungsfunktion $\chi(v)$ aus [6]

geben, um den aussichtsreichsten Knoten zu selektieren. Im Gesamten ergibt sich daher eine worst-case Laufzeit von $O(|V|^4 - |V|^2 \cdot |E|)$.

Heuristik H1 aus „The Regeneration Location Problem“ [6]

Die zweite in [6] vorgestellte Heuristik H1 versucht einen Spannbaum für den *Communication Graph* M zu bestimmen, der einerseits so wenige innere Knoten und andererseits so viele Blatt-Knoten wie nur möglich besitzt. Dabei wird eine Hilfsmethode $Tree(i)$ eingesetzt, die als Suchfunktion im Graphen fungiert. Grundidee ist es einen Knoten vorab als Startknoten auszuwählen und mit ihm die Methode $Tree(i)$ aufzurufen, die dann schrittweise den gesamten Spannbaum aufbaut. Im Allgemeinen haben Knoten mit einem relativ großen Knotengrad eher die Tendenz als Regeneratorknoten in einer optimalen Lösung zu dienen, als solche mit kleinem Knotengrad. Aus diesem Grund versucht die Heuristik vorrangig solche Knoten als Regeneratorknoten auszuwählen, die im aktuellen Schritt den größten Knotengrad besitzen. Trotzdem ist dies von Instanz zu Instanz unterschiedlich und es kann durchaus vorkommen, dass bei manchen Instanzen in der optimalen Lösung Regeneratoren an Knoten mit relativ kleinen Knotengraden installiert sind. Betrachtet man nun die Heuristik H1 näher, werden folgende Schritte durchgeführt:

1. Initialisierung: $S = L = \emptyset$, $C_i = \emptyset$ und $i.visited = false \; \forall i \in V$;
2. Finde den Knoten $i \in V$ mit kleinstem Knotengrad;
3. Aufruf von $R = Tree(i)$;

Angewendet auf die Beispielinstanz in *Abbildung 2.1* beginnt H1 damit den Startknoten mit kleinstem Knotengrad auszuwählen, nämlich 1. Dadurch wird $Tree(1)$ als Ausgangspunkt

input : *Communication Graph* $M = (V, E')$, Startknoten i
output : Menge von Regeneratorknoten R

```
1  Tree(i) {
2      i.visit ← true;
3      foreach (Nachbarknoten j ∈ N(i) & j ∉ S) do
4          C_i ← C_i ∪ {j};
5      end
6      if (S == ∅) then
7          S ← S ∪ {i};
8      else
9          S ← S ∪ i ∪ C_i;
10     end
11     while (∃ Knoten x ∈ C_i mit x.visit == false) do
12         Knoten c ← jener Knoten aus C_i, der noch nicht besucht wurde und den höchsten
           Knotengrad D_c im Graphen {V,E\E{S}} hat;
13         if (D_c > 0) then          // ∃ noch mind. zwei Knoten im Graph
14             R ← R ∪ {c};
15             R = Tree(c);
16         else                  // Grad == 0 -> letzter Knoten erreicht
17             return R;
18         end
19     end
20 }
```

Algorithmus 2.2: Hilfsalgorithmus $Tree(i)$ aus [6]

für die Spannbaumfunktion aufgerufen. Es wird $C_i = \{2, 4\}$ und $S = \{1\}$ gesetzt. Der Knotengrad von 2 beträgt zwei, jener von Knoten 4 ist fünf. Daher wird 4 als Regeneratorknoten ausgewählt, $R = \{4\}$ aktualisiert und $Tree(4)$ aufgerufen. Sogleich wird $C_4 = \{2, 3, 5, 6\}$ und $S = \{1, 2, 3, 4, 5, 6\}$ gesetzt. Die Knotengrade, abzüglich der Kanten, wo beide Endpunkte in S liegen, betragen für die Knoten in C_4 für $2 = 0$, $3 = 1$, $5 = 1$ und $6 = 2$. Aus diesem Grund wird der Knoten 6 selektiert, R auf $\{4, 6\}$ erweitert und $Tree(6)$ aufgerufen. In diesem Aufruf wird $C_6 = \{7, 8\}$ und $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ belegt. Beide Knoten aus C_6 haben abzüglich der Kanten aus S einen Knotengrad von null. Daher wird kein weiterer Regenerator hinzugefügt und die Rekursion arbeitet sich wieder nach oben. Da es auf keiner Ebene mehr möglich ist einen Regenerator hinzuzufügen, stellt $R = \{4, 6\}$ eine gültige Regeneratorbelegung für das Netzwerk dar, um es vollständig zu versorgen. Gäbe es in der Auswahlphase des nächsten Knotens zwei Knoten, die denselben höchsten Knotengrad aufweisen, würde einer von ihnen zufällig mit gleicher Wahrscheinlichkeit ausgewählt.

Die Laufzeit von H1 beträgt, wie in [6] gezeigt $O(|V| + |E|)$. Dabei ist ein Zeitaufwand von $O(|V|)$ notwendig, um alle Knoten zu überprüfen und jenen mit kleinstem Knotengrad auszuwählen. Anschließend wird die Hilfsmethode $Tree(i)$ so oft aufgerufen, wie es Knoten gibt, die noch unbesuchte Nachbarn besitzen. Da dies in jedem Schritt immer weiter reduziert

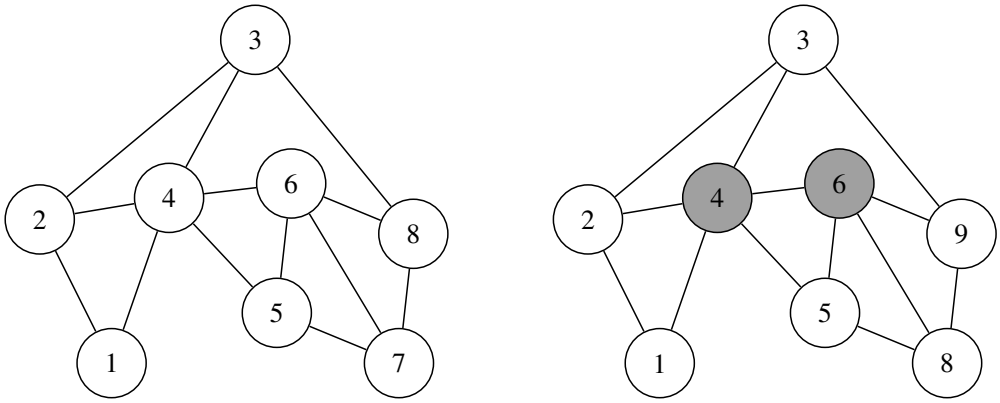


Abbildung 2.1: Illustration der H1 Heuristik anhand eines Beispiels

wird und auch bei der Überprüfung der Knoten stets jene Kanten ausgeblendet werden, die in $E(S)$ liegen, ist der Aufwand für den rekursiven Aufwand durch $O\left(\sum_{i=1}^{N} Grad(i)\right) = O(|E|)$ beschränkt. Daher ergibt sich die Gesamtlaufzeit von $O(|V| + |E|)$.

Heuristik H2 aus „The Regeneration Location Problem“ [6]

Die Heuristik H2 bildet eine Hybridstrategie aus dem Greedy-Algorithmus und der Heuristik H1. Dabei wird in jeder Iteration der Knoten mit dem jeweils kleinsten Knotengrad bestimmt und anschließend alle Nachbarn des Knotens auf den besten Standort für einen Regenerator hin untersucht. Dies geschieht dadurch, dass jener Nachbarknoten ausgewählt wird, der den größten Knotengrad besitzt. Anschließend wird dieser Knoten selektiert und in die Lösungsmenge aufgenommen. Sofort werden die neu entstandenen direkten Kanten in den Communidation Graph eingefügt, sowie die eliminierten NDC-Paare entfernt. Die Heuristik H2 ähnelt der Heuristik H1 in der Auswahl des Knotens mit jeweils kleinsten Knotengrad und verwebt dieses Vorgehen geschickt mit dem Greedy Algorithmus durch die Aktualisierung des *Communication Graphen* nach der Platzierung des Regenerators und dem iterativen Wiederholen des Auswahlschritts. In *Algorithmus 2.3* sind die einzelnen Schritte der H2 Heuristik veranschaulicht.

Betrachtet man die Laufzeit, so benötigt es $O(|V|)$, um den Knoten mit dem kleinsten Grad auszuwählen. Die Hilfsmethode $wähleMaxNachbar(i)$ wird abhängig von $Grad(i)$ aufgerufen, das Update des *Communication Graph*s erfolgt in $O(|V|^2)$. Da der Aufruf der Methode $wähleMaxNachbar(i)$ in der Laufzeitanalyse als konstanter Faktor wegfällt, bleibt für jede Iteration ein Aufwand von $O(|V|^2)$ über. Da maximal $|V|$ Regeneratoren gesetzt werden können, besitzt die Heuristik H2 eine Gesamtlaufzeit im worst-case von $O(|V|^3)$.

input : *Communication Graph* $M = (V, E')$
output : Menge von Regeneratorknoten R

```
1  H2(M) {
2      Initialisierung: R = ∅;
3      while (∃ v ∈ V mit Grad(v) < |V| − 1) do
4          Knoten i ← Knoten mit kleinstem Grad in V;
5          Knoten r ← wähleMaxNachbar(i,R);
6          R ← R ∪ {r};
7          Aktualisiere M;
8      end
9      return R;
10 }
11 wähleMaxNachbar(v,R) {
12     v_max ← null;
13     foreach (Nachbarknoten u ∈ N(v)) do
14         if (Grad(v_max) < Grad(u)) then
15             v_max ← u;
16         end
17     end
18     return v_max;
19 }
```

Algorithmus 2.3: Illustration der H2 Heuristik aus [6]

2.2 Greedy Randomized Adaptive Search für das RLP

Duarte, Resende und Marti haben in [10] auf der Metaheuristics International Conference 2011 (MIC2011) einen Greedy Randomized Adaptive Search (GRASP, siehe *Kapitel 1.6*) Ansatz vorgestellt, der in der Praxis sehr gute Lösungen für das RLP berechnet. Dabei benutzen sie dieselbe Transformation zum *Communication Graph* inklusive Preprocessing, wie in [6] beschrieben und erweitern den einfachen Greedy Algorithmus durch Randomisierung. Darüber hinaus haben sie eine Nachbarschaftsstruktur für eine lokale Suche definiert, die dem Post-Processing aus [6] entspricht. Gemäß dem GRASP Aufbau werden in mehreren Iterationen die Phasen „Konstruktion“ und „lokale Verbesserung“ durchgeführt. Dabei erstellt in jeder Iteration die randomisierte Konstruktionsheuristik zunächst eine Ausgangslösung, die anschließend mit Hilfe der vorgestellten Nachbarschaftsstruktur in Form einer lokalen Suche verbessert wird. Durch den Einsatz von zufallsbasierten Auswahlverfahren in der Konstruktionsheuristik ist der Ansatz in der Lage lokalen Optima zu entkommen und für ein breites Spektrum an Instanzen qualitativ hochwertige Lösungen zu berechnen.

input : *Communication Graph* $M = (V, E')$
output : Menge von Regeneratorknoten R

```
1  CG(M) {
2      Initialisierung: R = ∅;
3      while (∃ v ∈ V mit Grad(v) < |V| − 1) do
4          CL ← {V\R}};
5          foreach (Knoten u ∈ CL) do
6              g[u] ← χ(u);
7          end
8          RCL = {u ∈ CL | g[u] ≥ (g_min + α ∗ (g_max − g_min))};
9          w ← Zufallsauswahl aus RCL;
10         R ← R ∪ {w};
11         Aktualisiere Communication Graph M;
12     end
13     return R;
14 }
```

Algorithmus 2.4: CG Konstruktionsheuristik aus [10]

Konstruktionsheuristiken für das RLP aus „GRASP for the Regenerator Location Problem“ [10] und „Randomized Heuristics for the Regenerator Location Problem“ [9]

Diese Konstruktionsheuristik versucht genauso, wie der Greedy Algorithmus in [6], in jedem Schritt immer einen weiteren Knoten als Regenerator zu identifizieren. Dabei wird jedesmal zuerst eine Candidate List (CL) an möglichen Regeneratorknoten erstellt. Anschließend wird für jeden Knoten $v \in CL$ der Wert der Hilfsfunktion $\chi(v)$ berechnet, die die Anzahl der eliminierten NDC-Paare bestimmt falls in dem Knoten ein Regenerator platziert wird. Anhand eines Toleranzwerts $\alpha \in [0; 1]$, wird aus der CL eine Restricted Candidate List (RCL) erstellt, in der nur noch jene Knoten enthalten sind, die wirklich aussichtsreiche Positionen für Regeneratoren darstellen. Anschließend wird aus der RCL ein Knoten zufällig bestimmt und der Menge der Regeneratorknoten R hinzugefügt. Durch die Verwendung der RCL wird ein Randomisierungsmechanismus in den Konstruktionsprozess eingebaut, der durch den Wert Alpha gesteuert werden kann. Dadurch kann das Festsitzen in lokalen Optima reduziert werden und mehr Vielfalt hinsichtlich der unterschiedlichen Beschaffenheit einer Lösung erzielt werden. In *Algorithmus 2.4* ist die Funktionsweise dieser Konstruktionsheuristik dargestellt.

Duarte, Marti, Resende und Silva stellen in [9] weitere Varianten der zufallsbasierten Lösungssuche vor. Dabei haben sie die drei in [6] beschriebenen Basisheuristiken hergenommen und jeweils durch Randomisierung erweitert. Die Erweiterung des Greedy Algorithmus ist genau jene, die in [10] als CG Heuristik vorgestellt wurde.

Die Heuristik H1 versucht, wie in *Kapitel 11* beschrieben, einen Spannbaum im Graphen mit so wenig inneren und so vielen Blattknoten wie möglich zu finden. Dabei wird in jeder Iteration jener Knoten zur Lösung hinzugefügt, der den höchsten Knotengrad bei Vernachlässigung aller

Kanten, die mit dem bisher gefundenen Spannbaumn in Zusammenhang stehen, besitzt. Die weiterentwickelte, zufallsbasierte Heuristik C1 wählt nun nicht den Knoten mit maximalen Grad aus, sondern erstellt eine RCL, in der alle Knoten größer gleich einem bestimmten Schwellwert bezüglich des Knotengrads enthalten sind. Anschließend wird ein Knoten zufällig aus der RCL gewählt, die Lösung erweitert und der *Communication Graph* aktualisiert.

Schließlich versucht die Heuristik H2, wie in *Kapitel 20* beschrieben, in jeder Iteration zuerst den Knoten mit kleinstem Knotengrad zu bestimmen und anschließend in seiner Nachbarschaft den Knoten mit größtem Knotengrad, um ihn als Regenerator der Lösung hinzuzufügen. Die Heuristik C2 adaptiert erneut die Auswahl des aussichtsreichsten Knotens für einen Regenerator, indem sie eine RCL abhängig vom Knotengrad der möglichen Kandidatenknoten erstellt. Wie bereits bei der Heuristik C1 wird der nächste Regeneratorknoten zufällig aus der RCL ausgewählt und zur Lösung hinzugefügt.

Nachbarschaftsstruktur 2x1Move aus „GRASP for the Regenerator Location Problem“ [10]

In [10] haben Duarte et al. eine Nachbarschaftsstruktur für die lokale Suche beschrieben, die auf dem in [6] vorgestellten Post-Processing Mechanismus aufbaut. Grundlegende Idee dahinter ist es Regeneratorknoten einer bestehenden Lösung zu eliminieren, damit das Netzwerk mit weniger Regeneratoren so gut wie möglich versorgt ist. Limitierender Faktor ist das Theorem aus [6], dass in einer gültigen Lösung jeder Knoten aus dem *Communication Graph* mit zumindest einem Regeneratorknoten verbunden sein muss. Daher ist die Wahrscheinlichkeit nach der Entfernung eines Regenerators noch immer eine gültige Lösung zu erhalten eher gering, obgleich es trotzdem in manchen Fällen möglich ist. Viel wahrscheinlicher ist es durch das Ersetzen von zwei Regeneratorknoten durch einen anderen die Lösung nachhaltig zu verbessern. Die Autoren sprechen dabei von einem 2×1 Move bzw. 1×0 Move, falls es gelingt einen Regenerator gänzlich zu entfernen.

Ein detaillierter Einblick in die lokale Suche in dieser Nachbarschaftsstruktur wird in *Algorithmus 2.5* gegeben. Grundsätzlich verfolgt der Ansatz eine next improvement Strategie, sodass die Nachbarschaft einer Lösung nur so lange durchsucht wird, bis die erste Verbesserung gefunden wird. Dies geschieht indem für alle beliebigen Knotenpaare u, v einer bestehenden, gültigen Lösung R ein erfolgreicher 2×1 Move versucht wird. Dabei wird jeweils auch automatisch ein 1×0 Move durchgeführt, falls die beiden Regeneratoren durch einen Regenerator in Knoten u oder v ersetzt werden können. Um den lokalen Verbesserungsversuch auszuführen, werden zunächst beide Regeneratorknoten u, v aus der Lösung R vorübergehend entfernt und $R' = R \backslash \{u, v\}$. Eine Menge von Kandidatenknoten für die Platzierung von Regeneratoren C wird als $C = V \backslash R'$ initialisiert. Anschließend wird für jeden Knoten in den vereinigten Nachbarschaften $N(u) \cup N(v)$ die Schnittmenge seiner Nachbarschaft mit der Lösung R' gebildet. Sollte diese Schnittmenge leer sein, also der jeweilige Knoten mit einem Regenerator aus R' keine direkte Verbindung besitzen, muss dieser Knoten auf alle Fälle mit einem neuen Regenerator versorgt werden. In Folge dessen wird die Kandidatenmenge C reduziert, indem sie mit der Nachbarschaft des Knotens geschnitten wird. Sollte die Menge C zu einem Zeitpunkt leer sein, so ist es nicht möglich den 2×1 Move durchzuführen, da es keinen Knoten gibt, der alle

input : *Communication Graph* $M = (V, E')$, gültige Lösung R
output : bessere Lösung R'

```
1  foreach (i, j) ∈ R do
2  |  call 2x1Move(i, j);
3  end
   // 2 × 1 Move für Knotenpaar (i, j)
4  2x1Move(i, j) {
5  |  R' ← {i, j};
6  |  C ← V\R';
7  |  foreach u ∈ {N(i) ∪ N(j)} do
8  |  |  if N(u) ∩ R' = ∅ then
9  |  |  |  C ← C ∩ N(u);
10 |  |  end
11 |  |  if C = ∅ then
12 |  |  |  return null;
13 |  |  end
14 |  end
15 |  while C ≠ ∅ do
16 |  |  Wähle Knoten u aus C;
17 |  |  R' ← R' ∪ {u};
18 |  |  if Lösung ist gültig then
19 |  |  |  return R';
20 |  |  else
21 |  |  |  C ← C\{u};
22 |  |  |  R' ← R'\{u};
23 |  |  end
24 |  end
25 |  return null;
26 }
```

Algorithmus 2.5: Nachbarschaftsstrutkur 2×1 Move aus [10] inklusive Erweiterung

durch die Entfernung der beiden Regeneratoren nicht verbundenen Knoten wieder verbindet. Ist die Kandidatenmenge C jedoch am Ende mit zumindest einem Knoten gefüllt, kann mit der abschließenden Überprüfung der Gültigkeit einer Lösung begonnen werden. Dieser Schritt wird in der Literatur in [6], [10] und [9] weitgehend vernachlässigt, ist aber unabdingbar, um sicherzustellen, dass keine ungültige Lösungen akzeptiert werden. Stellt man sich die Lösung eines RLP gemäß des Theorems aus *Kapitel1.2* als zusammenhängender Spannbaum vor, so funktioniert die bisher beschriebene Vorgangsweise bei zwei Knoten, die Blattknoten des Spannbaums sind sehr gut. Werden allerdings zwei Knoten gewählt, die im Spannbaum auf inneren Knoten liegen, kann es vorkommen, dass durch ihre Entfernung der Spannbaum zerreist, der neue Regeneratorknoten es nicht schafft ihn wieder zusammenhängend zu machen und trotzdem alle Knoten im Graphen gemäß dem Theorem aus [6] zumindest einen Regeneratorknoten als Nachbar besitzen.

In diesem Fall würde eine ungültige Lösung von der lokalen Suche als neue, gültige Lösung des RLP akzeptiert werden. Um diesen Irrtum auf alle Fälle zu vermeiden, wird abschließend der vielversprechendste Knoten (z.B. aufgrund seines Knotengrads) aus C ausgewählt und überprüft, ob die erweiterte Löung R' zusammenhängend ist. Ist dies der Fall, handelt es sich um einen erfolgreichen 2×1 Move und die Lösung wurde verbessert. Kann hingegen mit keinem Knoten aus C ein zusammenhängender Spannbaum von Regeneratorknoten erstellt werden, so ist es nicht möglich den 2×1 Move durchzuführen.

2.3 Zusammenhang des RLP mit dem MLSTP und dem MCDSP

Das Maximum Leaf Spanning Tree Problem (MLSTP) [16] [14] [8] und das Minimum Connected Dominating Set Problem (MCDSP) [17] [3] sind zwei äquivalente Probleme, die in der Literatur bereits sehr ausführlich beschrieben wurden. Stellt man sich den Spannbaum eines ungerichteten, zusammenhängenden Graphen vor, so wird beim MLSTP ein Knoten *Leaf* oder *Blatt* genannt, wenn er zu genau einer Kante im Spannbaum inzident ist. Dabei soll ein Spannbaum gefunden werden, der die Anzahl der Blätter maximiert. Betrachtet man das MCDSP, so soll zu einem Graph G eine Knotenmenge S gefunden werden, bei der der durch S induzierte Subgraph zusammenhängend ist und S eine dominierende Menge (siehe *Kapitel 1.5*) auf G bildet. Werden nun beide Probleme verglichen, so ist leicht zu sehen, dass die inneren Knoten eines Spannbaums mit k Blättern eine zusammenhängende dominierende Menge der Größe $|V| - k$ darstellen. Dies gilt natürlich umgekehrt genauso. Bei beiden Problemen handelt es sich um NP-schwierige Probleme, die nicht in polynomieller Zeit in Bezug zur Problemgröße optimal gelöst werden können.

Der Zusammenhang der beiden Probleme mit dem RLP ist am einfachsten durch die beidseitige Betrachtung von MLSTP und MCDSP zu erklären. Zu diesem Zweck stellt man sich als Probleminstanz einen allgemeinen, ungerichteten Graphen G, sowie eine Menge an NDC-Knotenpaaren N vor. Wird nun für diese Instanz eine gültige Lösung L_{RLP} bestimmt, repräsentiert L_{RLP} alle inneren Knoten eines Spannbaums für den Graphen G. Dies ergibt sich aus der Definition des RLP durch die Forderung, dass jedes Knotenpaar $(u, v) \in N$ entweder durch eine Kante direkt, oder durch einen Pfad zusammenhängender Regeneratorknoten verbunden sein muss. Durch die Spannbaumeigenschaft und die gleichzeitige Minimierung der inneren Regeneratorknoten ist bereits ersichtlich, dass die gefundene Lösung des RLP auch eine gültige Lösung des MLSTP darstellt, also $L_{MLSTP} = L_{RLP}$. Es werden zwar im Zuge der Optimierung jeweils unterschiedliche Ziele verfolgt, jedoch findet in beiden Fällen eine Minimierung der inneren Spannbaumknoten statt. Auch beim MCDSP stellt das gesuchte *Connected Dominating Set* einen Spannbaum dar, bei dem im Zuge der Optimierung die inneren Knoten minimiert werden, wodurch $L_{MLSTP} = L_{RLP} = L_{MCDSP}$ gilt. Durch diese Beobachtungen handelt sich beim MLSTP, MCDSP und RLP um äquivalente Probleme, deren optimale Lösungen in den übrigen Problemen ebenso optimale Lösungen darstellen. Dadurch ist auch jede Lösungsmethode, die für eine der drei Probleme konzipiert wurde für die beiden anderen anwendbar bzw. die gefundenen Lösungen ebenfalls als gültige Lösungen der anderen Probleme verwendbar.

2.4 Maximum Clique Problem & Maximum Independent Set Problem

Maximum Clique Problem

Beim Maximum Clique Problem (MCP) [32] [11] handelt es sich im Fall eines zusammenhängenden, ungerichteten Graphen um ein NP-schwieriges Problem, das die Bestimmung der größten Clique (Clique siehe *Kapitel 1.5*) zum Ziel hat. Als Zielfunktionskriterium wird dabei die Kardinalität der Clique verwendet. Betrachtet man das RLP, so existiert im aktualisierten *Communication Graph* nach Bestimmung einer gültigen Lösung eine maximale Clique der Größe $|V|$. Dabei werden in den in der Literatur beschriebenen Lösungsverfahren [6] vorwiegend Knoten durch die Anzahl an beeinflussten NDC-Knotenpaaren als Regeneratorknoten für die resultierende Endlösung ausgewählt. Würde man das Problem aus einem anderen Blickwinkel betrachten, so wäre die schrittweise Erweiterung, oder Zusammenfassung von Cliques solange, bis die maximale Größe von $|V|$ erreicht ist eine weitere Möglichkeit das RLP zu lösen. Dabei können für die Bestimmung und Erweiterung dieser Cliques viele Mechanismen, die im Zuge der Lösung des MCP verwendet werden, durchaus nützlich sein. In *Kapitel 3.3* wird diese Idee für eine Konstruktionsheuristik des RLP umgesetzt.

Für die formale Definition des Maximum Clique Problems für einen beliebigen, ungerichteten Graphen $G = (V, E)$ bestehend aus der Menge der Knoten $V = \{1, 2, , \ldots, n\}$ und der Menge der Kanten $E \subseteq V \times V$ wird ein Subgraph

$$G_S = (S, E'), E' = E \cap S \times S \tag{2.1}$$

herangezogen, der durch die Knotenmenge $S \subseteq V$ induziert wird. Damit die Clique-Eigenschaft erfüllt ist, muss der Subgraph G_S vollständig sein, d.h. es gilt

$$\forall u, v \in S, (u, v) \in E'. \tag{2.2}$$

Beim MCP wird nun gefordert, dass jene Clique im Graphen gefunden wird, die die maximale Kardinalität besitzt, also für die Zielfunktion $f(S)$ und optimale Lösung S^*

$$\begin{aligned} f(S) &= |S| \\ f(S^*) &\geq f(S), \forall S \in \mathcal{P}(V) \end{aligned} \tag{2.3}$$

gilt.

Als Lösungsvarianten für das MCP haben sich unterschiedliche Strategien bewährt. Diese reichen von enumerativen Algorithmen, über heuristische Verfahren bis hin zu exakten Algorithmen. Bei den enumerativen Algorithmen haben vor allem Bron und Kerbosch [2] durch das Einführen von backtracking Methoden einen maßgeblichen Einfluss auf die Entwicklung der Verfahren in diesem Bereich. Im Bereich des heuristischen Lösungsansatzes des MCP werden vor allem sequentielle greedy Heuristiken und Heuristiken mit Randomisierung eingesetzt. Die von Kopf und Ruhe erforschten [27] so genannten sequentiellen greedy Algorithmen versuchen dabei eine maximale Clique durch das Hinzufügen des nächsten „besten“ oder durch das Herausnehmen des „schlechtesten“ Knotens zu erreichen. Als exakte Lösungsverfahren haben sich

besonders Branch-and-Bound Algorithmen zur Auffindung einer maximalen Clique etabliert und als drastische Kürzung zu enumerativen Verfahren erwiesen. Dies geschieht indem nach Auffinden der ersten Clique nur noch Cliques mit einer höheren Kardinalität betrachtet werden.

Maximum Independent Set Problem

Das Maximum Independent Set Problem (MISP) [34] ist durch den Zusammenhang von Cliques (siehe *Kapitel 1.5*) und Independent Sets (stabile Menge, siehe *Kapitel 1.5*) eng mit dem Maximum Clique Problem verbunden. Betrachtet man einen beliebigen, ungerichteten Graph $G = (V, E)$ mit Knotenmenge $V = \{1, 2, \ldots, n\}$ und Kantenmenge $E \subseteq V \times V$, ist der entsprechende Komplementärgraph $\overline{G} = (V, \overline{E})$ mit $\overline{E} = \{(u, v) \in V \times V | (u, v) \notin E\}$ gegeben. Jede Knotenmenge S, die eine Clique in G darstellt, ist automatisch auch eine stabile Menge in $\overline{G}$. Das gleiche gilt umgekehrt natürlich genauso. Darüber hinaus trifft dies auch für die Maximums- und Minimumsbedingung zu, sodass eine maximale oder minimale Clique in G auch eine maximale oder minimale stabile Menge in $\overline{G}$ darstellt. Daher kann jedes MCP durch geschickte Umformung anhand des Komplementärgraphs in ein MISP verwandelt werden.

Für die Definition des MISP wird der Graph $G = (V, E)$ betrachtet. Eine Lösung des MISP ist eine Knotenmenge S, die den Subgraph $G(S) = (S, E \cap S \times S)$ induziert. Damit es sich um eine gültige Lösung handelt, muss für alle Knotenpaare aus S die Bedingung

$$\forall u, v \in S : (u, v) \notin E \tag{2.4}$$

erfüllt sein. Die optimale Lösung S^* optimiert die Kardinalität von S, sodass

$$\begin{aligned} f(S) &= |S| \\ f(S^*) &\geq f(S), \forall S \in \mathcal{P}(V) \end{aligned} \tag{2.5}$$

gilt. Das MISP ist ebenfalls ein NP-schwieriges Problem, d.h. es kann höchstwahrscheinlich keine optimale Lösung in polynomieller Zeit in Bezug zum Input gefunden werden.

Durch den engen Zusammenhang mit dem MCP können die gleichen Überlegungen bezüglich der Ableitung von Lösungsmethoden für das RLP auch für das MISP angestellt werden. Dabei besitzt der aktualisierte *Communication Graph* einer gültigen Lösung für das RLP einen komplett unzusammenhängenden Komplementärgraph. Es gibt also eine maximale stabile Menge der Größe $|V|$. Als mögliche Vorgehensweise kann daher schrittweise jener Knoten zur Lösung hinzugefügt werden, der am Komplementärgraph in der aktuell größten stabilen Menge die meisten Kanten in $\overline{G}$ eliminiert, sodass die größte stabile Menge in $\overline{G}$ weiter wächst. Diese Idee wurde in zwei Konstruktionsheuristiken in *Kapitel 3.1* und *Kapitel 3.2* als Hauptkonzept eingebaut.

Als mögliche Lösungsmethoden des MISP wurden genauso wie beim MCP sowohl exakte, als auch heuristische Ansätze entwickelt. Den ersten exakten Algorithmus zur Lösung des MISP haben Tarjan und Trojanowski [34] mit einer Laufzeit von $O(1.2599^n)$ entdeckt. Aufbauend auf ihren Ergebnissen hat zunächst Jian [24] die Komplexität weiter auf $O(1.2346^n)$ und später Robson [33] auf $O(1.2108^n)$ verbessert. Im Bereich der heuristischen Methoden wurden in der Literatur sowohl unterschiedliche Metaheuristiken wie GRASP oder Ant Colony Optimization, als auch evolutionäre Algorithmen zur Lösung des MISP vorgeschlagen.

KAPITEL 3

Lösungsansatz

Die in der Literatur beschriebenen und in *Kapitel 2.1* und *Kapitel 2.2* vorgestellten Heuristiken, bilden die Ausgangslage für die Erarbeitung neuer Ansätze. Dabei werden einerseits neue Mechanismen erforscht, die es ermöglichen sollen noch schneller Lösungen zu generieren und andererseits bestehende Lösungsansätze hinsichtlich möglicher Verbesserungspotenziale durchleuchtet, um die allgemeine Lösungsqualität zu steigern.

Im Zuge der Entwicklung von verbesserten Konstruktionsheuristiken für das RLP haben sich Methoden, die besser die Struktur des Graphen miteinbeziehen, wie z.B. durch die Berechnung von stabilen Mengen oder Cliques (siehe *Kapitel 1.5*), als sehr gute Hilfsmittel zur Bestimmung von guten Lösungen herausgestellt. Bei der bisher effektivsten in der Literatur beschriebenen Konstruktionsheuristik, die in *Kapitel 2.2* vorgestellt wurde, stellt die Berechnung der Hilfsfunktion $\chi(v)$ für alle Kandidatenknoten v den kritischen Punkt hinsichtlich der Laufzeit dar. Durch den Einsatz von Markierungsalgorithmen können in Kombination mit strukturausnutzenden Methoden hier deutlich schnellere und bessere Ergebnisse erzielt werden. Insgesamt wurden drei verschiedene Strategien entwickelt, die einerseits durch die Bestimmung von stabilen Mengen (siehe *Kapitel 3.1*, bzw. *Kapitel 3.2*) und andererseits durch das Auffinden von Cliques im *Communication Graph* (siehe *Kapitel 3.3*) die Konstruktion von Lösungen für eine beliebige Instanz des RLP durchführen.

3.1 Independent Set Konstruktionsheuristik

Die Independent Set Konstruktionsheuristik versucht anhand der Bestimmung von stabilen Mengen (siehen *Kapitel 1.5*) aus der Struktur des Graphen wichtige Informationen für den Optimierungsschritt zu gewinnen. Dabei wird das in der Literatur (in [6]) eingeführte Konzept der NDC-Knotenpaare erweitert. NDC-Knotenpaare geben Auskunft über das Fehlen einer direkten Kante zwischen jeweils zwei Knoten. Eine stabile Menge mit Kardinalität $n \geq 2$ enthält implizit die Information über genau $\sum_{k=1}^{n-1} k = \frac{n \cdot (n-1)}{2}$ nicht direkt verbundener Knotenpaare aus M. Gelingt es nun einen Kandidatenknoten zu finden, der sehr viele Knoten der stabilen Menge als

direkte Nachbarn besitzt, eliminiert ein Regenerator an dieser Stelle gleichzeitig mehrere NDC-Knotenpaare aus M. Dies ähnelt dem in [10] beschriebenen Vorgehen, jeweils den Knoten, der die größte Anzahl an NDC-Paaren eliminiert, zu selektieren, jedoch ohne für jeden Kandidatenknoten alle existierenden NDC-Knotenpaare in seiner Nachbarschaft zu überprüfen. Es reicht aus lediglich die Nachbarschaft hinsichtlich der stabilen Menge zu evaluieren.

Natürlich ist es nicht einfach für jede Situation die geeignete stabile Menge im Graphen zu finden, da es üblicherweise viele solcher Mengen mit unterschiedlichen Kardinalitäten gibt. In der Theorie erscheint es es optimal, wenn möglichst große stabile Mengen im Graphen gefunden werden und diese obendrein noch so liegen, dass in jeder Iteration ein weiterer Knoten aus der optimalen Lösung für die Instanz des RLP identifiziert wird. In der Praxis ist die Bestimmung von maximalen stabilen Mengen selbst ein NP-schwieriges Problem (siehe [34]) und die Lage bzw. das Aussehen der stabilen Menge oft nicht beeinflussbar. Trotzdem reicht es aus nicht die größte aller stabilen Mengen exakt zu bestimmen, sondern so große stabile Mengen wie möglich im Graphen zu identifizieren, um annähernd die gleiche Auswahlsgüte für einen Regenerator zu erreichen, wie dies mit der Evaluierung der Hilfsfunktion $\chi(v)$ in [6] der Fall ist. Durch die mehrmalige Durchführung des Konstruktionsschritts innerhalb einer Metaheuristik können etwaige Einschränkungen durch die Auswahl von nicht-optimalen stabilen Mengen hinsichtlich Größe und Lage ohnehin ausgeglichen werden. Darüber hinaus ist es möglich bei der Existenz einer maximalen stabilen Menge mit Kardinalität eins sofort durch die Platzierung eines Regenerators eine gültige Lösung zu erzeugen. Aus der Eigenschaft der maximalen stabilen Menge folgt, dass alle übrigen Knoten eine direkte Kante zu dem einzigen Knoten in der stabilen Menge haben müssen. Dadurch eliminiert die Platzierung eines Regenerators in diesem Knoten alle noch bestehenden NDC-Paare.

Grob lässt sich das Vorgehen der Independent Set Konstruktionsheuristik folgendermaßen skizzieren:

1. Initialisierung der Kandidatenknoten
2. Solange es noch NDC-Knotenpaare gibt:
    a) Bestimmung von mehreren stabilen Mengen im Komplementärgraph von M, sodass jeder Knoten in zumindest einer stabilen Menge vorkommt
    b) Überprüfung ob es eine stabile Menge mit Kardinalität eins gibt
    c) Bestimmung des Kandidatenknotens, der die beste Bewertung hinsichtlich der Anzahl an adjazenten Knoten aus den stabilen Mengen besitzt
    d) Erweiterung der Lösung durch den gefundenen Knoten und Aktualisierung des *Communication Graphen*

Da es durch die NP-Schwierigkeit des Maximum Independent Set Problems ohnehin nicht möglich ist in polynomieller Zeit abhängig von der Problemgröße immer die größte stabile Menge im Graphen exakt zu bestimmen, wird zu diesem Zweck ein Näherungsverfahren benutzt. Dabei gibt es prinzipiell zwei mögliche Vorgehensweisen, nämlich entweder die direkte Berechnung von stabilen Mengen am *Communication Graph*, oder die Bestimmung von Cliques im Komplementärgraph $\overline{M}$. Dies ist durch den Zusammenhang gegeben, dass jede stabile Menge

im *Communication Graph* M eine Clique im Komplementärgraph $\overline{M}$ darstellt (siehe *Kapitel 1.5*). Beide Vorgehensweisen liefern dabei die gleichen Ergebnisse, es ändert sich lediglich die Art der Auswahl je nachdem ob eine stabile Menge oder eine Clique bestimmt wird.

Im Zuge der Independent Set Konstruktionsheuristik wird die Bestimmung von Cliques am Komplementärgraph vorgenommen. Zu diesem Zweck wird die Knotenmenge U, die jene Knoten bezeichnet die sich in noch keiner Clique befinden, mit den Knoten aus V initialisiert. Anschließend wird für die Berechnung einer Clique C jeweils eine neue Menge U' gebildet, die jene Knoten enthält, die mit den bisherigen Knoten aus C direkt verbunden sind und dadurch mögliche Kandidaten zur Erweiterung der Clique darstellen. Aus der Menge U' wird in jeder Iteration jener Knoten als Clique-Knoten identifiziert, der die meisten adjazenten Nachbarn aus U' besitzt. Sollten mehrere Knoten die gleiche maximale Anzahl an adjazenten Nachbarn besitzen, wird einer von ihnen zufällig als nächster Clique Knoten ausgewählt. Anschließend wird U' um jene Knoten reduziert, die keine direkte Kante zum zuletzt hinzugefügten Knoten besitzen. Sobald U' leer ist wurde eine maximale Clique C in $\overline{M}$ gefunden, U durch das Entfernen der Knoten aus C entsprechend aktualisiert und solange mit der Konstruktion von Cliques fortgefahren, bis U leer ist. Durch dieses Vorgehen ist jeder Knoten aus M in zumindest einer maximalen stabilen Menge enthalten. Alternativ zu der Auswahl des Knotens mit den meisten Nachbarn aus U', können Cliques auch durch Auswahl des Knotens mit dem a) kleinsten Knotengrad, b) größten Knotengrad, oder c) einem zufälligen Knoten aus U' konstruiert werden. Im Rahmen dieser Diplomarbeit wurden in den dokumentierten Tests in *Kapitel 4* ausschließlich die vorgestellte Strategie der meisten verbleibenden Nachbarknoten als Konstruktionsmechanismus für Cliques innerhalb der Independent Set Heuristik verwendet.

Nach der Bestimmung aller stabilen Mengen erfolgt die Bewertung der einzelnen Kandidatenknoten v anhand der Bewertungsfunktion $\Phi(v)$. Ziel ist es jenen Knoten zu finden, an dessen Stelle ein Regenerator sehr viele Knoten unterschiedlicher stabiler Mengen verbinden würde. Um dies zu erreichen werden in der Bewertung für einen bestimmten Kandidatenknoten v mehrere Brüche gebildet, die jeweils das Verhältnis zwischen den Nachbarknoten von v in der stabilen Menge $I_j,\ j = 1, \ldots, |I|$ zu ihrer jeweiligen Größe $|I_j|$ darstellen. Durch die Multiplikation der einzelnen Brüche werden jene Kandidatenknoten bevorzugt, die viele Knoten von vielen stabilen Mengen verbinden, was zum gewünschten Effekt führt, dass typischwerweise eher „zentrale“ Knoten als attraktive Orte für Regeneratoren identifiziert werden. Die Funktion $\Phi(v)$ sieht folgendermaßen aus:

$$\Phi(v) = \prod_{j=1}^{|I|} \frac{\min(1, |N(v) \cap I_j|)}{|I_j|}. \tag{3.1}$$

Anschließend wird der Kandidatenknoten mit der höchsten Bewertung als Regeneratorknoten ausgewählt und zur Lösung R hinzugefügt. Sollten zwei Kandidatenknoten die genau gleiche, beste Bewertung aufweisen, wird jener bevorzugt, der den größeren Knotengrad besitzt, bzw. im Fall von gleichen Knotengraden einer von ihnen zufällig ausgewählt. Am Ende jeder Iteration wird der *Communication Graph* durch Hinzufügen neu entstandener Kanten aktualisiert und falls weitere NDC-Knotenpaare existieren mit der nächsten Iteration und der neuerlichen Bestimmung aller Cliques im Komplementärgraph $\overline{M}$ fortgefahren.

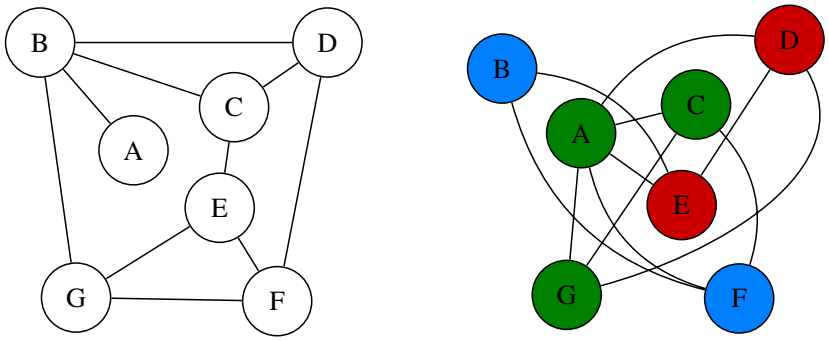
input : *Communication Graph* $M = (V, E')$
output : Menge von Regeneratorknoten R

```
1  IndependentSetHeuristik(M) {
2      R ← ∅;
3      C ← V\L;
4      while !M.isFullyConnected() do
5          if ∃ v ∈ V mit |N(v)| == |V| − 1 then
6              return R ∪ {v};
7          end
8          I ← berechne alle stabilen Mengen im Graphen;
9          v_best ← null;
10         foreach v ∈ C do
11             if Φ(v) > Φ(v_best) then
12                 v_best ← v;
13             end
14         end
15         R ← R ∪ {v_best};
16         C ← C\{v_best};
17         Aktualisiere M;
18     end
19     return R;
20 }
```

Algorithmus 3.1: Pseudocode der Independent Set Konstruktionsheuristik

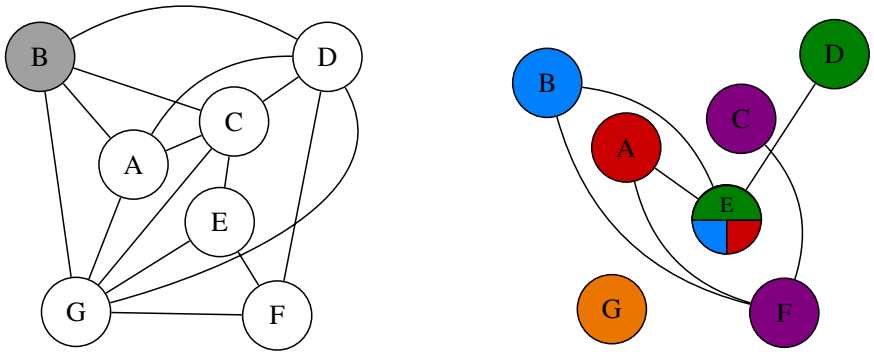
Demonstration der Funktionsweise anhand eines praktischen Beispiels

Eine detaillierte Beschreibung des Algorithmus in Form eines Pseudocodes ist in *Algorithmus 3.1* gegeben. Um einen Einbick in die Funktionsweise der Heuristik zu geben, wird sie auf ein praktisches Beispiel in *Abbildung 3.1* mit 7 Knoten angewendet. Da zu Beginn noch kein Regenerator gesetzt wurde, wird die Menge der Kandidatenknoten C mit der Menge aller Knoten V initialisiert. Anschließend wird mit der ersten Iteration und der Bestimmung der Regeneratorknoten begonnen. Zunächst wird überprüft, ob es im *Communication Graph* einen Knoten mit Knotengrad gleich $|V| - 1$ gibt, was in der ersten Iteration nicht der Fall ist. Als nächstes wird die Menge der stabilen Mengen I mit Hilfe der Auswahl der meisten übereinstimmenden Nachbarknoten bestimmt. Diese findet insgesamt drei stabile Mengen für den *Communication Graph*, nämlich $I = \{(A, C, G), (B, F), (D, E)\}$. Nun wird für jeden Kandidatenknoten v die Bewertungsfunktion $\Phi(v)$ evaluiert und jener Knoten mit dem höchsten Wert als ausichtsreichste Position für einen Regenerator ausgewählt. Beispielhaft für den Knoten C berechnet, ist der Wert der Bewertungsfunktion $\Phi(C) = 1/3 \cdot 1/2 \cdot 2/2 = 0,17$. Die übrigen Bewertungsfunktionswerte sind $\Phi(A) = 0,08$, $\Phi(B) = 0,25$, $\Phi(D) = 0,17$, $\Phi(E) = 0,17$, $\Phi(F) = 0,17$ und $\Phi(G) = 0,17$. Dadurch wird der Kandidatenknoten B mit der höchsten Bewertung von $0,25$ ausgewählt und $R = \{B\}$ erweitert. Anschließend werden neu entstandene direkte Verbindun-



Communication Graph M **Komplementärgraph** $\overline{M}$

Abbildung 3.1: Communication- und Komplementärgraph der Beispielinstanz für die Anwendung der IndependentSet Konstruktionsheuristik zu Beginn der ersten Iteration



Communication Graph M **Komplementärgraph** $\overline{M}$

Abbildung 3.2: Communication- und Komplementärgraph der Beispielinstanz für die Anwendung der IndependentSet Konstruktionsheuristik zu Beginn der zweiten Iteration

gen im *Communication Graph* M als direkte Kanten eingefügt und in der nächsten Iteration die Menge I neu berechnet. Nach Einfügen der Kanten $(A, D), (A, C), (A, G), (C, G)$ und (D, G) sind die neuen stabilen Mengen $I = \{(A, E), (B, E), (C, F), (D, E), (G)\}$. Da G eine stabile Menge mit Kardinaliät eins bildet, bzw. der Knotengrad von G, $Grad(G) = |V| - 1$ beträgt, ist die sofortige Bestimmung einer gültigen Lösung für die Beispielinstanz mit $R = \{B, G\}$ möglich, was gleichzeitig die optimale Lösung für diese Instanz darstellt.

Laufzeitanalyse der Independent Set Konstruktionsheuristik

Bei der Analyse der Independent Set Konstruktionsheuristik hinsichtlich ihrer Laufzeit muss zunächst die Bestimmung aller stabiler Mengen im Graphen betrachtet werden. Da jeder Knoten in mehr als einer stabilen Menge vorkommen kann, bedeutet dies im schlechtesten Fall einen Aufwand von $O(|V|^2)$. Anschließend wird für jeden Kandidatenknoten v die Bewertungsfunk-

tion $\Phi(v)$ evaluiert. Für die Berechnung des Funktionswerts muss für jeden Kandidatenknoten überprüft werden, mit wie vielen Knoten er pro stabiler Menge direkt benachbart ist. Nachdem die stabilen Mengen so gewählt sind, dass jeder Knoten in zumindest einem vorkommt, müssen insgesamt pro Kandidatenknoten $|V|$ Vergleiche vorgenommen werden plus einem linearen Faktor, der jenen Knoten entspricht, die in mehreren stabilen Mengen vorkommen. Durch den Umstand, dass im Laufe der Optimierung die stabilen Mengen einerseits immer kleiner werden und andererseits die Anzahl an Kandidatenknoten immer weiter reduziert wird, kann die Anzahl an notwendigen Bewertungsschritten insgesamt im schlechtesten Fall mit $O(|V|^2)$ abgeschätzt werden. Die Aktualisierung des *Communication Graphen* kann in $O\left(\frac{|N(v_{best})|^2}{2}\right)$ Durchläufen bewerkstelligt werden, wobei $N(v_{best})$ den Knotengrad des besten Kandidatenknotens bezeichnet, der auf alle Fälle $< |V|$ ist. Insgesamt werden für die Independent Set Konstruktionsheuristik im schlechtesten Fall maximal $O(|V|)$ Iterationen durchgeführt, wodurch sich eine Gesamtlaufzeit von $O\left(|V| \cdot \left(|V|^2 + |V|^2 + \left(\frac{|N(v_{best})|^2}{2}\right)\right)\right) = O(|V|^3)$ ergibt.

3.2 Simplified Independent Set Konstruktionsheuristik

Im Gegensatz zur Independent Set Konstruktionsheuristik berechnet die Simplified Independent Set Konstruktionsheuristik in jeder Iteration nur eine einzige stabile Menge mit Hilfe eines Markierungsalgorithmus direkt am *Communication Graph*. Dabei ist das Ziel eine so große bzw. repräsentative stabile Menge wie nur möglich zu finden, damit die Auswahl des nächsten Kandidatenknotens möglichst gut erfolgen kann. Aus diesem Grund wird in die Bewertung von Kandidatenknoten neben der Anzahl an benachbarten Knoten der stabilen Menge auch ein Faktor miteinbezogen, der Auskunft über die Nähe zu bereits identifizierten Regeneratorknoten gibt. Generell wird bei der Simplified Independent Set Heuristik der in [6] beschriebene Grundsatz ausgenutzt, dass eine gültige Lösung des RLP stets als Spannbaum mit Regeneratorknoten an allen inneren Knoten konstruiert werden kann. Dadurch ist es vor allem in späteren Iterationen wichtig Kandidatenknoten, die sich in der Nähe von bereits platzierten Regeneratoren befinden, als attraktive Positionen für Regeneratoren zu identifizieren, um die Spannbaumeigenschaft einer gültigen Lösung zu erreichen. Die Bewertungsfunktion Φ bezieht daher auch eine Distanzinformation über den kürzesten Pfad $SP(u, v)$ zwischen zwei Knoten $u, v \in V$ mit ein. Darüber hinaus stellt I_1 die zu Beginn jeder Iteration bestimmte stabile Menge dar, $N(v)$ die adjazenten Knoten zum Kandidatenknoten v, sowie R die Menge der bereits ausgewählten Regeneratorknoten. Die Funktion Φ sieht daher folgendermaßen aus:

$$\Phi(v) = \frac{|N(v) \cap I_1|}{|I_1|} + \frac{|R|}{|V|} \cdot \frac{\max_{u \in V, w \in R} SP(u, w)}{\max(\min_{w \in R} SP(v, w), 1)}. \tag{3.2}$$

Der erste Bruch stellt aus Sicht des Kandidatenknotens v das Verhältnis zwischen den direkt benachbarten Knoten der stabilen Menge und ihrer Gesamtgröße dar. Dieser Teil der Bewertung soll in allen Phasen der Simplified Independent Set Heuristik einen gleich großen Einfluss auf die Gesamtbewertung haben. Der zweite Teil der Funktion versucht die Distanz zu bereits identifizierten Regeneratorknoten miteinzubeziehen. Diese Information soll jedoch erst in späteren

input : *Communication Graph* $M = (V, E')$
output : Menge von Regeneratorknoten R

```
1  SimplifiedIndependentSetHeuristik(M) {
2      R ← ∅;
3      C ← V\L;
4      while !M.isFullyConnected() do
5          if ∃ v ∈ V mit |N(v)| == |V| − 1 then
6              return R ∪ {v};
7          end
8          I ← berechne eine stabile Menge;
9          v_best ← null;
10         foreach v ∈ C do
11             if Φ(v) > Φ(v_best) then
12                 v_best ← v;
13             end
14         end
15         R ← R ∪ {v_best};
16         C ← C\{v_best};
17         Aktualisiere M;
18     end
19     return R;
20 }
```

Algorithmus 3.2: Pseudocode für die Simplified Independent Set Konstruktionsheuristik

Iterationen eine deutliche Auswirkung auf die Bewertung der Kandidatenknoten haben und wird daher mit dem Verhältnis zwischen der aktuellen Anzahl an identifizierten Regeneratorknoten $|R|$ und der gesamten Anzahl an Knoten $|V|$ gewichtet. Die Distanzinformation selbst dividiert den in der jeweiligen Iteration maximalen kürzesten Pfad zwischen einem Knoten $u \in V$ und einem Regeneratorknoten $w \in R$ durch den kürzesten Pfad zwischen dem aktuellen Knoten v und einem Regeneratorknoten $w \in R$, bzw. 1 falls R leer ist. Dadurch werden in späteren Iterationen Knoten besser bewertet, die eine vergleichsweise kurze Entfernung zu einem bereits identifizierten Regeneratorknoten besitzen. Anschließend wird der Kandidatenknoten mit der höchsten Bewertung als nächster Regeneratorknoten ausgewählt und zur Lösung R hinzugefügt. Sollten zwei Kandidatenknoten die genau gleiche, beste Bewertung besitzen, wird jener bevorzugt der den größeren Knotengrad besitzt, bzw. bei Gleichheit des Knotengrads eine Zufallsauswahl zwischen den beiden vorgenommen. Am Ende jeder Iteration wird der *Communication Graph* durch Hinzufügen neu entstandener Kanten aktualisiert und falls weitere NDC-Knotenpaare existieren mit der nächsten Iteration fortgefahren. Ein Pseudocode, der die Funktionsweise der Simplified Independent Set Heuristik illustriert ist in *Algorithmus 3.2* zu finden.

Wie bereits erwähnt erfolgt die Bestimmung der stabilen Menge I_1 direkt am *Communication Graph* mit Hilfe eines Markierungsalgorithmus. Das Vorgehen ähnelt dabei in etwa der Bestimmung der Cliques bei der Independent Set Heuristik. Zuerst werden alle Knoten aus V als

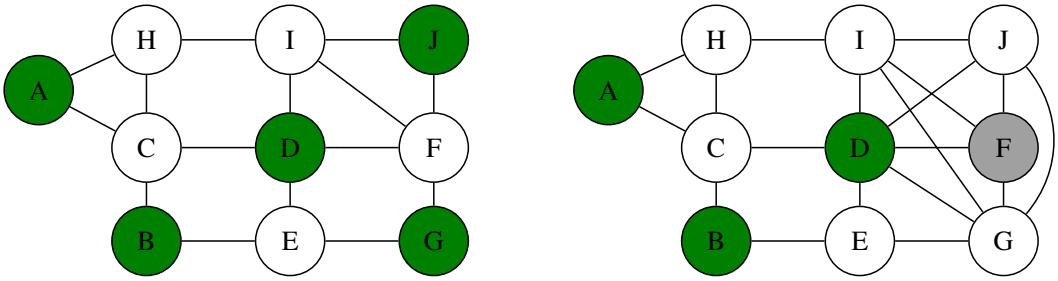


Abbildung 3.3: *Communication Graph* der Beispielinstanz für die Anwendung der Simplified Independent Set Konstruktionsheuristik zu Beginn der ersten und zweiten Iteration

„noch nicht markiert“ betrachtet. Nun wird jener Knoten ausgewählt, der am wenigsten gemeinsame adjazente Nachbarn mit noch nicht markierten Knoten besitzt. Sollten zwei Knoten die genau gleiche Anzahl solcher Nachbarknoten besitzen, wird einer von ihnen zufällig ausgewählt. Der betreffende Knoten wird zur stabilen Menge I_1 hinzugefügt, seine adjazenten Nachbarn als „markiert“ betrachtet und mit der Auswahl des nächsten unmarkierten Knotens fortgefahren, solange bis alle Knoten entweder Bestandteil von I_1 sind, oder den Status „markiert“ aufweisen. Die resultierende stabile Menge I_1 stellt eine maximale, d.h. nicht weiter erweiterbare stabile Menge auf M dar. Alternativ ist es auch denkbar den nächsten Knoten für die stabile Menge jeweils anhand des a) niedrigsten Knotengrads, b) höchsten Knotengrads, oder c) durch eine Zufallsauswahl auszuwählen. Jedoch benutzt die Simplified Independent Set Heuristik in den dokumentierten Ergebnissen in *Kapitel 4* ausschließlich die beschriebene Auswahstrategie, die die minimale Anzahl an adjanzenten unmarkierten Knoten bevorzugt.

Demonstration der Funktionsweise anhand eines praktischen Beispiels

Um den Ablauf der Simplified Independent Set Konstruktionsheuristik besser zu verstehen, wird in *Abbildung 3.1* eine Demonstration anhand einer klassischen RLP Instanz durchgeführt. Im Initialisierungsschritt wird die Menge der Kandidatenknoten C mit der gesamten Knotenmenge V initialisiert. Da es keinen Knoten gibt, der mit allen anderen Knoten direkt benachbart ist und eine Sofortlösung zulassen würde, wird eine stabile Menge I nach der beschriebenen Auswahlstrategie erzeugt. Dabei werden zunächst alle Knoten aus V als unmarkiert betrachtet und es wird jener mit der kleinsten Anzahl an adjazenten, unmarkierten Knoten (bei gleicher Anzahl zwischen mehreren Knoten per Zufall) als erster Knoten für die stabile Menge ausgewählt. In diesem Fall wird zuerst A bestimmt und anschließend dieser Knoten, sowie all seine direkten Nachbarn $N(A) = \{H, C\}$ als „markiert“ betrachtet. Die verbleibenden unmarkierten Knoten sind $\{B, D, E, F, G, I, J\}$. Diese Prozedur wird solange durchgeführt, bis es keine unmarkierten Knoten mehr gibt und schließlich die stabile Menge $I_1 = \{A, B, D, G, J\}$ gefunden wurde. Anschließend erfolgt die Bewertung aller Kandidatenknoten C anhand der Bewertungsfunktion Φ. Beispielhaft am Knoten I berechnet, beträgt der Koeffizient für den Verbindungsgrad der stabilen Menge $2/5$, also $0,4$. Der Distanzkoeffizient zu einem bereits fixierten Regenerator ist

null, da bisher noch kein Regenerator platziert wurde. Der Wert der Bewertungsfunktion für den Knoten I ist daher $0,4$. Die übrigen berechneten Werte sind $\Phi(A) = 0$, $\Phi(B) = 0$, $\Phi(C) = 0,6$, $\Phi(D) = 0$, $\Phi(E) = 0,6$, $\Phi(F) = 0,6$, $\Phi(G) = 0$, $\Phi(H) = 0,2$ und $\Phi(J) = 0$. Den größten Wert in der ersten Iteration haben daher die Knoten C, E und F. Bei gleichem Bewertungsfunktionswert werden zunächst Knoten mit höherem Knotengrad bevorzugt, da sie durch die größere Anzahl an Nachbarn mehr NDC-Paare eliminieren. Aus diesem Grund bleiben nur noch C und F übrig, da beide den Knotengrad vier besitzen. Durch die erneute Pattstellung wird einer der verbleibenden Knoten zufällig als Regeneratorknoten ausgewählt und der Lösung $R = \{F\}$ hinzugefügt. Als nächstes wird der *Communication Graph* mit den neu entstandenen Verbindungen aktualisiert und nachdem weitere NDC-Paare existieren die zweite Iteration gestartet. Wiederrum gibt es keinen Knoten mit Knotengrad $|V| - 1 = 8$, daher wird eine neue stabile Menge mit Hilfe der Auswahlstrategie bestimmt, nämlich $I_1 = \{A, B, D\}$. Da nun die Menge an Regeneratorknoten nicht leer ist, wird der maximale kürzeste Pfad zwischen einem Knoten und Regenerator $\max_{u \in V, w \in R} SP(u, w) = 3$ benötigt. Beispielhaft demonstriert am Knoten D, beträgt daher der Verbindungsgradskoeffizient null, jedoch der Distanzkoeffizient dank der Distanz des kürzesten Pfads von D zu F von $SP(D, F) = 1$ mehr als null. Die Gesamtbewertung des Knotens D setzt sich dadurch als $\Phi(D) = (0 + 1/9 \cdot 3 = 1/3) = 0,3$ zusammen. Die übrigen Werte der Kandidatenknoten sind $\Phi(A) = 0,1$, $\Phi(B) = 0,1$, $\Phi(C) = 1,15$, $\Phi(E) = 0,82$, $\Phi(G) = 0,63$, $\Phi(H) = 0,48$, $\Phi(I) = 0,63$ und $\Phi(J) = 0,63$. Mit einer Bewertung von $1,15$ ist der Knoten C der aussichtsreichste Kandidat für einen Regenerator und wird der Lösung $R = \{F, C\}$ hinzugefügt. Nachdem der *Communication Graph* aktualisiert wurde, wird die dritte Iteration gestartet. Nun besitzt der Knoten D einen Knotengrad von $|V| - 1 = 8$ und es kann dadurch die Lösung durch Hinzufügen des Knotens D als gültige Lösung abgeschlossen werden. Schlussendlich ist die Instanz mit $R = \{F, C, D\}$ – also mit drei Regeneratoren – optimal gelöst.

Laufzeitanalyse der Simplified Independent Set Konstruktionsheuristik

Bei der Analyse der Laufzeit der Simplified Independent Set Konstruktionsheuristik muss zunächst die Bestimmung der stabilen Menge I_1 pro Iteration betrachtet werden. Dabei werden für die Auswahl des ersten Knotens insgesamt $|V|$ Vergleiche durchgeführt. Nimmt man $N(v)$ als den durchschnittlichen Knotengrad im Graph an, so werden insgesamt $N(v) \cdot |V|$ Schritte für den ersten Knoten benötigt. Da die Anzahl an unmarkierten Knoten rapide abnimmt, entsteht ein Gesamtaufwand von $O(N(v) \cdot \log |V|)$ zur Bestimmung der stabilen Menge. Die Berechnung der Bewertungsfunktion pro Kandidatenknoten bedeutet einen Aufwand von $O(|V| \cdot |I_1|)$, während die Auswahl des besten Kandidatenknotens nur $O(|V|)$ Durchläufe benötigt. Die Aktualisierung des *Communication Graphen* erfolgt in $O\left(\frac{|N(v_{best})|^2}{2}\right)$ Schritten, wobei $N(v_{best})$ den Grad des besten Kandidatenknotens bezeichnet. Da pro Iteration immer ein Regeneratorknoten ausgewählt wird, kann es maximal $O(|V|)$ Iterationen geben, wodurch sich eine Gesamtlaufzeit von $O\left(|V| \cdot \left(N(v) \cdot \log |V| + |V| \cdot |I_1| + |V| + \frac{|N(v_{best})|^2}{2}\right)\right) = O(|V|^2)$ ergibt.

3.3 Clique Konstruktionsheuristik

Die Clique Konstruktionsheuristik versucht im Gegensatz zu den bereits vorgestellten Verfahren stark zusammenhängende Knotenverbände im Graphen zu identifizieren. Diese Knotenverbände, die in der Graphentheorie als Clique bezeichnet werden (siehe *Kapitel 1.5*), besitzen die Eigenschaft, dass alle Knoten innerhalb der Clique untereinander vollständig vernetzt sind. Das bedeutet für jedes Knotenpaar $u, v \in C$ gilt, dass im *Communication Graph* eine entsprechende Kante $(u, v) \in E'$ existieren muss. Der Grundgedanke dahinter ist, dass der *Communication Graph* durch das Einfügen neu entstandener Verbindungen durch die Platzierung von Regeneratoren bei einer gültigen Lösung zu einem vollständigen Graph wird, der aus einer einzigen Clique besteht. Entfernt man nun Schritt für Schritt die Regeneratoren, zerfällt die große Clique in mehrere kleine Cliques, die eine immer kleinere Kardinalität besitzen. Die ausschlaggebende Beobachtung ist daher, dass in einer noch nicht bearbeiteten RLP Instanz von vornherein unterschiedliche Cliques mit Kardinalität größer als eins bestehen. Gelingt es nun in wenigen Schritten durch die Platzierung von Regeneratoren diese Cliques zu einer einzigen großen Clique zusammenzuschließen, können sehr gute Lösungen für das RLP erzeugt werden. Dabei wird folgendermaßen vorgegangen:

1. Initialisierung aller Kandidatenknoten mit den Knoten aus V
2. Solange es noch NDC-Knotenpaare gibt:
    a) Überprüfen der Möglichkeit einer vollständigen Sofortlösung mit einem einzigen zusätzlichen Regenerator
    b) Berechnung aller Cliques für den *Communication Graph*, sodass alle Knoten in genau einer Clique vorkommen (in diesem Fall auch Cliques der Größe eins möglich)
    c) Bewertung aller Kandidatenknoten anhand der Bewertungsfunktion Φ und Auswahl des besten Knotens für die Platzierung des nächsten Regenerators
    d) Aktualisierung des *Communication Graph* M

Die Bestimmung der Cliques erfolgt wie auch schon die Bestimmung der stabilen Menge der Simplified Independent Set Heuristik direkt am *Communication Graph*. Zu diesem Zweck werden zunächst alle Knoten aus V der Menge U hinzugefügt, die die Menge aller Knoten repräsentiert, die bisher in noch keiner Clique untergebracht sind. Anschließend wird jener Knoten bestimmt, der die meisten adjazenten Nachbarn aus U besitzt und als erster Knoten für die aktuelle Clique ausgewählt. Die Menge $U' \subseteq U$ bildet alle weiteren möglichen Kandidatenknoten für diese Clique, die nur noch adjazente Knoten zum ausgewählten Clique knoten aus U enthält. Daraufhin werden weitere Cliqueknoten identifiziert, die jeweils die meisten adjazenten Knoten in U' besitzen, bis U' leer ist. Sollten zwei Knoten die gleiche Anzahl adjazenter Knoten aus U' aufweisen, so wird einer von ihnen zufällig ausgewählt. Die resultierende Clique wird den gefundenen Cliques hinzugefügt und alle ihre Knoten aus U entfernt. Dieses Vorgehen wird solange wiederholt, bis U leer ist und dadurch jeder Knoten in genau einer Clique untergebracht ist. Alternativ wäre es auch möglich den Knoten mit a) größtem Knotengrad, oder b) die zufällige Auswahl eines Knotens vorzunehmen. In den dokumentierten Resultaten in *Kapitel 4*

```
input  : Communication Graph M = (V, E')
output : Menge von Regeneratorknoten R
1  CliqueKonstruktionsHeuristik(M) {
2      R ← ∅;
3      C ← V\L;
4      while !M.isFullyConnected() do
5          if ∃ v ∈ V mit |N(v)| == |V| − 1 then
6              return R ∪ {v};
7          end
8          P ← berechne alle Cliques im Graphen;
9          v_best ← null;
10         foreach v ∈ C do
11             if Φ(v) > Φ(v_best) then
12                 v_best ← v;
13             end
14         end
15         R ← R ∪ {v_best};
16         C ← C\{v_best};
17         Aktualisiere M;
18     end
19     return R;
20 }
```

Algorithmus 3.3: Pseudocode für die Clique Konstruktionsheuristik

wird für die Clique Konstruktionsheuristik ausschließlich die Auswahlstrategie der maximalen übereinstimmenden adjazenten Knoten eingesetzt. Ein ausführlicher Pseudocode für die Clique Konstruktionsheuristik ist in *Algorithmus 3.3* gegeben.

Sind alle Knoten in genau einer Clique untergebracht, müssen die Kandidatenknoten gemäß ihrer Fähigkeit mindestens zwei Cliques möglichst vollständig zu verbinden bewertet werden. Die Bewertungsfunktion Φ berechnet dabei für benachbarte Cliques die Anzahl der direkten Nachbarn des Kandidatenknotens und stellt sie jeweils in ein Verhältnis zu der jeweiligen Cliquegröße. Dabei ist die Idee, dass Knoten mit wenigen Nachbarn, die eine eventuell kleinere Clique vollständig in die eigene Clique integrieren, besser bewertet werden, als Knoten, die zwar viele Nachbarknoten besitzen, aber eine Nachbarclique nur zu einem kleinen Prozentsatz abdecken. Die Bewertungsfunktion Φ sieht daher folgendermaßen aus, wobei $C_i,\ i = 1, \ldots, |C|$ die benachbarten Cliques, sowie $N(v)$ alle adjazenten Knoten eines Kandidatenknotens v darstellen:

$$\Phi(v) = \sum_{i=1}^{|C(v)|} \frac{|N(v) \cap C_i|}{|C_i|}. \tag{3.3}$$

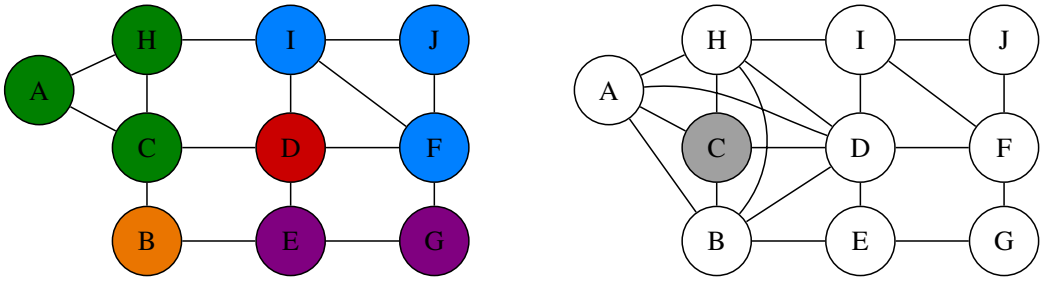


Abbildung 3.4: *Communication Graph* der Beispielinstanz für die Anwendung der Clique Konstruktionsheuristik zu Beginn der ersten und zweiten Iteration

Anschließend wird der Kandidatenknoten mit der größten Summe von Überdeckungsgraden als nächster Knoten für die Platzierung eines Regenerators ausgewählt. Sollten zwei Knoten die exakt gleiche Bewertung besitzen, so wird jener bevorzugt, der den größeren Knotengrad aufweist. Bei neuerlicher Gleichheit wird einer von ihnen zufällig ausgewählt. Ist die Lösung um den gefundenen Knoten erweitert, wird er aus den Kandidatenknoten entfernt und dem *Communication Graph* alle neu entstandenen Knotenverbindungen in Form von Kanten hinzugefügt. Die Schritte von der Bestimmung der Cliques bis zur Aktualisierung des *Communication Graphen* werden so lange wiederholt, bis der *Communication Graph* vollständig ist.

Demonstration der Funktionsweise anhand eines praktischen Beispiels

Um die Vorgangsweise der Clique Konstruktionsheuristik zu veranschaulichen, wird sie auf eine Beispielinstanz des RLP angewendet, illustriert in *Abbildung 3.4*. Zu Beginn werden die Kandidatenknoten mit der gesamten Knotenmenge V initialisiert. Da es in der ersten Iteration natürlich noch NDC-Paare im *Communication Graph* gibt, werden alle Cliques nach der beschriebenen Auswahlstrategie bestimmt. Dabei wird die Menge $U = V$ gesetzt und so lange der Knoten aus U bzw. U' mit den meisten Nachbarn ebenfalls aus U bzw. U' zu der aktuellen Clique hinzugefügt, bis alle Knoten in genau einer Clique untergebracht sind und die Menge U leer ist. In der ersten Iteration würden als Kandidaten für den ersten Clique-Knoten die Knoten C, D, F, I in Frage kommen, da sie alle vier Nachbarn aus U besitzen. Durch die Pattstellung wird der Knoten C zufällig ausgewählt. Nun wird die Menge $U' \subseteq U$ gebildet, in der nur jene Knoten aus U enthalten sind, die zu C direkt benachbart sind, also $U' = \{A, B, D, H\}$. Als nächstes wird aus U' der Knoten mit den meisten Nachbarn aus U' ausgewählt. Dafür kommt entweder der Knoten A oder H in Frage. Da es durch das Hinzufügen von A und H zu Clique keine weiteren Knoten mehr in U' gibt, ist die erste Clique $\{A, C, H\}$ fertig und es kann nach der Entfernung der drei Knoten aus U mit der Bestimmung der übrigen Cliques aus U fortgefahren werden, solange bis U leer ist. Schließlich werden für die erste Iteration die Cliques $\{A, C, H\}$, $\{F, I, J\}$, $\{E, G\}$, $\{B\}$ und $\{D\}$ gefunden. Nun erfolgt die Bewertung der Kandidatenknoten anhand der Bewertungsfunktion Φ. Der Knoten D besitzt drei benachbarte Cliques, nämlich $\{A, C, H\}$, $\{F, I, J\}$ und $\{E, H\}$. Um den Knoten D zu bewerten, wird nun die Summe seiner jeweiligen

Überdeckungsgrade anhand der direkt benachbarten Knoten gebildet. Das wäre in diesem Fall $\Phi(D) = 1/3 + 2/3 + 1/2 = 1,5$. Die übrigen Werte der Kandidatenknoten sind $\Phi(A) = 0$, $\Phi(B) = 0,83$, $\Phi(C) = 2$, $\Phi(E) = 2$, $\Phi(F) = 1,5$, $\Phi(G) = 0,33$, $\Phi(H) = 0,33$, $\Phi(I) = 1,33$ und $\Phi(J) = 0$. Da die Knoten C und E die höchste Bewertung haben, werden sie anhand ihrer Knotengrade weiter differenziert und durch $Grad(C) = 4 > Grad(E) = 3$ der Knoten C der Lösung $R = \{C\}$ hinzugefügt. Anschließend werden neu entstandene Verbindungen im *Communication Graph* in Form von zusätzlichen Kanten eingefügt und mit der zweiten Iteration fortgefahren. Erneut gibt es im Graphen keinen Knoten mit Knotengrad gleich $|V|-1$, daher werden wiederrum alle Cliques des Graphen bestimmt. Es ergeben sich $\{A, B, C, D, H\}$, $\{F, I, J\}$ und $\{E, G\}$. Beispielhaft am Knoten D demonstriert ergibt sich durch die benachbarten Cliques $\{F, I, J\}$ und $\{E, G\}$ ein Bewertungsfunktionswert von $\Phi(D) = 2/3 + 1/2 = 1,17$. Die übrigen Werte der Bewertungsfunktion sind $\Phi(A) = 0$, $\Phi(B) = 0,5$, $\Phi(E) = 0,4$, $\Phi(F) = 0,7$, $\Phi(G) = 0,33$, $\Phi(H) = 0,33$, $\Phi(I) = 0,4$ und $\Phi(J) = 0$. Dadurch ist der aussichtsreichste Knoten für einen Regenerator der Knoten D und die Lösung, sowie der *Communication Graph* werden aktualisiert. In der dritten Iteration besitzt der Knoten F nun einen Knotengrad von $Grad(F) = |V| - 1$ und kann somit direkt der Lösung $R = \{C, D, F\}$ hinzugefügt werden. Damit ist eine gültige Lösung mit drei Regeneratoren für diese Instanz des RLP gefunden, die gleichzeitig die optimale Lösung darstellt.

Laufzeitanalyse der Clique Konstruktionsheuristik

Um die Laufzeit der Clique Konstruktionsheuristik zu analysieren, muss die Berechnung der Cliques für den *Communication Graph*, die Bewertung der Kandidatenknoten und die Aktualisierung des *Communication Graphen* miteinbezogen werden. Für die Bestimmung der Cliques muss jeder Knoten in genau einer Clique untergebracht werden, was bei einem durchschnittlichen Knotengrad $N(v)$ einen Aufwand von $O(N(v) \cdot |V|)$ bedeutet. Für die Berechnung der Bewertungsfunktion $\Phi(v)$ für jeden Kandidatenknoten muss zunächst in $O(|V|)$ Schritten festgestellt werden, welche Cliques die jeweiligen benachbarten Cliques darstellen. Anschließend kann ebenfalls mit einem Aufwand von $O(|V|)$ die Bewertung jedes einzelnen Knotens durchgeführt werden. Daher ergibt sich im schlechtesten Fall ein Gesamtaufwand für die Bewertung der Kandidatenknoten von $O(|V|^2)$. Die Auswahl des besten Kandidatenknotens benötigt $O(|V|)$ Durchläufe. Die Aktualisierung des *Communication Graphen* erfolgt in $O\left(\frac{|N(v_{best})|^2}{2}\right)$ Schritten, wobei $N(v_{best})$ den Knotengrad des besten Kandidatenknotens bezeichnet. Da maximal $|V|$ Iterationen stattfinden, ergibt sich eine Gesamtlaufzeit für die Clique Konstruktionsheuristik von $O\left(|V| \cdot \left(N(v) \cdot |V| + |V|^2 + |V| + \left(\frac{|N(v_{best})|^2}{2}\right)\right)\right) = O(|V|^3)$.

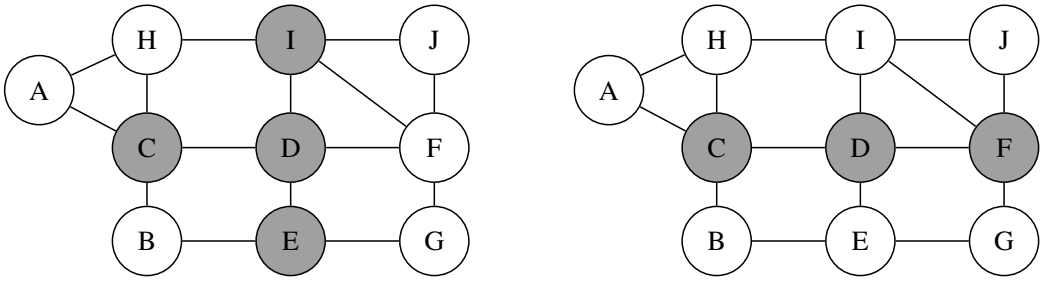


Abbildung 3.5: suboptimale und optimale Lösung für eine Instanz des RLP, bei der die lokale Suche mit 2x1 Moves keine Verbesserung findet

3.4 Destruct & Recreate Strategien für die lokale Suche

Wie bereits in *Kapitel 2.2* beschrieben, haben Chen et al. in [6] eine Basisstrategie der lokalen Suche vorgestellt. Dabei werden sogenannte 2x1 Moves durchgeführt, bei denen jeweils zwei Regeneratorknoten gegen einen einzigen ausgetauscht werden, ohne die Gültigkeit der Lösung zu verlieren. Problematisch dabei ist, dass die Lösung nur in jenen Fällen verbessert werden kann, wo der Graph eine Verbesserung durch den Austausch von lediglich zwei Knoten zulässt. Betrachtet man das Beispiel in *Abbildung 3.5*, so wurde auf der linken Seite für die Beispielinstanz eine suboptimale, aber gültige Ausgangslösung mit vier Knoten generiert. Auf der rechten Seite ist die tatsächlich optimale Lösung mit drei Knoten dargestellt. Wendet man nun die lokale Suche mit 2x1 Moves auf die Ausgangslösung an, werden jeweils zwei Regeneratorknoten hinsichtlich der Möglichkeit sie durch einen einzigen zu ersetzen betrachtet. Da drei der vier Regeneratoren, nämlich die Knoten H, I, E nicht Teil der optimalen Lösung sind, sollten sie nun schrittweise entfernt werden. Entfernt man beispielsweise H und I, so müssten im klassischen 2x1 Move die Knoten A, H und J mit genau einem neuen Regenerator verbunden werden. Leider ist dies nicht möglich, da durch die Anordnung der Kanten immer mindestens ein Knoten keine Verbindung zu einem Regenerator hat und dadurch das Theorem aus [6] nicht erfüllt ist, wodurch es sich um keine gültige Lösung handelt. Das gleiche Problem tritt auf, wenn man versucht die übrigen Kombinationen der Knoten H, I, E zu entfernen. Genauso ist es nicht möglich spezielle 2x1 Moves, bei denen beide Regeneratoren zunächst entfernt werden, jedoch anschließend der neue Regenerator wieder in einem der beiden platziert wird, mit dem Regeneratorknoten D durchzuführen. Solche 1x0 Moves funktionieren in dieser Instanz mit der vorliegenden Lösung nicht, da für alle Regeneratorknoten genau ein Knoten in ihrer Nachbarschaft existiert, der keine direkte Verbindung zu einem anderen Regeneratorknoten besitzt und wiederrum ihre Entfernung aus der Lösung das Theorem aus [6] für eine gültige Lösung verletzten würde. Aus diesen Gründen ist es nicht möglich diese Ausgangslösung unter Zuhilfenahme der lokalen Suche mit 2x1 Moves zu verbessern, obwohl die optimale Lösung weniger Knoten beinhaltet.

Um diesen Schwachpunkt der bestehenden lokalen Suche zu verbessern, wurden Destruct & Recreate Strategien für die lokale Suche implementiert. Dabei wird im Gegensatz zum Entfernen von lediglich zwei Regeneratorknoten ein größerer Teil der Lösung temporär entfernt.

Anschließend wird mit Clique und Independent Set Techniken, die bereits bei den Konstruktionsheuristiken in den *Kapiteln 3.1–3.3* verwendet wurden, versucht eine gültige Lösung für das Problem zu erzeugen. Gelingt dies mit weniger Regeneratoren, als in der Ausgangslösung, so wird im Zuge der lokalen Suche die neue Lösung als bisher beste Lösung für die Instanz des RLP und damit als Ausgangspunkt für die nächste Iteration weiter verwendet.

Insgesamt wurden drei Nachbarschaftsstrukturen für die lokale Suche implementiert, nämlich die IndependentSet Nachbarschaft, die Simplified Independent Set Nachbarschaft und die Clique Nachbarschaft. Dabei sind alle drei im Grunde gleich aufgebaut und unterscheiden sich hauptsächlich in der Art und Weise, wie sie die reduzierte Lösung wieder zu einer gültigen Lösung rekonstruieren. Die grobe Vorgangsweise sieht folgendermaßen aus:

1. Die Gültige Ausgangslösung wird durch ein bestimmtes Verfahren (z.B. Clustersuche, Zufall,...) teilweise „zerstört“

2. Der *Communication Graph*, sowie die möglichen Kandidatenknoten werden für die reduzierte Lösung entsprechend angepasst

3. Solange NDC-Knotenpaare existieren, also der *Communication Graph* nicht vollständig verbunden ist:

   a) Aus der Menge der Kandidatenknoten wird jener bestimmt, der den besten nächsten Regeneratorplatz unter Verwendung einer speziellen Heuristik (z.B. Independent Set, Clique,...) darstellt

   b) Der ausgewählte Knoten wird zur reduzierten Lösung hinzugefügt und der *Communication Graph*, sowie die Menge der Kandidatenknoten aktualisiert

4. Falls die neue Lösung weniger Knoten, als die Ausgangslösung besitzt, wird die gültige Lösung als neue beste Lösung für die Instanz verwendet

Reduktion der Ausgangslösung

Bei der teilweisen „Zerstörung“ der Ausgangslösung ist das Ziel suboptimale Bereiche der Lösung temporär zu entfernen und durch die Identifikation besserer Regeneratorknoten wieder aufzubauen. Einfachstes Verfahren ist eine Zufallauswahl, bei der für jeden Knoten der Lösung entschieden wird, ob er in der reduzierten Löung erhalten bleibt, oder temporär entfernt wird. Daher gibt es keine Garantie dafür, dass überhaupt Teile der Lösung rekonstruiert werden, bei denen noch eine Verbesserung möglich ist. Auf der anderen Seite kann zufällig eine Regeneratorknotenkombination aus der Lösung entfernt werden, die andere Verfahren durch ihre Bewertungsfunktion nicht auswählen würden. Dadurch ist es möglich lokalen Optima im Zuge der lokalen Suche zu entkommen.

Ein weiteres Verfahren als Umsetzung des Destruction Prinzips ist die Eliminierung von Regeneratorknoten, deren Nachbarn gleichzeitig auch von anderen Regeneratoren versorgt werden. Zu diesem Zweck wird für jeden Knoten aus der Ausgangslösung eine Bewertung erstellt, bei dem er einen positiven Punkt für jeden Nachbarn, für den er der einzige Regenerator in seiner

Umgebung ist bekommt und einen negativen für jeden Nachbarn, der auch andere Regeneratoren in seiner Nachbarschaft besitzt. Die Bewertungsfunktion $\Phi(v) \; \forall \; v \in R$ setzt sich aus

$$\Phi(v) = \frac{v_{positiv}}{v_{negativ}} \tag{3.4}$$

zusammen. Anschließend werden die Bewertungen der Regeneratorknoten aufsteigend sortiert und die jeweils beste und schlechteste Bewertung ermittelt. Für die schlussendliche Bestimmung der reduzierten Lösung wird eine gewichtete Zufallsauswahl vorgenommen, die abhängig von der Bewertung eines Knotens mit einer gewichteten Wahrscheinlichkeit das Entfernen des Knotens aus der Lösung beschließt. Dabei wird für jeden Knoten $v \in R$ abhängig von seinem Wert $w_v = \Phi(v)$, den minimalen und maximalen Werten w_{min} und w_{max} und dem Zufallswert $r \in [0, 1)$ die Entscheidung

$$\Psi(v) = \begin{cases} true & \text{wenn } r < w_v/(w_{min} + w_{max}) \\ false & \text{sonst} \end{cases} \tag{3.5}$$

über die Aufnahme in die reduzierte Lösung getroffen.

Durch diesen Mechanismus werden Regeneratorknoten, die viele Knoten in ihrer Nachbarschaft besitzen, die auch von anderen Regeneratoren aus der Lösung versorgt werden mit hoher Wahrscheinlichkeit temporär entfernt, sowie solche, die viele Knoten als einzigen Regenerator benachbarn ziemlich sicher erhalten. Dadurch wird ermöglicht Teile des Graphen, die vorher mit vielen Regeneratoren ausgestattet waren, in der Rekonstruktionsphase möglicherweise besser zu versorgen.

Da die Knotenmenge der optimalen Lösung eines RLP immer einen Spannbaum im Graphen repräsentiert, ist die Identifikation von suboptimalen Regeneratorästen eine weitere Möglichkeit Teile der Lösung zu entfernen. Dabei wird im Gegensatz zur zuvor beschriebenen Clustersuche, bei der viele Regeneratoren innnerhalb einer Region entfernt werden, vor allem auf die Entfernung von umständlichen Ästen im Regeneratorspannbaum Wert gelegt. Dies hat den Hintergrund, dass zentrale Regeneratoren, von denen oft mehrere Äste in Form von Regeneratorpfaden ausgehen, bereits gut platziert sind, aber die Verbindungen zwischen den Drehscheiben manchmal suboptimal sind. Zu diesem Zweck wird für jeden Lösungsknoten die Anzahl der Regeneratoren in seiner direkten Nachbarschaft bestimmt. Durch die Beschaffenheit des Spannbaums gibt es meistens zumindest zwei Lösungsknoten mit jeweils einem und unbestimmt viele, die jeweils zwei oder mehrere Regeneratorknoten als direkte Nachbarn besitzen. Nun werden die Regeneratoren hinsichtlich ihrer benachbarten Regeneratorknoten durchsucht und alle entfernt, die zwei oder weniger besitzen. Sollte es keine solche Knoten geben und daher ein ganzer Regeneratorcluster vorliegen, wird die gerundete Hälfte der Regeneratoren, beginnend bei jenen mit den wenigsten benachbarten Lösungsknoten entfernt. Schlussendlich bleiben die zuvor zentral platzierten Regeneratoren über und die Lösung kann anhand dieser markanten Punkte neu rekonstruiert werden.

input : *Communication Graph* $M = (V, E')$, Ausgangslösung R
output : Verbesserte Lösung R'

```
1  IndependentSetNachbarschaft(M, R) {
2      R' ← ∅;
       // Destruction durch Cluster-Rating Strategie
3      foreach v ∈ R do
4          if Ψ(v) = true then
5              R' ← R' ∪ {v};
6          end
7      end
8      C ← V\R';
       // Setup Communication Graph M
9      foreach v ∈ R' do
10         foreach (i, j) ∈ N(R') × N(R') do
11             M.E' ← M.E' ∪ {(i, j)};
12         end
13     end
       // Rekonstruktion der Lösung
14     while !M.isFullyConnected() do
           // Siehe Code in Algorithmus 3.1
15     end
16     return R';
17 }
```

Algorithmus 3.4: Pseudocode für die Independent Set Nachbarschaft

Demonstration der Funktionsweise anhand eines praktischen Beispiels

Illustrierend für alle drei entwickelten Nachbarschaftsstrukturen für die lokale Suche ist in *Algorithmus 3.4* die Funktionsweise der IndependentSet Nachbarschaft anhand eines Pseudocodes abgebildet. Dabei wird die zuvor beschriebene Entscheidungsfunktion des Clusterprinzips $\Psi(v)$ $\forall v \in R$ als Zerstörungsmechanismus für die Ausgangslösung benutzt. In *Abbildung 3.6* wird beispielhaft eine Iteration der lokalen Suche exemplarisch auf eine bestehende Ausgangslösung angewendet, um die Vorgehensweise zu demonstrieren.

Als erstes wird die neue Lösung R' als leere Menge initialisiert und mit der Bewertung der Regeneratorknoten der Ausgangslösung begonnen. Zu diesem Zweck müssen die Funktionswerte $\Phi(v)$ und in weiterer Folge $\Psi(v)$ für jeden Knoten aus R berechnet werden. Beispielhaft am Regeneratorknoten C demonstriert, würde sich der Wert der Funktion $\Phi(C)$ aus einem positiven Bewertungspunkt durch die alleinige Versorgung des Knotens A, sowie drei negativen Punkten durch die anderen Regeneratoren in der Nachbarschaft bei den Knoten B, D, H zusammensetzen. Dadurch ergibt sich für $\Phi(C)$ der Wert $1/3 = 0,3$. Die übrigen Regeneratorknoten besitzen die Werte $\Phi(D) = 3$, $\Phi(E) = 0,5$ und $\Phi(I) = 0,3$. Dabei ist die kleinste Bewertung $w_{min} = 0,3$ und die größte Bewertung $w_{max} = 3$. Nun wird durch die Evaluierung der Funktion

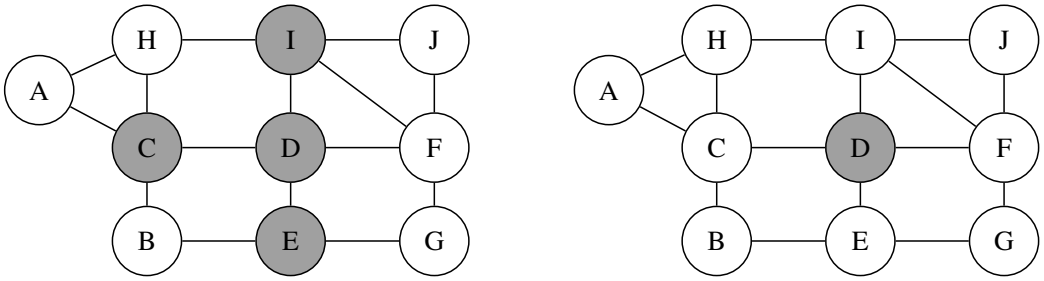


Abbildung 3.6: Demonstration einer Iteration der IndependentSet Nachbarschaft mit Entscheidungsfunktion $\Psi(v)$ für jeden Knoten $v \in R$

$\Psi(v)$ für jeden Knoten bestimmt, ob er in der reduzierten Lösung erhalten bleibt, oder temporär entfernt wird. Für den Knoten C ergibt sich die Frage, ist ein Zufallswert $r \in [0, 1)$ kleiner als $\Phi(C)/(w_{min}+w_{max})$. Da der Wert von $\Phi(C)$ ohnehin bereits der kleinste war und die Differenz von $\Phi(D)$ zu $\Phi(C)$ so enorm ist, ist es nach der Normierung sehr unwahrscheinlich, dass C in der Lösung erhalten bleibt, da $r < 0,099$ sein muss. Die übrigen normierten Werte innerhalb der Entscheidungsfunktion $\Psi(v)$ sind für den Knoten $D = 0,899$, für den Knoten $E = 0,15$ und für den Knoten $I = 0,099$. Als Zufallswerte r werden beispielsweise für den Knoten $C = 0,477$, $D = 0,856$, $E = 0,498$ und $I = 0,521$ gezogen. Daher verbleibt lediglich der Knoten D in der reduzierten Lösung $R' = \{D\}$. Nun wird der *Communication Graph* unter der Berücksichtigung von R' erstellt und die zusätzlichen Kanten (C, I), (C, E), (C, F), (E, F) und (E, I) eingefügt. Anschließend erfolgt die Rekonstruktion der Lösung durch die Anwendung des Independent Set Konstruktionsprinzips aus *Kapitel 3.1*. Durch die Fixierung des Knotens D wird dort leicht die optimale Lösung $R' = \{C, D, F\}$ gefunden und als neue beste Lösung im Prozess der lokalen Verbesserung verwendet.

Analyse der Laufzeit

Die Laufzeit der Destruction Nachbarschaftsstrukturen gliedert sich in zwei Teile. Diese sind zum Einen die Zerstörung der Ausgangslösung und zum Anderen ihre neuerliche Rekonstruktion. Dabei werden pro Iteration für jeden Regeneratorknoten in der Zerstörungsphase je nach Auswahlstrategie für Random $O(|R|)$, für die Bestimmung von Ästen $O(|R| + |R|)$ und für die Entscheidung durch die Funktionen $\Phi(v)$ und $\Psi(v)$ $O(|R| \cdot |V|)$ Schritte benötigt. Daher ergibt sich für die Zerstörungsphase ein worst-case Aufwand von $O(|R| \cdot |V|)$. Die Rekonstruktionsphase kann durch die Verwendung der zuvor in *Kapitel 3.1–3.3* beschriebenen Konstruktionsstrategien im Fall der Independent Set und Clique Heuristik in $O(|V|^3)$ erledigt werden, sowie bei der Simplified Independent Set Heuristik in $O(|V|^2)$. Daher ergibt sich ein Gesamtaufwand für die Nachbarschaftsstrukturen der lokalen Suche von $O(|R| \cdot |V| + |V|^3) = O(|V|^3)$ bzw. $O(|R| \cdot |V| + |V|^2) = O(|V|^2)$.

Kürzel + Art	Konstruktionsheuristik	Nachbarschaftsstruktur(en)
CG-GRASP	CG	2x1 Move
IS-GRASP	Independent Set	Independent Set
SIS-GRASP	Adv. Independent Set	Adv. Independent Set
CL-GRASP	Clique	Clique
Hyb-GRASP	Independent Set	Clique
VNS1	Independent Set	lokale Suche: 2x1 Move, Shaking: Clique – Ind.Set – Adv.Ind.Set
VNS2	Adv. Independent Set	lokale Suche: 2x1 Move, Shaking: Adv.Ind.Set – Ind.Set – Clique
RVNS	Clique	RandomShuffle(Ind.Set, Clique, Adv.Ind.Set)

Tabelle 3.1: Übersicht aller implementierten Metaheuristiken

3.5 Verwendete Metaheuristiken

Da die vorgestellten Konstruktionsheuristiken aus *Kapitel 3.1–3.3* alle Randomisierung beinhalten, eignen sie sich sehr gut für den Einsatz innerhalb einer Metaheuristik. Daher wurden für alle drei entwickelten Konstruktionsheuristiken jeweils zunächst ein GRASP (siehe *Kapitel 1.6*) formuliert, wobei die Konstruktionsheuristik mit der entsprechenden namensgleichen Nachbarschaftsstruktur verknüpft wurde, z.B. Clique Konstruktionsheuristik + Clique Nachbarschaftsstruktur. Durch die komplementäre Vorgehensweise der Clique und Independent Set Heuristiken, die sich durch ihren gegenseitigen Zusammenhang Auszeichnen, hat sich die zusätzliche Implementierung eines hybriden GRASP-Ansatzes angeboten. Dieser verbindet eine Clique Konstruktionsheuristik und eine Independent Set Nachbarschaftsstruktur. Insgesamt wurden daher vier GRASP Heuristiken implementiert, die jeweils durch den Parameter GRASP-Iterationen und Nachbarschafts-Iterationen beeinflusst werden können. Die GRASP-Iterationen bestimmen wie oft eine Lösung konstruiert werden soll, die Nachbarschafts-Iterationen wie oft eine Nachbarschaftsstruktur auf eine Lösung im Zuge der lokalen Suche angewendet werden soll, wenn keine Lösungsverbesserung stattfindet.

Darüber hinaus wurden auch VNS-Heuristiken (siehe *Kapitel 1.6*) zusammengestellt und auf die einzelnen Instanzen angewendet. Dabei wurden zwei klassische Basisansätze implementiert, die beide die 2x1 Nachbarschaftsstruktur zur lokalen Suche einsetzen und jeweils alle drei entwickelten „Shaking“-Nachbarschaften in unterschiedlicher Reihenfolge beinhalten. In der ersten Variante sind sie nach dem Zuverlässigkeitsgrad der verwendeten Strategie basierend aus dem Vergleich der Konstruktionsheuristiken im ersten Experiment (siehe *Kapitel 4.3*) in der Reihenfolge „Clique - Independent Set - Simplified Independent Set“ Nachbarschaftsstruktur geordnet. Bei der zweiten Ordnung wird nach dem Kriterium Laufzeit sortiert, was der Reihenfolge „Simplified Independent Set - Independent Set - Clique“ Nachbarschaftsstruktur entspricht. Abschließend wurde auch eine Variante der RVNS mit einer Clique Konstruktionsheuristik und allen drei entwickelten „Shaking“-Nachbarschaftsstrukturen implementiert, deren Reihenfolge nach jeder kompletten Anwendung aller Nachbarschaftsstrukturen ohne Verbesserung zufällig umgeordnet wird. Eine Übersicht über alle implementierten Ansätze ist in *Tabelle 3.1* zu finden.

KAPITEL 4

Ergebnisse

Die in Kapitel *Kapiel 3.1–Kapitel 3.4* beschriebenen Konstruktions- und Metaheuristiken, sowie Mechanismen der lokalen Suche wurden zur Durchführung von Vergleichstest in Java 1.7.0_04 als in einer Java Virtual Machine ausführbares Programm implementiert. Sämtliche darauf aufbauende Testergebnisse, die in diesem Kapitel vorgestellt werden, wurden auf dem Cluster des Arbeitsbereichs für Algorithmen und Datenstrukturen an der TU Wien durchgeführt, der aus Intel Core 2 Quad CPUs mit 2,83 GHz besteht.

4.1 Probleminstanzen und Testsätze

Wie in [6] beschrieben, gibt es zunächst drei verschiedene Arten von RLP Instanzen. Diese sind *Randomly Generated Networks* (oder kurz *Random Networks*), *Networks with Random Distances* und *Euclidean Networks*. Der Unterschied zwischen den verschiedenen Arten besteht dabei jeweils in der unterschiedlichen Generierung der Instanz. Beispielsweise bei den *Randomly Generated Networks* wird im Erstellungsprozess direkt der *Communication Graph* aufgesetzt und abhängig von einem Parameter p, der den Prozentsatz an NDC-Knotenpaaren angibt, die Anzahl an bestehenden Kanten im *Communication Graph* zufällig zugewiesen. Die *Networks with Random Distances* werden neben der Anzahl an Knoten anhand von drei weiteren Parametern q, a, b generiert, die jeweils die Anzahl der NDC-Knotenpaare beeinflussen. Dabei stellt q den Prozentsatz von Kanten dar, die größer als $d_{\max}$ sind und $a, b \in (0, 1]$ das Auswahlintervall $[a \cdot d_{\max}, b \cdot d_{\max}]$ mit $0 < a \leq b \leq 1$ für die Längen der übrigen Kanten. Genauso wie bei den *Euclidean Networks* ist ein zuvor aufgestellter minimaler Spannbaum die Voraussetzung für die Sicherstellung der Lösbarkeit der RLP-Instanz. Bei den *Euclidean Networks* werden Knoten zufällig auf einem 100×100 Quadrat angeordnet und anschließend durch den Parameter $d_{\max}$ und die euklidischen Distanzen zwischen den Knoten die Anzahl an NDC-Knotenpaaren bestimmt.

In *Abbildung 4.1* sind zwei Beispiele von *Random Networks* Instanzen mit einer Instanzparameterkonfiguration von $n = 60; p = 0,8$ und $n = 40; p = 0,1$ dargestellt. Sie sind ein guter Anhaltspunkt, um einen Eindruck von der Beschaffenheit der verschiedenen Instanzgraphen zu

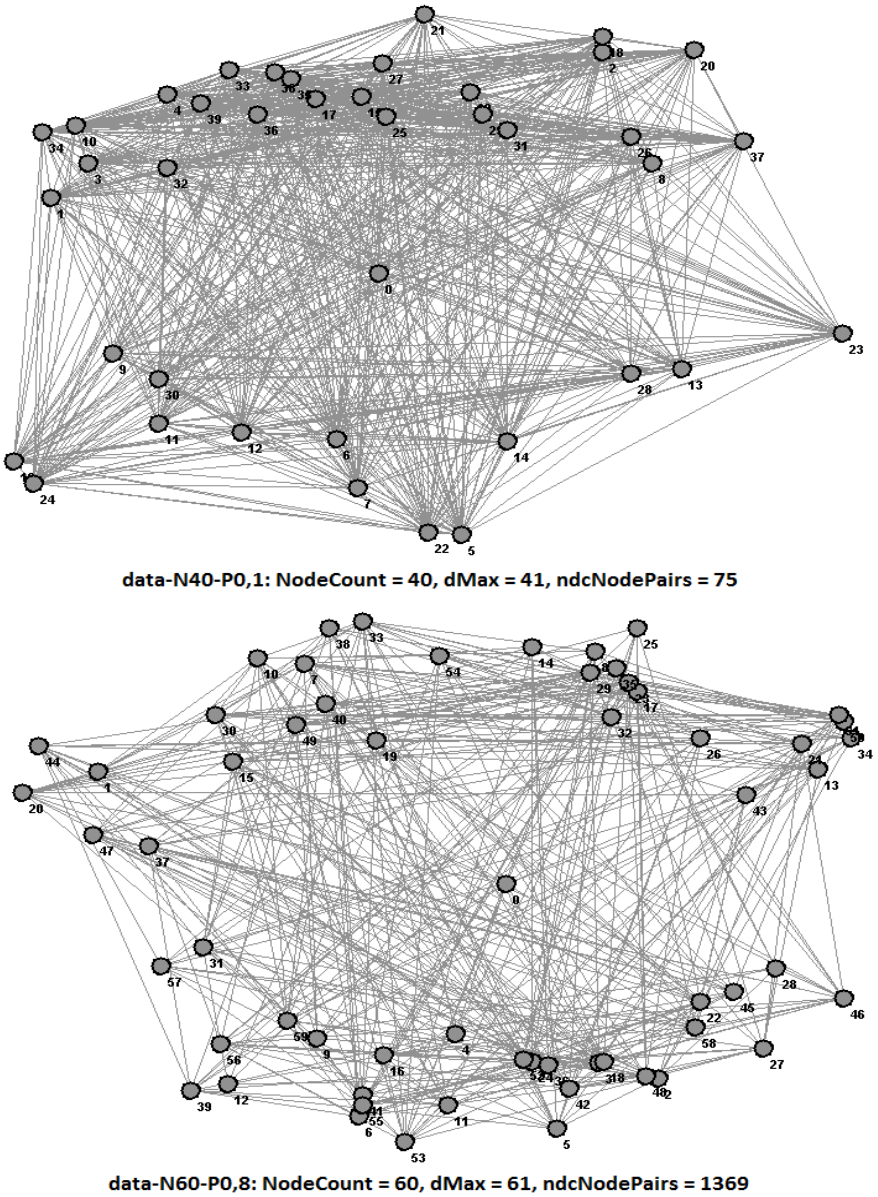
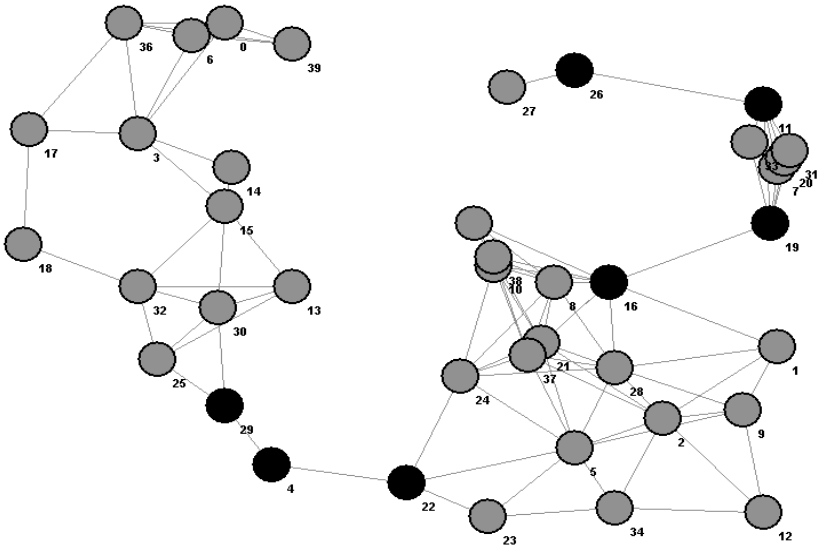


Abbildung 4.1: Beispiele für *Random Networks* Instanzen des RLP

bekommen. In *Abbildung 4.2* werden zum Vergleich zwei Instanzen von *Euclidean Networks* mit einer Parameterkonfiguration von $n = 40; d_{\max} = 30$ und $n = 80; d_{\max} = 50$ präsentiert. Es ist sehr deutlich zu sehen, dass *Random Networks* mit großem p-Wert eher dünne Graphen darstellen und diese auch dementsprechend viele NDC-Knotenpaare aufweisen. Auf der anderen Seite sind Instanzen mit kleinem p-Wert als dichte Graphen anzusehen. *Euclidean Networks* stellen selbst bei sehr großem $d_{\max}$ zwar dichte Graphen dar, die allerdings bei weitem nicht so



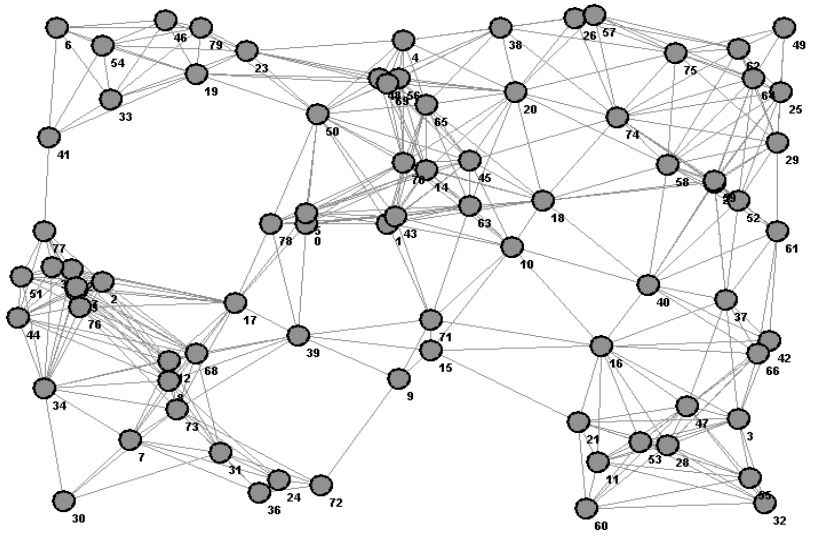


Abbildung 4.2: Beispiele für *Euclidean Networks* Instanzen des RLP

viele Kanten im Vergleich zur Anzahl der Knoten enthalten, wie die *Random Networks* Instanzen. Ist der Parameter $d_{\max}$ klein, so handelt es sich um dünne Graphen, deren Kanten meist nur einen minimalen Spannbaum mit ein paar vereinzelten zusätzlichen Verbindungen formen. Daher stellen *Euclidean Networks* Instanzen grundsätzlich eher dünne Graphen dar, die spannbaumartig zusammengesetzt sind und Instanzen der *Random Networks* eher dichte Graphen dar, die wolkenartig zusammengesetzt sind.

Im Zuge der Evaluierung der entwickelten Heuristiken wurden insgesamt vier verschiedene Mengen von Testinstanzen angelegt. Zwei der vier Testsätze wurden von den Autoren von [6] zur Verfügung gestellt und beinhalten *Random Networks* Instanzen mit insgesamt fünf verschiedenen Werten für $p \in \{10\%, 30\%, 50\%, 70\%, 90\%\}$, sowie Anzahl an Knoten $n \in \{40, 60, 80, 100, 200, 300, 400, 500\}$. Die beiden anderen Testsätze enthalten *Euclidean Networks* Instanzen mit $d_{\max}$–Werten von $d_{\max} \in \{20, 30, 40, 50\}$ mit der jeweiligen Anzahl an Knoten von $n \in \{40, 60, 80, 100, 200, 300, 400, 500\}$. Im Speziellen enthält RN1 die übermittelten Testinstanzen mit $n \in \{40, 60, 80, 100\}$ Knoten und RN2 die größeren Instanzen mit $n \in \{200, 300, 400, 500\}$ Knoten. In EN1 sind die selbst generierten *Euclidean Networks* mit $n \in \{40, 60, 80, 100\}$ untergebracht, sowie in EN2 jene mit $n \in \{200, 300, 400, 500\}$. Pro Instanzkonfiguration (Bsp: *Random Networks*, $p = 30\%$, $n = 40$) gibt es in jedem Testsatz 10 verschiedene Instanzen dieser Konfiguration. Das bedeutet der Umfang der Testsätze beträgt für RN1 200, EN1 160, RN2 200 und EN2 160 Instanzen.

4.2 Vorgehensweise bei der Evaluierung

Um die entwickelten Konstruktions- und Metaheuristiken sowohl untereinander, als auch mit den in der Literatur [10], [6] beschriebenen Heuristiken zu vergleichen, werden anhand der Testsätze primär die Merkmale **„Qualität der Endlösung“**, und **„Laufzeit“** evaluiert. Zu diesem Zweck wurden zwei verschiedene Experimente durchgeführt. Die „Qualität der Endlösung“ entspricht dabei der Anzahl an benötigten Regeneratoren aus denen sich eine gültige Lösung zusammensetzt. Dabei werden als Vergleichswerte best-known Lösungen aus zahlreichen zuvor durchgeführten Testläufen von unterschiedlichen Heuristiken herangezogen, um so insgesamt eine möglichst fixe Vergleichsgröße zu schaffen. Unter „Laufzeit“ wird die gesamte CPU-Zeit verstanden, die eine Heuristik benötigt um eine gültige Lösung ausgehend vom *Communication Graph* des RLP zu berechnen. Die Erstellungsdauer des *Communication Graphen* wird dabei nicht berücksichtigt.

Im ersten Experiment wird für alle Heuristiken eine maximale Anzahl an Iterationen vorgegeben. Dabei führen Konstruktionsheuristiken beispielsweise 100 Konstruktionsschritte für eine Instanz durch und vergleichen anschließend untereinander die Abweichungen von den best-known Lösungen der Instanzen. Bei den GRASP Ansätzen bestimmt die vorgeschriebene Anzahl an Iterationen wie oft eine Lösung konstruiert und anschließend durch die lokale Suche verbessert wird. Bei den VNS und RVNS Metaheuristiken legt die Anzahl an Iterationen fest wie oft das Shaking mit anschließender lokalen Suche bzw. nur das Shaking durchgeführt wird. Durch die unterschiedlich langen Laufzeiten der verschiedenen Metaheuristiken und dem gleichzeitig hohen Grad an Randomisierung ist es im Zuge des Experiments nicht möglich eine stichhaltige Aussage über die Gesamtgüte von zwei Metaheuristiken zu treffen. Beispielsweise könnte eine VNS, die bei gleicher Anzahl an Iterationen wie ein GRASP Ansatz durch die oftmalige Anwendung von Nachbarschaftsstrukturen möglicherweise eine längere Laufzeit aufweist, bessere Ergebnisse für viele Instanzen liefern, aber im Allgemeinen bei gleicher Laufzeit schlechter sein. Daher gibt das erste Experiment in erster Linie Auskunft über die unterschiedlichen Laufzeiten der einzelnen Heuristiken und über die Leistungsfähigkeit der unterschiedlichen Konstruktionsheuristiken, die die Ausgangslösungen für die Metaheuristiken bereitstellen.

Beim zweiten Experiment werden ausschließlich Metaheuristiken untereinander verglichen und jeweils die maximale Laufzeit pro Instanz vorgegeben. Durch diese Maßnahme haben beispielsweise GRASP Heuristiken, die nur eine Nachbarschaftsstruktur im Zuge der lokalen Suche einsetzen, die Möglichkeit durch die geringere Laufzeit pro Iteration den Suchraum öfters zu durchsuchen. Genauso wird dadurch auch für die VNS Heuristiken ein einheitlicher Rahmen geschaffen, wie oft sie die Anwendung des Shakings und der lokalen Suche wiederholen dürfen. Die vorgegebene Maximallaufzeit pro Instanz schafft daher die Grundlage für einen fairen Vergleich zwischen den verschiedenen Metaheuristiken.

Im Zuge der beiden Experimente werden alle zu evaluierenden Heuristiken auf alle Instanzen innerhalb der vier definierten Testsätze angewendet. Dabei werden im Zuge des ersten Experiments pro Instanzkonfiguration folgende Werte bestimmt:

- die Anzahl der durchschnittlich benötigten Regeneratoren $\oslash$-*Reg*,
- die Anzahl $\#$-*Best* wie oft die Lösungsqualität der konstruierten Lösung der best-known Lösung der Instanz entsprach,
- die durchschnittlich benötigte CPU-Laufzeit zum Auffinden der best-known Lösung pro Instanz $\oslash$-t_{Best} [ms] und
- die durchschnittliche zur Berechnung benötigte CPU-Laufzeit pro Instanz $\oslash$-*LZ* [s].

Da das zweite Experiment eine unterschiedliche Beschaffenheit im Gegensatz zum ersten Experiment aufweist, werden im Zuge dessen teilweise andere Werte pro Instanzkonfiguration bestimmt. Diese sind:

- die durchschnittlich ausgeführten Iterationen $\oslash$-*Iter* pro Instanz,
- die Anzahl der durchschnittlich benötigten Regeneratoren $\oslash$-*Reg*,
- die Anzahl $\#$-*Best* wie oft die Lösungsqualität der konstruierten Lösung der best-known Lösung der Instanz entsprach,
- die durchschnittlich benötigte CPU-Zeit, bis die best-known Lösung einer Instanz gefunden wurde $\oslash$-t_{Best} [ms] und
- die zuvor festgelegte CPU-Laufzeit für alle Instanzen einer Konfiguration Σ-*LZ* [ms].

Um die statistische Signifikanz der Ergebnisse zu überprüfen werden die Gesamtergebnisse jedes Experiments einem Wilcoxon-Vorzeichen-Rang-Test (siehe [20]) unterzogen. Bei diesem Test wird keine Annahme über die Verteilung der zugrundeliegenden Population benötigt und die Hypothese, dass ein Verfahren im Mittel kleinere Ergebniswerte liefert als ein anderes überprüft. Dafür werden die Ergebnisse von jeweils zwei Heuristiken als gepaarte Stichproben pro Instanzkonfiguration gegenübergestellt. Die notwendigen Wilcoxon Tests wurden mit Hilfe der Programmiersprache R (siehe [23]) in Form eines Scriptaufrufs durchgeführt. Der Aufruf der Wilcoxon Standardimplementierung in R wird im konkreten Szenario durch den Befehl

$$wilcox.test(X_1, X_2, \text{alternative = “one.sided”, paired = true, conf.level = 0.95}) \tag{4.1}$$

	RN1			EN1		
Heuristik	⌀-Reg	#-Best	⌀-LZ	⌀-Reg	#-Best	⌀-LZ
CG	**4,96**	**129.349 (64,7%)**	2.015,15s	12,89	55.551 (34,7%)	2.712,94s
IS	5,07	119.339 (59,7%)	181,15s	**12,71**	**63.842 (39,9%)**	420,56s
SIS	5,38	95.967 (48,0%)	**74,31s**	14,75	21.340 (13,3%)	**205,73s**
CL	5,00	121.966 (61,0%)	506,21s	12,81	57.633 (36,0%)	498,99s
Iterationen		200.000 (100%)			160.000 (100%)	

	RN2			EN2		
Heuristik	⌀-Reg	#-Best	⌀-LZ	⌀-Reg	#-Best	⌀-LZ
CG	**7,48**	**13.904 (67,3%)**	48.787,78s	14,45	98 (0,5%)	40.434,34s
IS	7,79	12.330 (59,7%)	1.128,81s	**12,95**	**1.858 (11,1%)**	1.551,14s
SIS	8,48	9.644 (46,7%)	**312,01s**	15,45	43 (0,2%)	**489,30s**
CL	7,53	13.775 (66,7%)	9.550,29s	13,53	333 (2,0%)	3.447,55s
Iterationen		20.650 (100%)			16.720 (100%)	

Tabelle 4.1: Zusammenfassung der Ergebnisse für die Konstruktionsheuristiken auf den Testsätzen RN1, EN1, RN2 und EN2

erreicht. Dabei sind X_1 und X_2 die zu vergleichenden Ergebnisse, der Parameter „alternative“ wird als einseitiger Test eingestellt, die Stichproben paarweise betrachtet, sowie das Konfidenzniveau auf 95% gesetzt.

4.3 Erstes Experiment: Max-Iterationen Evaluierung

Vergleich der Konstruktionsheuristiken

Im Zuge des ersten Experiments berechnen die Konstruktionsheuristiken in den kleineren Testsätzen RN1 und EN1 jeweils 100 Lösungen pro Instanz, sowie in den größeren Testsätzen aufgrund der längeren Laufzeit jeweils 20 Lösungen pro Instanz. Die Zusammenfassung der gefundenen Ergebnisse ist in *Tabelle 4.1* dargestellt. Eine genaue Auflistung der Ergebnisse im Detail ist im Anhang in *Tabelle A.1* bis *Tabelle A.4* zu finden.

Die Ergebnisse der Konstruktionsheuristiken zeigen, dass die in der Literatur [10] publizierte CG Konstruktionsheuristik vor allem bei den *Random Networks* gute Ergebnisse liefert. Im Testsatz RN1 stellt sie mit 64,7% erreichten best-known Lösungen und einer durchschnittlichen Anzahl an Regeneratoren von 4,96 den Spitzenreiter dar. Ebenso sind die erreichten 67,3% aller best-known Lösungen in RN2 ein Indiz dafür, dass diese Heuristik bei dichten Graphen gute Ergebnisse liefert. Bei dünneren Graphen in den Testsätzen EN1 und EN2 wurden weitaus schlechtere Ergebnisse erzielt, nämlich nur 34,7% bzw. 0,5% der best-known Lösungen erreicht. Auffällig war, dass die durchschnittliche Laufzeit pro Instanz in allen Testsätzen weitaus länger als jene der entwickelten Clique und Independent Set Heuristiken war.

Die Independent Set Heuristik (IS) erzielte auf den Testsätzen RN1 und RN2 mit jeweils 59,7% gefundenen best-known Lösungen im Vergleich zu den anderen Heuristiken eine Platzie-

rung im Mittelfeld, jedoch auf den Testsätzen EN1 und EN2 mit 39,9% und 11,1% die besten Ergebnisse. Dies legt die Vermutung nahe, dass die Independet Set Konstruktionsheuristik in der Lage ist für dünne Graphen im Vergleich zu den übrigen Heuristiken qualitativ bessere Lösungen zu konstruieren. In punkto durchschnittlicher Laufzeit pro Iteration belegte sie auf allen Testsätzen den zweiten von vier Plätzen und besitzt damit im Allgemeinen ein gutes Verhältnis zwischen Laufzeit und Lösungsqualität.

Die Clique Konstruktionsheuristik (CL) konnte vor allem auf den dichteren Graphen der *Random Networks* Instanzen in den Testsätzen RN1 und RN2 mit 61,0% und 66,7% der errechneten best-known Lösungen überzeugen. Auf den Testsätzen EN1 und EN2 belegte sie mit 36,0% und 2,0% erreichten best-known Lösungen jeweils den zweiten Platz unter den vier evaluierten Heuristiken. Im Vergleich zu den Independent Set Heuristiken wies die Clique Konstruktionsheuristik praktisch immer eine längere durchschnittliche Laufzeit pro Instanz auf.

Die Simplified Independent Set Konstruktionsheuristik (SIS) schneidet bezogen auf die Qualität der erzielten Lösungen mit 48,0%, 13,3%, 46,7% und 0,2% der erreichten best-known Lösungen in den Testsätzen RN1, EN1, RN2 und EN2 im Vergleich zu den übrigen Heuristiken stets schlechter ab. Demgegenüber steht allerdings die beste durchschnittliche Laufzeit in allen Testsätzen. Dies ist vor allem dadurch zu erklären, dass die Simplified Independent Set Heuristik in jeder Iteration immer nur mit einem einzigen Independent Set arbeitet, wohingegen die beiden anderen entwickelten Heuristiken mehrere Independent Sets bzw. Cliques betrachten, bzw. die CG Heuristik in jeder Iteration in der Bewertungsfunktion pro Kandidatenknoten viele Nachbarknoten miteinbeziehen muss. Dadurch eignet sich die Simplified Independent Set Heuristik trotz der schlechteren Lösungsqualität überaus gut dazu in Metahueristiken eingesetzt zu werden, um es diesen Heuristiken zu ermöglichen viele Iterationen innerhalb einer kurzen Zeit durchzuführen.

Überprüfung der statistischen Signifikanz der berechneten Ergebnisse

Um die Unterschiede erreichter Ergebnisse in den Bereichen durchschnittliche Anzahl an Regeneratoren und Summe der gefundenen best-known Lösungen zu überprüfen, wird ein gepaarter Wilcoxon Vorzeichenrang-Test herangezogen. Dabei ist die Nullhypothese H_0 stets, dass eine bestimmte Heuristik A angewendet auf einen bestimmten Instanztyp (RN, EN oder beides) signifikant bessere Ergebnisse als eine andere Heuristik B liefert. Unter Verwendung eines Konfidenzniveaus von 95% wird die Nullhypothese mit einer Irrtumswahrscheinlichkeit die dem errechneten p-Wert entspricht entweder angenommen oder verworfen. Da es sich um einen einseitige Test handelt, wird im Fall einer Irrtumswahrscheinlichkeit von $< 2,5\%$ ein signifikanter Unterschied zwischen den beiden Heuristiken angenommen. Der Test wurde für alle Paare von Heuristiken jeweils zweimal mit vertauschten Plätzen durchgeführt, sodass vor allem bei Ergebnissen mit einem geringen Unterschied, also einer hohen Irrtumswahrscheinlichkeit trotzdem jenes Ergebnis angenommen werden kann, das die kleinere Irrtumswahrscheinlichkeit besitzt. Die statistische Auswertung der beiden Kriterien sind in *Tabelle 4.2* und *Tabelle 4.3* zu finden.

Hinsichtlich der durchschnittlichen Anzahl an Regeneratoren (siehe *Tabelle 4.2*) erreicht die CG Heuristik bei RN Instanzen signifikant bessere Ergebnisse als alle übrigen Heuristiken. An zweiter Stelle liegt bei dieser Art von Instanzen eindeutig die Clique Heuristik, sowie an dritter Stelle die Independent Set Heuristik, die sich mit einer gerundeten Irrtumswahrschein-

A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert
CG	IS	RN	0,000	CG	SIS	EN	0,000	CG	SIS	RN+EN	0,000
CG	SIS	RN	0,000	IS	CG	EN	0,000	IS	CG	RN+EN	**0,072**
CG	CL	RN	0,002	IS	SIS	EN	0,000	IS	SIS	RN+EN	0,000
IS	SIS	RN	0,000	IS	CL	EN	0,000	IS	CL	RN+EN	**0,164**
CL	IS	RN	0,000	CL	CG	EN	0,000	CL	CG	RN+EN	0,005
CL	SIS	RN	0,000	CL	SIS	EN	0,000	CL	SIS	RN+EN	0,000

Tabelle 4.2: Auswertung der statistischen Signifikanz der für die Konstruktionsheuristiken berechneten durchschnittlichen Regeneratoren pro Instanz durch den Wilcoxon Test mit der Nullhypothese $H_0 : A$ „besser als“ B

A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert
CG	IS	RN	0,000	CG	SIS	EN	0,000	CG	SIS	RN+EN	0,000
CG	SIS	RN	0,000	IS	CG	EN	0,000	CG	IS	RN+EN	0,**277**
CG	CL	RN	0,000	IS	SIS	EN	0,000	CG	CL	RN+EN	0,**185**
IS	SIS	RN	0,000	IS	CL	EN	0,010	IS	SIS	RN+EN	0,000
CL	IS	RN	0,002	CL	CG	EN	**0,042**	CL	IS	RN+EN	**0,401**
CL	SIS	RN	0,000	CL	SIS	EN	0,000	CL	SIS	RN+EN	0,000

Tabelle 4.3: Auswertung der statistischen Signifikanz der für die Konstruktionsheuristiken berechneten erreichten best-known Lösungen durch den Wilcoxon Test mit der Nullhypothese $H_0 : A$ „besser als“ B

lichkeit von 0% gegen die Simplified Independent Set Heuristik durchsetzt. Bei den *Euclidean Networks* bildet die Independent Set Konstruktionsheuristik den unangefochtenen Spitzenreiter mit einer gerundeter Irrtumswahrscheinlichkeit von 0% im direkten Vergleich mit allen anderen Heuristiken. Die zweitbesten Ergebnisse liefert die Clique Konstruktionsheuristik, die sich auch eindeutig gegen die CG und Simplified Independent Set Heuristik durchsetzt. Den dritten Platz belegt die CG Heuristik vor der SIS Heuristik.

Betrachtet man sowohl *Random Networks* als auch *Euclidean Networks* gemeinsam, so bilden die Clique und Independent Set Konstruktionsheuristik die Spitzenreiter. Sie weisen beide im Gegensatz zu den anderen beiden Heuristiken signifikant bessere Ergebnisse auf, besitzen aber im Vergleich zueinander keinen statistisch signifikanten Unterschied. Die Independent Set Heuristik würde mit einer Irrtumswahrscheinlichkeit von 16,4% als die bessere hinsichtlich der durchschnittlichen Anzahl an Regeneratoren pro Instanz angesehen werden. Hingegen mit einer Irrtumswahrscheinlichkeit von gerundeten 0% liefert die CG Heuristik bei RN+EN Instanzen zusammen signifikant bessere Ergebnisse als die Simplified Independent Set Heuristik und belegt dadurch den dritten von vier Plätzen im Gesamtvergleich.

Hinsichtlich der Summe an erreichten best-known Lösungen (siehe *Tabelle 4.3*) verhalten sich die signifikanten Unterschiede zwischen den Heuristiken beinahe genauso wie bei der Auswertung der durchschnittlichen Regeneratoren pro Instanz. Lediglich bei den *Euclidean Networks* ist kein statistisch signifikanter Unterschied zwischen den Ergebnissen der Clique und CG Heuristik mehr festzustellen. Mit einer Irrtumswahrscheinlichkeit von 4,2% liefert die Cli-

	RN1				EN1			
Heuristik	⌀-Reg	#-Best	$\varnothing t_{\text{Best}}$	⌀-LZ	⌀-Reg	#-Best	$\varnothing t_{\text{Best}}$	⌀-LZ
CG-GRASP	4,56	967 (96,7%)	854,0ms	272,4s	11,98	768 (96,0%)	1.836,4ms	611,8s
IS-GRASP	**4,53**	991 (99,1%)	269,6ms	125,3s	**11,95**	793 (99,1%)	324,5ms	314,8s
SIS-GRASP	4,54	981 (98,1%)	**144,4ms**	**65,5s**	12,00	757 (94,6%)	347,8ms	**124,0s**
CL-GRASP	4,54	989 (98,9%)	444,2ms	342,4s	11,96	787 (98,4%)	450,7ms	433,6s
Hyb-GRASP	**4,53**	991 (99,1%)	199,9ms	158,6s	**11,95**	794 (99,3%)	**186,2ms**	326,3s
RVNS	4,57	951 (95,1%)	665,7ms	155,9s	11,99	764 (95,5%)	664,9ms	231,4s
VNS1	**4,53**	**994 (99,4%)**	883,4ms	484,2s	**11,95**	**795 (99,4%)**	1.105,4ms	829,2s
VNS2	4,54	990 (99,0%)	563,3ms	241,3s	**11,95**	793 (99,1%)	980,6ms	539,0s

Tabelle 4.4: Ergebnisse des Vergleichs der Metaheuristiken im Zuge des ersten Experiments auf den Testsätzen RN1 und EN1

que Heuristik signifikant bessere Ergebnisse. Genauso ist bei der gemeinsamen Betrachtung von *Euclidean* und *Random Networks* kein statistisch signifikanter Unterschied zwischen der Clique und CG Heuristik, der Independent Set und CG Heuristik, sowie der Clique und Independent Set Heuristik feststellbar.

Vergleich der Metaheuristiken

Im Zuge des ersten Experiments wurden auch die Metaheuristiken unter einer vorgegebenen Anzahl an maximalen Iterationen überprüft. Dabei wurden für die Testsätze RN1 und EN1 jeweils 100 Iterationen pro Instanz durchgeführt, sowie bei den größeren Testsätze RN2 und EN2 je nach Komplexität der Instanz zwischen 5 und 20 Iterationen. Wie bereits eingangs erwähnt können durch die unterschiedlichen Gesamtlaufzeiten der einzelnen Metaheuristiken keine exakten Aussagen über die Überlegenheit einer Heuristik gegenüber einer anderen getroffen werden, da bei gleich langer Laufzeit Qualitätsdefizite ausgeglichen werden könnten. Im Zuge dieses Experiments wird daher nur ein Richtwert für die Lösungsqualität im Bezug zu einer festgesetzten Iterationszahl bestimmt, sowie die durchschnittliche Laufzeit einer Iteration evaluiert. Ist eine Metaheuristik beispielsweise nicht in der Lage bei einer langen Laufzeit ähnlich viele best-known Lösungen wie die Heuristiken mit einer kürzeren Laufzeit zu erreichen, ist es ziemlich wahrscheinlich, dass sie bei einem Test mit gleichen Laufzeiten schlechtere Ergebnisse liefert.

Die Übersicht über die erzielten Ergebnisse der Metaheuristiken im ersten Experiment auf den Testsätzen RN1 und EN1 ist in *Tabelle 4.4* zu finden. Die Ergebnisse im Detail sind im Anhang in *Tabelle A.5* bis *Tabelle A.6* aufgelistet. Der CG GRASP Ansatz aus [10] findet im Testsatz RN1 mit durchschnittlich 4,56 benötigten Regeneratoren 96,7% aller best-known Lösungen. Ein ähnliches Ergebnis wird von der Heuristik auch auf EN1 mit 96,0% erreichten best-known Lösungen erzielt. Auffällig ist, dass die durchschnittliche Gesamtlaufzeit in beiden Testsätzen im Vergleich zu den übrigen Verfahren sehr lang ist und auch die best-known Lösungen meistens erst später gefunden werden.

Unter den entwickelten GRASP Ansätzen erzielen die IS-GRASP und Hyb-GRASP Heuristik mit jeweils rund 99% gefundenen best-known Lösungen auf beiden Testsätzen die besten Ergebnisse. Ihre durchschnittliche Laufzeit pro Instanz und die Zeit bis zum Auffinden einer best-known Lösung liegt ebenfalls im Spitzenfeld. Die CL-GRASP und SIS-GRASP Heuristik

	RN2				EN2			
Heuristik	⌀-Reg	#-Best	$⌀t_{\text{Best}}$	⌀-LZ	⌀-Reg	#-Best	$⌀t_{\text{Best}}$	⌀-LZ
CG-GRASP	7,78	499 (49,9%)	211,99s	13.416,9s	12,30	163 (20,4%)	440,98s	12.067,4s
IS-GRASP	7,17	746 (74,6%)	43,26s	1.612,9s	**11,55**	567 (70,9%)	24,40s	1.818,3s
SIS-GRASP	7,53	568 (56,8%)	**7,09s**	**502,8s**	12,42	152 (19,0%)	**21,79s**	**562,6s**
CL-GRASP	**7,09**	794 (79,4%)	169,70s	11.933,7s	11,85	382 (47,8%)	148,25s	5.505,6s
Hyb-GRASP	**7,09**	**799 (79,9%)**	43,18s	2.972,5s	11,58	**596 (74,5%)**	30,37s	2.331,2s
RVNS	7,36	623 (62,3%)	118,00s	4.486,9s	12,21	250 (31,3%)	86,53s	2.702,3s
VNS1	7,28	681 (68,1%)	381,74s	15.630,8s	11,67	482 (60,3%)	191,51s	8.755,8s
VNS2	7,42	610 (61,0%)	81,94s	4.339,0s	11,71	464 (58,0%)	79,89s	3.569,8s

Tabelle 4.5: Ergebnisse des Vergleichs der Metaheuristiken im Zuge des ersten Experiments auf den Testsätzen RN2 und EN2

liegen in punkto Lösungsqualität im Vergleich zu allen anderen Lösungsverfahren im Mittelfeld. Dafür kann die SIS-Heuristik mit ihrer kurzen durchschnittlichen Laufzeit pro Instanz glänzen und legt die Vermutung nahe, dass sie sich bei einem Test mit gleichen Laufzeiten hinsichtlich der Lösungsqualität noch steigern kann.

Die VNS Ansätze erreichen mit über 99% gefundenen best-known Lösungen in beiden Testsätzen die besten Ergebnisse. Demgegenüber steht allerdings auch zumindest im Fall der VNS1 Heuristik die längste durchschnittliche Laufzeit pro Instanz. Dennoch zeigt sich, dass die angeschlossene lokale Suche einen deutlichen Unterschied bei der Lösungsqualität bringt, indem man den Unterschied der durchschnittlich benötigten Regeneratoren der VNS und RVNS Heuristiken vergleicht, der auf beiden Testsätzen ca. 0,4 durchschnittliche Regeneratoren beträgt. Im Allgemeinen schneidet in diesem ersten Experiment die RVNS Heuristik auf beiden Testsätzen am schlechtesten ab.

Aufgrund der langen Laufzeiten wurde die Evaluierung auf den größeren Testsätzen RN2 und EN2 mit weniger Konstruktionsschritten pro Instanz durchgeführt. Die zusammengefassten Ergebnisse sind in *Tabelle 4.5* zu finden. Die Detailergebnisse sind im Anhang in *Tabelle A.7* bis *Tabelle A.8* aufgelistet. Im Gegensatz zu den Resultaten der kleineren Testsätze sind in manchen Bereichen durchaus Veränderungen festzustellen. Beispielsweise verzeichnen die CG-GRASP Heuristik, die RVNS Heuristik und die VNS Ansätze deutliche Einbußen, was die Anzahl an gefundenen best-known Lösungen betrifft. Auffällig ist, dass die Hyb-GRASP Heuristik in beiden Testsätzen die besten Ergebnisse hinsichtlich der Lösungsqualität erzielt. Die CL-GRASP und IS-GRASP bilden auf dem Testsatz RN2 ihre ersten Verfolger. Bei den Instanzen des Testsatzes EN2 bleibt die CL-GRASP Heuristik jedoch in Punkto Lösungsqualität auf der Strecke und somit bilden dort die Independent Set Strategien die Spitzenposition. Betrachtet man die durchschnittlichen Laufzeiten pro Instanz, so setzt sich auf beiden Testsätzen wie auch schon bei den kleineren Instanzen die SIS-GRASP Heuristik eindeutig durch. Durch die Betrachtung der Steigerungsraten der Heuristiken auf den verschiedenen Testsätzen RN1, RN2 und EN1, EN2 bestätigt sich die in *Kapitel 3* bei der Laufzeitanalyse gefundenen Ergebnisse, dass die SIS-GRASP Metaheuristik im Allgemeinen immer schneller Lösungen für beliebige Instanzen berechnet, als die übrigen Heuristiken.

Generell kann festgehalten werden, dass die Ergebnisse der Testsätze RN2 und EN2 den Trend der Resultate für die Testsätze RN1 und EN1 im Zuge des ersten Experiments fortsetzen.

	RN1				EN1			
Heuristik	⌀-Iter	⌀-Reg	#-Best	⌀-t_{Best}	⌀-Iter	⌀-Reg	#-Best	⌀-t_{Best}
CG-GRASP	365,1	4,54	985 (98,5%)	1,81s	4.626,2	11,97	783 (97,9%)	2,29s
IS-GRASP	2.604,1	**4,53**	**1000 (100%)**	0,33s	1.049,0	**11,94**	**800 (100%)**	0,81s
SIS-GRASP	3.009,8	**4,53**	**1000 (100%)**	0,31s	1.769,0	11,95	795 (99,4%)	0,73s
CL-GRASP	2.280,9	**4,53**	996 (99,6%)	0,47s	970,3	11,95	793 (99,1%)	0,72s
Hyb-GRASP	2.506,9	**4,53**	**1000 (100%)**	**0,24s**	1.014	**11,94**	**800 (100%)**	**0,33s**
RVNS	1.241,3	4,54	990 (99,0%)	1,34s	46.592,7	11,95	792 (99,0%)	1,35s
VNS1	558,1	**4,53**	993 (99,3%)	0,82s	22.140,4	11,95	794 (99,3%)	0,96s
VNS2	698,2	**4,53**	994 (99,4%)	0,72s	23.377,2	11,95	798 (99,8%)	1,22s

Tabelle 4.6: Ergebnisse des Vergleichs der Metaheuristiken im Zuge des zweiten Experiments auf den Testsätzen RN1 und EN1

Daher soll im zweiten Experiment umso mehr überprüft werden, ob diese Unterschiede bei gleichen Laufzeiten noch immer vorhanden sind, oder ob sich die Ergebnisse annähern, bzw. ob sie weiter auseinander klaffen.

4.4 Zweites Experiment: Max-Laufzeit Evaluierung

Um die verschiedenen Metaheuristiken vergleichen zu können, wurde im zweiten Experiment für jede Instanz eine maximale Laufzeit festgesetzt. Dadurch haben Heuristiken, die eine kürzere Laufzeit besitzen, aber in gleich viel Iterationen im Vergleich zu anderen schlechtere Lösungen berechnen die Chance durch die einheitliche Laufzeit Defizite auszugleichen. Besonders die GRASP Ansätze sind darauf ausgelegt viele Iterationen pro Instanz durchzuführen, um so suboptimale Lösungen durch das breitflächige Durchsuchen des Lösungsraums zu vermeiden. Die vorgeschriebenen Laufzeiten pro Instanz wurden im Zuge des Experiments so gewählt, dass jede Heuristik mindestens ca. 50-150 Iterationen pro Instanz durchführen kann. Dabei werden bei den VNS Ansätzen durch den ständigen Neubeginn bei der ersten Nachbarschaft, wenn die Lösung verbessert wurde, oft mehrere Anwendungen der Nachbarschaftsstrukturen innerhalb einer Iteration durchgeführt. Bei GRASP Heuristiken entspricht die Anzahl der Iterationen genau der Anzahl an konstruierten Lösungen. Neben den durchschnittlich benötigten Regeneratoren pro Instanz und der Summe der gefundenen best-known Lösungen ist in diesem Zusammenhang auch die Zeit bis zum Auffinden der besten Lösung ein interessanter Parameter. Die zusammengefassten Ergebnisse der Durchführung des zweiten Experiments auf den Testsätzen RN1 und EN1 ist in *Tabelle 4.6* zu finden. Die Detailergebnisse sind im Anhang in *Tabelle A.9* bis *Tabelle A.12* vermerkt.

Die CG-GRASP Metaheuristik findet auf dem Testsatz RN1 98,5% aller best-known Lösungen und erreicht damit im Vergleich zu den übrigen Heuristiken das schlechteste Ergebnis mit durchschnittlich 4,54 Regeneratoren pro Instanz. Bei 365,1 durchschnittlichen Iterationen pro Instanz benötigte es meistens 1,81 Sekunden, bis die beste Lösung gefunden wurde. Im Testsatz EN1 bleibt die CG-GRASP Heuristik mit 97,7% errechneten best-known Lösungen und 11,97 durchschnittlichen Regeneratoren ebenfalls hinter den übrigen Heuristiken zurück, owbohl sie mit 4.626,2 Iterationen im Mittelfeld liegt. Die beste Lösung wird hier erst nach durchschnittlich

2,29 Sekunden gefunden.

Die IS-GRASP und Hybrid-GRASP Heuristik finden auf beiden Testsätzen jeweils 100% aller best-known Lösungen. Sie können auf dem Testatz RN1 mit 4,53 durchschnittlichen Regeneratoren, 2.604,1 bzw. 2.506,9 Iterationen pro Instanz und dem Auffinden der besten Lösung nach 0,33 bzw. 0,24 Sekunden überzeugen.

Ähnlich gut präsentiert sich auch die SIS-GRASP Heuristik, die auf dem Testsatz RN1 alle best-known Lösungen findet und auf dem Testsatz EN1 mit 99,4% aller best-known Lösungen und 11,95 durchschnittlichen Regeneratoren im Vergleich den vierten von acht Plätzen belegt.

Die VNS Metaheuristiken erzielten zwar gute Ergebnisse, kommen jedoch nicht ganz an die besten GRASP Ergebnisse heran. Auf beiden Testsätzen erreicht die RVNS mit jeweils 99,0% best-known Lösungen die schlechtesten Ergebnisse, die im Gesamtvergleich auf beiden Testsätzen den vorletzten Platz bedeuten. Trotz der großen Anzahl an Iterationen ist dieses Abschneiden im Gegensatz zu den beiden VNS Ansätzen dadurch zu erklären, dass bei der VNS durch die zwischengeschaltete lokale Suche zwar mehr Zeit pro Iteration benötigt wird, jedoch die Ergebnisse eine Spur besser als bei reinen Shaking Nachbarschaften der RVNS ausfallen. Die VNS Ansätze bilden mit 99,3% und 99,4% bzw. 99,3% und 99,8% best-known Lösungen gemeinsam mit der Clique-GRASP und SIS-GRASP Heuristik das Mittelfeld im Vergleich der Heuristiken zueinander. Trotzdem finden sie durch die Anwendung der Shaking-Nachbarschaften die beste Lösung im Schnitt nach ca. einer Sekunde, was bei den GRASP Heuristiken auf den getesteten Instanzen deutlich schneller passiert. Die große Anzahl an Iterationen der VNS Ansätze auf den Instanzen des EN1 Testsatzes ist durch die Instanzzusammensetzung zu erklären, die aus hauptsächlich dünnen Graphen besteht. Dadurch wird im Preprocessing (siehe *Kapitel 1.3*) der Graph oft in kleinere Subgraphen aufgespalten für die sehr schnell die Bedingung über die Verbesserungsmöglichkeit einer Lösung im Zuge einer Nachbarschaftsstruktur bestimmt werden kann, jedoch die Neuberechnung einer neuen Lösung mit anschließender lokalen Suche im Vergleich deutlich länger dauert.

Generell hat sich die Vermutung bestätigt, dass sich die Ergebnisse der Heuristiken durch längere Laufzeiten aneinander annähern. Auch die Überlegung, dass beispielswiese die SIS-GRASP Heuristik bei gleicher Laufzeit ihre Defizite bei der Lösungsqualität verkleinern kann ist eingetreten. Trotzdem kristallisiert sich der Trend heraus, dass die CG-Konstruktionsheuristik zwar die zuverlässigsten Ergebnisse berechnet, jedoch die CG-GRASP Heuristik insgesamt im Vergleich zu den entwickelten Clique und Independent Set Mechanismen abfällt.

In *Tabelle 4.7* sind die zusammengefassten Ergebnisse der Testsätze RN2 und EN2 aufgelistet. Die Resultate im Detail sind im Anhang in *Tabelle A.11* bis *Tabelle A.12* zu finden. Durch die bereits im ersten Experiment festgestellten langen Laufzeiten bei Instanzen mit 400 oder 500 Knoten, wurde das Zeitbudget pro Instanz so bemessen, dass vor allem Instanzen mit 200 und 300 Knoten im Fokus stehen, d.h. dort eine Mindestanzahl von ca. 30-80 Iterationen pro Heuristik erzeilt wird.

Die CG-GRASP Heuristik erzielt bei der geringsten Anzahl an Iterationen mit 42,8% bzw. 18,6% aller best-known Lösungen im Vergleich zu den anderen Heuristiken die schlechtesten Ergebnisse auf beiden Testsätzen. Im Schnitt wurde auf dem Testsatz RN2 nach 95,16 Sekunden die beste Lösung gefunden, wodurch sie in diesem Kriterium vor den VNS Heuristiken und der CL-GRASP Heuristik liegt. Die besten Resultate erreicht wie auch schon in den kleineren Test-

	RN2				EN2			
Heuristik	⌀-Iter	⌀-Reg	#-Best	⌀-t_{Best}	⌀-Iter	⌀-Reg	#-Best	⌀-t_{Best}
CG-GRASP	21,7	7,97	428 (42,8%)	95,16s	9,0	12,57	149 (18,6%)	488,41s
IS-GRASP	187,9	7,10	819 (81,9%)	55,97s	41,5	11,39	670 (83,8%)	**34,17s**
SIS-GRASP	297,6	7,24	720 (72,0%)	55,37s	143,9	11,80	375 (46,9%)	95,99s
CL-GRASP	29,1	7,17	741 (74,1%)	128,26s	18,2	11,88	377 (47,1%)	147,70s
Hyb-GRASP	85,0	**7,07**	**848 (84,8%)**	**40,03s**	34,4	**11,37**	**688 (86,0%)**	36,65s
RVNS	69,2	7,37	613 (61,3%)	42,65s	42,1	11,98	353 (44,1%)	129,25s
VNS1	24,9	7,43	590 (59,0%)	193,85s	11,5	11,80	392 (49,0%)	240,65s
VNS2	75,3	7,38	675 (67,5%)	100,68s	25,4	11,62	508 (63,5%)	79,58s

Tabelle 4.7: Ergebnisse des Vergleichs der Metaheuristiken im Zuge des zweiten Experiments auf den Testsätzen RN2 und EN2

sätzen die Hyb-GRASP Heuristik. Mit 84,8% und 86,0% der gefundenen best-known Lösungen setzt sie sich deutlich im Gegensatz zu den übrigen Heuristiken ab. Auch in der durchschnittlichen Dauer bis eine best-known Lösung gefunden wurde, liegt sie mit einem ersten und einem zweiten Platz sehr gut im Vergleich. Die einzige Heuristik die auf den großen Testsätzen noch einigermaßen mit der Hyb-GRASP Heuristik mithalten kann ist die IS-GRASP Heuristik. Mit 81,9% und 83,8% aller best-known Lösungen erreicht sie auf beiden Testsätzen konstant gute Ergebnisse. Ebenbürtig sind sich die SIS-GRASP und CL-GRASP Heuristik, die best-known Lösungen im Bereich von 72-74% auf dem Testsatz RN2 und 46-47% auf dem Testsatz EN2 finden. Dahinter liegen wie schon in den kleineren Testsätzen die VNS und RVNS Ansätze mit rund 60% bzw. 45% aller best-known Lösungen. Einzig auf dem Testsatz EN2 sticht die VNS2 Heuristik mit 63,5% gefundenen best-known Löungen heraus. Dieser Unterschied zwischen der VNS1 und VNS2 Heuristik deutet darauf hin, dass die effektivere Reihenfolge der Shaking-Nachbarschaften auf dem Testsatz EN2 jene ist, wo die Independent Set Nachbarschaften als erstes vor der Clique Nachbarschaft angewendet werden. Dies könnte auch aus der Überlegenheit der IS-GRASP gegenüber der CL-GRASP Heuristik auf diesem Testsatz geschlossen werden, um so möglichst früh gute Lösungen zu erlangen.

Betrachtet man die Gesamtergebnisse des zweiten Experiments, so kristallisieren sich drei leistungsfähige Heuristiken heraus. Diese sind die IS-GRASP, SIS-GRASP und Hyb-GRASP Heuristik. Alle drei erreichen auf den Testsätzen RN1 und EN1 im vorgegebenen Zeitbudget alle best-known Lösungen. Die Hyb-GRASP Heuristik schafft es obendrein auch noch auf den großen Testsätzen ihre hohe Lösungsqualität aufrecht zu halten und schließt die Testserie mit den insgesamt besten Ergebnissen ab. Dicht gefolgt von der IS-GRASP Heuristik, die zwar im Vergleich ein bisschen weniger best-known Lösungen auf den großen Testsätzen findet, aber nicht wirklich weit abgeschlagen ist. Die SIS-GRASP Heuristik muss auf den Testsätzen RN2 und EN2 leider deutliche Einbußen hinnehmen, was durch die geringere Anzahl an Iterationen im Vergleich zu den kleinen Testsätzen zu erklären ist. Durch die einfachere Funktionsweise benötigt die SIS-GRASP Heuristik je nach Komplexität der Instanz anscheinend eine gewisse Mindestanzahl an Iterationen, um mit der Qualität der übrigen Heuristiken mitzuhalten. Dadurch ergibt sich die Tendenz, dass sie mit steigender Anzahl an Knoten und der proportional steigenden Laufzeit im Vergleich zur IS-GRASP oder Hyb-GRASP Heuristik aufgrund der kleiner

A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert
Hyb	CG	RN	0,000	Hyb	CG	EN	0,000	Hyb	CG	RN+EN	0,000
Hyb	CL	RN	0,004	Hyb	CL	EN	0,000	Hyb	CL	RN+EN	0,000
Hyb	IS	RN	**0,154**	Hyb	IS	EN	**0,069**	Hyb	IS	RN+EN	**0,065**
Hyb	SIS	RN	0,003	Hyb	SIS	EN	0,000	Hyb	SIS	RN+EN	0,000
Hyb	RVNS	RN	0,000	Hyb	RVNS	EN	0,000	Hyb	RVNS	RN+EN	0,000
Hyb	VNS1	RN	0,001	Hyb	VNS1	EN	0,000	Hyb	VNS1	RN+EN	0,000
Hyb	VNS2	RN	0,002	Hyb	VNS2	EN	0,000	Hyb	VNS2	RN+EN	0,000
IS	CG	RN	0,000	IS	CG	EN	0,000	IS	CG	RN+EN	0,000
IS	CL	RN	0,010	IS	CL	EN	0,000	IS	CL	RN+EN	0,000
IS	SIS	RN	0,003	IS	SIS	EN	0,000	IS	SIS	RN+EN	0,000
IS	RVNS	RN	0,000	IS	RVNS	EN	0,000	IS	RVNS	RN+EN	0,000
IS	VNS1	RN	0,001	IS	VNS1	EN	0,000	IS	VNS1	RN+EN	0,000
IS	VNS2	RN	0,001	IS	VNS2	EN	0,000	IS	VNS2	RN+EN	0,000
CL	CG	RN	0,000	CL	CG	EN	0,000	CL	CG	RN+EN	0,000
SIS	CL	RN	**0,450**	SIS	CL	EN	**0,074**	SIS	CL	RN+EN	**0,177**
CL	RVNS	RN	0,000	CL	RVNS	EN	0,006	CL	RVNS	RN+EN	0,000
CL	VNS1	RN	0,001	VNS1	CL	EN	**0,044**	CL	VNS1	RN+EN	**0,076**
CL	VNS2	RN	**0,062**	VNS2	CL	EN	0,000	VNS2	CL	RN+EN	**0,079**
SIS	CG	RN	0,000	SIS	CG	EN	0,000	SIS	CG	RN+EN	0,000
SIS	RVNS	RN	0,005	SIS	RVNS	EN	0,004	SIS	RVNS	RN+EN	0,000
SIS	VNS1	RN	0,001	VNS1	SIS	EN	**0,448**	SIS	VNS1	RN+EN	0,003
SIS	VNS2	RN	0,009	VNS2	SIS	EN	0,000	VNS2	SIS	RN+EN	**0,101**
VNS2	CG	RN	0,000	VNS2	CG	EN	0,000	VNS2	CG	RN+EN	0,000
VNS2	RVNS	RN	**0,238**	VNS2	RVNS	EN	0,000	VNS2	RVNS	RN+EN	0,001
VNS2	VNS1	RN	**0,211**	VNS2	VNS1	EN	0,000	VNS2	VNS1	RN+EN	0,002
VNS1	CG	RN	0,000	VNS1	CG	EN	0,000	VNS1	CG	RN+EN	0,000
RVNS	VNS1	RN	**0,150**	VNS1	RVNS	EN	0,000	VNS1	RVNS	RN+EN	**0,031**
RVNS	CG	RN	0,000	RVNS	CG	EN	0,000	RVNS	CG	RN+EN	0,000

Tabelle 4.8: Ergebnisse des Wilcoxon Tests für die durchschnittliche Anzahl an Regeneratoren der Metaheuristiken im zweiten Experiment mit der Nullhypothese $H_0 : A$ „besser als“ B

werdenden Anzahl an Iterationen voraussichtlich weiter abfallen wird.

Überprüfung der statistischen Signifikanz der berechneten Ergebnisse

Zur Überprüfung der statistischen Signifikanz der erzielten Ergebnisse wird wie auch schon beim ersten Experiment ein Wilcoxon Test herangezogen. Auch hier lautet die Nullhypothese H_0 „eine bestimmte Heuristik A angewendet auf einen bestimmten Instanztyp (RN, EN oder beides) liefert signifikant bessere Ergebnisse als eine andere Heuristik B“. Sofern der berechnete p-Wert größer als 2,5% ist, wird ein signifikanter Unterschied zwischen den beiden Heuristiken angenommen. In *Tabelle 4.8* und *Tabelle 4.9* sind die Ergebnisse der statistischen Auswertung für die Kriterien durchschnittliche Anzahl an Regeneratoren und Summe der gefundenen best-known Lösungen zu finden.

In beiden Kriterien bildet die Mehrheit an gefundenen Ergebnissen tatsächlich auch statistisch signifikante Unterschiede. So liefern die IS-GRASP und Hyb-GRASP Heuristik in beiden Kriterien im Vergleich zu den übrigen Heuristiken die signifikant besten Ergebnisse, weisen jedoch untereinander keinen signifikanten Unterschied auf. Lediglich mit einer Fehlerwahrschein-

A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert	A	B	Testsatz	p-Wert
Hyb	CG	RN	0,000	Hyb	CG	EN	0,000	Hyb	CG	RN+EN	0,000
Hyb	CL	RN	0,001	Hyb	CL	EN	0,000	Hyb	CL	RN+EN	0,000
Hyb	IS	RN	**0,154**	Hyb	IS	EN	**0,061**	Hyb	IS	RN+EN	**0,039**
Hyb	SIS	RN	0,003	Hyb	SIS	EN	0,000	Hyb	SIS	RN+EN	0,000
Hyb	RVNS	RN	0,000	Hyb	RVNS	EN	0,000	Hyb	RVNS	RN+EN	0,000
Hyb	VNS1	RN	0,001	Hyb	VNS1	EN	0,000	Hyb	VNS1	RN+EN	0,000
Hyb	VNS2	RN	0,001	Hyb	VNS2	EN	0,000	Hyb	VNS2	RN+EN	0,000
IS	CG	RN	0,000	IS	CG	EN	0,000	IS	CG	RN+EN	0,000
IS	CL	RN	0,010	IS	CL	EN	0,000	IS	CL	RN+EN	0,000
IS	SIS	RN	0,003	IS	SIS	EN	0,000	IS	SIS	RN+EN	0,000
IS	RVNS	RN	0,000	IS	RVNS	EN	0,000	IS	RVNS	RN+EN	0,000
IS	VNS1	RN	0,001	IS	VNS1	EN	0,000	IS	VNS1	RN+EN	0,000
IS	VNS2	RN	0,001	IS	VNS2	EN	0,000	IS	VNS2	RN+EN	0,000
CL	CG	RN	0,000	CL	CG	EN	0,000	CL	CG	RN+EN	0,000
CL	SIS	RN	**0,286**	CL	SIS	EN	**0,479**	CL	SIS	RN+EN	**0,296**
CL	RVNS	RN	0,000	CL	RVNS	EN	**0,096**	CL	RVNS	RN+EN	0,000
CL	VNS1	RN	0,001	VNS1	CL	EN	**0,162**	CL	VNS1	RN+EN	0,013
CL	VNS2	RN	**0,058**	VNS2	CL	EN	0,000	VNS2	CL	RN+EN	**0,058**
SIS	CG	RN	0,000	SIS	CG	EN	0,000	SIS	CG	RN+EN	0,000
SIS	RVNS	RN	0,002	SIS	RVNS	EN	**0,064**	SIS	RVNS	RN+EN	0,000
SIS	VNS1	RN	0,001	VNS1	SIS	EN	**0,210**	SIS	VNS1	RN+EN	0,021
SIS	VNS2	RN	0,016	VNS2	SIS	EN	0,000	VNS2	SIS	RN+EN	0,015
VNS2	CG	RN	0,000	VNS2	CG	EN	0,000	VNS2	CG	RN+EN	0,000
VNS2	RVNS	RN	0,022	VNS2	RVNS	EN	0,000	VNS2	RVNS	RN+EN	0,000
VNS2	VNS1	RN	0,012	VNS2	VNS1	EN	0,000	VNS2	VNS1	RN+EN	0,000
VNS1	CG	RN	0,000	VNS1	CG	EN	0,000	VNS1	CG	RN+EN	0,000
RVNS	VNS1	RN	**0,171**	VNS1	RVNS	EN	0,003	VNS1	RVNS	RN+EN	**0,076**
RVNS	CG	RN	0,000	RVNS	CG	EN	0,000	RVNS	CG	RN+EN	0,000

Tabelle 4.9: Ergebnisse des Wilcoxon Tests für die Summe an best-known Lösungen der Metaheuristiken im zweiten Experiment mit der Nullhypothese $H_0 : A$ „besser als“ B

lichkeit von ca. 6,5% würde man die Resultate der Hyb-GRASP Heuristik in Summe auf allen getesteten Instanzen als besser ansehen. Die CL-GRASP und SIS-GRASP Heuristik belegen im Gesamtvergleich den geteilten dritten Platz, da sie untereinander ebenfalls keine signifikanten Unterschiede in ihren Ergebnissen aufweisen, sich jedoch gegenüber den verbleibenden Heuristiken absetzen können. Lediglich die VNS2 Heuristik schafft es im Gegensatz zur CL-GRASP Heuristik Ergebnisse ohne signifikanten Unterschied zu erzielen, wodurch sie im Vergleich der Heuristiken den fünften Platz belegt. Dahinter kann sich die RVNS Heuristik mit einem signifikanten Unterschied von der VNS1 Heuristik in beiden Kriterien absetzen und den letzten Platz mit einem signifikanten Unterschied zu allen anderen Heuristiken platziert sich die CG-GRASP Heuristik. Generell sind leichte Abweichungen bei den erzielten Ergebnissen hinsichtlich der durchschnittlichen Anzahl an Regeneratoren und der Summe an best-known Lösungen zu sehen, jedoch ist ein allgemeiner Trend erkennbar. Einzige Ausnahme bildet der Vergleich der SIS-GRASP und CL-GRASP Heuristik. Dort sind zwar in beiden Kriterien keine signifikante Unterschiede erkennbar, jedoch geht die Tendenz hinsichtlich der durchschnittlichen Regeneratoren eher zur SIS-GRASP Heuristik, wohingegen bei der Summe an best-known Lösungen die CL-GRASP Heuristik eher bessere Ergebnisse liefert.

KAPITEL 5

Conclusio und zukünftige Arbeiten

Das Regenerator Location Problem versucht in einem optischen Netzwerk so wenige Regeneratoren wie nur möglich zu platzieren, damit alle Knoten miteinander ohne Einschränkungen kommunizieren können. Diese Platzierung von Regeneratoren ist notwendig, da ein optisches Signal nach einer gewissen zurückgelegten Distanz $d_{\max}$ periodisch regeneriert werden muss, damit eine störungsfreie Kommunikation möglich ist. Zur Lösung des RLP wurden in der Literatur bisher GRASP Ansätze und exakte Verfahren in Form eines Branch-and-Cut Algorithmus beschrieben. Dabei hat sich gezeigt, dass exakte Verfahren bei Instanzen, die mehr als 100 Knoten enthalten aufgrund der langen Laufzeit nicht mehr einsetzbar sind. Daher hat sich der CG-GRASP Ansatz als die bisher beste Heuristik zur Lösung des RLP herausgestellt, da sie bei Instanzen mit wenigen Knoten an die Ergebnisse der exakten Verfahrens herankommt und auch für große Instanzen Lösungen berechnen kann. Der *Communication Graph* ist ein ungemein hilfreiches Werkzeug, das durch einen Transformationsprozess aus dem eigentlichen Instanzgraphen gewonnen wird und den tatsächlich benötigten Bedarf an Regeneratoren einfacher darstellt. Darüber hinaus ist es auch möglich in Form eines Preprocessings Knoten von vornherein als Lösungsknoten zu identifizieren, bzw. den *Communication Graph* in mehrere Sub-Graphen aufzuteilen, die parallel gelöst werden können.

Im Zuge dieser Diplomarbeit wurden alternative Möglichkeiten für neue Konstruktionsheuristiken und Nachbarschaftsstrukturen für die lokale Suche untersucht. Dabei haben sich die Bestimmung von Cliques und Independent Sets im *Communication Graph* als sehr interessante Mechanismen entpuppt. Idee ist es einerseits Cliques im *Communication Graph* durch die Platzierung von Regeneratoren immer weiter zusammenzufassen, bis nur noch eine einzige Clique bestehen bleibt und die identifizierten Regeneratoren eine gültige Lösung des RLP darstellen. Andererseits stellt ein Independent Set eine Menge an NDC-Knotenpaaren dar, von denen möglichst viele pro Iteration durch die Platzierung eines Regenerators eliminiert werden sollen. Diese Ideen wurden in weiterer Folge in Konstruktionsheuristiken und Nachbarschaftsstrukturen für die lokale Suche eingebettet und schlussendlich als Teil von GRASP und VNS Metaheuristiken getestet.

Die Evaluierung der Ergebnisse wurde auf insgesamt vier Testsätzen, die sich aus *Random*- und *Euclidean Networks* Instanzen mit jeweils bis zu 500 Knoten zusammensetzen durchgeführt. Dabei wurden insgesamt zwei Experimente realisiert, wobei einmal eine maximale Anzahl an Iterationen und einmal eine maximale Laufzeit vorgegeben war. Da sich im Zuge des Tests bei festgesetzten Iterationen ein deutlicher Unterschied in den Laufzeiten der einzelnen Heuristiken gezeigt hat, lässt vor allem das zweite Experiment mit festgesetzten, gleichen Laufzeiten stichhaltigere Aussagen bezüglich der Überlegenheit von einer Heuristik gegenüber einer anderen auf den jeweiligen Testsätzen zu. Nach Auswertung aller Ergebnisse zeigt sich, dass die Independent Set und Clique Ansätze durch ihre kurze Laufzeit und dadurch hohe Anzahl an Iterationen ausgesprochen gute Ergebnisse liefern. Besonders der Hyb-GRASP-Ansatz, der eine Independent Set Konstruktionsheuristik und eine Clique Nachbarschaftsstruktur verwendet, konnte auf allen Testsätzen am deutlichsten überzeugen und die meisten best-known Lösungen erreichen. Um die Ergebnisse auf ihre statistische Signifikanz hin zu überprüfen wurde ein Wilcoxon Test durchgeführt, der im Großen und Ganzen die gefundenen Resultate mit geringer Fehlerwahrscheinlichkeit bestätigte. Insgesamt belegten im Vergleich der gefundenen best-known Lösungen auf allen Instanzen die Hyb-GRASP und IS-GRASP Heuristik den geteilten ersten Platz, gefolgt von der SIS-GRASP und CL-GRASP Heuristik auf dem geteilten dritten Platz. Dahinter erreichten die VNS und RVNS Heuristiken die nächstbesten Ergebnisse und konnten sich dadurch ebenfalls gegen die in der Literatur beschriebene CG-GRASP Heuristik durchsetzen. Dadurch scheinen Strategien, die die Struktur des Graphen in einer geschickten Art und Weise ausnützen und dadurch in kürzerer Zeit gute Lösungen berechnen können durchaus konkurrenzfähig im Vergleich zu bereits bestehenden Ansätzen zu sein.

Um die gefundenen Resultate noch weiter zu verbessern könnten speziell für große Instanzen weiterführende Mechanismen erforschft werden, damit bei einer Auswahl von vielen großen Cliques bzw. Independent Sets trotzdem immer jene Menge an Knoten ausgewählt wird, die im Endeffekt zu der besten Lösung führt. Auch könnte im Sinne der SIS-GRASP Strategie, die mit sehr einfachen Maßnahmen im Zuge eines GRASP Ansatzes mit Abstand am schnellsten durchaus robuste Ergebnisse liefert, durch neue Ideen die Laufzeit speziell bei größeren Instanzen noch weiter verringert werden, um dort noch bessere Lösungen zu finden.

ANHANG A

Anhang

	RN1		∅-Reg				#-Best				Σ-LZ(s)			
	p	Iter	CG	IS	SIS	CL	CG	IS	SIS	CL	CG	IS	SIS	CL
$N = 40$	0.1	1.000	**1,40**	**1,40**	**1,40**	**1,40**	**10.000**	**10.000**	**10.000**	**10.000**	33,77	4,42	**3,15**	21,38
	0.3	1.000	**2,00**	2,02	2,04	2,01	**10.000**	9.758	9.635	9.938	30,39	10,82	**4,63**	32,13
	0.5	1.000	**3,02**	3,06	3,30	3,04	**9.826**	9.394	6.983	9.552	50,43	11,57	**8,66**	30,81
	0.7	1.000	**4,72**	4,76	5,20	4,77	5.831	**5.922**	2.991	5.553	87,12	28,63	**15,31**	55,07
	0.9	1.000	9,89	10,07	10,56	**9,85**	**2.904**	2.201	1.021	2.616	180,38	64,99	**34,38**	101,86
$N = 60$	0.1	1.000	**1,90**	**1,90**	**1,90**	**1,90**	**10.000**	**10.000**	**10.000**	**10.000**	833,81	21,94	**9,92**	165,57
	0.3	1.000	**2,06**	2,19	2,26	2,13	**9.425**	8.057	7.359	8.695	604,90	44,32	**13,68**	140,01
	0.5	1.000	**3,30**	3,39	3,82	3,35	**7.021**	6.054	1.927	6.503	252,75	53,13	**52,42**	129,52
	0.7	1.000	**5,14**	5,38	5,85	5,31	**6.666**	4.955	2.026	5.496	392,03	94,45	**40,94**	209,22
	0.9	1.000	**11,37**	11,63	12,40	11,49	**1.751**	1.414	399	936	908,23	218,68	**111,76**	422,35
$N = 80$	0.1	1.000	**1,90**	**1,90**	**1,90**	**1,90**	**10.000**	**10.000**	**10.000**	**10.000**	2.734,01	56,52	**19,61**	420,49
	0.3	1.000	**2,67**	2,68	2,80	2,68	**6.329**	6.248	4.960	6.175	1.053,40	49,34	**27,81**	257,76
	0.5	1.000	**3,65**	3,78	3,97	3,83	**5.535**	4.203	2.816	3.725	1.347,02	93,06	**46,96**	410,12
	0.7	1.000	**5,75**	5,84	6,34	5,82	**2.499**	1.926	492	1.906	1.443,31	187,37	**99,80**	575,22
	0.9	1.000	12,42	12,57	13,43	**12,38**	**945**	766	209	836	3.202,66	565,37	**226,79**	1.163,15
$N = 100$	0.1	1.000	**2,00**	**2,00**	**2,00**	**2,00**	**10.000**	**10.000**	**10.000**	**10.000**	1.879,60	51,01	**27,08**	433,98
	0.3	1.000	**2,83**	2,91	2,94	2,88	**8.674**	7.860	7.606	8.156	9.802,95	142,10	**53,04**	646,91
	0.5	1.000	**4,00**	4,02	4,22	**4,00**	**9.000**	8.755	6.893	**9.000**	3.674,61	187,15	**88,46**	871,98
	0.7	1.000	**6,04**	6,19	6,80	6,05	2.174	1.699	595	**2.318**	3.682,94	388,96	**168,21**	1.340,70
	0.9	1.000	**13,22**	13,74	14,49	13,27	**769**	127	55	561	8.108,63	1.349,23	**433,61**	2.696,03
	Gesamt	20.000	**4,96**	5,07	5,38	5,00	**129.349**	119.339	95.967	121.966	40.302,92	3.623,04	**1.486,21**	10.124,26

Tabelle A.1: Evaluierung der Konstruktionsheuristiken im Zuge des ersten Experiments in RN1 mit jeweils 1.000 konstruierten Lösungen pro Instanz, siehe *Kapitel 4.3*

	EN1		⌀-Reg				#-Best				Σ-LZ(s)			
	$d_{\max}$	Iter	CG	IS	SIS	CL	CG	IS	SIS	CL	CG	IS	SIS	CL
$N = 40$	20	1.000	**12,70**	12,71	13,88	12,86	**8.018**	7.897	5.071	6.369	67,92	35,55	**24,62**	32,58
	30	1.000	12,89	12,83	13,78	**12,82**	6.291	6.722	3.412	6.752	79,32	36,59	**24,37**	37,02
	40	1.000	**11,65**	11,88	12,94	11,93	**8.541**	6.688	4.822	5.722	67,00	37,32	**25,07**	29,19
	50	1.000	**12,39**	**12,39**	14,27	12,53	**9.348**	9.121	3.987	7.715	68,03	31,99	**19,13**	35,50
$N = 60$	20	1.000	12,59	**12,55**	14,28	12,81	3.177	**4.087**	614	2.041	708,45	185,03	**113,64**	189,58
	30	1.000	13,67	13,52	16,69	**13,43**	2.768	4.034	259	**4.390**	666,94	167,52	**116,67**	168,82
	40	1.000	12,69	12,59	15,01	**12,57**	3.190	3.594	255	**5.086**	747,41	212,32	**208,53**	212,99
	50	1.000	12,81	**12,30**	14,59	12,55	2.543	**4.388**	1.028	2.845	909,91	197,45	**125,98**	214,56
$N = 80$	20	1.000	13,10	**12,75**	14,95	12,93	2.298	**4.102**	501	3.573	2.656,35	484,15	**259,62**	465,41
	30	1.000	13,39	**13,27**	15,33	13,36	1.774	**2.525**	151	1.850	2.636,66	471,00	**223,35**	551,82
	40	1.000	12,56	**12,44**	14,89	12,46	2.768	3.027	406	**3.454**	2.690,36	455,72	**226,67**	492,34
	50	1.000	12,93	**12,56**	14,59	12,60	1.607	2.624	403	**2.971**	2.678,44	487,30	**230,39**	484,01
$N = 100$	20	1.000	13,81	13,38	16,18	**13,37**	143	1.003	9	**1.651**	7.470,57	1.027,08	**486,23**	1.329,79
	30	1.000	13,16	**12,84**	14,69	13,07	1.471	**1.623**	182	1.052	7.505,89	977,39	**402,04**	1.269,19
	40	1.000	13,18	13,23	15,17	**13,09**	1.202	1.210	142	**1.386**	7.290,01	1.010,81	**410,21**	1.250,64
	50	1.000	12,78	**12,19**	14,78	12,62	412	**1.197**	98	776	7.163,81	911,71	**395,21**	1.220,45
	Gesamt	16.000	12,89	**12,71**	14,75	12,81	55.551	**63.842**	21.340	57.633	43.407,05	6.728,93	**3.291,72**	7.983,88

Tabelle A.2: Evaluierung der Konstruktionsheuristiken im Zuge des ersten Experiments in EN1, siehe *Kapitel 4.3*

	RN2		⌀-Reg				#-Best				Σ-LZ(s)			
	p	Iter	CG	IS	SIS	CL	CG	IS	SIS	CL	CG	IS	SIS	CL
$N = 200$	0.1	200	**2,00**	**2,00**	**2,00**	**2,00**	**2.000**	**2.000**	**2.000**	**2.000**	3.973,34	62,46	**35,87**	610,07
	0.3	180	**3,00**	**3,00**	3,04	**3,00**	**1.800**	**1.800**	1.724	**1.800**	7.816,85	134,71	**61,14**	1.278,74
	0.5	150	**4,45**	4,69	5,12	4,58	**829**	467	37	623	9.972,52	252,46	**104,28**	1.918,61
	0.7	120	**7,05**	7,41	8,29	7,17	**478**	283	27	411	8.492,93	459,03	**155,04**	2.400,19
	0.9	100	**16,42**	17,20	18,37	16,62	**65**	2	0	19	16.008,05	1.444,47	**349,94**	4.133,64
$N = 300$	0.1	200	**2,00**	**2,00**	**2,00**	**2,00**	**2.000**	**2.000**	**2.000**	**2.000**	19.373,82	197,05	**119,54**	2.171,43
	0.3	180	3,05	3,19	3,46	**3,04**	1.708	1.459	976	**1.729**	39.317,32	480,21	**225,36**	5.601,17
	0.5	150	**4,99**	5,03	5,65	**4,99**	615	573	206	**617**	40.891,27	977,76	**350,17**	9.900,34
	0.7	80	**7,90**	8,17	9,21	7,92	**87**	22	0	77	33.341,96	1.162,54	**330,96**	8.218,78
	0.9	60	**18,42**	19,33	20,86	18,68	**27**	7	1	15	57.865,38	4.361,38	**753,26**	13.554,83
$N = 400$	0.1	150	**2,00**	**2,00**	**2,00**	**2,00**	**1.500**	**1.500**	**1.500**	**1.500**	45.452,92	339,73	**187,20**	4.162,86
	0.3	120	3,56	3,63	3,78	**3,43**	530	440	265	**689**	83.650,10	937,61	**376,62**	14.892,28
	0.5	80	**5,04**	5,25	5,98	5,06	**694**	535	78	678	69.485,60	1.345,75	**439,13**	15.958,00
	0.7	60	**8,43**	8,82	9,91	8,44	**344**	139	2	337	86.911,24	2.320,43	**610,81**	20.255,88
	0.9	10	**19,89**	20,89	22,82	20,17	**17**	1	0	12	33.184,40	1.719,24	**318,97**	7.392,72
$N = 500$	0.1	80	**2,00**	**2,00**	**2,00**	**2,00**	**800**	**800**	**800**	**800**	58.570,17	386,56	**243,18**	4.565,59
	0.3	80	3,87	**3,84**	3,98	3,85	108	**128**	20	118	156.133,84	1.398,77	**516,97**	30.166,21
	0.5	50	5,42	5,65	6,25	**5,33**	290	173	8	**335**	121.914,80	1.783,14	**543,18**	25.694,94
	0.7	10	8,92	9,21	10,43	**8,91**	8	1	0	**10**	39.283,12	855,20	**199,07**	8.724,19
	0.9	5	**21,28**	22,42	24,46	21,44	4	0	0	**5**	44.116,02	1.957,67	**319,55**	9.405,23
	Gesamt	2.065	**7,48**	7,79	8,48	7,53	**13.904**	12.330	9.644	13.775	975.755,63	22.576,16	**6.240,22**	191.005,71

Tabelle A.3: Evaluierung der Konstruktionsheuristiken im Zuge des ersten Experiments in RN2, siehe *Kapitel 4.3*

	EN2		∅-Reg				#-Best				Σ-LZ(s)			
	$d_{\max}$	Iter	CG	IS	SIS	CL	CG	IS	SIS	CL	CG	IS	SIS	CL
$N = 200$	20	300	13,84	**13,13**	15,12	13,52	35	**175**	8	18	36.571,42	2.308,18	**767,35**	3.662,18
	30	300	13,99	**12,95**	15,33	13,44	21	**586**	23	141	38.833,24	2.282,26	**767,76**	4.298,88
	40	300	13,49	**12,46**	14,44	13,00	2	**261**	4	48	38.031,10	2.225,57	**742,81**	3.868,12
	50	300	13,10	**12,34**	14,66	12,73	28	**269**	0	81	34.428,49	2.169,99	**736,99**	3.884,14
$N = 300$	20	100	14,78	**13,26**	15,86	13,92	4	**175**	3	11	71.465,92	2.628,21	**815,85**	6.008,04
	30	100	14,52	**12,88**	15,27	13,44	1	**63**	1	9	71.163,44	2.592,30	**807,90**	5.746,21
	40	100	14,10	**12,67**	15,26	13,38	7	**136**	1	3	67.374,54	2.636,88	**780,21**	5.693,47
	50	100	13,90	**12,52**	14,74	13,07	0	**147**	2	13	67.502,52	2.429,09	**781,19**	5.716,71
$N = 400$	20	15	15,33	**13,71**	16,26	14,18	0	**9**	0	2	39.646,27	1.186,13	**315,49**	2.954,34
	30	15	14,68	**13,33**	15,69	13,97	0	**8**	0	2	34.322,31	1.001,00	**289,56**	2.692,74
	40	15	14,41	**12,89**	15,71	13,49	0	**4**	0	1	33.378,97	910,24	**283,09**	2.689,22
	50	15	14,34	**12,62**	15,29	13,27	0	**20**	0	1	33.692,15	894,31	**278,74**	2.741,21
$N = 500$	20	3	15,73	**13,67**	16,27	14,17	0	**1**	**1**	0	19.427,49	405,47	**115,29**	1.364,09
	30	3	15,20	**13,27**	15,87	13,57	0	0	0	**3**	18.447,49	355,45	**114,61**	1.216,67
	40	3	15,03	**12,80**	15,87	13,60	0	**3**	0	0	24.790,88	439,46	**120,37**	1.167,02
	50	3	14,77	**12,73**	15,57	13,80	0	**1**	0	0	17.873,19	353,71	**111,57**	1.457,72
	Gesamt	1.672	14,45	**12,95**	15,45	13,53	98	**1.858**	43	333	646.949,40	24.818,23	**7.828,78**	55.160,74

Tabelle A.4: Evaluierung der Konstruktionsheuristiken im Zuge des ersten Experiments in EN2, siehe *Kapitel 4.3*

RN1		$N = 40$					$N = 60$					$N = 80$					$N = 100$					Gesamt
		0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	
Iterationen		100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	2.000
∅-Reg	CG-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,22	**2,00**	**2,70**	**3,90**	5,46	11,98	4,56
	IS-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	4,88	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	**11,06**	**2,00**	**2,70**	3,94	**5,20**	**11,80**	**4,53**
	SIS-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	4,82	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,10	**2,00**	**2,70**	3,92	5,34	11,90	4,54
	CL-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	4,84	**10,20**	**1,90**	2,32	**3,20**	**5,00**	11,14	**2,00**	**2,70**	**3,90**	5,22	**11,80**	4,54
	Hyb-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	4,84	10,22	**1,90**	2,34	**3,20**	**5,00**	**11,06**	**2,00**	**2,70**	**3,90**	**5,20**	11,82	**4,53**
	RVNS	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	4,88	10,34	**1,90**	2,34	**3,20**	**5,00**	11,20	**2,00**	**2,70**	3,92	5,54	11,96	4,57
	VNS1	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	2,34	**3,20**	**5,00**	11,08	**2,00**	**2,70**	**3,90**	**5,20**	**11,80**	**4,53**
	VNS2	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	4,82	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,16	**2,00**	**2,70**	3,92	**5,20**	**11,80**	4,54
#-Best	CG-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	39	**50**	**50**	**50**	37	41	967
	IS-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	46	**50**	**50**	**50**	**50**	**50**	**47**	**50**	**50**	48	**50**	**50**	991
	SIS-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**	**50**	**50**	**50**	**50**	45	**50**	**50**	49	43	45	981
	CL-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	48	**50**	**50**	49	**50**	**50**	43	**50**	**50**	**50**	49	**50**	989
	Hyb-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	48	49	**50**	48	**50**	**50**	**47**	**50**	**50**	**50**	**50**	49	991
	RVNS	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	46	43	**50**	48	**50**	**50**	40	**50**	**50**	49	33	42	951
	VNS1	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	48	**50**	**50**	46	**50**	**50**	**50**	**50**	**50**	**994**
	VNS2	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**	**50**	**50**	**50**	**50**	42	**50**	**50**	49	**50**	**50**	990
∅-tBest(ms)	CG-GRASP	**1**	**1**	**1**	8	23	18	12	14	87	422	18	81	163	668	2.548	134	158	428	4.024	8.270	854,0
	IS-GRASP	**1**	**1**	2	8	39	2	3	7	132	235	4	116	55	77	1.180	13	22	87	677	2.730	269,6
	SIS-GRASP	**1**	**1**	**1**	7	**12**	**1**	**2**	**6**	**65**	**72**	4	**31**	**24**	**68**	**709**	19	**12**	129	**606**	**1.118**	**144,4**
	CL-GRASP	2	2	3	13	29	12	5	18	158	340	14	423	280	158	1.635	76	85	275	1.689	3.666	444,2
	Hyb-GRASP	**1**	**1**	2	**6**	19	5	3	9	78	220	7	232	79	72	967	29	31	**58**	768	1.410	199,9
	RVNS	**1**	2	3	29	71	7	**2**	14	74	729	18	182	179	472	1.655	40	40	182	1.814	7.799	665,7
	VNS1	**1**	**1**	**1**	9	51	**1**	3	18	177	493	2	302	245	312	3.849	6	32	197	3.238	8.729	883,4
	VNS2	**1**	**1**	2	13	39	**1**	3	17	184	393	**1**	142	95	167	1.821	**4**	18	231	2.067	6.066	563,3
∅-LZ pro Inst.(s)	CG-GRASP	3,2	3,5	7,1	12,0	37,7	83,3	31,7	35,2	72,2	220,0	79,0	89,8	119,1	212,2	705,2	657,0	298,9	316,8	630,6	1.833,4	272,4
	IS-GRASP	1,9	3,9	7,9	16,2	45,1	11,7	12,2	24,4	54,7	163,5	18,3	76,1	69,4	127,6	416,9	**54,8**	76,3	122,0	256,6	946,8	125,3
	SIS-GRASP	**1,1**	**2,4**	**6,0**	**11,7**	**18,9**	**7,1**	**8,1**	**20,2**	**54,0**	**59,7**	**17,7**	**68,1**	**36,0**	**65,3**	**169,9**	93,6	**52,4**	**83,4**	**199,7**	**334,8**	**65,5**
	CL-GRASP	7,5	7,2	16,7	40,7	67,1	60,6	24,4	60,6	123,3	294,9	75,3	310,7	255,2	371,1	901,9	361,8	364,9	500,3	835,2	2.168,5	342,4
	Hyb-GRASP	3,5	4,9	9,7	20,1	47,4	25,8	15,7	32,4	65,6	175,4	34,5	124,0	126,2	166,5	472,9	143,5	138,2	189,7	342,1	1.034,4	158,6
	RVNS	4,2	5,9	10,6	19,1	35,9	27,3	18,7	36,8	66,4	139,0	38,7	141,3	118,5	173,2	390,4	161,6	167,5	233,2	416,1	913,9	155,9
	VNS1	9,9	14,0	26,4	48,1	112,5	71,6	44,7	96,2	198,3	474,6	95,8	383,6	324,3	519,8	1.368,3	408,0	438,9	630,1	1.231,8	3.187,3	484,2
	VNS2	6,1	10,5	17,7	32,8	72,0	36,0	31,2	53,4	110,9	272,8	51,4	138,1	155,3	238,1	715,3	173,7	185,9	263,1	643,0	1.618,8	241,3

Tabelle A.5: Evaluierung der Metaheuristiken im Zuge des ersten Experiments in RN1, siehe *Kapitel 4.3*

	EN1	$N = 40$				$N = 60$				$N = 80$				$N = 100$				Gesamt
		20	30	40	50	20	30	40	50	20	30	40	50	20	30	40	50	
	Iterationen	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	1.600
∅-Reg	CG-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	12,82	11,92	11,62	12,02	**12,30**	11,64	11,80	12,32	11,74	11,78	11,18	11,98
	IS-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	12,08	**12,30**	**11,60**	**11,70**	**12,20**	**11,70**	11,72	**11,04**	**11,95**
	SIS-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	11,82	**12,80**	12,08	**11,60**	**12,00**	12,34	**11,60**	11,78	12,34	11,76	11,80	11,24	12,00
	CL-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	12,30	**11,70**	11,76	11,10	11,96
	Hyb-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	11,72	**12,20**	**11,70**	**11,70**	11,10	**11,95**
	RVNS	**12,50**	**12,50**	**11,50**	**12,30**	11,86	**12,80**	11,94	**11,60**	12,02	**12,30**	11,62	11,74	12,26	11,82	11,86	11,20	11,99
	VNS1	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	**12,20**	**11,70**	11,72	11,08	**11,95**
	VNS2	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	**12,20**	11,72	11,74	11,08	**11,95**
#-Best	CG-GRASP	**50**	**50**	**50**	**50**	**50**	49	49	49	49	**50**	48	45	44	48	46	41	768
	IS-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	46	**50**	**50**	**50**	**50**	**50**	49	**48**	793
	SIS-GRASP	**50**	**50**	**50**	**50**	49	**50**	41	**50**	**50**	48	**50**	46	43	47	45	38	757
	CL-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	45	**50**	47	45	787
	Hyb-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**	**50**	**50**	45	794
	RVNS	**50**	**50**	**50**	**50**	47	**50**	48	**50**	49	**50**	49	48	47	44	42	40	764
	VNS1	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	46	**795**
	VNS2	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	48	46	793
∅-*t*Best(ms)	CG-GRASP	13	13	6	6	226	383	519	330	1.251	542	1.482	746	6.659	4.225	6.753	6.229	1.836,4
	IS-GRASP	6	6	6	4	40	43	123	44	380	**143**	233	**190**	**382**	585	1.760	1.247	324,5
	SIS-GRASP	**4**	**4**	**3**	3	66	45	120	102	**127**	178	168	228	1.958	601	1.006	951	347,8
	CL-GRASP	6	8	5	5	79	71	217	90	457	342	445	450	465	703	2.558	1.310	450,7
	Hyb-GRASP	5	6	5	4	**35**	**35**	**76**	**42**	187	161	**140**	251	426	**351**	**569**	**686**	**186,2**
	RVNS	**4**	5	**3**	**1**	114	78	281	127	457	367	357	476	1.989	1.202	2.985	2.192	664,9
	VNS1	**4**	7	**3**	3	65	102	368	147	651	277	406	486	2.143	1.685	6.475	4.865	1.105,4
	VNS2	10	10	4	4	126	109	199	236	498	272	318	851	2.090	2.889	4.188	3.886	980,6
∅-LZ pro Inst.(s)	CG-GRASP	24,7	25,4	16,0	23,2	161,5	167,1	183,3	274,7	647,3	612,8	555,5	581,9	1.591,2	1.570,7	1.884,6	1.468,7	611,8
	IS-GRASP	22,3	22,9	19,4	18,7	129,6	117,2	129,4	140,5	344,3	353,2	335,2	350,1	761,3	737,9	901,9	653,2	314,8
	SIS-GRASP	**12,1**	**12,2**	**12,7**	**11,0**	**63,0**	**57,5**	**71,5**	**80,0**	**142,7**	**130,8**	**138,0**	**133,2**	**330,8**	**273,3**	**271,7**	**244,4**	**124,0**
	CL-GRASP	23,6	26,1	22,2	26,2	154,0	134,5	159,4	203,8	395,7	465,8	408,2	421,9	1.132,9	1.054,8	1.274,3	1.034,4	433,6
	Hyb-GRASP	22,7	22,8	20,0	19,9	134,6	116,2	128,0	151,0	351,4	368,8	341,4	343,3	787,0	790,1	900,2	724,1	326,3
	RVNS	16,4	18,6	15,5	16,1	95,8	83,0	98,0	107,0	249,6	248,7	252,3	246,0	574,1	532,4	638,7	509,9	231,4
	VNS1	62,5	69,9	48,9	58,5	322,4	312,9	351,7	439,7	897,7	938,1	859,3	890,6	2.003,0	1.919,9	2.311,5	1.781,4	829,2
	VNS2	48,9	52,6	37,7	43,8	233,8	221,1	246,3	307,0	616,1	614,9	570,4	614,3	1.294,5	1.212,3	1.404,6	1.105,8	539,0

Tabelle A.6: Evaluierung der Metaheuristiken im Zuge des ersten Experiments in EN1, siehe *Kapitel 4.3*

RN2		$N = 200$					$N = 300$					$N = 400$					$N = 500$					Gesamt
		0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	
Iterationen		50	40	35	30	20	20	15	8	5	5	20	15	8	5	5	20	15	8	5	2	331
$\varnothing$-Reg	CG-GRASP	**2,00**	**3,00**	4,04	6,98	16,20	**2,00**	**3,00**	5,00	8,28	19,70	**2,00**	3,16	5,16	8,98	21,54	**2,00**	3,80	5,68	9,34	23,82	7,78
	IS-GRASP	**2,00**	**3,00**	**4,00**	**6,76**	15,14	**2,00**	**3,00**	4,90	7,50	17,82	**2,00**	**3,00**	5,00	**7,98**	19,32	**2,00**	3,16	**5,00**	8,60	21,24	7,17
	SIS-GRASP	**2,00**	**3,00**	**4,00**	6,98	15,56	**2,00**	**3,00**	5,00	8,10	18,78	**2,00**	**3,00**	4,98	8,82	20,60	**2,00**	3,36	5,46	9,22	22,78	7,53
	CL-GRASP	**2,00**	**3,00**	**4,00**	**6,76**	15,08	**2,00**	**3,00**	4,84	7,42	17,58	**2,00**	**3,00**	**4,96**	8,00	**18,96**	**2,00**	3,06	**5,00**	8,36	**20,70**	**7,09**
	Hyb-GRASP	**2,00**	**3,00**	**4,00**	6,88	**15,06**	**2,00**	**3,00**	**4,82**	**7,40**	**17,52**	**2,00**	**3,00**	5,00	8,00	19,00	**2,00**	**3,04**	**5,00**	**8,28**	20,72	**7,09**
	RVNS	**2,00**	**3,00**	**4,00**	7,00	15,94	**2,00**	**3,00**	4,94	7,86	18,52	**2,00**	3,02	4,98	8,32	19,98	**2,00**	3,34	5,08	8,82	21,34	7,36
	VNS1	**2,00**	**3,00**	**4,00**	6,90	15,36	**2,00**	**3,00**	4,86	7,80	18,22	**2,00**	**3,00**	4,98	8,12	19,76	**2,00**	3,10	**5,00**	8,78	21,70	7,28
	VNS2	**2,00**	**3,00**	**4,00**	6,96	15,76	**2,00**	**3,00**	4,92	7,96	18,42	**2,00**	**3,00**	5,00	8,68	20,24	**2,00**	3,14	5,02	8,96	22,32	7,42
#-Best	CG-GRASP	**50**	**50**	48	21	1	**50**	**50**	20	0	0	**50**	42	39	2	0	**50**	10	16	0	0	499
	IS-GRASP	**50**	**50**	**50**	**32**	31	**50**	**50**	25	25	7	**50**	**50**	45	**46**	19	**50**	42	**50**	20	4	746
	SIS-GRASP	**50**	**50**	**50**	21	12	**50**	**50**	20	0	0	**50**	**50**	46	9	1	**50**	32	27	0	0	568
	CL-GRASP	**50**	**50**	**50**	**32**	31	**50**	**50**	28	29	14	**50**	**50**	**47**	45	**27**	**50**	47	**50**	32	**12**	794
	Hyb-GRASP	**50**	**50**	**50**	26	**32**	**50**	**50**	**29**	**30**	**20**	**50**	**50**	45	45	**27**	**50**	**48**	**50**	**36**	11	**799**
	RVNS	**50**	**50**	**50**	20	2	**50**	**50**	23	7	1	**50**	49	46	31	2	**50**	33	46	9	4	623
	VNS1	**50**	**50**	**50**	25	19	**50**	**50**	27	10	6	**50**	**50**	46	39	3	**50**	45	**50**	11	0	681
	VNS2	**50**	**50**	**50**	22	6	**50**	**50**	24	2	3	**50**	**50**	45	14	0	**50**	43	49	2	0	610
$\varnothing$-tBest(ms)	CG-GRASP	567	997	13.277	11.379	325.031	1.985	9.962	10.057	-	-	6.100	121.545	99.522	883.064	-	14.749	973.986	707.571	-	-	211.986,1
	IS-GRASP	55	121	**368**	6.295	29.434	161	415	4.169	**9.167**	131.715	384	1.965	**2.145**	**9.989**	148.565	676	13.126	**7.085**	86.154	413.155	43.257,2
	SIS-GRASP	37	77	757	3.582	**16.654**	118	**282**	**587**	-	-	272	**1.108**	2.180	23.034	**45.132**	548	**5.105**	14.016	-	-	**7.093,1**
	CL-GRASP	295	859	2.177	19.778	87.589	822	4.069	24.098	53.721	383.185	1.990	16.758	38.875	61.682	394.875	3.863	147.255	69.461	501.115	1.581.490	169.697,9
	Hyb-GRASP	119	276	696	2.300	23.422	348	1.131	5.626	16.283	75.639	858	4.921	5.532	14.428	126.653	1.694	30.609	15.568	**83.083**	454.385	43.178,6
	RVNS	305	710	4.098	**1.064**	45.750	1.086	711	9.141	40.708	**22.591**	2.775	14.324	7.164	35.141	1.525.719	5.707	132.996	41.442	280.537	**188.105**	118.003,7
	VNS1	31	75	3.605	25.355	240.781	99	1.925	44.981	177.452	805.716	226	27.771	35.892	158.920	4.088.947	483	177.830	148.000	1.315.036	-	381.743,4
	VNS2	**18**	**29**	1.377	8.612	139.551	**60**	586	6.644	89.813	710.636	**125**	3.559	6.955	113.428	-	**304**	34.530	26.625	331.986	-	81.935,4
$\varnothing$-LZ pro Inst.(s)	CG-GRASP	1.398,3	2.022,1	2.116,2	3.387,5	7.915,6	1.951,1	4.994,6	2.918,9	3.194,5	11.510,0	6.141,5	12.007,6	10.189,3	11.069,5	39.050,4	14.987,3	29.201,0	26.044,9	29.910,2	48.328,3	13.416,9
	IS-GRASP	128,9	228,5	388,8	802,6	2.423,7	157,6	312,7	331,4	493,1	2.650,0	378,3	801,8	854,4	1.403,3	6.861,9	688,7	1.675,8	1.921,0	2.868,4	6.888,0	1.612,9
	SIS-GRASP	**90,5**	**147,4**	**217,6**	**558,9**	**625,6**	**118,0**	**210,3**	**236,2**	**190,8**	**591,1**	**277,8**	**473,6**	**432,4**	**606,1**	**1.037,8**	**543,4**	**929,8**	**801,1**	**990,7**	**977,5**	**502,8**
	CL-GRASP	702,0	1.772,3	2.669,7	4.203,1	7.416,6	820,6	2.969,7	3.370,7	3.814,5	10.046,0	2.045,2	9.824,0	10.191,2	12.956,5	33.528,6	4.228,6	31.680,0	27.602,8	33.287,7	35.544,3	11.933,7
	Hyb-GRASP	293,7	545,5	778,8	1.252,7	2.820,9	352,8	827,7	812,6	914,6	3.181,7	865,7	2.523,3	2.251,7	2.855,1	8.803,3	1.700,0	7.060,7	5.821,0	6.819,0	8.969,4	2.972,5
	RVNS	328,1	673,5	1.087,9	1.734,3	2.935,7	404,0	1.219,3	1.291,1	1.775,2	4.437,1	967,2	4.000,4	3.749,1	5.452,3	11.670,2	1.871,1	11.053,4	10.642,8	12.556,1	11.889,6	4.486,9
	VNS1	821,9	1.832,8	3.142,5	5.377,8	11.122,5	961,8	3.515,2	3.510,4	4.702,0	16.142,7	2.423,1	11.703,8	11.515,4	16.041,3	53.594,1	4.597,7	31.832,7	32.251,8	38.201,3	59.325,4	15.630,8
	VNS2	381,7	526,7	800,4	1.644,9	4.841,5	457,8	721,1	697,8	1.127,0	6.512,0	1.108,4	1.777,7	1.820,6	3.138,0	20.210,2	2.013,6	3.659,3	3.926,5	7.115,2	24.300,5	4.339,0

Tabelle A.7: Evaluierung der Metaheuristiken im Zuge des ersten Experiments in RN2, siehe *Kapitel 4.3*

	EN2	$N = 200$				$N = 300$				$N = 400$				$N = 500$				Gesamt
		20	30	40	50	20	30	40	50	20	30	40	50	20	30	40	50	
Iterationen		20	20	20	20	10	10	10	10	5	5	5	5	2	2	2	2	148
∅-Reg	CG-GRASP	12,22	12,08	11,48	11,40	12,60	12,12	12,12	11,56	12,96	12,68	12,10	12,16	13,26	12,90	12,52	12,56	12,30
	IS-GRASP	11,74	11,64	11,08	**11,02**	11,94	**11,38**	11,20	11,12	**12,06**	11,88	**11,40**	11,22	12,42	11,64	**11,38**	11,72	**11,55**
	SIS-GRASP	12,04	12,14	11,36	11,56	12,62	12,08	11,94	11,78	13,16	12,86	12,38	12,26	13,66	13,08	13,00	12,80	12,42
	CL-GRASP	11,82	11,86	11,14	**11,02**	12,16	11,76	11,62	11,22	12,42	12,22	11,74	11,44	12,86	12,20	11,96	12,16	11,85
	Hyb-GRASP	**11,68**	**11,60**	**11,06**	11,94	**11,88**	**11,38**	**11,16**	**11,06**	12,16	**11,82**	11,46	**11,12**	**12,18**	**11,60**	11,48	**11,66**	11,58
	RVNS	12,08	11,96	11,28	11,32	12,24	12,06	11,82	11,40	12,88	12,64	12,22	11,96	13,50	12,84	12,52	12,70	12,21
	VNS1	11,74	11,74	11,10	11,08	11,98	11,45	11,40	11,14	12,34	11,90	11,58	11,48	12,50	11,84	11,64	11,84	11,67
	VNS2	11,76	11,78	11,10	**11,02**	11,92	11,56	11,40	11,28	12,32	12,00	11,72	11,58	12,52	11,96	11,58	11,82	11,71
#-Best	CG-GRASP	19	22	21	25	11	7	9	21	8	5	10	1	0	0	3	1	163
	IS-GRASP	43	43	41	44	43	**41**	40	39	**42**	26	**35**	39	24	**22**	**31**	14	567
	SIS-GRASP	28	18	27	18	15	8	11	12	2	3	2	5	2	0	0	1	152
	CL-GRASP	39	32	38	44	32	22	19	34	25	16	20	29	11	6	10	5	382
	Hyb-GRASP	**46**	**45**	**42**	**48**	**46**	**41**	**42**	**42**	38	**29**	32	**44**	**36**	21	26	**18**	**596**
	RVNS	26	27	31	29	28	11	13	29	11	4	7	14	7	2	6	5	250
	VNS1	43	38	40	41	42	37	30	38	30	25	26	26	21	13	19	13	482
	VNS2	42	36	40	44	44	32	30	32	29	22	20	21	23	11	23	15	464
∅-tBest(s)	CG-GRASP	104,13	41,11	71,81	48,67	204,82	357,21	222,52	182,15	652,55	627,31	552,13	701,26	-	-	950,48	1.457,58	440,98
	IS-GRASP	4,54	6,84	6,63	4,47	**9,39**	**16,00**	16,29	9,10	**20,59**	31,99	**24,12**	**20,77**	55,81	**43,98**	**39,91**	79,95	24,40
	SIS-GRASP	**3,76**	**2,15**	**4,07**	**3,72**	13,06	17,93	**5,92**	**7,16**	57,00	**27,14**	32,78	38,93	**47,75**	-	-	**43,67**	**21,79**
	CL-GRASP	10,87	17,17	14,06	10,64	67,43	71,35	73,15	40,75	121,25	184,96	172,71	111,23	315,56	373,33	296,65	490,97	148,25
	Hyb-GRASP	5,77	6,76	6,98	3,88	11,11	20,23	19,95	14,92	43,59	43,62	39,45	31,78	49,37	61,19	58,91	68,45	30,37
	RVNS	15,04	12,01	9,25	10,34	41,12	48,30	26,70	26,96	97,08	329,39	225,75	114,90	108,22	80,30	115,55	123,59	86,53
	VNS1	23,23	31,38	27,57	20,13	65,04	83,04	123,12	78,98	230,16	211,71	198,25	253,20	466,88	447,72	242,14	561,58	191,51
	VNS2	15,26	14,37	10,15	11,92	27,98	50,90	64,19	34,15	96,92	86,90	108,25	102,94	160,01	150,84	134,49	208,91	79,89
∅-LZ pro Inst.(s)	CG-GRASP	4.273,1	4.396,7	4.237,5	3.938,3	11.221,5	10.630,5	10.554,0	9.882,8	17.723,8	17.281,8	15.922,4	16.880,0	17.071,5	16.669,7	16.237,9	16.156,8	12.067,4
	IS-GRASP	1.134,4	1.178,4	1.112,9	1.032,9	1.936,1	1.862,7	1.869,7	1.770,5	2.606,2	2.402,2	2.252,6	2.173,7	2.075,4	1.945,5	1.819,1	1.921,1	1.818,3
	SIS-GRASP	**339,1**	**365,4**	**331,5**	**328,6**	**607,2**	**552,1**	**561,5**	**547,2**	**739,0**	**780,7**	**681,1**	**707,4**	**626,2**	**606,0**	**642,3**	**585,6**	**562,6**
	CL-GRASP	2.144,9	2.384,1	2.289,3	2.106,2	5.070,8	5.056,3	4.832,7	4.706,5	7.776,7	7.482,1	7.390,3	7.453,6	7.563,5	7.445,3	6.833,9	7.552,6	5.505,6
	Hyb-GRASP	1.285,9	1.363,1	1.295,4	1.205,8	2.426,7	2.352,3	2.297,4	2.202,6	3.335,0	3.137,3	2.898,3	2.950,0	2.795,1	2.534,5	2.474,5	2.744,5	2.331,2
	RVNS	1.058,6	1.157,1	1.066,3	1.055,6	2.485,8	2.301,9	2.144,3	2.184,8	3.797,4	3.627,0	3.158,0	3.705,6	3.694,1	4.281,3	3.966,0	3.553,6	2.702,3
	VNS1	3.624,5	3.796,7	3.482,1	3.445,1	7.858,7	7.408,6	7.269,4	6.988,1	12.559,5	12.154,9	11.400,1	10.966,9	12.587,5	12.762,8	11.360,1	12.428,5	8.755,8
	VNS2	1.916,7	2.028,6	1.861,7	1.748,3	3.438,3	3.337,1	3.290,3	3.062,8	4.966,2	4.439,2	4.416,5	4.216,6	4.682,7	4.638,0	4.409,9	4.663,1	3.569,8

Tabelle A.8: Evaluierung der Metaheuristiken im Zuge des ersten Experiments in EN2, siehe *Kapitel 4.3*

RN1		$N = 40$					$N = 60$					$N = 80$					$N = 100$					Gesamt
		0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	
Laufzeit pro Inst.(s)		1,2	2,0	4,0	6,0	12,0	8,0	12,0	16,0	40,0	80,0	20,0	30,0	40,0	80,0	160,0	40,0	80,0	120,0	160,0	600,0	1.511,2
∅-Iterationen	CG-GRASP	550,2	598,6	588,0	513,3	250,0	306,5	546,5	451,2	654,9	350,5	255,2	242,8	236,8	375,4	220,3	211,1	72,4	260,5	290,1	327,9	365,1
	IS-GRASP	16.871,3	533,3	541,5	418,5	293,0	9.831,3	**1.012,5**	**626,3**	792,5	486,9	**14.359,8**	862,4	589,3	635,3	383,3	1.086,2	**609,7**	868,9	634,3	645,4	2.604,1
	SIS-GRASP	**19.269,5**	**854,6**	**631,8**	**495,6**	**487,2**	**10.264,3**	605,9	502,7	**929,8**	**1.438,0**	14.074,2	**1.064,6**	**964,1**	**1.266,1**	**1.041,4**	**1.240,4**	567,6	**1.659,5**	**1.011,3**	**1.827,5**	**3.009,8**
	CL-GRASP	19.004,8	272,7	215,1	180,9	146,2	9.466,0	486,9	260,1	327,3	273,8	13.116,0	268,9	163,7	217,2	188,2	249,3	103,7	201,1	194,7	282,0	2.280,9
	Hyb-GRASP	19.021,0	413,8	424,6	348,6	274,7	9.737,5	752,8	472,5	641,9	453,5	13.404,0	529,2	381,0	483,5	344,1	546,6	289,5	543,4	481,3	595,6	2.506,9
	RVNS	6.498,2	344,2	351,2	367,7	347,2	4.150,6	624,2	432,1	622,5	590,0	6.483,3	455,6	354,3	469,4	431,1	487,6	241,0	465,5	432,5	677,0	1.241,3
	VNS1	3.238,2	143,8	136,0	136,7	98,7	2.025,4	253,6	163,4	221,0	166,4	3.212,2	178,3	127,3	160,8	118,2	194,5	85,5	164,1	145,7	192,1	558,1
	VNS2	3.306,5	192,1	232,8	220,9	142,8	2.164,6	380,8	292,2	428,5	287,4	3.354,5	333,3	285,3	350,2	221,5	375,7	256,4	413,4	343,4	380,9	698,2
∅-Regeneratoren	CG-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,10	**2,00**	**2,70**	**3,90**	5,34	11,86	4,54
	IS-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	**11,00**	**2,00**	**2,70**	**3,90**	**5,20**	**11,80**	**4,53**
	SIS-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	**11,00**	**2,00**	**2,70**	**3,90**	**5,20**	**11,80**	**4,53**
	CL-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,08	**2,00**	**2,70**	**3,90**	**5,20**	**11,80**	**4,53**
	Hyb-GRASP	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	**11,00**	**2,00**	**2,70**	**3,90**	**5,20**	**11,80**	**4,53**
	RVNS	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	4,82	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,14	**2,00**	**2,70**	**3,90**	5,24	**11,80**	4,54
	VNS1	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,14	**2,00**	**2,70**	**3,90**	**5,20**	**11,80**	**4,53**
	VNS2	**1,40**	**2,00**	**3,00**	**4,30**	**8,90**	**1,90**	**2,00**	**3,00**	**4,80**	**10,20**	**1,90**	**2,30**	**3,20**	**5,00**	11,12	**2,00**	**2,70**	**3,90**	**5,20**	**11,80**	**4,53**
#-Best	CG-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	45	**50**	**50**	**50**	43	47	985
	IS-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**1000**
	SIS-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**1000**
	CL-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	46	**50**	**50**	**50**	**50**	**50**	996
	Hyb-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	1000
	RVNS	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**	**50**	**50**	**50**	**50**	43	**50**	**50**	**50**	48	**50**	990
	VNS1	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	43	**50**	**50**	**50**	**50**	**50**	993
	VNS2	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	44	**50**	**50**	**50**	**50**	**50**	994
∅-$t_{\text{Best}}(ms)$	CG-GRASP	**1**	**1**	2	9	28	6	10	15	86	486	17	135	297	819	3.751	40	550	440	5.949	23.520	1.808,1
	IS-GRASP	**1**	**1**	2	8	23	2	3	**6**	101	380	4	125	78	**84**	1.808	8	36	302	765	2.820	327,9
	SIS-GRASP	**1**	**1**	**1**	17	**18**	2	4	11	**28**	**62**	3	**19**	**38**	103	**1.020**	7	32	**95**	2.925	**1.757**	307,2
	CL-GRASP	2	2	4	10	51	7	6	14	185	276	16	125	258	200	2.861	34	207	348	1.884	2.989	474,0
	Hyb-GRASP	**1**	**1**	2	**6**	19	3	4	8	163	273	7	138	105	129	1.299	15	62	100	**684**	1.838	**242,9**
	RVNS	**1**	2	3	24	73	7	**1**	12	341	980	18	199	234	417	3.433	40	92	1.233	4.074	15.561	1.337,3
	VNS1	**1**	**1**	**1**	9	42	**1**	5	20	178	595	2	326	230	273	2.610	6	111	576	2.982	8.473	822,1
	VNS2	**1**	**1**	**1**	10	49	**1**	2	15	109	362	**1**	101	96	195	4.269	**4**	**30**	207	2.270	6.619	717,2

Tabelle A.9: Evaluierung der Metaheuristiken im Zuge des zweiten Experiments in RN1, siehe *Kapitel 4.4*

	EN1	$N = 40$				$N = 60$				$N = 80$				$N = 100$				Gesamt
		20	30	40	50	20	30	40	50	20	30	40	50	20	30	40	50	
Laufzeit pro Inst.(s)		12,0	12,0	12,0	12,0	80,0	80,0	80,0	80,0	160,0	160,0	160,0	160,0	360,0	360,0	360,0	360,0	2.448,0
$\varnothing$-Iterationen	CG-GRASP	**47.198,3**	2.616,4	4.139,3	15.533,0	431,2	1.146,7	553,3	388,6	262,6	279,6	284,4	273,0	231,3	233,3	193,5	254,5	4.626,2
	IS-GRASP	4.381,4	1.282,7	1.661,9	2.535,3	642,7	1.213,5	717,0	500,0	480,2	472,3	490,7	463,3	482,5	495,4	411,8	552,9	1.049,0
	SIS-GRASP	5.357,8	2.037,6	2.418,5	3.678,1	1.151,3	1.906,1	1.195,1	609,5	1.186,7	1.128,6	1.061,3	1.234,8	1.103,5	**1.356,1**	**1.353,6**	1.526,2	1.769,0
	CL-GRASP	5.191,7	1.060,5	1.437,9	2.496,2	511,4	956,3	535,7	404,5	425,5	356,0	414,3	380,4	343,4	356,2	282,3	372,5	970,3
	Hyb-GRASP	4.262,4	1.181,7	1.620,9	2.449,2	604,7	1.185,2	720,5	479,3	470,0	459,2	485,4	468,1	470,3	466,3	399,0	509,0	1.014,5
	RVNS	23.914,2	**23.748,3**	**37.570,7**	**32.398,2**	**54.654,7**	**55.702,1**	**93.946,7**	**33.695,3**	**66.865,2**	**69.050,0**	**43.083,0**	**13.928,6**	**154.314,7**	695,3	580,2	**41.336,0**	**46.592,7**
	VNS1	11.235,7	10.751,6	17.772,3	15.024,5	25.249,8	25.978,6	45.182,7	16.094,6	32.520,8	33.622,2	20.377,7	6.369,2	73.987,3	191,5	156,9	19.730,1	22.140,4
	VNS2	11.857,8	11.799,7	19.137,2	16.415,7	25.959,2	27.676,9	46.367,0	17.517,7	34.958,9	34.026,7	21.930,1	6.453,4	78.953,5	303,5	260,8	20.417,2	23.377,2
$\varnothing$-Regeneratoren	CG-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	11,76	12,26	**11,70**	11,78	11,14	11,97
	IS-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	**12,20**	**11,70**	**11,70**	**11,00**	**11,94**
	SIS-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	12,22	**11,70**	**11,70**	11,08	11,95
	CL-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	12,22	**11,70**	11,74	11,08	11,95
	Hyb-GRASP	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	**12,20**	**11,70**	**11,70**	**11,00**	**11,94**
	RVNS	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	**12,20**	11,72	**11,70**	11,14	11,95
	VNS1	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	**12,20**	**11,70**	11,72	11,10	11,95
	VNS2	**12,50**	**12,50**	**11,50**	**12,30**	**11,80**	**12,80**	**11,90**	**11,60**	**12,00**	**12,30**	**11,60**	**11,70**	**12,20**	**11,70**	**11,70**	11,04	11,95
#-Best	CG-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	47	47	**50**	46	43	783
	IS-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**800**
	SIS-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**	**50**	46	795
	CL-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**	48	46	793
	Hyb-GRASP	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**800**
	RVNS	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**	43	792
	VNS1	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	45	794
	VNS2	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	48	798
$\varnothing$-$t_{\text{Best}}(ms)$	CG-GRASP	17	10	7	6	425	281	307	272	1.222	603	1.698	2.169	9.118	5.077	8.570	6.808	2.286,9
	IS-GRASP	6	5	6	4	43	44	**99**	62	1.433	**121**	222	**257**	437	784	2.303	7.087	807,1
	SIS-GRASP	6	**3**	**3**	**3**	106	48	435	162	**142**	421	172	310	4.201	1.376	2.594	**1.630**	725,8
	CL-GRASP	6	12	6	5	75	50	217	97	373	244	189	766	3.420	830	2.687	2.531	719,3
	Hyb-GRASP	**5**	7	5	4	**40**	**33**	100	**47**	220	168	**166**	295	**302**	**639**	**710**	2.534	**329,7**
	RVNS	38	37	48	20	455	389	600	337	817	549	854	944	1.959	4.993	5.123	4.424	1.349,2
	VNS1	20	42	36	9	314	318	437	162	467	442	447	1.506	3.451	1.366	4.582	1.748	959,2
	VNS2	42	86	62	75	484	797	500	311	488	461	475	1.369	1.607	3.527	4.665	4.689	1.227,4

Tabelle A.10: Evaluierung der Metaheuristiken im Zuge des zweiten Experiments in EN1, siehe *Kapitel 4.4*

RN2		$N = 200$					$N = 300$					$N = 400$					$N = 500$					Gesamt
		0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	0.1	0.3	0.5	0.7	0.9	
Laufzeit pro Inst.(s)		200,0	500,0	530,0	570,0	600,0	520,0	560,0	600,0	640,0	700,0	480,0	520,0	560,0	580,0	600,0	400,0	480,0	560,0	640,0	700,0	10.940,0
$\varnothing$-Iterationen	CG-GRASP	100,6	96,6	85,2	50,8	14,7	21,0	3,5	16,1	9,4	2,7	15,0	1,0	4,0	2,0	1,0	4,9	2,0	1,1	1,0	1,0	21,7
	IS-GRASP	824,0	912,4	467,6	221,2	52,4	359,4	126,2	145,5	62,1	13,0	266,4	37,0	53,6	20,3	3,9	112,9	43,3	23,3	11,0	1,8	187,9
	SIS-GRASP	**1.117,4**	**1.381,3**	**673,1**	**474,7**	**274,4**	**521,9**	**136,2**	**293,1**	**101,9**	**60,6**	**346,5**	**44,9**	**93,9**	**65,8**	**29,5**	**148,9**	**77,8**	**58,6**	**36,7**	**15,9**	**297,6**
	CL-GRASP	173,4	112,5	68,8	40,4	15,9	57,5	9,8	13,8	7,9	3,0	46,3	2,0	3,9	1,7	1,0	18,7	1,8	1,2	1,0	1,0	29,1
	Hyb-GRASP	395,3	374,9	237,6	139,6	45,0	143,3	38,9	59,1	33,6	10,6	112,0	8,5	19,4	9,8	2,9	46,1	10,0	7,3	4,3	1,1	85,0
	RVNS	359,5	305,6	173,9	89,0	42,7	128,3	26,6	40,2	20,4	8,8	104,2	5,4	10,6	6,0	2,9	42,6	7,0	4,2	3,0	2,9	69,2
	VNS1	142,6	110,0	58,6	28,4	10,0	49,4	8,7	13,3	5,8	1,7	40,4	1,2	3,0	1,4	1,0	16,9	1,7	1,0	1,0	1,0	24,9
	VNS2	286,7	380,4	230,1	107,5	26,4	110,8	48,1	66,0	28,6	4,9	90,0	14,5	24,5	9,3	1,2	40,1	20,7	11,2	3,9	1,0	75,3
$\varnothing$-Regeneratoren	CG-GRASP	**2,00**	**3,00**	**4,00**	6,98	16,48	**2,00**	3,02	4,98	8,12	19,62	**2,00**	3,92	5,44	9,26	22,48	**2,00**	3,90	5,98	10,04	24,18	7,97
	IS-GRASP	**2,00**	**3,00**	**4,00**	**6,62**	14,96	**2,00**	**3,00**	**4,58**	7,02	17,50	**2,00**	**3,00**	**4,90**	**7,98**	19,64	**2,00**	**3,00**	**5,00**	8,40	21,34	7,10
	SIS-GRASP	**2,00**	**3,00**	**4,00**	6,74	14,94	**2,00**	**3,00**	4,74	7,48	17,88	**2,00**	**3,00**	4,94	8,10	19,86	**2,00**	3,18	**5,00**	8,98	21,86	7,24
	CL-GRASP	**2,00**	**3,00**	**4,00**	6,78	14,96	**2,00**	**3,00**	4,76	7,06	17,64	**2,00**	3,12	4,98	8,06	19,70	**2,00**	3,62	5,06	8,78	**20,88**	7,17
	Hyb-GRASP	**2,00**	**3,00**	**4,00**	6,70	**14,90**	**2,00**	**3,00**	4,66	**7,00**	**17,40**	**2,00**	**3,00**	4,94	**7,98**	**19,22**	**2,00**	3,12	**5,00**	**8,20**	21,18	**7,07**
	RVNS	**2,00**	**3,00**	**4,00**	6,86	15,76	**2,00**	**3,00**	4,88	7,76	18,42	**2,00**	3,16	5,00	8,44	19,94	**2,00**	3,62	5,18	8,88	21,50	7,37
	VNS1	**2,00**	**3,00**	**4,00**	6,90	15,56	**2,00**	**3,00**	4,80	7,72	18,58	**2,00**	3,18	5,00	8,60	20,30	**2,00**	3,60	5,34	8,92	22,00	7,43
	VNS2	**2,00**	**3,00**	**4,00**	6,78	15,50	**2,00**	**3,00**	4,66	7,60	18,38	**2,00**	**3,00**	5,00	8,12	20,92	**2,00**	3,18	**5,00**	8,98	22,40	7,38
#-Best	CG-GRASP	**50**	**50**	**50**	21	2	**50**	49	20	1	0	**50**	4	24	0	0	**50**	5	2	0	0	428
	IS-GRASP	**50**	**50**	**50**	**39**	37	**50**	**50**	**38**	49	18	**50**	**50**	**50**	**46**	10	**50**	**50**	**50**	30	2	819
	SIS-GRASP	**50**	**50**	**50**	33	38	**50**	**50**	28	26	7	**50**	**50**	48	41	7	**50**	41	**50**	1	0	720
	CL-GRASP	**50**	**50**	**50**	31	37	**50**	**50**	33	47	13	**50**	44	46	42	12	**50**	19	47	11	**9**	741
	Hyb-GRASP	**50**	**50**	**50**	35	**40**	**50**	**50**	37	**50**	**26**	**50**	**50**	48	**46**	**26**	**50**	44	**50**	**40**	6	**848**
	RVNS	**50**	**50**	**50**	27	8	**50**	**50**	25	12	3	**50**	42	45	25	8	**50**	19	41	6	2	613
	VNS1	**50**	**50**	**50**	25	12	**50**	**50**	26	14	3	**50**	41	45	16	1	**50**	20	33	4	0	590
	VNS2	**50**	**50**	**50**	31	12	**50**	**50**	33	20	1	**50**	**50**	45	39	1	**50**	41	**50**	2	0	675
$\varnothing$-t_{Best}(ms)	CG-GRASP	399	987	17.518	39.648	106.449	2.770	40.452	**9.582**	73.758	-	6.113	411.550	101.625	-	-	14.634	278.871	323.049	-		95.160,3
	IS-GRASP	48	111	**390**	36.282	46.619	170	925	20.309	19.646	76.168	363	4.963	8.060	**20.710**	178.339	722	**14.293**	**6.905**	80.899	603.480	55.970,1
	SIS-GRASP	36	72	764	29.312	**34.191**	114	**795**	23.855	52.781	79.629	287	**4.265**	8.316	44.036	153.130	544	19.589	13.000	587.384	-	55.373,7
	CL-GRASP	255	853	2.612	**13.058**	59.229	1.014	11.099	19.903	46.686	174.816	1.887	44.714	27.777	65.341	262.174	4.301	95.538	72.014	396.122	1.265.847	128.262,0
	Hyb-GRASP	100	267	604	26.826	34.575	421	2.786	20.029	**18.892**	71.907	862	17.206	7.764	24.176	90.514	1.724	31.692	14.686	**66.908**	368.757	40.034,8
	RVNS	296	725	3.241	31.046	235.147	1.063	1.195	12.852	57.895	**22.807**	2.684	21.970	**770**	41.565	**73.927**	5.707	53.247	11.562	87.242	**188.105**	**31.129,1**
	VNS1	34	83	3.027	22.806	110.650	103	3.240	25.450	162.084	222.135	235	72.234	7.648	240.910	1.811.669	483	124.225	78.215	797.884	-	204.590,7
	VNS2	**20**	**22**	1.321	26.871	108.784	**56**	1.729	23.761	127.809	272.052	**128**	9.761	6.277	60.939	917.929	**304**	31.051	31.385	292.802	-	106.260,9

Tabelle A.11: Evaluierung der Metaheuristiken im Zuge des zweiten Experiments in RN2, siehe *Kapitel 4.4*

	EN2	$N = 200$				$N = 300$				$N = 400$				$N = 500$				Gesamt
		20	30	40	50	20	30	40	50	20	30	40	50	20	30	40	50	
Laufzeit pro Inst.(s)		600,0	600,0	600,0	600,0	800,0	800,0	800,0	800,0	480,0	530,0	560,0	610,0	700,0	720,0	770,0	850,0	10.820,0
$\varnothing$-Iterationen	CG-GRASP	27,9	27,3	22,4	29,7	6,7	7,1	7,2	7,4	1,1	1,1	1,3	1,3	1,0	1,0	1,0	1,0	9,0
	IS-GRASP	106,2	108,5	83,7	115,8	41,7	43,0	43,3	45,8	9,3	10,6	12,0	13,7	5,9	7,2	8,5	8,9	41,5
	SIS-GRASP	**367,4**	**357,5**	**352,2**	**376,0**	**143,2**	**150,1**	**144,8**	**155,6**	**35,0**	**29,1**	**33,8**	**48,6**	**21,0**	**26,5**	**29,5**	**32,7**	**143,9**
	CL-GRASP	56,6	51,3	41,2	57,4	15,9	16,3	16,0	17,1	2,7	3,1	3,2	3,3	1,4	1,6	1,8	1,8	18,2
	Hyb-GRASP	95,0	91,2	71,6	100,3	33,0	34,4	34,4	36,5	7,1	8,0	8,7	9,5	4,2	5,1	5,8	6,0	34,4
	RVNS	124,1	118,7	97,2	128,1	37,2	39,9	39,3	43,4	6,4	6,7	7,7	8,9	3,3	3,2	5,2	4,8	42,1
	VNS1	33,9	33,2	27,3	35,4	10,1	10,6	11,3	11,2	1,4	1,5	1,8	2,0	1,0	1,0	1,2	1,1	11,5
	VNS2	67,2	65,9	57,5	73,3	25,2	26,0	28,1	28,7	4,0	5,8	6,4	7,0	1,9	2,3	3,2	3,8	25,4
$\varnothing$-Regeneratoren	CG-GRASP	12,24	12,02	11,26	11,32	12,76	12,26	12,12	11,60	13,70	13,42	12,88	12,66	13,68	13,22	13,02	13,02	12,57
	IS-GRASP	11,68	11,52	**10,92**	**10,90**	11,86	**11,28**	11,06	11,00	12,02	11,76	11,30	**11,08**	**11,90**	11,42	11,26	11,34	11,39
	SIS-GRASP	11,78	11,76	11,00	11,08	12,08	11,58	11,54	11,20	12,68	12,30	11,72	11,70	12,58	12,04	11,88	11,94	11,80
	CL-GRASP	11,74	11,82	11,14	10,94	12,08	11,54	11,46	11,12	12,58	12,32	11,86	11,78	12,96	12,34	12,08	12,28	11,88
	Hyb-GRASP	**11,66**	**11,50**	10,96	**10,90**	**11,84**	**11,28**	**11,02**	**10,96**	**12,00**	**11,66**	**11,24**	11,14	11,96	**11,38**	**11,16**	**11,28**	**11,37**
	RVNS	11,78	11,76	11,04	11,02	12,08	11,64	11,46	11,24	12,88	12,54	12,16	11,86	13,28	12,60	12,06	12,32	11,98
	VNS1	11,74	11,76	11,06	10,94	11,98	11,62	11,46	11,18	12,72	12,22	11,96	11,64	12,70	12,12	11,76	12,00	11,80
	VNS2	11,68	11,64	11,06	10,94	11,90	11,44	11,32	11,18	12,42	11,94	11,54	11,40	12,42	11,82	11,50	11,68	11,62
#-Best	CG-GRASP	19	24	32	29	7	5	9	20	1	0	0	1	1	0	1	0	149
	IS-GRASP	46	49	**49**	**50**	47	**46**	47	45	44	27	35	**46**	**40**	29	37	33	670
	SIS-GRASP	41	37	45	41	36	31	23	35	16	5	16	16	13	4	6	10	375
	CL-GRASP	43	34	38	48	36	33	27	39	20	4	12	17	6	5	9	6	377
	Hyb-GRASP	**47**	**50**	47	**50**	**48**	**46**	**49**	**47**	**45**	**32**	**38**	43	37	**31**	**42**	**36**	**688**
	RVNS	41	37	43	44	37	28	27	33	9	4	8	12	5	3	16	6	353
	VNS1	43	37	42	48	41	29	27	36	12	8	10	19	13	8	16	3	392
	VNS2	46	43	42	48	45	38	34	36	24	20	24	30	19	14	27	18	508
$\varnothing$-$\mathbf{t}_{\text{Best}}(m.s)$	CG-GRASP	94.944	55.588	63.969	66.643	105.680	87.178	153.613	112.222	359.646	-	-	370.457	3.781.395	-	609.568	-	488.408,6
	IS-GRASP	**5.901**	8.914	19.272	7.054	**10.817**	25.900	**20.472**	**20.787**	28.714	53.930	**26.854**	34.223	**59.270**	73.625	**58.394**	**92.550**	**34.167,3**
	SIS-GRASP	22.880	20.112	**7.539**	29.734	38.733	48.658	84.512	43.496	116.674	181.464	157.445	71.468	155.998	204.780	199.214	153.148	95.990,9
	CL-GRASP	16.144	49.662	17.834	13.971	45.004	82.123	94.343	40.729	110.660	193.442	183.405	160.268	441.612	289.685	288.318	335.920	147.695,0
	Hyb-GRASP	11.265	**8.796**	14.874	**3.533**	12.611	**21.063**	28.977	25.384	**27.654**	**52.582**	37.346	**34.007**	73.645	**62.117**	77.707	94.788	36.646,8
	RVNS	27.517	32.813	31.259	30.932	45.351	95.234	69.476	69.803	180.725	197.281	234.917	166.415	156.892	199.052	135.108	395.285	129.253,8
	VNS1	22.154	35.283	36.156	21.880	39.373	111.088	60.065	59.515	240.725	320.440	210.108	104.360	573.478	618.304	282.936	1.114.590	240.653,4
	VNS2	23.747	27.984	23.319	19.071	28.502	55.188	66.471	47.063	111.492	104.652	102.569	80.311	135.894	168.122	120.212	158.736	79.583,3

Tabelle A.12: Evaluierung der Metaheuristiken im Zuge des zweiten Experiments in EN2, siehe *Kapitel 4.4*

Literaturverzeichnis

[1] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[2] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[3] S. Butenko, X. Cheng, C. Oliveira, and PM Pardalos. A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. *Recent Developments in Cooperative Control and Optimization*, 3(1):61–73, 2004.

[4] D.A.R. Chaves, C.F.C.L.C. Ayres, R.V.B. Carvalho, H.A. Pereira, C.J.A. Bastos-Filho, and J.F. Martins-Filho. Sparse regeneration placement for translucent optical networks using multiobjective evolutionary algorithms considering quality of service and capital cost. In *Microwave and Optoelectronics Conference (IMOC), 2009 SBMO/IEEE MTT-S International*, pages 417–422. SBMO/IEEE MTT-S International, 2009.

[5] S. Chen, I. Ljubic, and S. Raghavan. The Generalized Regenerator Location Problem. In *Proceedings of the 2009 International Network Optimization Conference (INOC 2009), Pisa, Italy*, 2009.

[6] S. Chen, I. Ljubic, and S. Raghavan. The Regenerator Location Problem. *Networks*, 55(3):205–220, 2010.

[7] S. Chen and S. Raghavan. The Regenerator Location Problem. In *Proceedings of the International Network Optimization Conference (INOC 2007)*. INOC, 2007.

[8] Guoli Ding, Thor Johnson, and Paul D. Seymour. Spanning trees with many leaves. *Journal of Graph Theory*, 37(4):189–197, 2001.

[9] A. Duarte, R. Marti, M.G.C. Resende, and R.M.A. Silva. Randomized heuristics for the Regenerator Location Problem. Technical report, AT&T Labs Research, Florham Park, New Jersey, 2010. URL http://www.research.att.com/ mgcr/doc/gpr-regenloc.pdf, accessed: 2012-11-30.

[10] A. Duarte, M. Resende, and R. Marti. GRASP for the Regenerator Location Problem. In *Proceedings of the 9th Metaheuristics International Conference (MIC2011), Udine, Italy*, pages 111–121, 2011.

[11] John D. Eblen, Charles A. Phillips, Gary L. Rogers, and Michael A. Langston. The maximum clique enumeration problem: algorithms, applications and implementations. In *Proceedings of the 7th international conference on Bioinformatics research and applications*, ISBRA'11, pages 306–319. Springer-Verlag, 2011.

[12] Thomas A. Feo and Mauricio G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71, 1989.

[13] Thomas A. Feo and Mauricio G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[14] Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, Daniel Raible, and Peter Rossmanith. An exact algorithm for the Maximum Leaf Spanning Tree problem. *Theoretical Computer Science*, 412(45):6290–6302, 2011.

[15] Michele Flammini, Alberto Marchetti Spaccamela, Gianpiero Monaco, Luca Moscardelli, and Shmuel Zaks. On the complexity of the regenerator placement problem in optical networks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures. Calgary, AB, Canada*, pages 154–162. ACM, 2009.

[16] Tetsuya Fujie. An exact algorithm for the maximum leaf spanning tree problem. *Computers & Operations Research*, 30(13):1931–1944, 2003.

[17] Sudipto Guha and Samir Khuller. Approximation Algorithms for Connected Dominating Sets. *Algorithmica*, 20(4):374–387, 1998.

[18] Pierre Hansen, Pierre Hansen, Nenad Mladenovic, Nenad Mladenovic, and Les Cahiers Du Gerad. A Tutorial on Variable Neighborhood Search. Technical report, Les Cahiers du GERAD, HEC Montreal and GERAD, 2003.

[19] Pierre Hansen and Nenad Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[20] Myles Hollander and Douglas A. Wolfe. *Nonparametric Statistical Methods, 2nd Edition*. Wiley-Interscience, 2 edition, 1999.

[21] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[22] Kim Hyung-Joon. Articulation points detection algorithm. Unpublished manuscript., 2006. http://www.ibluemojo.com/school/articul_algorithm.html, accessed: 2012-09-30.

[23] R Ihaka. R: Past and future history. In *Computer Science and Statistics, Proceedings of the 30th Symposium on the Interface (1998)*, pages 392–396. Interface Foundation of North America, 1998.

[24] T Jian. An O(20.304n) Algorithm for Solving Maximum Independent Set Problem. *IEEE Transactions on Computers*, 35(9):847–851, 1986.

[25] S. Kartalopoulos. *Overview of DWDM Devices and Networks*, chapter 4, pages 101–145. Wiley-IEEE Press, 2004.

[26] S.-W. Kim, S.-W. Seo, and S. C. Kim. Regenerator placement algorithms for connection establishment in all-optical networks. In *IEEE Global Telecommunications Conference*, pages 1205–1209. IEEE Communications Society, 2000.

[27] Roland Kopf and Günther Ruhe. A computational study of the weighted independent set problem for general graphs. *Foundations of Control Engineering*, 12(4):167–180, 1987.

[28] Fernando A. Kuipers, Anteneh Beshir, Ariel Orda, and Piet Van Mieghem. Impairment-aware path selection and regenerator placement in translucent optical networks. In *ICNP*, pages 11–20. IEEE Computer Society, 2010.

[29] Nenad Mladenovic and Pierre Hansen. Variable Neighborhood Search. *Computers & OR*, 24(11):1097–1100, 1997.

[30] G. Mohan, C. Siva Ram Murthy, and Arun K. Somani. Efficient algorithms for routing dependable connections in WDM optical networks. *IEEE/ACM Transactions on Networking*, 9(5):553–566, October 2001.

[31] B. Mukherjee. WDM optical communication networks: progress and challenges. *IEEE Journal on Selected Areas in Communications*, 18(10):1810–1824, 2000.

[32] Panos M. Pardalos and Jue Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994.

[33] J. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.

[34] R. Tarjan and A. Trojanowski. Finding a Maximum Independent Set. *SIAM Journal on Computing*, 6(3):537–546, 1977.

[35] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[36] Weiyi Zhang, Jian Tang, Kendall E. Nygard, and Chonggang Wang. REPARE: Regenerator Placement and Routing Establishment in Translucent Networks. In *Proceedings of the Global Communications Conference, 2009 (GLOBECOM 2009). Honolulu, Hawaii, USA*, pages 1–7. IEEE, 2009.

[37] Benyuan Zhu, David Peckham, Man Yan, Thierry Taunay, and John Fini. Recent Progress in Transmission Fibers for Capacity beyond 100-Tbit/s. In *Optical Fiber Communication Conference*, page OW1D.5. Optical Society of America, 2012.