Technische Universität Berlin

Fakultät IV - Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik

Bachelorarbeit

Verbesserung einer Greedy-Heuristik für Dominating Set: Dynamische Probleme und Turbo-Charging

Frank Dymek

Erstgutachter: Prof. Dr. Rolf Niedermeier
Zweitgutachter: Prof. Dr. Toby Walsh
Betreuer: Hendrik Molter, Prof. Dr. Rolf Niedermeier

28. Februar 2017

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.
Berlin, den

.....
Unterschrift

Abstract

A very common way to solve NP-hard problems heuristically is to solve them with a greedy heuristic. They are so popular, because they are very simple. In this work we look at a greedy heuristic for Dominating Set. We analyse how this greedy heuristic can be improved by turbo-charging. For this purpose we implement a turbo-charging greedy heuristic, which was presented by Downey et al. [5]. Based on this we engineer and implement own improvments for the heuristic. To analyse the quality of the turbo-charging greedy heuristic with and without our improvements, we conduct experiments. The experiments show that we can reduce the average size of the dominating sets by 2.45 %, if we are using our best improvement and the turbo-charging mechanism instead of only using the greedy heuristic.

The turbo-charging is based on dynamic problems. So we also look at the area of dynamic problems. We show that Dynamic Vertex Cover and Dynamic Set Cover are fixed-parameter tractable with the respect to the edit-parameter. For their size restricted versions Restricted Size Dynamic Vertex Cover and Restricted Size Dynamic Set Cover we show that they are W[1]-hard and W[2]-hard, respectively, when parameterised by the combined parameter of the edit- and the increment-parameter.

Zusammenfassung

Eine sehr weit verbreitete Art, um NP-schwere Probleme heuristisch zu lösen, sind die Greedy-Heuristiken. Sie sind deshalb so beliebt, weil sie sehr einfach aufgebaut sind. Wir betrachten in dieser Arbeit eine Greedy-Heuristik für Dominating Set. Dabei untersuchen wir, wie gut sie sich durch Turbo-Charging verbessern lässt, indem wir die in dem Konzept von Downey et al. [5] vorgestellte Turbo-Charging-Greedy-Heuristik implementieren. Basierend darauf entwickeln und implementieren wir eigene Verbesserungen der Heuristik. Um die Qualität der Turbo-Charging-Greedy-Heuristik mit und ohne unsere Verbesserungen zu untersuchen, führen wir Experimente durch. Mithilfe dieser Experimente stellen wir fest, dass wir mit unserer, auf die Größe des Dominating Sets bezogen, besten Verbesserung und der Nutzung des Turbo-Charging gegenüber der Greedy-Heuristik eine durchschnittliche Verkleinerung der Dominating Sets um $2,45\,\%$ erreichen können.

Das Turbo-Charging beruht dabei auf dynamischen Problemen. Deswegen betrachten wir neben der Heuristik auch das Gebiet der dynamischen Probleme. Dabei zeigen wir, dass Dynamic Vertex Cover und Dynamic Set Cover fixed-parameter tractable bezüglich der Veränderung in der Eingabe sind. Für ihre größenbeschränkten Varianten Restricted Size Dynamic Vertex Cover und Restricted Size Dynamic Set Cover zeigen wir, dass sie W[1]-schwer beziehungsweise W[2]-schwer bezogen auf die kombinierte Veränderung von Eingabe und Lösung sind.

Inhaltsverzeichnis

1 Einführung **6**
1.1 Literaturüberblick 7
1.2 Beitrag dieser Arbeit 8

2 Notationen und Grundlegendes **8**
2.1 Graphentheorie 8
2.2 Komplexitätstheorie 9
2.3 Problemdefinitionen 10

3 Dynamische Probleme **11**
3.1 Dynamic Dominating Set 14
3.2 Dynamic Vertex Cover 16
3.3 Dynamic Set Cover 21

4 Turbo-Charging-Heuristik für Dominating Set **26**
4.1 Algorithmus 28
4.2 Implementierung 35
4.3 Heuristische Verbesserungen 37
4.4 Erweiterung für Set Cover 39

5 Experimente **40**
5.1 Daten 40
5.2 Aufbau 41
5.3 Resultate 42

6 Ausblick **47**

1 Einführung

Viele wichtige Berechnungsprobleme sind NP-schwer. Daher werden oft Heuristiken verwendet, um sie zu lösen. Wir werden hier eine Turbo-Charging-Greedy-Heuristik für DOMINATING SET von Downey et al. [5] vorstellen und ihre Qualität verbessern.

Das Problem DOMINATING SET ist ein Graphproblem. Dabei ist ein ungerichteter Graph $G = (V, E)$ und eine Zahl k gegeben. Gesucht ist eine Knotenmenge $S \subseteq V$ der Größe k, sodass jeder Knoten in G Teil dieser Menge S ist oder über eine Kante mit einem Knoten in dieser Menge S verbunden ist. Ist das der Fall, so nennen wir S ein *Dominating Set* für G.

DOMINATING SET findet in vielen Bereichen Anwendung. Wir können uns zum Beispiel folgende Situation vorstellen: Wir sind der Leiter einer Wanderausstellung. Um die Ausstellung zu beaufsichtigen und Fragen zu beantworten wollen wir Mitarbeiter einstellen. Jeder Mitarbeiter kann dabei den Raum, in dem er sich aufhält und die davon abgehenden Räume beaufsichtigen. Aus ökonomischen Gründen wollen wir möglichst wenige Leute einstellen. Es drängt sich also folgende Frage auf: Wie viele Leute müssen wir für die Ausstellung mindestens einstellen?

Damit wir dieses Problem als DOMINATING SET-Problem lösen können, überführen wir es in einen Graphen. Dazu wird jeder Raum der Ausstellung als Knoten dargestellt. Zwischen zwei Knoten ziehen wir eine Kante, wenn die entsprechenden Räume direkt miteinander verbunden sind. Bei einer Ausstellung mit acht Räumen ergibt sich dann zum Beispiel der in Abbildung 1 dargestellte Graph G.

Um die kleinste Anzahl benötigter Mitarbeiter zu finden, suchen wir nun für den Graphen G ein kleinstmögliches Dominating Set. Mit genauem Hinsehen können wir feststellen, dass $S = \{b, f\}$ ein kleinstmögliches Dominating Set für G ist. Die Knoten dieses Dominating Sets sind in Abbildung 2 schwarz

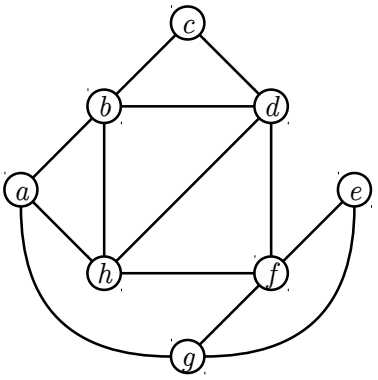

Abbildung 1: Graph G.

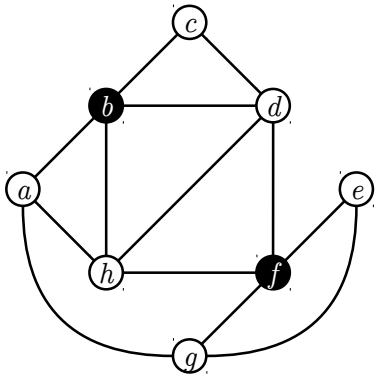

Abbildung 2: Graph G mit Dominating Set.

gefärbt. Wir müssen also in die Räume b und f je einen Mitarbeiter stellen.

Bei dieser kleinen Instanz haben wir die Lösung relativ einfach sehen können. Da Dominating Set NP-schwer ist [9], ist es jedoch für große Instanzen sehr aufwendig eine Lösung zu berechnen. In diesen Fällen können wir unter anderem auf eine Turbo-Charging-Greedy-Heuristik von Downey et al. [5] zurückgreifen.

Das Besondere an einer Turbo-Charging-Heuristik ist, dass sie im Lösungsprozess wieder ein paar Schritte zurückgehen kann, falls sie keine Lösung gefunden hat. Dadurch schlägt sie hoffentlich einen besseren Weg ein und findet letzten Endes möglicherweise doch noch ein Dominating Set der gesuchten Größe. Turbo-Charging bezeichnet also den Vorgang des Zurückgehens, um zu versuchen eine bessere Lösung zu finden.

Um das Turbo-Charging umzusetzen, bedienen wir uns als Unterprozedur der Lösung sogenannter dynamischer Probleme. Bei dynamischen Problemen erhalten wir zwei „ähnliche“ Instanzen I und I' und zusätzlich eine Lösung S für die Instanz I. Wir suchen nun eine zu S „ähnliche“ Lösung S' für die Instanz I'. Die Ähnlichkeit der Instanzen I und I' und der Lösungen S und S' kann dabei von Problem zu Problem unterschiedlich definiert sein.

1.1 Literaturüberblick

Schon 2013 entwickelten Hartung und Niedermeier [11] einen Turbo-Charging-Mechanismus. Sie wollten damit eine Greedy-Heuristik für ein Graph Coloring-Problem verbessern. Die Experimente, die sie dazu durchführten zeigten, dass ihnen das auch gelungen war. Die Idee Turbo-Charging auf eine Heuristik anzuwenden, setzte sich danach weiter fort. Downey et al. [5] zeigten 2014, dass Dynamic Dominating Set fixed-parameter tractable bezüglich der Edge-Edit-Distance ist und beschrieben auf der Grundlage dieses dynamischen Problems eine Turbo-Charging-Greedy-Heuristik für Dom-

INATING SET. Danach zeigten Abu-Khzam et al. [1] und Krithika et al. [13] für eine Reihe weiterer dynamischer Probleme, dass sie fixed-parameter tractable sind und haben so den Weg dafür frei gemacht Turbo-Charging auf weitere Probleme auszudehnen. Abu-Khzam et al. [1] haben sich dabei auch RESTRICTED SIZE DYNAMIC DOMINATING SET, welches eine größenbeschränkte Variante von DYNAMIC DOMINATING SET ist, gewidmet. Letztes Jahr haben Gaspers et al. [10] Turbo-Charging auf zwei Greedy-Heuristiken für das Treewidth-Problem angewandt. In ihren Experimenten konnten auch sie eine leichte Verbesserung durch das Turbo-Charging aufzeigen.

1.2 Beitrag dieser Arbeit

Wir untersuchen in dieser Arbeit die Komplexität von DYNAMIC VERTEX COVER und DYNAMIC SET COVER, sowie von deren größenbeschränkten Varianten RESTRICTED SIZE DYNAMIC VERTEX COVER und RESTRICTED SIZE DYNAMIC SET COVER. Dabei zeigen wir, dass DYNAMIC VERTEX COVER und DYNAMIC SET COVER fixed-parameter tractable bezüglich des Unterschieds in der Eingabe sind. RESTRICTED SIZE DYNAMIC VERTEX COVER und RESTRICTED SIZE DYNAMIC SET COVER sind hingegen W[1]-schwer beziehungsweise W[2]-schwer bezüglich des kombinierten Parameters aus dem Unterschied in der Eingabe und dem Unterschied in der Lösung.

Des Weiteren entwickeln wir drei Verbesserungen für die von Downey et al. [5] beschriebene Turbo-Charging-Greedy-Heuristik für DOMINATING SET. Diese Verbesserungen vergleichen wir durch Experimente mit der ursprünglichen Greedy-Heuristik und der Greedy-Heuristik mit dem Turbo-Charging-Mechanismus von Downey et al. [5]. Mit der, auf die Größe der Dominating Sets bezogen, besten Verbesserung und der Nutzung des Turbo-Charging-Mechanismus können wir dabei gegenüber der Greedy-Heuristik eine durchschnittliche Verkleinerung der Dominating Sets um $2,45\,\%$ erzielen.

2 Notationen und Grundlegendes

Hier führen wir einige Definitionen und Notationen ein, die wir im weiteren Verlauf verwenden.

2.1 Graphentheorie

In Anlehnung an Diestel [4] führen wir im Folgenden einige für uns wichtige Begriffe der Graphentheorie ein:

- Wir nennen einen Graphen *einfach*, wenn er ungerichtet ist und keine Mehrfachkanten oder Schleifen enthält.
- Ein Graph ist *endlich*, wenn die Anzahl der Kanten und Knoten endlich ist.

- Immer wenn im Folgenden von einem *Graphen* G die Rede ist, ist ein einfacher, endlicher Graph $G = (V, E)$ mit einer Knotenmenge V und einer Kantenmenge $E \subseteq \{\{v_1, v_2\} \mid v_1, v_2 \in V \wedge v_1 \neq v_2\}$ gemeint.
- Wir bezeichnen die *Knotenmenge* eines Graphen G als $V(G)$ und die *Kantenmenge* als $E(G)$.
- Einen Graphen G bezeichnen wir als *zusammenhängend*, wenn für alle Knoten $u, w \in V(G)$ mit $u \neq w$ eine Folge $(u = v_1, v_2, \ldots, v_p = w)$ existiert, sodass für alle $i \in \mathbb{N}$ mit $1 \leq i \leq p-1$ gilt, dass $\{v_i, v_{i+1}\} \in E(G)$.
- Falls nicht anders definiert, so bezeichnen wir mit $n = |V(G)|$ die Anzahl der Knoten eines Graphen G.
- Die *Dichte* $\frac{2|E(G)|}{n(n-1)}$ eines Graphen G definieren wir als den Quotienten aus der tatsächlichen und der maximal möglichen Anzahl an Kanten in G.
- Die *offene Nachbarschaft* $N_G(S) = \{v \in V(G) \setminus S \mid \{v, s \in S\} \in E(G)\}$ einer Knotenmenge $S \subseteq V(G)$ im Graphen G ist die Menge aller Knoten $v \in V(G) \setminus S$, die durch eine Kante mit einem Knoten $s \in S$ verbunden sind.
- Die *geschlossene Nachbarschaft* $N_G[S] = N_G(S) \cup S$ einer Knotenmenge $S \subseteq V(G)$ im Graphen G ist die Menge aller Knoten $v \in V(G)$, die entweder selbst in S enthalten sind oder durch eine Kante mit einem Knoten $s \in S$ verbunden sind.
- Der *Knotengrad* $\deg(v) = |N_G(\{v\})|$ eines Knotens $v \in V(G)$ in einem Graphen G ist die Anzahl der Knoten in der offenen Nachbarschaft $N_G(\{v\})$ der einelementigen Knotenmenge $\{v\}$ in G.
- Die *Vertex-Edit-Distance* $d_v(S, S') = |S \setminus S' \cup S' \setminus S|$ zwischen zwei Knotenmengen S und S' ist die Anzahl der Elemente, die in genau einer dieser Mengen enthalten sind.
- Die *Edge-Edit-Distance* $d_e(G, G') = |E \setminus E' \cup E' \setminus E|$ zwischen zwei Graphen $G = (V, E)$ und $G' = (V, E')$ ist die Anzahl der Kanten, die in genau einer der Kantenmengen der beiden Graphen enthalten sind.

2.2 Komplexitätstheorie

Hier führen wir grundlegende, für diese Arbeit relevante Begriffe der Komplexitätstheorie sein. Dabei schauen wir uns zuerst einige Begriffe der klassischen Komplexität an, die auch von Garey und Johnson [9] beschrieben werden.

Die wichtigsten Komplexitätsklassen in der Komplexitätstheorie sind die Klassen P und NP. Die Klasse P enthält dabei alle Entscheidungsprobleme, die sich deterministisch in Polynomialzeit lösen lassen. Die Klasse NP enthält wiederum alle Entscheidungsprobleme, die sich nichtdeterministisch in Polynomialzeit lösen lassen. Insbesondere gilt also, dass $\mathrm{P} \subseteq \mathrm{NP}$. Dabei nehmen wir an, dass $\mathrm{P} \neq \mathrm{NP}$ ist.

Ein Entscheidungsproblem A ist auf ein anderes Entscheidungsproblem B *reduzierbar*, wenn eine total berechenbare Funktion f existiert, sodass für eine beliebige Instanz I von Problem A gilt, dass I genau dann eine Ja-Instanz von A ist, wenn $f(I)$ eine Ja-Instanz von B ist. Existiert diese Funktion f, so nennen wir sie eine *Reduktion* von Problem A auf Problem B.

Die Entscheidungsprobleme, die wir in dieser Arbeit betrachten, sind alle *NP-vollständig*. Wir bezeichnen dabei ein Problem $A \in \mathrm{NP}$ als NP-vollständig, wenn sich jedes Problem $B \in \mathrm{NP}$ in Polynomialzeit auf dieses Problem A reduzieren lässt.

Zwei Probleme A und B sind *polynomialzeitäquivalent*, wenn sie sich gegenseitig in Polynomialzeit aufeinander reduzieren lassen.

Neben der klassischen Komplexität spielt auch die parametrisierte Komplexität für uns eine Rolle. Siehe dazu auch Cygan et al. [3], Downey und Fellows [6] und Flum und Grohe [7]. In der parametrisierten Komplexität werden die Probleme bezüglich eines Parameters $k \in \mathbb{N}$ in Komplexitätsklassen eingeteilt. Der Parameter k sollte dabei im Allgemeinen deutlich kleiner als die Eingabegröße n sein. Für uns interessant sind dabei die Komplexitätsklassen FPT, W[1] und W[2].

In der Komplexitätsklasse FPT befinden sich alle Probleme, welche *fixed-parameter tractable* bezüglich eines Parameters k sind. Diese Probleme können in Zeit $\mathcal{O}(f(k)p(n))$ gelöst werden, wobei f eine beliebige berechenbare Funktion und p ein Polynom ist.

Über die Komplexitätsklassen W[1] und W[2] ist für uns nur wichtig zu wissen, dass $\mathrm{FPT} \subseteq \mathrm{W[1]} \subseteq \mathrm{W[2]}$ gilt. Wir gehen dabei davon aus, dass $\mathrm{FPT} \neq \mathrm{W[1]} \neq \mathrm{W[2]}$ ist. Das heißt insbesondere, dass wir annehmen, dass W[1]-schwere und W[2]-schwere Probleme nicht in FPT enthalten sind.

Eine Reduktion von Problem B mit dem Parameter k_B auf das Problem A mit dem Parameter k_A bezeichnen wir als *FPT-Reduktion*, falls sie in Zeit $\mathcal{O}(f(k_B)p(n))$ durchgeführt werden kann und der Parameter k_A nur von dem Parameter k_B abhängt.

Ein Problem A ist *W[i]-schwer* mit $i \in \{1, 2\}$ bezüglich eines Parameters k_A, wenn eine FPT-Reduktion von einem, bezüglich eines Parameters k_B, W[i]-schweren Problem B auf dieses Problem A existiert.

2.3 Problemdefinitionen

In Kapitel 3 werden wir uns die dynamischen Versionen der folgenden NP-vollständigen Probleme angucken und ihre Komplexität untersuchen:

Bei dem NP-vollständigen Problem Dominating Set [9] haben wir einen Graphen G und eine Zahl $k \in \mathbb{N}$ gegeben und fragen uns, ob ein Dominating Set S der Größe k für G existiert. Eine Menge wird als Dominating Set für einen Graphen bezeichnet, falls für alle Knoten dieses Graphen gilt, dass sie von dieser Menge dominiert werden. Ein Knoten wird von einer Menge dominiert, wenn er entweder Teil der Menge ist oder über eine Kante mit einem Knoten in dieser Menge verbunden ist.

Dominating Set (DS)

Eingabe:	Ein Graph $G = (V, E)$ und eine natürliche Zahl $k \in \mathbb{N}$.
Frage:	Existiert ein Dominating Set $S \subseteq V$ für G mit $\|S\| \leq k$, sodass $N_G[S] = V$?

Bei dem NP-vollständigen Problem Vertex Cover [9] haben wir einen Graphen G und eine Zahl $k \in \mathbb{N}$ gegeben und fragen uns, ob ein Vertex Cover S der Größe k für G existiert. Eine Menge wird als Vertex Cover für einen Graphen bezeichnet, falls alle Kanten dieses Graphen mindestens einen Knoten im Vertex Cover als Endpunkt haben.

Vertex Cover (VC)

Eingabe:	Ein Graph $G = (V, E)$ und eine natürliche Zahl $k \in \mathbb{N}$.
Frage:	Existiert ein Vertex Cover $S \subseteq V$ für G mit $\|S\| \leq k$, sodass für alle $e = \{u, v\} \in E$ gilt, dass $u \in S$ oder $v \in S$?

Bei dem NP-vollständigen Problem Set Cover [9] haben wir ein Universum $\mathcal{U}$, eine Menge $\mathcal{F}$ von m Teilmengen $(T_i \in \mathcal{F}) \subseteq \mathcal{U}$ mit $1 \leq i \leq m$ und eine Zahl $k \in \mathbb{N}$ gegeben und fragen uns, ob ein Set Cover S der Größe k für $\mathcal{U}$ existiert. Eine Menge wird als Set Cover für ein Universum bezeichnet, wenn die Vereinigung der Teilmengen die es enthält, gleich $\mathcal{U}$ ist.

Set Cover (SC)

Eingabe:	Ein Universum $\mathcal{U}$; eine Menge $\mathcal{F}$ von m Teilmengen $T_i \in \mathcal{F}$ mit $T_i \subseteq \mathcal{U}$ und $1 \leq i \leq m$; eine natürliche Zahl $k \in \mathbb{N}$.
Frage:	Existiert ein Set Cover $S \subseteq \mathcal{F}$ für $\mathcal{U}$ aus höchstens k Teilmengen $T_i \in \mathcal{F}$, sodass $\bigcup_{T_i \in S} = \mathcal{U}$?

3 Dynamische Probleme

Wir erklären die dynamischen Probleme an unserem Beispiel mit der Wanderausstellung aus Kapitel 1. Dazu stellen wir uns vor, dass unsere Ausstellung in die nächste Stadt zieht. Dabei müssen wir uns an die veränderten Gegebenheiten anpassen. Wir haben zwar wieder die gleiche Anzahl an Räumen zur Verfügung, die Räume sind jedoch leicht anders angeordnet. Es

haben sich vier Zugänge geändert.

Wir suchen jetzt ausgehend von unserer Lösung S für den alten Standort eine neue Aufstellung für die Mitarbeiter, sodass wieder für jeden Raum gilt, dass sich entweder ein Mitarbeiter im Raum selbst oder in einem der Nachbarräume befindet. Diese Aufstellung soll sich dabei kaum von der alten unterscheiden. Das heißt, dass wir nur die Möglichkeit haben die Aufstellung unverändert zu lassen, einen zusätzlichen Mitarbeiter einem Raum zuzuweisen oder einen Mitarbeiter abzuziehen. Der Unterschied zwischen der alten und der neuen Aufstellung beträgt also höchstens eins. Bei so einer kleinen Instanz wie in unserem Beispiel klingt dieses Vorgehen nur unnötig kompliziert und es wäre hier aufgrund der geringen Größe in der Tat sinnvoller erneut nach einer optimalen Lösung zu suchen. Bei großen Instanzen ist es allerdings deutlich einfacher sich den geringen Unterschieden anzupassen und ausgehend von der alten Lösung eine ähnliche, neue zu suchen.

Auch für die Mitarbeiter hat dieses Konzept einen Vorteil. So haben sich die bestehenden Mitarbeiter schon in ihre Bereiche eingearbeitet, wissen über die Ausstellungsstücke in den Räumen, für die sie zuständig sind, Bescheid und können so alle Fragen der Besucher beantworten. Wenn wir nun eine neue Aufstellung finden, die die alte komplett verwirft, müssten sich alle Mitarbeiter neu einarbeiten. Mit unserer Einschränkung muss allerdings höchstens ein Mitarbeiter neu eingewiesen werden.

Ir bleiben also bei unserem kleinen Beispiel und übertragen es auf Graphen. Dabei bezeichnen wir den Graphen G, der die alte Ausstellung widerspiegelt und den Graphen G', der die neue Anordnung repräsentiert. Da sich die Anzahl der Räume nicht ändert, gilt $V(G) = V(G')$. Für die Änderung der Zugänge löschen wir die Kanten von b nach h und von e nach f und fügen die Kanten von d nach e und von g nach h hinzu. Die Graphen G und G' sind sich also „ähnlich“. Genauer gesagt ist die Edge-Edit-Distance $d_e(G, G') \leq 4$.

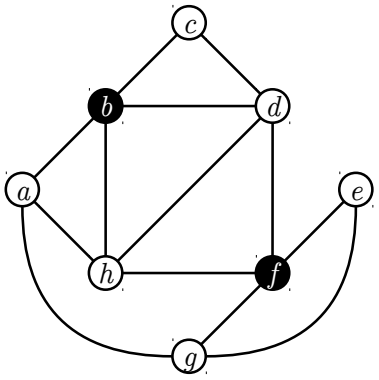

Abbildung 3: Graph G mit altem Dominating Set.

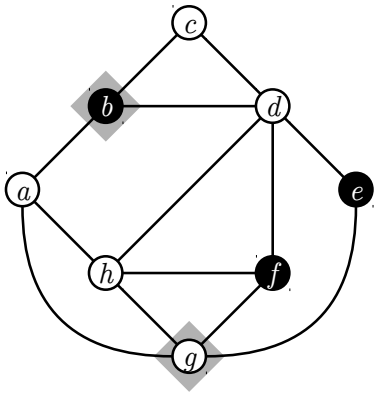

Abbildung 4: Graph G' mit neuem und alternativem Dominating Set.

Für den Graphen G hatten wir vorhin bereits ein Dominating Set $S = \{b, f\}$ gefunden. Die Knoten dieser Räume sind in Abbildung 3 schwarz gekennzeichnet.Jetzt suchen wir ein neues Dominating Set S', welches um eins von dem alten Dominating Set S abweicht. Das bedeutet, dass die Vertex-Edit-Distance $d_v(S, S') \leq 1$ ist. Dabei können wir uns die Ähnlichkeit der Graphen G und G' zunutze machen.

Wenn wir nämlich einfach einen zusätzlichen Mitarbeiter dem Raum e zuordnen, so haben wir ein neues Dominating Set $S' = \{b, e, f\}$ gefunden. Dieses Dominating Set S' ist in Abbildung 4 schwarz dargestellt. Es gäbe jedoch auch ein Dominating Set mit nur zwei Mitarbeitern. In der Abbildung 4 sind das die grau hinterlegten Knoten b und g. Genau dort liegt das Problem. Es ist zwar einfacher mittels eines dynamischen Problems eine Lösung zu finden, als sie direkt zu berechnen. Diese Lösung ist dann aber in der Regel nicht optimal. Da die in dieser Arbeit betrachteten Probleme NP-vollständig sind, würde es allerdings auch zu lange dauern bei großen Instanzen eine optimale Lösung zu berechnen.

Von unserem Beispiel ausgehend verallgemeinern wir das nun auf ein beliebiges dynamisches Graphproblem P. Wir haben also einen Graphen G und einen Graphen G'. Die Ähnlichkeit dieser Graphen spiegelt sich darin wider, dass $V(G) = V(G')$ gilt und die Edge-Edit-Distance $d_e(G, G')$ kleiner oder gleich einem festgelegten $k \in \mathbb{N}$ ist. Außerdem haben wir eine Knotenmenge S, welche eine Lösung des Graphproblems P auf dem Graphen G ist. Gesucht ist nun eine Knotenmenge S' welche eine Lösung des Graphproblems P auf dem Graphen G' sein soll. Um die Ähnlichkeit von S und S' zu gewährleisten, soll die Vertex-Edit-Distance $d_v(S, S')$ kleiner oder gleich einem festgelegten $r \in \mathbb{N}$ sein.

Das Ganze können wir nun auch auf ein allgemeines dynamisches Problem X ausdehnen. Dieses dynamische Problem X enthält zwei „ähnliche“ Instanzen I und I' und außerdem eine Lösung S für die Instanz I. Die Frage ist nun, ob es für die zu I „ähnliche“ Instanz I' auch eine zu S „ähnliche“ Lösung S' gibt. Je nachdem, um welches Problem es sich handelt, ist die Ähnlichkeit dabei unterschiedlich definiert.

In diesem Kapitel werden wir die dynamischen Probleme Dynamic Dominating Set, Dynamic Vertex Cover und Dynamic Set Cover untersuchen. Wir können dabei schon jetzt festhalten, dass alle diese dynamischen Probleme NP-vollständig sind, da auch die Probleme, auf denen sie basieren, nämlich Dominating Set, Vertex Cover und Set Cover, NP-vollständig sind [9]. Das können wir uns ganz einfach überlegen, indem wir bei einem dynamischen Problem als Ausgangsinstanz eine triviale Instanz mit einer trivialen Lösung nehmen und als neue Instanz die Instanz, die wir eigentlich lösen wollen. Die Lösung der alten Instanz hilft uns in diesem Fall also nicht beim Lösen der neuen. Somit sind wir wieder beim Basisproblem, welches NP-vollständig ist, angelangt. Es wird sich allerdings herausstellen, dass die dynamischen Probleme fixed-parameter tractable bezüglich des Un-

terschieds in der Eingabe sind.

Denken wir nochmal an unser Beispiel zurück. Wir hatten vorhin festgestellt, dass das neue Dominating Set aus Abbildung 4 nicht optimal ist. Stellen wir uns nun vor, dass wir mit der Wanderausstellung noch ein paar Mal umziehen und jedes Mal eine um eins abweichende Aufstellung suchen. Irgendwann könnte es passieren, dass die Lösung, die uns das dynamische Problem liefert, lautet, dass wir in so gut wie jeden Raum einen Mitarbeiter stellen sollen. Wir stellen also fest, dass dynamische Probleme gut geeignet sind, um bei leichten Veränderungen eine Lösung zu erhalten. Ändert sich etwas jedoch kontinuierlich und wir wollen jedes Mal eine Lösung berechnen, so kann es dazu kommen, dass wir viel zu große Lösungen erhalten. Um dem entgegen zu wirken, könnte man festlegen, dass die neue Lösung nicht größer sein darf, als die alte. Dadurch wird es zwar möglicherweise schwieriger eine Lösung zu finden, dafür bleibt die Lösung auch bei wiederholter Ausführung klein. Genau dort setzen die größenbeschränkten dynamischen Probleme an. Sie sind den dynamischen Problemen sehr ähnlich. Der einzige Unterschied liegt darin, dass bei den größenbeschränkten dynamischen Problemen die neue Lösung S' nicht größer als die alte Lösung S sein darf.

Wir werden zu jedem dynamischen Problem, welches wir betrachten auch die dazugehörige größenbeschränkte Variante Restricted Size Dynamic Dominating Set, Restricted Size Dynamic Vertex Cover beziehungsweise Restricted Size Dynamic Set Cover untersuchen und feststellen, dass sie alle W[1]-schwer bezüglich des kombinierten Parameters (k, r), bestehend aus dem Unterschied k in der Eingabe und dem Unterschied r in der Lösung, sind. Die Parameter k und r werden dabei in den jeweiligen Unterkapiteln der einzelnen Probleme präziser definiert.

Im weiteren Verlauf wird es für uns jedoch entscheidend sein, dass die dynamischen Probleme fixed-parameter tractable bezüglich k sind. In Kapitel 4 präsentieren wir nämlich basierend auf einer Idee von Downey et al. [5] eine Turbo-Charging-Greedy-Heuristik für Dominating Set, die ausnutzt, dass das zugrundeliegende dynamische Problem Dynamic Dominating Set fixed-parameter tractable bezüglich k ist. Der Gedanke des Turbo-Charging lässt sich aber ohne Weiteres auch auf andere Probleme übertragen, solange das entsprechende dynamische Problem fixed-parameter tractable ist.

3.1 Dynamic Dominating Set

Schauen wir uns also das Problem Dynamic Dominating Set etwas genauer an. Beim Problem Dynamic Dominating Set ist ein Graph G und ein Dominating Set S für diesen Graphen gegeben. Außerdem ist ein zweiter, „ähnlicher“ Graph G' gegeben. Ähnlichkeit bedeutet, dass die Edge-Edit-Distance zwischen G und G' kleiner als ein vorgegebenes $k \in \mathbb{N}$ ist. Gesucht ist ein zu S „ähnliches“ Dominating Set S' für den Graphen G'. Hierbei bedeutet

ähnlich, dass die Vertex-Edit-Distance zwischen S und S' kleiner oder gleich einem vorgegebenen $r \in \mathbb{N}$ ist.

Dynamic Dominating Set (DDS)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Graph $G = (V, E)$; ein Graph $G' = (V, E')$ mit $d_e(G, G') \leq k$; ein Dominating Set $S \subseteq V$ für G.
Frage:	Existiert ein Dominating Set $S' \subseteq V$ für G, sodass $d_v(S, S') \leq r$ ist?

Downey et al. [5] haben gezeigt, dass DDS NP-vollständig und fixed-parameter tractable bezüglich der Edge-Edit-Distance k ist. Das bedeutet für uns, dass wir eine Turbo-Charging-Heuristik für Dominating Set entwickeln können. Das haben wir in Kapitel 4 nach der Idee von Downey et al. [5] auch getan. Diese Heuristik nutzt das im Folgenden beschriebene Vorgehen.

Um zu zeigen, dass DDS fixed-parameter tractable bezüglich k ist, haben Downey et al. [5] zuerst gezeigt, dass DDS polynomialzeitäquivalent zu Monotone Dynamic Dominating Set ist.

Das Problem Monotone Dynamic Dominating Set ist bis auf zwei Punkte identisch mit DDS. Erstens dürfen zu G nur Kanten hinzugefügt werden, um G' zu erzeugen, und zweitens muss S' alle Knoten enthalten, die bereits in S enthalten sind.

Monotone Dynamic Dominating Set (Mon-DDS)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Graph $G = (V, E)$; ein Graph $G' = (V, E')$ mit $d_e(G, G') \leq k$ und $E(G) \subseteq E(G')$; ein Dominating Set $S \subseteq V$ für G.
Frage:	Existiert ein Dominating Set $S' \subseteq V$ für G mit $d_v(S, S') \leq r$ und $S \subseteq S'$?

Downey et al. [5] haben gezeigt, dass sich Mon-DDS in Polynomialzeit auf Dominating A Vertex Cover reduzieren lässt.

Beim Problem Dominating A Vertex Cover ist ein Graph G und ein Vertex Cover C für diesen Graphen gegeben. Das Vertex Cover C ist dabei k Elemente groß. Weiterhin ist eine natürliche Zahl $r \leq k$ gegeben. Gesucht ist ein Dominating Set D für C in G, welches höchstens r Knoten enthält.

Dominating A Vertex Cover (Dom-A-VC)

Eingabe:	Ein Graph $G = (V, E)$; ein Vertex Cover $C \subseteq V$ für G der Größe k; eine natürliche Zahl $r \leq k$.
Frage:	Existiert eine Knotenmenge $D \subseteq V$ mit $\lvert D \rvert \leq r$, sodass $C \subseteq N_G[D]$?

Für Dom-A-VC haben Downey et al. [5] gezeigt, dass es fixed-parameter tractable bezüglich der Größe k des Vertex Covers C ist. Somit haben Downey et al. [5] folgenden Satz bewiesen.

Satz 1 ([5]). Dynamic Dominating Set *ist NP-vollständig, kann in der Laufzeit* $\mathcal{O}(\binom{k+2^k}{k} n^3 log^2(n))$ *gelöst werden und ist somit fixed-parameter tractable bezüglich der Edge-Edit-Distance* k.

Restricted Size Dynamic Dominating Set

Wie bereits erwähnt wäre es wünschenswert, wenn die Lösung auch bei mehrmaliger Hintereinanderausführung von DDS nicht immer größer werden könnte. Genau das liefert das Problem Restricted Size Dynamic Dominating Set. Das Problem Restricted Size Dynamic Dominating Set unterscheidet sich lediglich in einem Punkt von DDS. Für die neue Lösung S' wird zusätzlich dazu, dass die Vertex-Edit-Distance $d_v(S, S')$ zwischen der alten Lösung S und der neuen S' kleiner oder gleich einem vorgegebenen $r \in \mathbb{N}$ sein muss, gefordert, dass S' nicht größer sein darf als S. Es muss also auch gelten, dass $|S'| \leq |S|$ ist. Dadurch bleibt die Lösung klein.

Restricted Size Dynamic Dominating Set (RSDDS)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Graph $G = (V, E)$; ein Graph $G' = (V, E')$ mit $d_e(G, G') \leq k$; ein Dominating Set $S \subseteq V$ für G.
Frage:	Existiert ein Dominating Set $S' \subseteq V$ für G', sodass $d_v(S, S') \leq r$ und $\|S'\| \leq \|S\|$ ist?

Abu-Khzam et al. [1] haben gezeigt, dass die zusätzliche Größenbeschränkung bei RSDDS dafür sorgt, dass das Problem W[2]-schwer bezüglich des kombinierten Parameters (k, r) ist. Dafür haben sie Dominating Set auf RSDDS reduziert. Für uns können wir somit folgenden Satz festhalten.

Satz 2 ([1]). Restricted Size Dynamic Dominating Set *ist NP-vollständig und W[2]-schwer bezüglich des kombinierten Parameters* (k, r) *aus der Edge-Edit-Distance* k *und der Vertex-Edit-Distance* r.

Der Vorteil, den das dynamische Problem DDS gegenüber Dominating Set hatte, nämlich dass wir schneller zu einer Lösung kommen, wird durch die Größenbeschränkung also wieder zunichte gemacht.

3.2 Dynamic Vertex Cover

Als nächstes untersuchen wir das Problem Dynamic Vertex Cover, um festzustellen, ob wir auch für Vertex Cover eine Turbo-Charging-Heuristik entwickeln können. Das Problem Dynamic Vertex Cover ist vom

Prinzip her genau gleich wie DDS. Diesmal sind die Lösungen allerdings keine Dominating Sets, sondern Vertex Cover.

Wir haben also einen Graph G und ein Vertex Cover S für diesen Graphen gegeben. Außerdem ist ein zweiter, „ähnlicher“ Graph G' gegeben. Ähnlichkeit bedeutet wieder, dass die Edge-Edit-Distance zwischen G und G' kleiner als ein vorgegebenes $k \in \mathbb{N}$ ist. Gesucht ist diesmal ein zu S „ähnliches“ Vertex Cover S' für den Graphen G'. Hierbei bedeutet ähnlich genau so wie vorhin, dass die Vertex-Edit-Distance zwischen S und S' kleiner oder gleich einem vorgegebenen $r \in \mathbb{N}$ ist.

Dynamic Vertex Cover (DVC)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Graph $G = (V, E)$; ein Graph $G' = (V, E')$ mit $d_e(G, G') \leq k$; ein Vertex Cover $S \subseteq V$ für G.
Frage:	Existiert ein Vertex Cover $S' \subseteq V$ für G, sodass $d_v(S, S') \leq r$ ist?

Um mithilfe von DVC eine Turbo-Charging-Heuristik für Vertex Cover zu entwickeln, muss DVC fixed-parameter tractable bezüglich des kombinierten Parameters (k, r) sein. Um das zu zeigen, beweisen wir zunächst, dass DVC polynomialzeitäquivalent zu Monotone Dynamic Vertex Cover ist.

Das Problem Monotone Dynamic Vertex Cover ist bis auf zwei Einschränkungen identisch mit DVC. Die eine Einschränkung besagt, dass zu G nur Kanten hinzugefügt werden dürfen, um G' zu erzeugen, und die andere Einschränkung besagt, dass S' alle Knoten enthalten muss, die bereits in S enthalten sind.

Monotone Dynamic Vertex Cover (Mon-DVC)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Graph $G = (V, E)$; ein Graph $G' = (V, E')$ mit $d_e(G, G') \leq k$ und $E(G) \subseteq E(G')$; ein Vertex Cover $S \subseteq V$ für G.
Frage:	Existiert ein Vertex Cover $S' \subseteq V$ für G mit $d_v(S, S') \leq r$ und $S \subseteq S'$?

Lemma 1. Dynamic Vertex Cover *ist polynomialzeitäquivalent zu* Monotone Dynamic Vertex Cover.

Beweis. Um zu beweisen, dass DVC und Mon-DVC polynomialzeitäquivalent sind, zeigen wir polynomielle Reduktionen in beide Richtungen.

Die Reduktion von Mon-DVC auf DVC ist trivial, da Mon-DVC nur ein Spezialfall von DVC ist.

Also reduzieren wir nun DVC auf Mon-DVC. Dazu überführen wir die DVC-Instanz $I = (G, G', S, k, r)$ in eine Mon-DVC-Instanz I'. Hier-

für bilden wir einen neuen Graphen $G_- = (V, E)$ mit $V = V(G)$ und $E = E(G) \cap E(G')$ aus dem Graphen G, indem wir nur Kanten entfernen, sodass sich G' aus G_- bilden lässt, indem nur Kanten hinzugefügt werden. Aus $d_e(G, G') \leq k$ folgt, dass $d_e(G', G_-) \leq k$ ist. Dadurch haben wir jetzt eine gültige Instanz $I' = (G_-, G', S, k, r)$ von Mon-DVC.

Um die Reduktion von DVC auf Mon-DVC zu zeigen, zeigen wir nun, dass I genau dann eine Ja-Instanz von DVC ist, wenn I' eine Ja-Instanz von Mon-DVC ist. Dabei zeigen wir zuerst, dass I' eine Ja-Instanz von Mon-DVC ist, wenn I eine Ja-Instanz von DVC ist. Sei dazu I eine Ja-Instanz von DVC und das Vertex Cover S' eine Lösung für I. Wir bilden ein $S'' = S \cup S'$, sodass $d_v(S, S'') \leq r$ und $S' \subseteq S''$. Daraus folgt, dass S'' ein Vertex Cover für G' ist. Somit ist I' eine Ja-Instanz von Mon-DVC.

Nun zeigen wir, dass I eine Ja-Instanz von DVC ist, wenn I' eine Ja-Instanz von Mon-DVC ist. Dazu sei I' eine Mon-DVC-Instanz. Wenn I' eine Ja-Instanz von Mon-DVC ist, weil ein entsprechendes S' existiert, ist I aufgrund dieses S' auch eine Ja-Instanz von DVC.

Da sowohl die Reduktion von Mon-DVC auf DVC als auch die Reduktion von DVC auf Mon-DVC in Polynomialzeit durchgeführt werden kann, ist DVC polynomialzeitäquivalent zu Mon-DVC. □

Lemma 2. Monotone Dynamic Vertex Cover *ist fixed-parameter tractable bezüglich der Edge-Edit-Distance k, da es in der Laufzeit $\mathcal{O}(2^k n^2 log(n))$ gelöst werden kann.*

Beweis. Um zu zeigen, dass Mon-DVC fixed-parameter tractable bezüglich k ist, nehmen wir eine Mon-DVC-Instanz (G, G', S, k, r). Wir bilden den Graphen $H = (V, E_+)$ aus G' mit $V = V(G')$ und $E_+ = E(G') \setminus E(G)$, sodass in E_+ genau die Kanten enthalten sind, die sich in $E(G')$ aber nicht in $E(G)$ befinden. Da die Edge-Edit-Distance $d_e(G, G') = |E(G) \setminus E(G') \cup E(G') \setminus E(G)| \leq k$ ist, ist $|E_+| \leq k$. Nun entfernen wir alle Kanten aus H, die bereits durch das Vertex Cover S abgedeckt sind. Für den Graphen H lässt sich danach mittels Brute-Force ein kleinstes Vertex Cover T finden. Der Brute-Force-Algorithmus hat dabei die Laufzeit $\mathcal{O}(2^k)$, da in E_+ maximal k Kanten enthalten sind und es für jede Kante 2 Möglichkeiten gibt sie zu dominieren. Jetzt bilden wir ein Vertex Cover $T' = S \cup T$ für G'. Da T das kleinstmögliche Vertex Cover für H ist, ist T' ein Vertex Cover, welches die kleinste Vertex-Edit-Distance $d_v(S, T')$ hat. Wenn nun $d_v(S, T') \leq r$ ist, so existiert ein Vertex Cover $S' = T'$ für G', für das $d_v(S, S') \leq r$ gilt. Ist dies nicht der Fall existiert kein solches Vertex Cover. Somit ist Mon-DVC fixed-parameter tractable bezüglich k. □

Satz 3. Dynamic Vertex Cover *ist NP-vollständig, kann in der Laufzeit $\mathcal{O}(2^k n^2 log(n))$ gelöst werden und ist somit fixed-parameter tractable bezüglich der Edge-Edit-Distance k.*

Beweis. Aus Lemma 1 folgt, dass DVC polynomialzeitäquivalent zu Mon-DVC ist. Aus Lemma 2 folgt, dass Mon-DVC fixed-parameter tractable bezüglich k ist und in der Laufzeit $\mathcal{O}(2^k n^2 \log(n))$ gelöst werden kann. Da die Reduktion von DVC auf Mon-DVC die Laufzeit $\mathcal{O}(n^2 \log(n))$ hat und Mon-DVC innerhalb der Laufzeit $\mathcal{O}(2^k n^2 \log(n))$ gelöst werden kann, kann DVC ebenfalls in der Laufzeit $\mathcal{O}(2^k n^2 \log(n))$ gelöst werden und ist somit fixed-parameter tractable bezüglich k. Wie wir weiter oben in Kapitel 3 bereits festgestellt haben, ist DVC NP-vollständig, da Vertex Cover NP-vollständig ist [9]. □

Mit dem Wissen, dass DVC fixed-parameter tractable bezüglich k ist, könnten wir nun eine Turbo-Charging-Heuristik für Vertex Cover entwickeln, wie wir es in Kapitel 4 anhand der Idee von Downey et al. [5] für Dominating Set tun werden.

Bemerkung. Abu-Khzam et al. [1] behaupten, dass DVC W[1]-schwer bezüglich des kombinierten Parameters (k, r) ist. Krithika et al. [13] haben jedoch unabhängig von dieser Arbeit in dem Beweis von Abu-Khzam et al. [1] einen Fehler gefunden und gezeigt, dass DVC fixed-parameter tractable bezüglich der Edge-Edit-Distance k ist und innerhalb der Laufzeit $\mathcal{O}^*(1, 1740^k)$ gelöst werden kann. Dabei ist $\mathcal{O}^*(1, 1740^k)$ gleichbedeutend mit $\mathcal{O}(1, 1740^k p(n))$, wobei p ein Polynom ist.

Restricted Size Dynamic Vertex Cover

Auch für das Problem DVC schauen wir uns die größenbeschränkte Variante Restricted Size Dynamic Vertex Cover an. Restricted Size Dynamic Vertex Cover unterscheidet sich lediglich in einem Punkt von DVC. Wie auch schon bei RSDDS soll für die neue Lösung S' gelten, dass sie nicht größer sein darf als S. Es muss also gelten, dass $|S'| \leq |S|$ ist. Dadurch wird, wie bereits erwähnt, dem vorgebeugt, dass bei mehrmaliger Hintereinanderausführung die Lösung sehr groß werden kann.

Restricted Size Dynamic Vertex Cover (RSDVC)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Graph $G = (V, E)$; ein Graph $G' = (V, E')$ mit $d_e(G, G') \leq k$; ein Vertex Cover $S \subseteq V$ für G.
Frage:	Existiert ein Vertex Cover $S' \subseteq V$ für G', sodass $d_v(S, S') \leq r$ und $\|S'\| \leq \|S\|$ ist?

Satz 4. Restricted Size Dynamic Vertex Cover *ist NP-vollständig und W[1]-schwer bezüglich des kombinierten Parameters (k, r) aus der Edge-Edit-Distance k und der Vertex-Edit-Distance r.*

Um zu zeigen, dass RSDVC NP-vollständig und W[1]-schwer bezüglich des kombinierten Parameters (k, r) ist, werden wir das NP-vollständige und bezüglich der Größe k W[1]-schwere Problem Independent Set [3] [6] auf RSDVC reduzieren.

Beim Problem Independent Set ist ein Graph gegeben und gesucht wird ein Independent Set, also eine Knotenmenge S, für die gilt, dass kein Knoten $u \in S$ mit einem anderen Knoten $v \in S$ benachbart ist.

Independent Set (IS)

Eingabe:	Ein Graph $G = (V, E)$ und eine natürliche Zahl $k \in \mathbb{N}$.
Frage:	Existiert ein Independent Set $S \subseteq V$ für G, sodass $\|S\| = k$ und für alle $u, v \in S$ gilt, dass $\{u, v\} \notin E$?

Beweis. Um zu zeigen, dass RSDVC W[1]-schwer bezüglich des kombinierten Parameters (k, r) ist, nutzen wir die Konstruktion von Abu-Khzam et al. [1], mit der sie zeigen wollten, dass DVC W[1]-schwer bezüglich des kombinierten Parameters (k, r) ist.

Dazu reduzieren wir die Independent Set-Instanz $A = (H, p)$ mit dem Graphen H und einer natürlichen Zahl $p \in \mathbb{N}$ auf die folgende RSDVC-Instanz:

$$B = (G, G', S = V(H), k = \tfrac{1}{2}p(p+1), r = |S|),$$

wobei G der Graph ist, der entsteht, wenn man zu H noch $p + 1$ isolierte Knoten hinzufügt und G' der Graph, der entsteht, wenn aus den $p + 1$ isolierten Knoten eine Clique gebildet wird. Infolge dessen, dass $S = V(H)$ gilt, ist S ein Vertex Cover für den Graphen H. Zum besseren Verständnis ist die Konstruktion des Graphen G in Abbildung 5 dargestellt.

Da $E(G) = E(H)$ ist S auch ein Vertex Cover für G. Außerdem gilt, dass $d_e(G, G') \leq k$. Da $r = |S|$, muss r nicht weiter betrachtet werden, denn

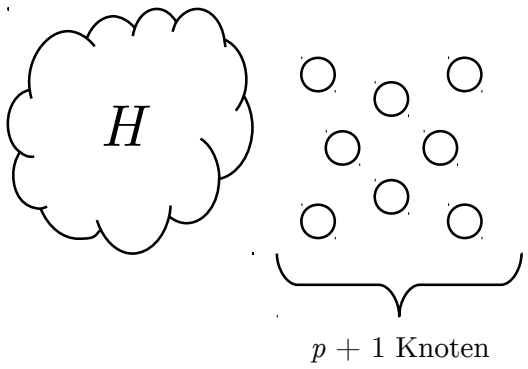

Abbildung 5: Aufbau des Graphen G.

damit $d_v(S, S') > |S|$ ist, müsste $|S'| > |S|$ sein.

Ein Vertex Cover S' für G' muss also p Knoten in der Clique enthalten. Da $|S'| \leq |S|$ sein soll, folgt daraus, dass aus dem Rest des Graphen G' nur noch maximal $|S| - p$ Knoten in das Vertex Cover S' aufgenommen werden können. Der Graph G' ohne die Clique entspricht genau dem Graphen H. Somit existiert ein Vertex Cover S' für G' genau dann, wenn ein Vertex Cover S_H für H existiert, welches höchstens $|S|-p$ Knoten hat. Angenommen es existiere ein solches Vertex Cover S_H. Dann würden die Knoten $E(H) \setminus S_H$ ein Indepentent Set I bilden, welches mindestens p groß ist. Ein solches Independent Set I existiert also genau dann, wenn ein S_H und somit ein S' existiert.

Somit haben wir die INDEPENDENT SET-Instanz A auf die RSDVC-Instanz B reduziert. Da es W[1]-schwer ist, eine Lösung für A zu finden [3] [6], ist es folglich auch W[1]-schwer eine Lösung für B zu finden. Außerdem kann in Polynomialzeit überprüft werden, ob eine Menge T ein Vertex Cover für G ist. Deshalb gilt zusätzlich RSDVC $\in$ NP und somit ist RSDVC NP-vollständig. □

Der Vorteil des dynamischen Problems DVC, dass es fixed-parameter tractable bezüglich k ist, geht also auch bei RSDVC wieder verloren.

3.3 Dynamic Set Cover

Als nächstes Problem untersuchen wir DYNAMIC SET COVER. Wie wir gleich sehen werden, hat es große Ähnlichkeiten mit DDS. Beim Problem DYNAMIC SET COVER haben wir ein Universum $\mathcal{U}$, in dem einzelne Elemente $u \in \mathcal{U}$ enthalten sind. Außerdem haben wir eine Menge $\mathcal{F}$, die m Teilmengen $T_i \subseteq \mathcal{U}$ enthält und eine Menge $\mathcal{F}'$, die m Teilmengen $T'_i \subseteq \mathcal{U}$ enthält, wobei $1 \leq i \leq m$. Dabei sind die Mengen $\mathcal{F}$ und $\mathcal{F}'$ „ähnlich“. Das bedeutet, dass die *Set-Edit-Distance*

$$d_s(\mathcal{F}, \mathcal{F}') = \Sigma_{j=1}^{m} |(T_j \in \mathcal{F}) \setminus (T'_j \in \mathcal{F}')| + |(T'_j \in \mathcal{F}') \setminus (T_j \in \mathcal{F})|$$

kleiner oder gleich einem vorgegebenen $k \in \mathbb{N}$ ist. Außerdem ist ein Set Cover $S \subseteq \mathcal{F}$ für das Universum $\mathcal{U}$ gegeben. Gesucht ist nun ein Set Cover $S' \subseteq \mathcal{F}'$ für das Universum $\mathcal{U}$. Dabei soll das Set Cover S' „ähnlich“ zu dem Set Cover S sein. Ähnlichkeit bedeutet in diesem Fall, dass die *Cover-Edit-Distance*

$$d_c(S, S') = |\{T_i \in S \mid T'_i \notin S'\} \cup \{T'_i \in S' \mid T_i \notin S\}|$$

kleiner gleich einem vorgegebenen $r \in \mathbb{N}$ ist.

Dynamic Set Cover (DSC)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Universum $\mathcal{U}$; zwei Mengen $\mathcal{F}, \mathcal{F}'$ von jeweils m Teilmengen $T_i \in \mathcal{F}$ beziehungsweise $T_i' \in \mathcal{F}'$ mit $T_i, T_i' \subseteq \mathcal{U}$ und $1 \leq i \leq m$, wobei $d_s(\mathcal{F}, \mathcal{F}') \leq k$; ein Set Cover $S \subseteq \mathcal{F}$ für $\mathcal{U}$.
Frage:	Existiert ein Set Cover $S' \subseteq \mathcal{F}'$ für $\mathcal{U}$, sodass $d_c(S, S') \leq r$?

Für das Problem DSC definieren wir $n = |U| + m$.

Lemma 3. Dynamic Set Cover *ist polynomialzeitäquivalent zu* Dynamic Dominating Set.

Beweis. Um zu beweisen, dass DSC und DDS polynomialzeitäquivalent sind, zeigen wir polynomielle Reduktionen in beide Richtungen.

Reduktion: DSC auf DDS Zunächst reduzieren wir DSC auf DDS. Dazu sei uns eine DSC-Instanz $I = (\mathcal{U}, \mathcal{F}, \mathcal{F}', S_{\text{DSC}}, k, r)$ mit den Mengen $\mathcal{F} = \{T_1, \ldots, T_m\}$ und $\mathcal{F}' = \{T_1', \ldots, T_m'\}$ gegeben, die wir auf die DDS-Instanz $I' = (G = (V, E), G' = (V, E'), S_{\text{DDS}}, k, r)$ reduzieren, wobei

- $T_V = \{t_1, \ldots, t_m\}$,
- $V = \mathcal{U} \cup T_V \cup \{w\}$,
- $E = \{\{t_i \in T_V, u \in \mathcal{U}\} \mid u \in T_i\} \cup \{\{t_i \in T_V, w\} \mid T_i \in \mathcal{F}\}$,
- $E' = \{\{t_i \in T_V, u \in \mathcal{U}\} \mid u \in T_i'\} \cup \{\{t_i \in T_V, w\} \mid T_i' \in \mathcal{F}'\}$,
- $S_{\text{DDS}} = \{t_i \in T_V \mid T_i \in S_{\text{DSC}}\}$.

In Abbildung 6 ist der Aufbau der DDS-Instanz I' zur Veranschaulichung grafisch dargestellt. Der Knoten w ist dabei dafür zuständig, dass alle $t_i \in T_V$ dominiert werden.

Aus der Konstruktion von S_{DDS} folgt, dass es ein Dominating Set für den Graphen G ist. Da $d_s(\mathcal{F}, \mathcal{F}') \leq k$ gelten muss, damit I eine gültige DSC-Instanz ist, gilt auch $d_e(G, G') \leq k$. Somit ist I' eine gültige DDS-Instanz. Um nun die Reduktion von DSC auf DDS zu zeigen, zeigen wir, dass I genau dann eine Ja-Instanz von DSC ist, wenn I' eine Ja-Instanz von DDS ist.

„⇒“ Dabei zeigen wir zuerst, dass I' mit $S_{\text{DDS}} = \{t_i \in T_V \mid T_i \in S_{\text{DSC}}\} \cup \{w\}$ eine Ja-Instanz von DDS ist, wenn I eine Ja-Instanz von DSC ist. Sei dazu I eine Ja-Instanz von DSC, das Set Cover S'_{DSC} eine Lösung für I und $S'_{\text{DDS}} = \{t_i \in T_V \mid T_i' \in S'_{\text{DSC}}\} \cup \{w\}$. Dadurch dass $w \in S'_{\text{DDS}}$ und w im Graphen G' mit allen Knoten $t_i \in T_V$, die die Teilmengen $T_i' \in \mathcal{F}'$ repräsentieren, verbunden ist, sind alle Knoten $t_i \in T_V$ und w von S'_{DDS} dominiert. Weiterhin ist jeder Knoten $u \in \mathcal{U} \subseteq V$ mit allen Knoten $t_i \in T_V \subseteq V$ verbunden. Da $\bigcup_{T_i' \in S'_{\text{DSC}}} = \mathcal{U}$ gilt, gilt auch $N_{G'}(S'_{\text{DDS}}) = \mathcal{U}$.

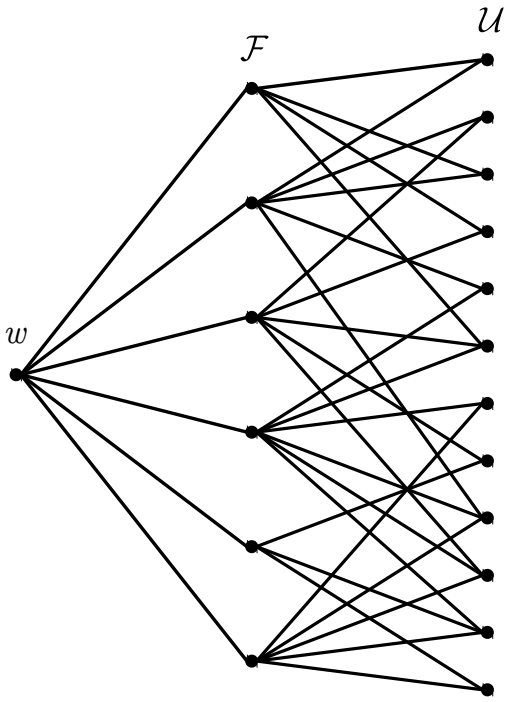

Abbildung 6: Illustration der Konstruktion der DDS-Instanz I'.

Daraus folgt, dass alle Knoten $u \in \mathcal{U} \subseteq V$ von S'_{DDS} dominiert werden. Also ist S'_{DDS} ein Dominating Set für den Graphen G'. Da $d_c(S_{\text{DSC}}, S'_{\text{DSC}}) \leq r$ gilt, gilt auch $d_v(S_{\text{DDS}}, S'_{\text{DDS}}) \leq r$ und somit ist I' eine Ja-Instanz von DDS.

„$\Leftarrow$“ Nun zeigen wir, dass I mit $S_{\text{DSC}} = S_T \cup S_u$, wobei $S_T = \{T_i \in \mathcal{F} \mid t_i \in S_{\text{DDS}}\}$ ist, und in der Menge S_u für jedes $u \in S_{\text{DDS}}$ die Teilmenge $T_i \in \mathcal{F}$, die unter denen, die dieses u enthalten, den kleinsten Index i hat, enthalten ist, sodass

$$S_u = \{T_i \in \mathcal{F} \mid \exists u \in T_i \cap S_{\text{DDS}}.\nexists T_j \in \mathcal{F}.u \in T_j \wedge j < i\}$$

ist, eine Ja-Instanz von DSC ist, wenn I' eine Ja-Instanz von DDS ist. Dazu sei I' eine Ja-Instanz von DDS, das Dominating Set S'_{DDS} eine Lösung für I' und $S'_{\text{DSC}} = S'_T \cup S'_u$ mit $S'_T = \{T'_i \in \mathcal{F}' \mid t_i \in S'_{\text{DDS}}\}$ und

$$S'_u = \{T'_i \in \mathcal{F}' \mid \exists u \in T'_i \cap S'_{\text{DDS}}.\nexists T'_j \in \mathcal{F}'.u \in T'_j \wedge j < i\}.$$

Dadurch haben wir erreicht, dass S'_{DSC} nur aus Teilmengen $T'_i \in \mathcal{F}'$ des Universums $\mathcal{U}$ besteht. Da jedes $t_i \in S'_{\text{DDS}}$ genau die $u \in \mathcal{U}$ dominiert, die das entsprechende $T'_i \in S'_T$ enthält, sind alle $u \in \bigcup_{T'_i \in S'_T}$ in Teilmengen im Set Cover S'_{DSC} enthalten. Alle anderen $u \in \mathcal{U}$ werden durch das Dominating Set S'_{DDS} dominiert, weil sie Teil des Dominating Sets sind.

Durch die Menge S'_u ist allerdings sichergestellt, dass im Set Cover S'_{DSC} nur Teilmengen $T'_i \subseteq \mathcal{U}$ enthalten sind. Daraus folgt, dass $\bigcup_{T_i \in S'_{\text{DSC}}} = \mathcal{U}$ gilt, und S'_{DSC} somit ein Set Cover für $\mathcal{U}$ ist. Da $d_v(S_{\text{DDS}}, S'_{\text{DDS}}) \leq r$ gilt, gilt auch $d_c(S_{\text{DSC}}, S'_{\text{DSC}}) \leq r$ und somit ist I eine Ja-Instanz von DSC. Daraus folgt, dass sich DSC in Polynomialzeit auf DDS reduzieren lässt.

Reduktion: DDS auf DSC Jetzt reduzieren wir DDS auf DSC. Dazu sei eine DDS-Instanz $I = (G = (V, E), G' = (V, E'), S_{\text{DDS}}, k, r)$ gegeben. Nun erstellen wir eine DSC-Instanz $I' = (\mathcal{U} = V, \mathcal{F}, \mathcal{F}', S_{\text{DSC}}, k, r)$ mit $\mathcal{F} = \{N_G[\{v\}] \mid v \in V\}$, $\mathcal{F}' = \{N_{G'}[\{v\}] \mid v \in V\}$ und $S_{\text{DSC}} = \{N_G[\{v\}] \mid v \in S_{\text{DDS}}\}$. Aufgrund dessen, dass S_{DDS} ein Dominating Set für den Graphen G ist, ist S_{DSC} ein Set Cover für das Universum $\mathcal{U}$. Da $d_e(G, G') \leq k$ gelten muss, damit I eine gültige DDS-Instanz ist, gilt wegen der Konstruktion von I' auch $d_s(\mathcal{F}, \mathcal{F}') \leq k$. Somit ist I' eine gültige DSC-Instanz. Um nun die Reduktion von DSC auf DDS zu zeigen, zeigen wir, dass I genau dann eine Ja-Instanz von DDS ist, wenn I' eine Ja-Instanz von DSC ist.

„⇒“ Dabei zeigen wir zuerst, dass I eine Ja-Instanz von DDS ist, wenn I' eine Ja-Instanz von DSC ist. Dazu sei I' eine Ja-Instanz von DSC, das Set Cover S'_{DSC} eine Lösung für I' und $S'_{\text{DDS}} = \{v \in V \mid N_{G'}[\{v\}] \in S'_{\text{DSC}}\}$. Aus der Konstruktion von S'_{DDS} geht hervor, dass es ein Dominating Set für G' ist. Da $|S_{\text{DSC}}| = |S_{\text{DDS}}|$ und $|S'_{\text{DSC}}| = |S'_{\text{DDS}}|$ gelten und außerdem $d_c(S_{\text{DSC}}, S'_{\text{DSC}}) \leq r$ gilt, damit S'_{DSC} eine Lösung der DSC-Instanz I ist, gilt auch $d_v(S_{\text{DDS}}, S'_{\text{DDS}}) \leq r$. Somit ist die DDS-Instanz I durch die Lösung S'_{DDS} eine Ja-Instanz.

„⇐“ Nun zeigen wir, dass I' eine Ja-Instanz von DSC ist, wenn I eine Ja-Instanz von DDS ist. Dazu sei I eine Ja-Instanz von DDS, das Dominating Set S'_{DDS} eine Lösung für I und $S'_{\text{DSC}} = \{N_{G'}[\{v\}] \mid v \in S'_{\text{DDS}}\}$. Aus der Konstruktion von S'_{DSC} geht hervor, dass es ein Set Cover für $\mathcal{U}$ ist. Da $|S_{\text{DDS}}| = |S_{\text{DSC}}|$ und $|S'_{\text{DDS}}| = |S'_{\text{DSC}}|$ gelten und außerdem gilt, dass $d_v(S_{\text{DDS}}, S'_{\text{DDS}}) \leq r$ ist, damit S'_{DDS} eine Lösung der DDS-Instanz I ist, gilt auch hier wieder, dass $d_c(S_{\text{DSC}}, S'_{\text{DSC}}) \leq r$. Somit ist die DSC-Instanz I' durch die Lösung S'_{DSC} eine Ja-Instanz. Daraus folgt, dass sich DDS in Polynomialzeit auf DSC reduzieren lässt.

Da sowohl die Reduktion von DSC auf DDS als auch die Reduktion von DDS auf DSC in Polynomialzeit durchgeführt werden kann, ist DSC polynomialzeitäquivalent zu DDS. □

Satz 5. Dynamic Set Cover *ist NP-vollständig, kann innerhalb der Laufzeit* $\mathcal{O}(\binom{k+2^k}{k} n^3 log^2(n))$ *gelöst werden und ist somit fixed-parameter tractable bezüglich der Set-Edit-Distance* k.

Beweis. Aus Lemma 3 folgt, dass DSC polynomialzeitäquivalent zu DDS ist. Da Downey et al. [5] gezeigt haben, dass DDS NP-vollständig und fixed-parameter tractable bezüglich k ist, da es in der Laufzeit $\mathcal{O}(\binom{k+2^k}{k} n^3 \log^2(n))$ gelöst werden kann, folgt aus der Polynomialzeitäquivalenz von DSC und

DDS, dass DSC ebenfalls NP-vollständig und fixed-parameter tractable bezüglich k ist. Da DSC in $\mathcal{O}(n^3\log^2(n))$ auf DDS reduziert werden kann und DDS in der Laufzeit $\mathcal{O}(\binom{k+2^k}{k}n^3\log^2(n))$ gelöst werden kann, kann DSC ebenfalls innerhalb der Laufzeit $\mathcal{O}(\binom{k+2^k}{k}n^3\log^2(n))$ gelöst werden. □

Somit könnten wir auch für Set Cover eine Turbo-Charging-Heuristik entwickeln. Im Kapitel 4.4 werden wir zeigen, wie sich durch ein paar einfache Anpassungen die von uns nach der Idee von Downey et al. [5] entwickelte Turbo-Charging-Heuristik für Dominating Set auch auf Set Cover anwenden lässt.

Restricted Size Dynamic Set Cover

Natürlich schauen wir uns auch für DSC die größenbeschränkte Variante Restricted Size Dynamic Set Cover an. Das Problem Restricted Size Dynamic Set Cover unterscheidet sich, wie auch die vorhergehenden größenbeschränkten Varianten, in nur einem Punkt von DSC. Für die neue Lösung S' wird zusätzlich gefordert, dass $|S'| \leq |S|$ ist. Dadurch wird, wie bei den anderen Problemen auch, die Lösung bei wiederholter Ausführung klein gehalten.

Restricted Size Dynamic Set Cover (RSDSC)

Eingabe:	Zwei natürliche Zahlen $k, r \in \mathbb{N}$; ein Universum $\mathcal{U}$; zwei Mengen $\mathcal{F}, \mathcal{F}'$ von jeweils m Teilmengen $T_i \in \mathcal{F}$ beziehungsweise $T_i' \in \mathcal{F}'$ mit $T_i, T_i' \subseteq \mathcal{U}$ und $1 \leq i \leq m$, wobei $d_s(\mathcal{F}, \mathcal{F}') \leq k$; ein Set Cover $S \subseteq \mathcal{F}$ für $\mathcal{U}$.
Frage:	Existiert ein Set Cover $S' \subseteq \mathcal{F}'$ für $\mathcal{U}$, sodass $d_c(S, S') \leq r$ und $\|S'\| \leq \|S\|$ ist?

Für das Problem RSDSC definieren wir $n = |U| + m$.

Satz 6. Restricted Size Dynamic Set Cover *ist NP-vollständig und W[2]-schwer bezüglich der Set-Edit-Distance k.*

Der Beweis von Satz 6 ist sehr ähnlich zu der Reduktion von DDS auf DSC.

Beweis. Um zu beweisen, dass RSDSC W[2]-schwer bezüglich k ist, reduzieren wir RSDDS auf RSDSC. Dazu sei eine RSDDS-Instanz $I = (G = (V, E), G' = (V, E'), S_{\text{RSDDS}}, k, r)$ gegeben. Nun erstellen wir eine RSDSC-Instanz $I' = (\mathcal{U} = V, \mathcal{F}, \mathcal{F}', S_{\text{RSDSC}}, k, r)$ mit $\mathcal{F} = \{N_G[\{v\}] \mid v \in V\}$, $\mathcal{F}' = \{N_{G'}[\{v\}] \mid v \in V\}$ und $S_{\text{RSDSC}} = \{N_G[\{v\}] \mid v \in S_{\text{RSDDS}}\}$. Aufgrund dessen, dass S_{RSDDS} ein Dominating Set für den Graphen G ist, ist S_{RSDSC} ein Set Cover für das Universum $\mathcal{U}$. Da $d_e(G, G') \leq k$ gelten muss, damit I eine gültige RSDDS-Instanz ist, gilt wegen der Konstruktion von I'

auch $d_s(\mathcal{F}, \mathcal{F}') \leq k$. Somit ist I' eine gültige RSDSC-Instanz. Um nun die Reduktion von RSDSC auf RSDDS zu zeigen, zeigen wir, dass I eine Ja-Instanz von RSDDS ist, genau dann, wenn I' eine Ja-Instanz von RSDSC ist.

„⇒“ Dabei zeigen wir zuerst, dass I eine Ja-Instanz von RSDDS ist, wenn I' eine Ja-Instanz von RSDSC ist. Dazu sei I' eine Ja-Instanz von RSDSC, das Set Cover S'_{RSDSC} eine Lösung für I' und $S'_{\text{RSDDS}} = \{v \in V \mid N_{G'}[\{v\}] \in S'_{\text{RSDSC}}\}$. Aus der Konstruktion von S'_{RSDDS} geht hervor, dass es ein Dominating Set für G' ist. Da $|S_{\text{RSDSC}}| = |S_{\text{RSDDS}}|$ und $|S'_{\text{RSDSC}}| = |S'_{\text{RSDDS}}|$ gelten und außerdem gelten muss, dass $|S'_{\text{RSDSC}}| \leq |S_{\text{RSDSC}}|$ und $d_c(S_{\text{RSDSC}}, S'_{\text{RSDSC}}) \leq r$ ist, damit S'_{RSDSC} eine Lösung der RSDSC-Instanz I ist, gelten auch $|S'_{\text{RSDDS}}| \leq |S_{\text{RSDDS}}|$ und $d_v(S_{\text{RSDDS}}, S'_{\text{RSDDS}}) \leq r$. Somit ist die RSDDS-Instanz I durch die Lösung S'_{RSDDS} eine Ja-Instanz.

„⇐“ Nun zeigen wir, dass I' eine Ja-Instanz von RSDSC ist, wenn I eine Ja-Instanz von RSDDS ist. Dazu sei I eine Ja-Instanz von RSDDS, das Dominating Set S'_{RSDDS} eine Lösung für I und $S'_{\text{RSDSC}} = \{N_{G'}[\{v\}] \mid v \in S'_{\text{RSDDS}}\}$. Aus der Konstruktion von S'_{RSDSC} geht hervor, dass es ein Set Cover für $\mathcal{U}$ ist. Da $|S_{\text{RSDDS}}| = |S_{\text{RSDSC}}|$ und $|S'_{\text{RSDDS}}| = |S'_{\text{RSDSC}}|$ gelten und außerdem $|S'_{\text{RSDDS}}| \leq |S_{\text{RSDDS}}|$ und $d_v(S_{\text{RSDDS}}, S'_{\text{RSDDS}}) \leq r$ gelten müssen, damit S'_{RSDDS} eine Lösung der RSDDS-Instanz I ist, gelten auch $|S'_{\text{RSDSC}}| \leq |S_{\text{RSDSC}}|$ und $d_c(S_{\text{RSDSC}}, S'_{\text{RSDSC}}) \leq r$. Somit ist die RSDSC-Instanz I' durch die Lösung S'_{RSDSC} eine Ja-Instanz. Daraus folgt, dass sich RSDDS in Polynomialzeit auf RSDSC reduzieren lässt.

Da RSDDS W[2]-schwer bezüglich k ist [1], ist also RSDSC ebenfalls und W[2]-schwer bezüglich k. Außerdem kann in Polynomialzeit überprüft werden, ob eine Menge R eine Lösung für I' ist. Deshalb gilt zusätzlich RSDSC $\in$ NP und somit ist RSDSC NP-vollständig. □

Auch bei DSC ist also die größenbeschränkte Variante RSDSC nicht fixed-parameter tractable. Zusammengefasst lässt sich also sagen, dass alle drei betrachteten dynamischen Probleme fixed-parameter tractable bezüglich k sind. Kommt jedoch die Größenbeschränkung hinzu, sind sie alle W[1]-schwer bezüglich des kombinierten Parameters (k, r).

4 Turbo-Charging-Heuristik für Dominating Set

In diesem Kapitel werden wir die Turbo-Charging-Greedy-Heuristik für Dominating Set von Downey et al. [5] untersuchen und hinsichtlich ihrer Qualität verbessern. Dazu schauen wir uns zuerst die zugrundeliegende Greedy-Heuristik an und erklären im Anschluss die Idee hinter dem Turbo-Charging der Heuristik. Danach betrachten wir die Turbo-Charging-Greedy-Heuristik von Downey et al. [5] anhand ihres Pseudocodes genauer und setzen uns mit den Besonderheiten der Implementierung auseinander. Anschließend gehen wir auf unsere Verbesserungen der Turbo-Charging-Heuristik von

Downey et al. [5] ein und erläutern die Unterschiede. Am Ende des Kapitels schildern wir, wie sich diese Turbo-Charging-Greedy-Heuristik auf SET COVER anwenden lässt.

Bei einer Heuristik geht es darum möglichst schnell eine möglichst gute Lösung zu finden. Wenn das Problem an sich schon in einer annehmbaren Zeit gelöst werden kann, ist eine Heuristik also nicht sinnvoll, da sie meist keine optimale Lösung findet. Bei NP-schweren Problemen hingegen wird häufig auf Heuristiken zurückgegriffen. Da DOMINATING SET NP-vollständig ist [9], ist das Verwenden einer Heuristik für DOMINATING SET dementsprechend sinnvoll.

Greedy-Heuristik Zunächst widmen wir uns der Greedy-Heuristik und erläutern ihre Funktionsweise. Dazu betrachten wir die Knoten des Graphen, den wir dominieren wollen, und sortieren sie der Reihe nach, von dem Knoten mit dem höchsten Knotengrad ausgehend, bis zu dem Knoten mit dem kleinsten Knotengrad. Nun suchen wir den ersten Knoten v, der nicht von unserer zu bildenden Menge D dominiert wird. Haben wir einen solchen Knoten v gefunden, suchen wir in der geschlossenen Nachbarschaft von v nach dem Knoten w, der die meisten noch nicht von D dominierten Knoten dominieren würde. Diesen Knoten w fügen wir nun in die Menge D ein. Jetzt suchen wir erneut den Knoten v, der unter den noch nicht von D dominierten Knoten den höchsten Knotengrad hat und fügen wieder aus dessen Nachbarschaft den Knoten w in D ein, der die meisten noch nicht von D dominierten Knoten dominieren würde. Das wiederholen wir so lange, bis alle Knoten von D dominiert werden.

Turbo-Charging Das Besondere an der Heuristik von Downey et al. [5] ist jedoch das Turbo-Charging. Um zu verstehen, wie das funktioniert, erläutern wir im Folgenden die Idee hinter dem Turbo-Charging-Mechanismus von Downey et al. [5]: Wir haben eine Instanz $I = (G, s)$ des Problems DOMINATING SET und eine Heuristik, mithilfe derer wir ein Dominating Set S für I finden wollen. Dieses Dominating Set soll nicht mehr als s Elemente enthalten. Dabei ist zu beachten, dass k bisher die maximale Größe des Dominating Sets war. Da k im Folgenden jedoch anders definiert wird, ist nun s die Höchstanzahl der Elemente im Dominating Set.

Wenn wir nun die Heuristik auf I anwenden, kann es dazu kommen, dass wir unser s so klein gewählt haben, dass die von der Heuristik gefundene Menge R zwar schon s Elemente enthält, aber noch kein Dominating Set für G ist. Dann kommt Turbo-Charging ins Spiel.

Den Knoten $v \in V(G)$, den wir betrachtet haben, als R die Größe s erreicht hat, nennen wir nach Downey et al. [5] den *moment of regret*. Nun bilden wir aus den bisher betrachteten Knoten, wie in Abbildung 7 zu sehen, eine Knotenmenge $T \subseteq V(G)$, die wir mit $R \subseteq T$ bereits dominieren. Aus

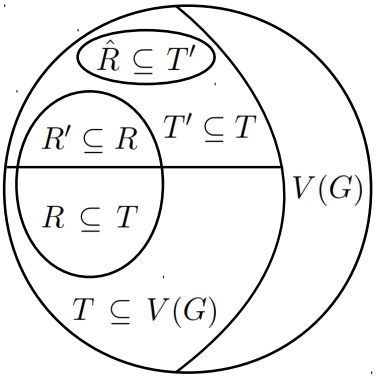

Abbildung 7: Schemenhafte Darstellung des Graphen G.

den k zuletzt betrachteten Knoten dieser Knotenmenge T bilden wir eine weitere Knotenmenge $T' \subseteq T$. Dabei gehen wir davon aus, dass die Heuristik die Knoten in einer wohldefinierten Reihenfolge abarbeitet und wir somit wissen, welche Knoten zuletzt betrachtet wurden. Die Knotenmenge T' nennen wir gemäß Downey et al. [5] das *Backup*, da wir sozusagen k Elemente in T zurückgehen, um ein neues Dominating Set R^* für T mit $|R^*| < |R|$ zu finden. Die Zahl k beschreibt also die Größe des Backups.

Mit den Elementen aus R, die nicht benötigt werden, um T zu dominieren, bilden wir eine weitere Menge $R' \subseteq R$. Dabei ist zu beachten, dass R' nicht unbedingt ein Dominating Set für T' ist. Nun suchen wir eine Menge $\hat{R} \subseteq T'$, sodass $R^* = \hat{R} \cup (R \setminus R')$ ein Dominating Set für T ist, sodass $|R^*| < |R|$ gilt.

Im Kapitel 4.1 erklären wir genauer, wie wir dieses Dominating Set R^* finden. Für den Augenblick können wir jedoch schon mal festhalten, dass der Algorithmus dazu fixed-parameter tractable bezüglich der Größe k der Knotenmenge T' ist.

Wenn ein Dominating Set R^* existiert, dominieren wir T mit einer kleineren Menge als vorher. Wenn wir nun $R = R^*$ setzen, gilt also wieder, dass $|R| < s$ ist. Wir können demzufolge mit der Heuristik fortfahren und weitere Elemente in R einfügen. Falls R, nachdem es wieder s Elemente enthält, immer noch kein Dominating Set für die G ist, können wir wieder Turbo-Charging anwenden, um R zu verkleinern. Diesen Vorgang wiederholen wir so lange, bis die Heuristik ein Dominating Set S für G gefunden hat, oder auch Turbo-Charging keine kleinere Teillösung R^* mehr findet.

4.1 Algorithmus

Nun gehen wir genauer auf die Turbo-Charging-Heuristik von Downey et al. [5] ein und entwickeln ausgehend von deren Idee eine Heuristik für Dom-

INATING SET. Unser Algorithmus bekommt als Eingabe einen Graphen G, eine natürliche Zahl k, die die Größe des Backups festlegt, und eine natürliche Zahl s, die die maximale Größe des Dominating Sets beschränkt. Der Algorithmus gibt uns aus, ob er ein Dominating Set der Größe s für G gefunden hat.

Dominating Set Heuristik (DSH)

Eingabe:	Ein Graph $G = (V, E)$; zwei natürliche Zahlen $k, s \in \mathbb{N}$, wobei k die Größe des Backups und s die maximale Größe des gesuchten Dominating Sets ist.
Ausgabe:	Wenn die Heuristik ein Dominating Set $D \subseteq V$ für G findet, sodass $\|D\| \leq s$ ist, wird *true* ausgegeben. Findet sie kein solches Dominating Set, wird *false* ausgegeben.

Da es eine Heuristik ist, kommt es vor, dass der Algorithmus ausgibt, dass er kein Dominating Set findet, welches höchstens s Knoten enthält, obwohl ein solches Dominating Set existiert. Um ein Dominating Set zu finden, vergrößert die Greedy-Heuristik die Zwischenlösung so lange, bis sie entweder ein Dominating Set für G ist, oder die maximale Größe s erreicht hat. Im zweiten Fall nutzt die Heuristik für das Turbo-Charging eine verschachtelte Unterprozedur: Sie ruft den Algorithmus Greedy-DS (Algorithmus 2) auf, der den Algorithmus Mon-DDS (Algorithmus 3) aufruft, der wiederum den Algorithmus Dom-A-VC (Algorithmus 4) aufruft, der dann die Zwischenlösung schließlich verkleinert, falls das durch den Turbo-Charging-Mechanismus möglich ist.

Bei seiner Suche nach einem Dominating Set L_D sortiert der Algorithmus DSH (Algorithmus 1) in Zeile 7 die Knoten im Graphen G in einer Liste L absteigend nach ihrem Knotengrad. Danach geht er diese Liste durch. Sobald er auf einen Knoten v trifft, der noch nicht dominiert ist, sucht der Algorithmus DSH (Algorithmus 1) in Zeile 19 in der geschlossenen Nachbarschaft von v nach dem Knoten mit dem höchsten Knotengrad, fügt ihn ins Dominating Set L_D ein und sucht wieder den nächsten noch nicht dominierten Knoten. Das macht er so lange, bis entweder alle Knoten dominiert sind oder die maximale Größe s des Dominating Sets L_D erreicht ist.

Wenn die maximale Größe des Dominating Sets L_D erreicht ist und der Graph G noch nicht dominiert ist, werden die Vorbereitungen fürs Turbo-Charging getroffen. Diese beginnen ab Zeile 24 des Algorithmus DSH (Algorithmus 1). Zu diesem Zeitpunkt ist die Liste L wie in Abbildung 8 aufgebaut. Ausgehend von dem ersten nicht dominierten Knoten v_{l+k+1} in L geht der Algorithmus in dieser Liste k bereits dominierte Knoten bis zum Knoten v_l zurück. Es wird eine Menge D_1 erzeugt, die die Knoten enthält, die als erstes ins Dominating Set L_D aufgenommen wurden, sodass D_1 alle Knoten bis einschließlich dem Knoten v_l in der Liste L dominiert. Außerdem wird eine Menge D_2 erzeugt, welche alle Knoten aus L_D enthält, die

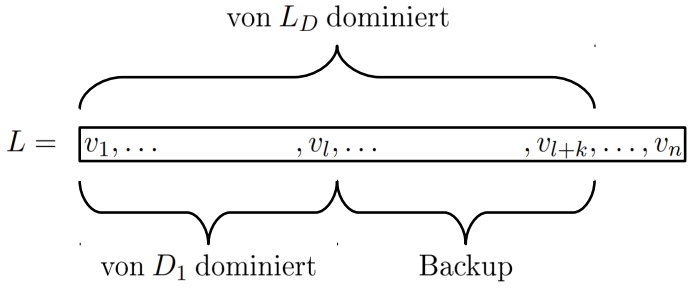

Abbildung 8: Veranschaulichung der Liste L.

nicht in D_1 enthalten sind. Für das Turbo-Charging wird in Zeile 41 des Algorithmus DSH (Algorithmus 1) der Graph G zusammen mit der Liste L, dem Knoten $v = v_{l+k}$, der Zahl k und den Mengen D_1 und D_2 dem Algorithmus Greedy-DS (Algorithmus 2) übergeben. Liefert das Turbo-Charging eine Lösung, macht der Algorithmus DSH (Algorithmus 1) wieder mit Zeile 11 weiter. Da die Liste L_D nach dem Turbo-Charging unsortiert ist, muss die Sortierung in Zeile 15 des Algorithmus DSH (Algorithmus 1) wiederhergestellt werden. Danach können weitere Elemente in das nun kleinere Dominating Set L_D eingefügt werden. Das abwechselnde Turbo-Charging und Erweitern von L_D geschieht so lange, bis entweder ein Dominating Set für den Graphen G gefunden werden konnte, oder auch das Turbo-Charging das Dominating Set nicht weiter verkleinern kann.

Algorithmus 1 Dominating Set Heuristik

1: **procedure** DSH(G, k, s)
2: **if** $|V(G)| \leq s$ **then return** *true*
3: **if** $s = 0$ **then return** *false*
4: $k' \leftarrow k$
5: $L \leftarrow$ eine leere Liste
6: **for all** $u \in V(G)$ **do** Füge u in L ein.
7: Sortiere L, sodass für alle $u, w \in L$ gilt, dass, wenn $\deg(u) > \deg(w)$ gilt, u in L vor w steht.
8: $i \leftarrow 1$
9: $L_D \leftarrow$ eine leere Liste
10: **while** $(|L_D| > 0)$ oder $i = 1$ **do** ▷ bis durch Turbo-Charging kein Dominating Set gefunden wird
11: $i \leftarrow |L_D| + 1$
12: **for all** $u \in V(G)$ **do** Markiere u als nicht dominiert.

13: **for all** $u \in V(G)$, sodass gilt $u \in N_G[L_D]$ **do**
14: Markiere u als dominiert.
15: Sortiere L_D chronologisch nach der Reihenfolge des Einfügens der Knoten.
16: **while** $i \leq s$ **do** ▷ bis die maximale Größe des Dominating Sets erreicht ist
17: **if** alle $u \in L$ sind dominiert **then return** *true*
18: $v \leftarrow$ erstes nicht dominiertes Element in L
19: $v \leftarrow$ Knoten in $N_G[\{v\}]$ mit den meisten nicht dominierten Nachbarn
20: **for all** $u \in N_G[\{v\}]$ **do** Markiere u als dominiert.
21: **if** alle $u \in L$ sind dominiert **then return** *true*
22: Füge v hinten an L_D an.
23: $i \leftarrow i + 1$
▷ Es folgt die Vorbereitung für Turbo-Charging.
24: $v \leftarrow$ erstes nicht dominiertes Element in L
25: $L' \leftarrow$ eine Kopie von L
26: **for all** $u \in L'$, sodass gilt u steht in L' hinter v **do**
27: Lösche u aus L'.
28: Lösche v aus L'.
29: **if** $|L'| < k$ **then** $k' \leftarrow |L'|$
30: $v \leftarrow$ letztes Element von L'
31: Lösche die letzten k' Elemente aus L'.
32: $D_1 \leftarrow \emptyset$
33: $j \leftarrow 1$
34: **while** Es existiert ein $u \in L'$, sodass gilt $u \notin N_G[D_1]$ **do**
35: $D_1 \leftarrow D_1 \cup \{$das j-te Element in $L_D\}$
36: $j \leftarrow j + 1$
37: $x \leftarrow j$-tes Element in L_D
38: $D_2 \leftarrow \{x\}$
39: **for all** u, sodass gilt u steht in L_D hinter x **do** $D_2 \leftarrow D_2 \cup \{u\}$
40: $L_D \leftarrow$ eine leere Liste
41: **for all** $u \in$ Greedy-DS$(G, L, v, k', D_1 \cup D_2)$ **do** ▷ Turbo-Charging
42: Füge u in L_D ein.
43: **return** *false*

Turbo-Charging Falls die Greedy-Heuristik bereits s Elemente in die Zwischenlösung L_D eingefügt hat und L_D allerdings noch kein Dominating Set für den Graphen G ist, wird, um das Turbo-Charging durchzuführen, der Algorithmus Greedy-DS (Algorithmus 2) aufgerufen. Er erhält als Eingabe einen Graphen H, eine nach absteigendem Knotengrad sortierte Liste L aller Knoten in $V(H)$, einen Knoten $v = v_{l+k}$ vor und einschließlich dem alle Knoten in L durch D dominiert sind, eine Zahl k, welche die Größe des Backups angibt und eine Menge $D = D_1 \cup D_2$, wobei D_1 alle Knoten bis einschließlich dem Knoten v_l in L dominiert. Falls ein Dominating Set D' existiert, welches alle Knoten in D und alle Knoten in L bis einschließlich dem Knoten v dominiert, wird dieses D' zurückgegeben. Ist dies nicht der Fall, wird die leere Menge zurückgegeben.

Greedy Improvement for Dominating Set (Greedy-DS)

Eingabe: Ein Graph $H = (V, E)$; eine nach absteigendem Knotengrad sortierte Liste $L = (v_1, \ldots, v_l, \ldots, v_{k+l} = v, \ldots, v_n)$ mit $v_i \in V$ und $n = |V|$, wobei v der moment of regret und k der Umfang des Backups ist; ein Dominating Set $D \subseteq V$ für die Knotenmenge $V' = \{v_1, \ldots, v\}$, wobei $D = D_1 \cup D_2$; ein Dominating Set $D_1 \subseteq D$ für die Knotenmenge $\{v_1, \ldots, v_l\}$; eine Knotenmenge $D_2 \subseteq D$, wobei $|D_2| \leq k$.

Ausgabe: Wenn ein Dominating Set $D' \subseteq V$ für die Knotenmenge $V' \cup D$ existiert, sodass $D_1 \subseteq D'$ und $|D'| < |D|$, wird D' ausgegeben. Existiert kein solches Dominating Set, wird die leere Menge ausgegeben.

Um ein Dominating Set D' zu finden, welches die Knoten in der Menge $U = \{v_1, \ldots, v\} \cup D$ dominiert, erzeugt der Algorithmus Greedy-DS (Al-

Algorithmus 2 Greedy Improvement for Dominating Set

```
1:  procedure Greedy-DS(H, L, v, k, D = D_1 ∪ D_2)
2:      if |D_2| = 1 then
3:          if {v_{l+1}, ..., v} ⊆ D_1 then return ∅
4:          return D_1
5:      if |D_1| ≤ 0 then return ∅
6:      x ← ein beliebiges Element in D_1
7:      U ← {v_1, ..., v} ∪ D
8:      G' ← der induzierte Subgraph von H auf U
9:      G ← G'
10:     for all u ∈ {v_{l+1}, ..., v} ∪ D_2 do
11:         if {x, u} ∉ E(G) then E(G) ← E(G) ∪ {{x, u}}
12:     return Mon-DDS(G, G', D_1, 2k, r = |D_2| − 1)
```

gorithmus 2) zwei identische induzierte Subgraphen G und G' von H auf U. Er wählt einen beliebigen Knoten $x \in D_1$ und fügt in G alle nicht bereits existierenden Kanten zwischen diesem Knoten x und jedem Knoten in der Menge $W = \{v_{l+1}, \ldots, v\} \cup D_2$ ein. Daraus folgt, dass D_1 ein Dominating Set für G ist. Da $|W| \leq 2k$ ist, ist die Edge-Edit-Distance $d_v(G, G')$ ebenfalls kleiner gleich $2k$.

Mon-DDS Nun suchen wir ein Dominating Set für G', welches mindestens um eins kleiner als D ist. Um das zu erreichen, können wir den Algorithmus Mon-DDS (Algorithmus 3) aufrufen, der das uns schon aus Kapitel 3.1 bekannte Problem Monotone Dynamic Dominating Set löst. Das Problem Monotone Dynamic Dominating Set ist ein Entscheidungsproblem. Wir gehen hier allerdings davon aus, dass uns der Algorithmus Mon-DDS (Algorithmus 3) eine Lösung oder die leere Menge zurückgibt.

Wir übergeben dem Algorithmus Mon-DDS (Algorithmus 3) die Graphen G und G', das Dominating Set D_1 für G, die Edge-Edit-Distance $2k$ und die Vertex-Edit-Distance $|D_2| - 1$, da wir wollen, dass das Dominating Set, welches uns zurückgegeben wird, kleiner als D ist. Die Rückgabe des Algorithmus Mon-DDS (Algorithmus 3) ist auch gleichzeitig die Rückgabe des Algorithmus Greedy-DS (Algorithmus 2).

Algorithmus 3 Monotone Dynamic Dominating Set

```
1:  procedure Mon-DDS(G, G', D, k, r)
2:      C ← V(G') \ N_{G'}[D]
3:      B ← N_{G'}(D)
4:      if k ≤ r then return D' ← D ∪ C
5:      for all v ∈ D do
6:          for all u ∈ V(G') do
7:              if {u, v} ∈ E(G') then E(G') ← E(G') \ {{u, v}}
8:              V(G') ← V(G') \ {v}
9:      for all v ∈ B do
10:         if N_{G'}({v}) ∩ C = ∅ then
11:             for all u ∈ V(G') do
12:                 if {u, v} ∈ E(G') then E(G') ← E(G') \ {{u, v}}
13:                 V(G') ← V(G') \ {v}
14:     for all v ∈ B do
15:         for all u ∈ B do
16:             if {u, v} ∈ E(G') then E(G') ← E(G') \ {{u, v}}
17:     D' ← Dom-A-VC(G', C, k, r)
18:     if D' ≠ ∅ then return D' ← D' ∪ D_1
19:     return ∅
```

Um nun das Problem Monotone Dynamic Dominating Set zu lösen, reduzieren wir es wie auch schon in Kapitel 3.1 auf Dominating A Vertex Cover. Dabei gehen wir auch bei dem Algorithmus Dom-A-VC (Algorithmus 4) zum Entscheidungsproblem Dominating A Vertex Cover davon aus, dass er uns eine Lösung oder die leere Menge zurückgibt.

Zuerst prüft der Algorithmus Mon-DDS (Algorithmus 3) in Zeile 4 jedoch, ob die Anzahl der Knoten in der Knotenmenge C, die die nicht von D dominierten Knoten enthält, kleiner als die erlaubte Vergrößerung r des Domianting Sets D für G ist. Wenn das zutrifft, ist $D' = C \cup D$ das gesuchte Dominating Set für G'. Ist das nicht der Fall, wendet der Algorithmus Mon-DDS (Algorithmus 3) ein paar Datenreduktionsregeln an: In den Zeilen 5 bis 8 löscht der Algorithmus Mon-DDS (Algorithmus 3) im Graphen G' alle Knoten $v \in D$ und deren ausgehende Kanten. In den Zeilen 9 bis 13 löscht der Algorithmus Mon-DDS (Algorithmus 3) im Graphen G' alle Knoten in der offenen Nachbarschaft von D, deren offene Nachbarschaft kein Knoten in C enthält und ihre ausgehenden Kanten. In den Zeilen 14 bis 17 löscht der Algorithmus Mon-DDS (Algorithmus 3) im Graphen G' schließlich alle Kanten zwischen Knoten in der offenen Nachbarschaft von D.

Da jetzt nur noch Kanten innerhalb oder ausgehend von C existieren, ist C ein Vertex Cover für G. Nun können wir den Algorithmus Dom-A-VC (Algorithmus 4) mit dem Graphen G, dem Vertex Cover C und den von dem Algorithmus Greedy-DS (Algorithmus 2) unverändert übernommenen Parametern k und r aufrufen. Findet der Algorithmus Dom-A-VC (Algorithmus 4) kein Dominating Set für C, so geben wir die leere Menge zurück. Wenn er jedoch ein Dominating Set D' für C findet, geben wir die Vereinigung $D' \cup D_1$ als Dominating Set für G' zurück.

Dom-A-VC Der Algorithmus Dom-A-VC (Algorithmus 4) prüft in den Zeilen 4 bis 6 zuerst für alle Knoten des Graphen G, die nicht im Vertex Cover C enthalten sind, ob ihre offenen Nachbarschaften gleich sind. Ist das

Algorithmus 4 Dominating A Vertex Cover

```
1:  procedure Dom-A-VC(G, C, k, r)
2:      if k ≤ r then return D ← C
3:      B ← V(G) \ C
4:      for all v ∈ B do
5:          for all u ∈ B do
6:              if v ≠ u und N_G({v}) = N_G({u}) then B ← B \ {u}
7:      B ← B ∪ C
8:      for all D ⊆ B, sodass gilt |D| = r do
9:          if C ⊆ N_G[D] then return D
10:     return ∅
```

der Fall, löscht er einen der beiden Knoten und alle von ihm ausgehenden Kanten. Danach ermittelt der Algorithmus mittels Brute-Force ein Dominating Set D für C und gibt dieses zurück.

4.2 Implementierung

Nachdem wir nun in Kapitel 4.1 die Heuristik vorgestellt haben, gehen wir hier auf die Besonderheiten unserer Implementierung ein.

Umgang mit Objektverweisen. Für jeden Graphen, jeden Knoten und jede Kante erstellen wir ein Objekt. Jeder Graph hat eine Knotenliste und eine Kantenmenge, in denen auf die in ihm enthaltenen Knoten und Kanten verwiesen wird. Wenn nun, wie beispielsweise in Zeile 8 des Algorithmus Greedy-DS (Algorithmus 2), vom Graphen H ein Subgraph G' erzeugt wird und wir im Subgraphen G' auf die in ihm enthaltenen Knoten und Kanten verweisen, gehören diese Knoten und Kanten nun sowohl zu H als auch zu G'. In Zeile 9 des Algorithmus Greedy-DS (Algorithmus 2) bilden wir nun noch einen neuen Graphen G, der wieder die gleichen Knoten und Kanten enthält. Wenn wir nun, wie in Zeile 11 des Algorithmus Greedy-DS (Algorithmus 2), bei einem dieser Graphen Kanten zwischen Knoten löschen oder hinzufügen, wird das für alle Graphen, die auf die entsprechenden Knoten verweisen, getan. Somit wäre in Zeile 12 des Algorithmus Greedy-DS (Algorithmus 2) die Edge-Edit-Distance $d_e(G, G') = 0$. Um das zu verhindern haben wir jedes Mal, wenn von einem Graphen G ausgehend ein neuer Graph G' erzeugt wird, auch alle Knoten und Kanten für G' neu erzeugt und in G' auf diese neuen Knoten und Kanten verwiesen.

Dadurch ist allerdings ein weiteres Problem aufgetreten. So gehen wir zum Beispiel in Zeile 10 des Algorithmus Greedy-DS (Algorithmus 2) über Knoten u, die aus L oder aus D_2 stammen. Die Knoten aus L und D_2 sind jedoch Teil des Graphen H. In Zeile 11 des Algorithmus Greedy-DS (Algorithmus 2) kann also in $E(G)$ gar keine Kante $\{x, u\}$ existieren und wir können diese Kante auch nicht zu $E(G)$ hinzufügen. Ebenfalls deutlich wird das Problem auch in Zeile 12 von des Algorithmus Greedy-DS (Algorithmus 2). Dort übergeben wir dem Algorithmus Mon-DDS (Algorithmus 3) unter anderem eine Knotenmenge D_1 die Knoten in dieser Menge sind jedoch Teil des Graphen H, den wir dem Algorithmus Mon-DDS (Algorithmus 3) gar nicht übergeben. Um dieses Problem zu beheben, müssen wir die Knoten mit den entsprechenden Pendants aus dem Graphen G' ersetzen, den wir im Anschluss betrachten. Auch müssen wir unter anderem bei der Rückgabe des Algorithmus Mon-DDS (Algorithmus 3) in Zeile 12 des Algorithmus Greedy-DS (Algorithmus 2) die Knoten in der zurückgegebenen Menge wieder durch Knoten aus H ersetzen.

Nutzen des Graphen G im Algorithmus Mon-DDS. Algorithmus Mon-DDS (Algorithmus 3) bekommt, wie es auch das Problem MONOTONE DYNAMIC DOMINATING SET beschreibt, unter anderem zwei Graphen G, G' und ein Dominating Set D für G und liefert ein Dominating Set D' für G'. Zur Berechnung von D' wird der Graph G jedoch nirgends benötigt. Deswegen müssen wir ihn dem Algorithmus Mon-DDS (Algorithmus 3) gar nicht übergeben und sparen uns demzufolge auch die Erzeugung dieses Graphen G im Algorithmus Greedy-DS (Algorithmus 2) in den Zeilen 9 bis 11.

Berechnung der Teilmengen. In Zeile 8 des Algorithmus Dom-A-VC (Algorithmus 4) gehen wir über alle Teilmengen $D \subseteq B$ mit $|D| = r$. Das Bilden dieser Teilmengen erwies sich jedoch als kleine Schwierigkeit. Um zu erklären, wie wir bei der Berechnung der Teilmengen vorgegangen sind, stellen wir uns vor, dass $|B| = 5$ und $r = 3$ ist. Somit wollen wir also alle dreielementigen Teilmengen einer fünfelementigen Menge berechnen. In Abbildung 9 ist zu erkennen, wie wir dabei vorgehen. Zuerst fügen wir allerdings alle Knoten $v \in B$ in eine Liste L_B ein, um damit eine wohldefinierte Reihenfolge der Knoten festzulegen.

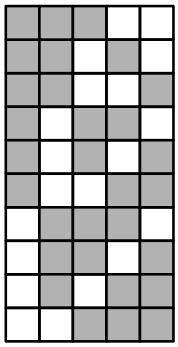

Abbildung 9: Vorgensweise zur Berechnung aller dreielementigen Teilmengen einer fünfelementigen Menge.

Als erste Teilmenge geben wir nun die ersten r Elemente der Liste L_B zurück und merken uns ihre Indizes $i_1 = 0, i_2 = 1, \ldots, i_r = r - 1$. Vor der Berechnung jeder weiteren Teilmenge addieren wir eins auf den größten Index i_a mit $a < L_B - 1$, den wir erhöhen können, ohne dass danach $i_a = i_{a+1}$ gilt. Haben wir diesen Index i_a erhöht, setzen wir alle Indizes i_b mit $b > a$ auf $i_a + b - a$. Damit haben wir erreicht, dass alle Indizes ab dem Index i_b genau eins größer sind als der vorhergehende Index. Nun geben wir eine Teilmenge, bestehend aus den Elementen, deren Positionen gerade durch die Indizes gespeichert sind, aus. Danach erhöhen wir nach dem obigen Schema wieder die Indizes. Lässt sich kein Index i_a finden, den wir erhöhen könnten, haben wir alle Teilmengen berechnet und geben die leere Menge zurück.

Rekonstruktion der Reihenfolge in L_D. Das Turbo-Charging durch den Algorithmus Greedy-DS (Algorithmus 2) in Zeile 41 des Algorithmus DSH (Algorithmus 1) gibt eine Menge zurück. Dadurch sind die Elemente in der Liste L_D danach unsortiert. Die Reihenfolge der Knoten in L_D ist aber relevant, wenn wir in den Zeilen 34 bis 39 des Algorithmus DSH (Algorithmus 1) die Liste L_D auf die Mengen D_1 und D_2 aufteilen. Um die Reihenfolge wiederherzustellen wird L_D deshalb in Zeile 15 des Algorithmus DSH (Algorithmus 1) sortiert. Danach soll die Reihenfolge der Knoten in L_D die Reihenfolge des Einfügens dieser Knoten widerspiegeln.

Um das zu gewährleisten, merken wir uns die Reihenfolge in L_D vor dem Turbo-Charging. Das machen wir, indem wir in Zeile 40 des Algorithmus DSH (Algorithmus 1) nur die Knoten, die in D_2 enthalten sind, aus L_D entfernen, da das gerade die Knoten sind, die wir durch das Turbo-Charging austauschen wollen. In dem Algorithmus Mon-DDS (Algorithmus 3) vereinigen wir D' in Zeile 18 nicht mit D_1 und geben somit nur die Knoten zurück, die wir zum Dominieren des Vertex Covers C benötigt haben. Für den Algorithmus Greedy-DS (Algorithmus 2) bedeutet das, dass wir in Zeile 12 dieses Algorithmus nur die Menge der Knoten zurückgeben, die zusätzlich zu D_1 benötigt werden, um alle Knoten in G' zu dominieren. Wenn wir nun in Zeile 42 des Algorithmus DSH (Algorithmus 1) alle Knoten dieser Menge hinten an die Liste L_D anfügen, bleibt die Sortierung nach der Reihenfolge des Einfügens der Knoten bestehen und wir sparen uns außerdem noch das Sortieren von L_D in Zeile 15 des Algorithmus DSH (Algorithmus 1).

4.3 Heuristische Verbesserungen

Im Nachfolgenden gehen wir auf unsere qualitativen Verbesserungen der Turbo-Charging-Heuristik von Downey et al. [5] ein. Dabei erklären wir unsere Idee, warum die Heuristik dadurch verbessert wird und beschreiben die Umsetzung der Verbesserung.

Dynamisch sortiert. Bei dieser Verbesserung wird die in dem Konzept von Downey et al. [5] vorgesehene Sortierung Liste L in Zeile 7 des Algorithmus DSH (Algorithmus 1) nicht vorgenommen. Stattdessen wird L direkt nach Zeile 16 des Algorithmus DSH (Algorithmus 1) absteigend nach der Anzahl der nicht dominierten Nachbarn sortiert. Dadurch werden bei der Suche nach einem Knoten, der als nächstes in das zu bildende Dominating Set aufgenommen werden soll, zuerst die Knoten betrachtet, die die meisten noch nicht dominierten Knoten dominieren.

Die Idee dahinter ist, dass es vielversprechender erscheint als nächstes einen Knoten zu betrachten, der viele noch nicht dominierte Knoten dominiert als einen, der zwar einen hohen Knotengrad hat, in dessen Umgebung aber womöglich schon fast alle Knoten dominiert sind.

In Kapitel 5 werden wir sehen, dass diese Intuition zumindest teilweise zutrifft.

Deutlich kleinere Teilmengen. Die Idee von Downey et al. [5] sieht vor, dass das bereits gefundene Dominating Set während des Turbo-Charging um genau ein Element verkleinert wird. Deswegen wird in Zeile 8 des Algorithmus Dom-A-VC (Algorithmus 4) nur nach Teilmengen $D \subseteq B$ gesucht, die r Elemente groß sind. Intuitiv wäre es jedoch sinnvoller mithilfe des Algorithmus Dom-A-VC (Algorithmus 4) ein kleinstmögliches Dominating Set D für das Vertex Cover C zu finden, damit nicht nach jedem Einfügen eines neuen Knotens wieder das Turbo-Charging durchgeführt werden muss.

Das setzen wir um, indem wir bei der Suche nach einem Dominating Set D für das Vertex Cover C D zuerst gleich der leeren Menge setzen und überprüfen, ob D C dominiert. Ist das nicht der Fall, betrachten wir nacheinander alle Teilmengen von B, die ein Element größer sind, und überprüfen, ob unter ihnen ein Dominating Set für C existiert. So erhöhen wir nach und nach die Anzahl der Elemente in D, bis wir entweder ein Dominating Set für C gefunden haben oder alle Teilmengen, die höchstens r Elemente haben, durchgegangen sind.

Wie wir in Kapitel 5 sehen werden hat sich diese Verbesserung auch bei unseren Experimenten bewährt.

Gleichgroße Teilmengen. Beim Turbo-Charging muss nach dem Konzept von Downey et al. [5] die gefundene Zwischenlösung verkleinert werden. Kann das vom Turbo-Charging nicht erreicht werden, gibt die Turbo-Charing-Greedy-Heuristik zurück, dass sie kein Dominating Set der gewünschten Größe s für den entsprechenden Graphen G finden konnte, obwohl vielleicht eines existiert.

Dort setzt unsere Idee an. Statt die Zwischenlösung S beim Turbo-Charging zwingend zu verkleinern, prüfen wir, nachdem wir festgestellt haben, dass das Turbo-Charging die Menge S nicht verkleinern kann, ob wir eine gleichgroße Menge S' finden können, die womöglich mehr Knoten als S dominiert und im besten Fall ein Dominating Set für G ist.

Um diese Idee umzusetzen, berechnen wir in Zeile 8 des Algorithmus Dom-A-VC (Algorithmus 4) zuerst wie bisher alle Teilmengen $D \subseteq B$, die r Elemente groß sind. Haben wir unter diesen allerdings kein Dominating Set für das Vertex Cover C finden können, berechnen wir danach noch die Teilmengen $D \subseteq B$, die ein Element größer als r sind. Existiert unter ihnen ein Dominating Set D' für C, so geben wir dieses zurück. Im Idealfall dominiert die daraus resultierende Zwischenlösung $S' = D' \cup D_1$, die wir in Zeile 18 des Algorithmus Mon-DDS (Algorithmus 3) bilden, den gesamten Graphen G. Ist dies nicht der Fall, wird der von uns veränderte Turbo-Charging-Mechanismus erneut aufgerufen. Wenn er diese Zwischenlösung S'

wieder nicht verkleinern kann, suchen wir abermals nach einer gleichgroßen Zwischenlösung. Damit wir dabei nicht noch einmal die vorhergehenden Zwischenlösungen finden, speichern wir uns direkt nach Zeile 5 des Algorithmus Greedy-DS (Algorithmus 2) die Menge D_2 als Element einer Menge F ab. Finden wir im Algorithmus Dom-A-VC (Algorithmus 4) nun ein Dominating Set D, welches genauso groß wie D_2 ist, überprüfen wir, ob diese Menge D bereits in F enthalten ist und wir sie somit schon als nicht zielführend abgespeichert haben. Ist sie dort enthalten, suchen wir weiter, bis wir ein Dominating Set $D \notin F$ für das Vertex Cover C gefunden oder alle Teilmengen von B, die $r + 1$ Elemente groß sind, betrachtet haben. Bevor wir im Algorithmus Dom-A-VC (Algorithmus 4) entweder in Zeile 2 oder in Zeile 9 ein Dominating Set D für C zurückgeben, speichern wir es ebenfalls als Element der Menge F ab, damit auch D bei einem möglichen, weiteren Turbo-Charging-Durchgang nicht noch einmal zurückgegeben wird.

In Kapitel 5 werden wir sehen, dass diese Vorgehensweise nicht nur intuitiv eine Verbesserung bewirkt, sondern auch in der Praxis häufig gute Ergebnisse liefert.

4.4 Erweiterung für Set Cover

Die Kapitel 4.1 vorgestellte Turbo-Charging-Heuristik lässt sich mit einer kleinen Anpassung so umbauen, dass sie sich auf Set Cover anwenden lässt.

Konzept. Die Idee dahinter haben wir schon in Kapitel 3.3 bei der Reduktion von DSC auf DDS beschrieben und in Abbildung 6 veranschaulicht. Beim Problem Set Cover haben wir ein Universum $\mathcal{U}$, eine Menge $\mathcal{F}$ von Teilmengen $T_i \in \mathcal{F}$ mit $T_i \subseteq \mathcal{U}$ und ein $k \in \mathbb{N}$ gegeben. Gesucht ist ein Set Cover $S \subseteq \mathcal{F}$ der Größe k für $\mathcal{U}$. Um dieses Problem auf Dominating Set zu übertragen, fassen wir es als Graphproblem auf. Wir bilden einen Graphen G, bei dem jeder Knoten entweder für eine Teilmenge T_i oder ein Element $u \in \mathcal{U}$ steht. Zwischen einem Knoten T_i und einem Knoten u verläuft eine Kante, falls $u \in T_i$ gilt.

Damit wir mit unserer Heuristik ein Set Cover S für $\mathcal{U}$ finden können, müssen wir noch einen Knoten w in den Graphen G einfügen. Diesen Knoten w verbinden wir mit allen T_i. Außerdem fügen wir noch einen Knoten mehr in den Graphen G ein, als $\mathcal{U}$ Elemente enthält, und verbinden sie alle mit w. Dadurch ist sichergestellt, dass w den höchsten Knotengrad in G hat und deswegen, aufgrund der Arbeitsweise der Heuristik, immer Teil des Dominating Sets D sein wird, welches die Heuristik als Lösung zurückgibt.

Außerdem müssen wir berücksichtigen, dass bei der Eingabe des Algorithmus Dom-A-VC (Algorithmus 4) auch Knoten, die Teilmengen T_i repräsentieren, im Vertex Cover sein können. Diese müssen wir entfernen, da wir sie eigentlich nicht dominieren wollen.

Nun rufen wir die Heuristik auf und wählen dabei die Größe des gesuchten Dominating Sets eins größer als k, da w zwar innerhalb des Dominating Sets D ist, aber nicht innerhalb des Set Covers S sein soll. Hat unsere Heuristik ein Dominating Set D gefunden, löschen wir also w daraus und ersetzen außerdem alle u, die sowohl in D als auch in $\mathcal{U}$ enthalten sind, durch eine beliebige Teilmenge T_i, die dieses u enthält. Danach enthält D nur noch Teilmengen T_i und ist somit das gesuchte Set Cover S.

Umsetzung. Die Erweiterung für Set Cover ist bereits implementiert. Die Implementierung greift allerdings nicht die oben beschriebene Konstruktion zurück. Stattdessen haben wir für alle Knoten T_i festgelegt, dass sie nicht von D dominiert werden müssen, damit D ein Dominating Set für G ist.

Das hat den Vorteil, dass wir nicht sicherstellen müssen, dass der Knoten w immer als erstes in das Dominating Set D eingefügt wird. Bei der bisherigen Herangehensweise ist das zwar der Fall, aber wenn wir in Zukunft Verbesserungen implementieren, die dafür sorgen, dass nicht unbedingt zuerst der Knoten mit dem höchsten Knotengrad eingefügt wird, können wir nicht mehr davon ausgehen.

Dadurch, dass wir uns den Knoten w sparen, stimmt außerdem die Größe des gesuchten Set Covers S mit der Größe des durch die Heuristik gelieferten Dominating Sets D überein. Wir müssen jedoch auch weiterhin alle u, die sowohl in D als auch in $\mathcal{U}$ enthalten sind, durch eine beliebige Teilmenge T_i, die dieses u enthält, ersetzen, damit wir ein gültiges Set Cover S erhalten.

5 Experimente

In diesem Kapitel widmen wir uns der Auswertung der Experimente. Wir haben die Qualität der in Kapitel 4 vorgestellten Turbo-Charging-Greedy-Heuristik für Dominating Set und ihrer in Kapitel 4.3 vorgestellten Verbesserungen untersucht. Zuerst gehen wir jedoch auf den Ursprung und die Beschaffenheit der Daten ein. Danach erläutern wir unsere Vorgehensweise beim Experimentieren und schließlich präsentieren wir die Resultate.

5.1 Daten

Wir haben die Heuristik anhand von 120 Instanzen analysiert. Diese Instanzen wurden von Jovanovic et al. [12] für ein Minimum Weighted Dominating Set-Problem erzeugt. Damit wir sie verwenden konnten, haben wir die Knotengewichte ignoriert. Die Graphen der verwendeten Instanzen sind in sechs Abstufungen zwischen 50 und 1000 Knoten groß und zusammenhängend. Für jede dieser Abstufungen haben wir 20 Graphen betrachtet. Wir haben die Heuristik nur auf Graphen untersucht, die mehr als doppelt so viele Kanten

wie Knoten haben, da die Heuristik ungeeignet für Graphen mit sehr geringer Dichte ist. In Tabelle 1 sehen wir die ungefähre durchschnittliche Dichte der Graphen, anhand derer wir die Heuristik untersucht haben, in Abhängigkeit von der Knotenanzahl.

Knoten	Maximum	Durch-schnitt	Minimum
50	0,816	0,510	0,204
150	0,268	0,130	0,045
300	0,111	0,052	0,017
500	0,080	0,044	0,016
800	0,031	0,018	0,006
1000	0,040	0,025	0,010

Tabelle 1: Dichte.

5.2 Aufbau

Wir haben die Heuristik in Java 8 implementiert und die Experimente auf einem Computer mit folgender Spezifikation durchgeführt:

- Betriebssystem: Ubuntu 16.04.2 LTS
- Prozessor: 4 GHz Intel Skylake i7-6700K
- Arbeitsspeicher: 16 GB DDR4 2133 MHz Dual Channel
- Festplatte: Samsung SSD 850 EVO 250 GB

Um die Turbo-Charging-Heuristik mit unterschiedlichen Größen des gesuchten Dominating Sets und des Backups und den verschiedenen Verbesserungen zu untersuchen, haben wir Bash-Skripte eingesetzt. Das hat die Durchführung der Experimente erheblich vereinfacht. Als Backupgrößen haben wir dabei 0, also kein Turbo-Charging, 50 und 100 gewählt. Für jede Variante der Heuristik, also für die Heuristik mit jeweils einer Verbesserung und für die Heuristik ohne Verbesserungen, haben wir ein eigenes Skript erstellt. In jedem dieser Skripte haben wir für jede Instanz, ausgehend von dem ohne Turbo-Charging gefundenen Dominating Set, die Größe des gesuchten Dominating Sets so lange jeweils um eins verkleinert, bis die Heuristik mit dem zu diesem Zeitpunkt festgelegten Backup beziehungsweise der in dem entsprechenden Skript untersuchten Verbesserung kein Dominating Set mehr gefunden hat. Konnte dabei für eine Instanz innerhalb einer halben Stunde kein Dominating Set gefunden werden, wurde die Heuristik abgebrochen und die nächste Instanz untersucht.

5.3 Resultate

Nun kommen wir zu den Ergebnissen der Experimente. Zuerst werden wir dabei die Resultate, die die Turbo-Charging-Greedy-Heuristik bei einer Backupgröße von 50 geliefert hat, betrachten. Uns interessieren hierbei besonders die Größen der gefundenen Dominating Sets. Deswegen werden wir ausführlich darauf eingehen, um wie viel die verschiedenen Varianten der Heuristik die durchschnittliche Größe der gefundenen Dominating Sets im Vergleich zu keinem Turbo-Charging und zu anderen Varianten verkleinern konnten. Das werden wir danach auch für eine Backupgröße von 100 analysieren. Zum Schluss werden wir die durchschnittlichen Größen der gefundenen Dominating Sets bei den Backupgrößen 50 und 100 miteinander vergleichen.

Backupgröße 50. Wie wir in Tabelle 2 sehen können, haben wir mit einer Backupgröße von 50 bereits sichtbare Verbesserungen gegenüber der Greedy-Heuristik ohne Turbo-Charging erzielt. Dabei fällt auf, dass mit der Verbesserung „dynamisch sortiert“ im Durchschnitt meist größere Dominating Sets gefunden werden als ohne eine Verbesserung. Das liegt daran, dass bei der Verbesserung „dynamisch sortiert“ bei einem kleinen Backup der Turbo-Charging-Mechanismus häufig keine kleinere Zwischenlösung findet.

Knoten	ohne Turbo-Charing	keine Verbesserung	dynamisch sortiert	gleichgroße Teilmengen	deutlich kleinere Teilmengen	Minimum	Maximum
50	5,25	4,10	4,10	3,60	4,10	2	7
150	24,20	22,60	22,90	22,35	22,60	8	40
300	53,45	52,15	52,35	51,95	51,80	20	91
500	73,20	71,80	72,30	71,65	71,75	35	117
800	148,95	147,60	147,75	147,60	147,75	81	244
1000	123,40	122,75	122,80	122,70	122,75	71	203

Tabelle 2: Durchschnitts- und Extremwerte der Größe der gefundenen Dominating Sets bei einer Backupgröße von 50 und Durchschnittswerte der Größe der gefundenen Dominating Sets ohne Turbo-Charging.

Auch sofort ins Auge fällt die Verbesserung „gleichgroße Teilmengen“ bei einer Knotenanzahl von 50. Dass sie dort häufig kleinere Dominating Sets gefunden hat als die anderen Varianten, liegt an der Idee des Turbo-Charging. Bei allen anderen Varianten muss beim Turbo-Charging die Zwischenlösung verkleinert werden. Wenn das kleinstmögliche Dominating Set jedoch nicht kleiner sondern gleich der Anzahl der Knoten in der Zwischenlösung ist, kann nur durch eine andere Auswahl der gleichen Anzahl an Knoten ein Dominating Set gefunden werden. Genau in diesen Fällen findet nur die Verbesserung „gleichgroße Teilmengen“ diese optimale Lösung.

Außerdem bemerken wir, dass wir bei einer Knotenanzahl von 800 die sowohl im Durchschnitt als auch insgesamt größten Dominating Sets gefunden haben. Wenn wir uns allerdings die Tabelle 1 angucken, stellen wir schnell

fest, dass das an der sehr geringen Dichte der Graphen mit 800 Knoten liegt.

In Tabelle 3 sehen wir die durchschnittliche relative Veränderung der Größe der gefundenen Dominating Sets. Dabei vergleichen wir zwischen keinem Turbo-Charging und einer Backupgröße von 50. Auch bei diesen prozentualen Werten können wir wieder interessante Beobachtungen anstellen.

	keine Verbesserung		dynamisch sortiert		gleichgroße Teilmengen		deutlich kleinere Teilmengen	
Knoten	Durchschnitt	Maximum	Durchschnitt	Maximum	Durchschnitt	Maximum	Durchschnitt	Maximum
50	-21,90	-40,00	-21,90	-40,00	-31,43	-60,00	-21,90	-40,00
150	-6,61	-16,67	-5,37	-12,50	-7,64	-16,67	-6,61	-16,67
300	-2,43	-11,11	-2,06	-11,11	-2,81	-16,67	-3,09	-11,11
500	-1,91	-7,50	-1,23	-5,08	-2,12	-7,89	-1,98	-7,50
800	-0,91	-3,62	-0,81	-3,33	-0,91	-3,62	-0,81	-4,44
1000	-0,53	-2,46	-0,49	-2,46	-0,57	-2,46	-0,53	-3,09

Tabelle 3: Prozentuale Veränderung der Größe der gefundenen Dominating Sets zwischen keinem Turbo-Charging und einer Backupgröße von 50.

Auffällig ist besonders, dass das Turbo-Charging bei steigender Knotenanzahl im Durchschnitt weniger nutzbringend ist. Die durchschnittliche Größe der gefundenen Dominating Sets mit Turbo-Charging nähert sich immer mehr der ohne Turbo-Charging an. Auch der Betrag der maximalen Veränderung fällt monoton mit einer steigenden Anzahl an Knoten. Das hat vor allem den Grund, dass mit steigender Größe des Graphen der Anteil des Backups sinkt. Deckte das Backup bei 50 Knoten noch den ganzen Graphen ab, so beträgt sein Anteil bei 1000 Knoten gerade einmal 5 %. Wenn das Backup einen kleineren Teil des Graphen ausmacht, bedeutet das, dass auch das Turbo-Charging auf einem prozentual kleineren Teil stattfindet. Mit der gleichen Backupgröße erzielen wir durch das Turbo-Charging auf kleinen Graphen also meist deutlich bessere Ergebnisse als auf großen. Zusätzlich dazu haben die größeren Graphen, wie wir in Tabelle 1 sehen, eine geringere Dichte. Dadurch können die im Algorithmus Mon-DDS (Algorithmus 3) verwendeten Datenreduktionsregeln in vielen Fällen nicht mehr so gut greifen und sortieren somit weniger Knoten und Kanten aus. Das sorgt zusammen mit der gestiegenen Größe der Graphen häufig dafür, dass das Turbo-Charging länger dauert und die Heuristik bei einigen Instanzen sogar abgebrochen wurde und deshalb nur ein größeres Dominating Set gefunden wurde. In Tabelle 4 sehen wir wie groß der Prozentsatz der abgebrochenen Instanzen jeweils ist. Im Folgenden gehen wir auch genauer auf die Bedeutung der Abbrüche ein.

In Tabelle 4 sehen wir wie die Mediane der Zeit, die die verschiedenen Varianten bei einer Backupgröße von 50 benötigt haben. Außerdem sind die damit eng verbundenen Abbrüche dargestellt.

Um zu verstehen, was die Abbrüche bedeuten, gehen wir hier noch mal etwas ausführlicher auf den Ablauf der Experimente ein: Um mit der Heuristik ein Dominating Set zu finden, müssen wir ihr unter anderem die gewünschte Größe s des gesuchten Dominating Sets als Parameter übergeben. Wie in

Knoten	keine Verbesserung		dynamisch sortiert		gleichgroße Teilmengen		deutlich kleinere Teilmengen	
	Zeit in Sekunden	Abbrüche in Prozent	Zeit in Sekunden	Abbrüche in Prozent	Zeit in Sekunden	Abbrüche in Prozent	Zeit in Sekunden	Abbrüche in Prozent
50	0,15	0	0,15	0	0,17	0	0,14	0
150	0,20	0	0,20	0	0,23	20	0,20	10
300	0,31	15	0,31	15	0,31	20	0,29	20
500	0,45	5	0,29	5	0,47	25	0,43	5
800	0,68	35	0,66	30	0,70	35	0,67	35
1000	0,47	5	0,47	5	0,47	10	0,46	10

Tabelle 4: Mediane der Laufzeiten und Prozentsatz der Abbrüche bei einer Backupgröße von 50.

Kapitel 5.2 beschrieben, rufen wir die Heuristik für jede Instanz mehrfach auf. Beim ersten Aufruf setzen wir s gleich der Größe des Dominating Sets, welches wir vorher bereits ohne Turbo-Charging gefunden haben. Bei jedem weiteren Aufruf verkleinern wir s um eins. Das machen wir so lange, bis die Heuristik entweder kein Dominating Set der Größe s mehr findet oder länger als 30 Minuten braucht. In beiden Fällen nehmen wir, um die Zeiten, die die Heuristik für die verschiedenen Instanzen gebraucht hat, miteinander zu vergleichen, die Zeit, die es gedauert hat das kleinste von der Heuristik für diese Instanz gefundene Dominating Set zu berechnen. Das bedeutet, dass wir nicht betrachten, wie lange es gedauert hat, bis die Heuristik herausgefunden hat, dass sie für ein bestimmtes s kein Dominating Set findet. Auch die 30 Minuten bis zum Abbruch der Heuristik werden nicht in die Zeiten mit eingerechnet. Stattdessen werden die Abbrüche separat aufgelistet.

Wenn wir bei einer Variante also eine kurze Zeit aber einen hohen Prozentsatz an Abbrüchen haben, bedeutet das nicht, dass die Heuristik mit dieser Variante schnell war. Stattdessen hat sie in der kurzen Zeit wahrscheinlich nur ein im Vergleich zur optimalen Lösung relativ großes Dominating Set gefunden. Hätten wir ihr länger als eine halbe Stunde Zeit gegeben, hätte sie dieses Dominating Set wahrscheinlich noch verkleinern können.

Eine Variante X der Heuristik, die lange braucht aber selten abgebrochen wird, ist also wahrscheinlich besser als eine Variante Y, die nur wenig Zeit benötigt und dabei jedoch viele Abbrüche produziert. Die kurze Zeit rührt in den meisten Fällen dann nämlich nur daher, dass die Variante Y es in 30 Minuten nicht geschafft hat ein Dominating Set der Größe zu finden, wie es Variante X in der vermeintlich längeren Zeit gefunden hat.

Um diese Problematik der Vergleichbarkeit zu umgehen, benutzen wir bei der Betrachtung der Zeit den Median statt des Durchschnitts. Dadurch haben die Ausreißer, die entstehen, weil eine Variante der Heuristik die Größe des Dominating Sets gerade noch innerhalb der Zeit weiter verkleinern konnte, während eine andere Variante abgebrochen wurde und deswegen bei ihr die Zeit der Ermittlung eines größeren Dominating Sets genommen wurde, keinen Einfluss auf den Mittelwert. Aber auch außerhalb dieses Kontextes gibt es einige Ausreißer, die den Durchschnitt verzerrt hätten und somit die

Nutzung des Medians erforderlich gemacht haben.

Anhand der Mediane in Tabelle 4 erkennt man gut, dass die Zeit in den meisten Fällen mit der Anzahl der Knoten steigt. Dass die Mediane der Zeit bei einer Knotenanzahl von 800 besonders hoch sind, liegt daran, dass, wie in Tabelle 1 zu erkennen, die durchschnittliche Dichte dieser Graphen am geringsten ist. Das erklärt auch die vielen Abbrüche bei in dieser Zeile. Verwunderlich hingegen ist der meist niedrige Prozentsatz an Abbrüchen bei den Knotenanzahlen 500 und 1000. Vielleicht erschlösse sich uns der Grund dafür, wenn wir jeweils die kleinstmöglichen Dominating Sets kennen würden.

Backupgröße 100. Nachdem wir uns nun die Werte für eine Backupgröße von 50 angesehen haben, betrachten wir im Folgenden, wie sich die Heuristik in den verschiedenen Varianten bei einer Backupgröße von 100 verhält. Dazu

Knoten	ohne Turbo-Charing	keine Verbesserung	dynamisch sortiert	gleichgroße Teilmengen	deutlich kleinere Teilmengen	Minimum	Maximum
50	5,25	4,10	4,10	3,60	4,10	2	7
150	24,20	22,65	22,40	22,50	23,05	7	41
300	53,45	52,00	51,95	51,85	51,80	20	91
500	73,20	71,60	71,20	71,45	70,80	34	117
800	148,95	147,70	147,30	147,70	147,40	81	244
1000	123,40	121,80	122,10	121,80	120,80	70	203

Tabelle 5: Durchschnitts- und Extremwerte der Größe der gefundenen Dominating Sets bei einer Backupgröße von 100 und Durchschnittswerte der Größe der gefundenen Dominating Sets ohne Turbo-Charging.

gucken wir uns als erstes die Tabelle 5 an. Dabei fällt auf, dass die Verbesserung „dynamisch sortiert“, anders als noch bei einer Backupgröße von 50, nun im Durchschnitt kleinere Dominating Sets findet als die Heuristik ohne Verbesserungen. Bei einem Vergleich mit Tabelle 2 fallen außerdem ein paar Werte auf, die nun größer sind als mit einem kleineren Backup. Das liegt daran, dass die Heuristik, wie in Tabelle 7 zu sehen, bei einer Backupgröße von 100 bereits häufig abgebrochen wird. Das ist auch der Grund, aus dem wir keine größeren Backupgrößen untersucht haben.

	keine Verbesserung		dynamisch sortiert		gleichgroße Teilmengen		deutlich kleinere Teilmengen	
Knoten	Durchschnitt	Maximum	Durchschnitt	Maximum	Durchschnitt	Maximum	Durchschnitt	Maximum
50	-21,90	-40,00	-21,90	-40,00	-31,43	-60,00	-21,90	-40,00
150	-6,40	-25,00	-7,44	-25,00	-7,02	-33,33	-4,75	-41,67
300	-2,71	-11,11	-2,81	-11,11	-2,99	-12,50	-3,09	-22,22
500	-2,19	-8,70	-2,73	-9,76	-2,39	-8,70	-3,28	-13,04
800	-0,84	-4,49	-1,11	-4,44	-0,84	-4,49	-1,04	-7,87
1000	-1,30	-4,04	-1,05	-4,12	-1,30	-4,04	-2,11	-9,28

Tabelle 6: Prozentuale Veränderung der Größe der gefundenen Dominating Sets zwischen keinem Turbo-Charging und einer Backupgröße von 100.

Bei den prozentualen Veränderungen in Tabelle 6 sind besonders die großen Maxima bei einer Anzahl von 150 Knoten auffällig. Sie kommen daher, dass die Backupgröße 100 bei diesen Instanzen zwei Drittel des Graphen abdeckt. Aus diesem Grund können die ohnehin schon kleinen Dominating Sets der Graphen, die unter denen mit 150 Knoten die höchste Dichte haben, um ein paar Knoten verkleinert werden und so das Maximum der jeweiligen Variante in die Höhe treiben.

Knoten	keine Verbesserung		dynamisch sortiert		gleichgroße Teilmengen		deutlich kleinere Teilmengen	
	Zeit in Sekunden	Abbrüche in Prozent	Zeit in Sekunden	Abbrüche in Prozent	Zeit in Sekunden	Abbrüche in Prozent	Zeit in Sekunden	Abbrüche in Prozent
50	0,15	0	0,15	0	0,17	0	0,16	0
150	0,22	40	0,24	35	0,25	45	0,18	75
300	0,36	35	0,31	30	0,37	35	0,29	45
500	0,48	35	0,58	35	0,51	35	0,43	40
800	0,67	50	0,82	40	0,71	50	0,58	50
1000	1,13	25	1,03	20	1,10	30	1,09	25

Tabelle 7: Mediane der Laufzeiten und Prozentsatz der Abbrüche bei einer Backupgröße von 100.

In Tabelle 7 sehen wir, wie oben bereits erwähnt, den Grund dafür, dass wir das Backup nicht größer als 100 gewählt haben. Schon bei dieser Größe braucht die Heuristik häufig länger als 30 Minuten und wird deshalb abgebrochen. Dabei ist nicht klar, warum bei einer Knotenanzahl von 1000 vergleichsweise wenige Abbrüche gibt. Auch der genaue Grund für den auffallend hohen Prozentsatz an Abbrüchen bei der Knotenanzahl 150 kombiniert mit der Verbesserung „deutlich kleinere Teilmengen“ bleibt uns verborgen. Vielleicht würde uns auch hier die optimale Lösung eine Erklärung liefern.

Vergleich der Backupgrößen. In diesem Abschnitt stellen wir nun die Backupgrößen gegenüber. Dabei sehen wir in Tabelle 8 zusätzlich zu den Auffälligkeiten, die wir bereits in den vorhergehenden Abschnitten behan-

Knoten	Backupgröße 50			Backupgröße 100		
	dynamisch sortiert	gleichgroße Teilmengen	deutlich kleinere Teilmengen	dynamisch sortiert	gleichgroße Teilmengen	deutlich kleinere Teilmengen
50	0,00	-12,20	0,00	0,00	-12,20	0,00
150	1,33	-1,11	0,00	-1,10	-0,66	1,77
300	0,38	-0,38	-0,67	-0,10	-0,29	-0,38
500	0,70	-0,21	-0,07	-0,56	-0,21	-1,12
800	0,10	0,00	0,10	-0,27	0,00	-0,20
1000	0,04	-0,04	0,00	0,25	0,00	-0,82

Tabelle 8: Prozentuale Durchschnittswerte der Veränderung der Größe der gefundenen Dominating Sets zwischen keiner Verbesserung und den verschiedenen Verbesserungen.

Backupgröße	keine Verbesserung	dynamisch sortiert	gleichgroße Teilmengen	deutlich kleinere Teilmengen
50	-1,74	-1,46	-2,01	-1,80
100	-2,01	-2,19	-2,23	-2,45

Tabelle 9: Prozentuale Durchschnittswerte der Veränderung der Größe der gefundenen Dominating Sets im Vergleich zu keinem Turbo-Charging.

delt haben, dass die Verbesserung „deutlich kleinere Teilmengen“ bei einer Backupgröße von 100 in einigen Fällen größere Dominating Sets liefert, als sie es bei einer Backupgröße von 50 getan hat. Das ist ganz einfach mit der höheren Anzahl an Abbrüchen zu erklären. Trotzdem ist diese Beobachtung besonders bemerkenswert, denn wie wir in Tabelle 9 sehen, liefert die Turbo-Charging-Greedy-Heuristik mit der Verbesserung „deutlich kleinere Teilmengen“ und einer Backupgröße von 100, verglichen mit allen von uns betrachteten Fällen, die durchschnittlich kleinsten Dominating Sets. Mit ihr und der Nutzung des Turbo-Charging-Mechanismus können wir gegenüber der Greedy-Heuristik ohne Turbo-Charging eine durchschnittliche Verkleinerung der Dominating Sets um $2,45\,\%$ erzielen.

Aufgrund unserer Feststellung, dass die Qualität der Verbesserung „deutlich kleinere Teilmengen“ bei einer Backupgröße von 100 teilweise wieder sinkt, ist es naheliegend, dass bei einer weiteren Vergrößerung des Backups unter den gleichen Bedingungen die Güte der Verbesserung „deutlich kleinere Teilmengen“ relativ schnell wieder abnimmt und wir deswegen auf diese Art nicht deutlich mehr als $2,45\,\%$ Verbesserung erreichen werden.

6 Ausblick

In dieser Arbeit haben wir gezeigt, dass Dynamic Vertex Cover und Dynamic Set Cover fixed-parameter tractable bezüglich des Unterschieds in der Eingabe sind. Für ihre größenbeschränkten Varianten Restricted Size Dynamic Vertex Cover und Restricted Size Dynamic Set Cover haben wir bewiesen, dass sie, bezogen auf den kombinierten Parameter aus dem Unterschied in der Eingabe und dem Unterschied in der Lösung, W[1]-schwer beziehungsweise W[2]-schwer sind.

In unseren Experimenten hat sich herausgestellt, dass die Turbo-Charging-Greedy-Heuristik mit der Verbesserung „deutlich kleinere Teilmengen“ und einer Backupgröße von 100 von den untersuchten Varianten die beste ist. Mit ihr haben wir gegenüber der reinen Greedy-Heuristik eine durchschnittliche Verkleinerung der Größe der Dominating Sets um $2,45\,\%$ bewirken können.

Wir überlegen nun in diesem Kapitel, was von den gerade genannten Ergebnissen ausgehend die nächsten Schritte sein könnten.

Da wir gezeigt haben, dass Dynamic Vertex Cover fixed-paramerter tractable bezüglich des Unterschieds in der Eingabe ist, können wir jetzt darauf aufbauend eine Turbo-Charging-Greedy-Heuristik für Vertex Cover entwickeln.

Wir können mithilfe unserer in Kapitel 4.4 vorgestellten Implementierung die Turbo-Charging-Heuristik auf Set Cover-Instanzen untersuchen. Dabei sollten wir die Datenreduktionsregeln von Weihe [14] implementieren.

In Kapitel 5 haben wir gesehen, dass mit einem größeren Backup die durchschnittliche Größe der Dominating Sets sinkt. Uns ist aber speziell bei der Verbesserung „deutlich kleinere Teilmengen“ aufgefallen, dass dieser Trend sich nicht ewig fortsetzen lässt. Die Berechnungszeit wird mit steigender Größe des Backups im Durchschnitt größer und deswegen wird die Heuristik häufiger abgebrochen. Dadurch ist irgendwann der Punkt erreicht, an dem die durchschnittliche Größe der Dominating Sets wieder steigt. Es wäre interessant diesen Punkt für jede Variante zu kennen. Dann könnten wir die maximale durchschnittliche Verbesserung jeder Variante der Turbo-Chaging-Heuristik gegenüber der bloßen Greedy-Heuristik ermitteln.

Wir haben in Kapitel 5 ebenfalls beobachten können, dass alle von uns entwickelten Verbesserungen sich im Mittel positiv auf die Qualität des Dominating Sets ausgewirkt haben. Es wäre daher interessant zu wissen, wie sich die Turbo-Charging-Heuristik verhalten würde, wenn wir mehrere Verbesserungen kombinieren würden.

Bis jetzt lang unser Fokus bei der Implementierung der Turbo-Charging-Greedy-Heuristik auf der Qualität der gefundenen Dominating Sets. Wir können uns nun stattdessen auch auf die Laufzeit konzentrieren und unseren Quellcode dahingehend verbessern.

Für die Vergleichbarkeit der Größen der gefundenen Dominating Sets wäre es interessant für jede Instanz die Größe des kleinstmöglichen Dominating Set zu wissen. Dazu könnten wir mittels integer-linear-programming die kleinstmöglichen Dominating Sets berechnen.

Bisher haben wir, bevor wir die Heuristik auf einem Graphen aufgerufen haben, keine Datenreduktionsregeln angewandt. Das können wir ändern und zum Beispiel die von Alber et al. [2] vorgestellten Datenreduktionsregeln für Dominating Set implementieren.

Die Instanzen, auf denen wir die Heuristik bislang untersucht haben, wurden alle von Jovanovic et al. [12] künstlich erzeugt. Es wäre jetzt interessant zu sehen, wie die Heuristik auf realen Daten arbeitet.

Fomin et al. [8] stellen einen Algorithmus vor, der in $\mathcal{O}(nm2^n)$ ein kleinstmögliches Set Cover für ein Universum $\mathcal{U}$ findet. Dabei steht n für die Anzahl der Elemente in $\mathcal{U}$ und m für die Anzahl der Teilmengen in $\mathcal{F}$. Wie wir in Kapitel 3.3 gesehen haben, sind sich Dominating Set und Set Cover sehr ähnlich. Es wäre also denkbar den Algorithmus Dom-A-VC (Algorith-

mus 4) durch den von Fomin et al. [8] beschriebenen Algorithmus für Set Cover zu ersetzen. Um diesen Algorithmus verwenden zu können, müssten wir dabei allerdings einige Anpassungen vornehmen.

Literatur

[1] F. N. Abu-Khzam, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. On the parameterized complexity of dynamic problems. *Theoretical Computer Science*, 607:426–434, 2015. 8, 16, 19, 20, 26

[2] J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for dominating set. *Journal of the ACM (JACM)*, 51(3):363–384, 2004. 48

[3] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. 10, 20, 21

[4] R. Diestel. *Graphentheorie*. Springer, 5th edition, 2016. 8

[5] R. G. Downey, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. Dynamic dominating set and turbo-charging greedy heuristics. *Tsinghua Science and Technology*, 19(4):329–337, 2014. 3, 4, 6, 7, 8, 14, 15, 16, 19, 24, 25, 26, 27, 28, 37, 38

[6] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*, volume 4. Springer, 2013. 10, 20, 21

[7] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. 10

[8] F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 245–256. Springer, 2004. 48, 49

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979. 7, 9, 11, 13, 19, 27

[10] S. Gaspers, J. Gudmundsson, M. Jones, J. Mestre, and S. Rümmele. Turbocharging treewidth heuristics. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation - IPEC 2016*, pages 13:1–13:13, 2016. 8

[11] S. Hartung and R. Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013. 7

[12] R. Jovanovic, M. Tuba, and D. Simian. Ant colony optimization applied to minimum weight dominating set problem. In *Proceedings of the 12th WSEAS International Conference on Automatic Control, Modelling & Simulation*, pages 322–326, 2010. 40, 48

[13] R. Krithika, A. Sahu, and P. Tale. Dynamic parameterized problems. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation - IPEC 2016*, pages 23:1–23:14, 2016. 8, 19

[14] K. Weihe. Covering trains by stations or the power of data reduction. *Proceedings of Algorithms and Experiments, ALEX*, pages 1–8, 1998. 48