Algorithmen zum Lösen von Vertex und Set Cover Instanzen zur Planung von Angriffen auf Netzwerke

–Stand: 29.06.2012–

Steve Göring

Zusammenfassung

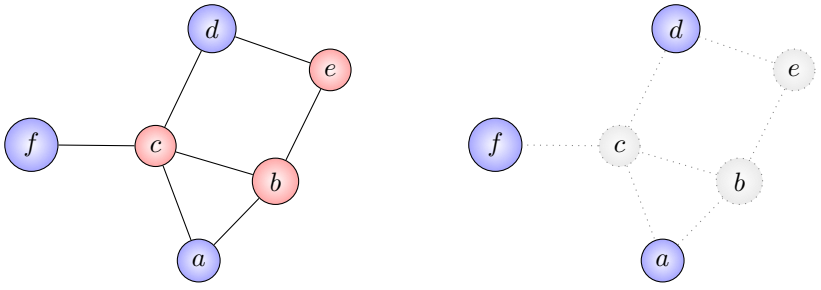
Angriffe auf Netzwerke betreffen meist nicht nur einzelne Knoten des Netzes. Das Planen solcher Angriffe kann durch Abstraktion des Netzwerkes als Graph stattfinden. Einen Totalausfall des Netzwerkes zu planen ähnelt dem Vertex-Cover- sowie Set-Cover-Problem. Die beiden Probleme sind *NP-vollständig*. In dieser Arbeit werden verschiedene Problemvarianten für allgemeine Fälle des Vertex-Cover- (ungewichtet, mit Größe k, gewichtet, ...) und des Set-Cover-Problems (ungewichtet und gewichtet) eingeführt. Weiterhin werden für die betrachteten Varianten Algorithmen beschrieben (Buss, Bar-Yehuda Even, Johnson, ...) und verglichen. Dabei handelt es sich um approximative, exakte oder randomisierte Algorithmen. Abschließend folgt eine Auswertung der Algorithmen bezüglich der möglichen Planung von Angriffen auf Netzwerke.

1 Einleitung

In den Medien hört man oft von Angriffen auf Netzwerke, beispielsweise zuletzt der Angriff auf die NASA Server, der als „Unknown Nasa Hack“ bekannt ist [Zeit]. Zur Realisierung von solchen Angriffen werden verschiedene Techniken verwendet. Dabei kommt es darauf an welchem Zweck der Angriff dienen soll, etwa dem Komplettausfall einer Webseite oder eines ganzen Webservices.

Auch Netzwerke, die nicht der Öffentlichkeit zugänglich sind, können ein Ziel darstellen, wie zum Beispiel virtuelle private Netzwerke von Firmen, die aufgebaut werden um transparent von verschiedenen Firmenstandorten zu arbeiten.

Im Vergleich zum Angriff auf einzelne Rechner ist es beim Attackieren von Netzwerken schwieriger die Angriffe zu planen. Besonders durch Techniken der Dezentralisierung und Replikation bedarf es intensiverer Planung bei einem Angriff genauso wie beim Analysieren eines Angriffsszenarios zum Schutz des Netzwerkes. Beispielsweise kann das in Abbildung 1(a) dar-



(a) Angriffsziele: b, c, e (b) nach dem Angriff

Abbildung 1: Beispielnetzwerk

gestellte vereinfachte Netzwerk betrachtet werden. Es besteht aus 6 Rechnern mit den Namen a, b, c, d, e und f. Die Topologie des Netzwerkes sei vorher ermittelt worden. Nun möchte ein potentieller „Angreifer“ mit möglichst wenig Aufwand so viel Schaden wie möglich anrichten, das heißt er möchte die Struktur des Netzes stark schädigen. Im Beispiel führt er einen Angriff auf die Knoten b, c und e aus und isoliert so den Rest (dargestellt in Abbildung 1(b)). Zum Verhindern eines Angriffs muss ein potenzieller „Beschützer“ des Netzwerkes die anfälligen Knoten identifizieren und besonders absichern.

Angriffe auf Netzwerke können unter Umständen nicht nur einzelne Knoten treffen, sondern auch benachbarte Knoten beeinflussen. Optimale Angriffe auf Overlay-Netzwerke zu betrachten kann unter Umständen jedoch sehr schwierig zu berechnen sein, da die Probleme dem Vertex-Cover und im allgemeinen Fall sogar dem schwierig zu approximierenden Set-Cover-Problem ähneln.

In diesem Hauptseminar sollen daher verschiedene Verfahren recherchiert und bewertet werden, welche die Vertex- beziehungsweise Set-Cover-Instanzen lösen. Insbesondere sollen dabei „angegriffene“ Knoten unterschiedliche Gewichte besitzen können und partielle Überdeckungen möglich sein. Von Interesse sind sowohl exakte Algorithmen mit niedriger Laufzeit als auch Approximationsalgorithmen.

In der Arbeit werden zunächst die Notationen und Problemvarianten in Kapitel 2 beschrieben. Anschließend werden in Kapitel 3 verschiedene Approximationsalgorithmen und exakte Algorithmen, die die aufgeführten Problemvarianten des Vertex-Cover und Set-Cover-Problems lösen, beschrieben. Danach folgt in Abschnitt 4 eine kurze Auswertung und Gegenüberstellung der verschiedenen Algorithmen. Abschließend folgt in Kapitel 5 ein Fazit.

2 Grundlagen

Im folgenden Kapitel werden einige Notationen festgelegt und formale Beschreibungen des Vertex-Cover und des Set-Cover-Problems dargestellt. Dabei werden auch verschiedene Varianten der Grundprobleme eingeführt und erläutert. Für die später beschriebenen Algorithmen ist das erforderlich.

2.1 Notationen

Zunächst sollen Notationen zur Beschreibung von Graphen und besondere Laufzeitnotationen festgelegt werden. Anschließend folgen Hinweise zur Genaugigkeitsnotation der Approximationsalgorithmen dieser Arbeit.

2.1.1 Graphennotationen

Ein *Netzwerk* wird üblicherweise als *(un)gerichteter Graph* $G = (V, E)$ aufgefasst. V stellt dabei die *Knotenmenge* und E die *Kantenmenge* dar. Im gerichteten Fall ist $E \subseteq V \times V$ und im ungerichteten ist $E \subseteq \{\{a, b\} \mid a \in V \wedge b \in V\}$. Für den weiteren Verlauf wird festgelegt, dass $|V| = n$ die Anzahl der Knoten und $|E| = m$ die Anzahl der Kanten beschreiben. Außerdem wird keine genauere Unterscheidung getroffen, ob der Graph gerichtet oder ungerichtet ist.

Ein *gewichteter Graph* $G = (V, E, w_v)$ ist, sofern nicht im Kontext anders beschrieben, ein Graph, dessen Knoten gewichtet sind. Dabei ist $w_v : V \to \mathbb{R}_+$ die Knotenkostenfunktion (Kostenfunktion). Ferner gilt für einen ungewichteten Graphen $\forall x \in V : w_v(x) = 1$.

2.1.2 Laufzeitnotationen

Für einige Algorithmen, insbesondere die mit exponentieller Laufzeit, ist der polynomielle Anteil in der Laufzeit von geringerer Bedeutung. In diesem Fall wird die Notation $\mathcal{O}^*$ benutzt. Alle Algorithmen mit polynomieller Laufzeit benutzen die übliche $\mathcal{O}$ Notation.

2.1.3 r-Approximationsalgorithmen

Im weiteren Verlauf werden unter anderen auch Approximationsalgorithmen beschrieben. Dabei bedeutet „r-Approximationsalgorithmus“, dass die gefundene Lösung maximal um den Faktor r von der optimalen Lösung abweicht, auch als Genauigkeit r eines Algorithmus bezeichnet.

2.2 Vertex-Cover-Problem

Unter einem Vertex-Cover versteht man in der Graphentheorie eine Menge von Knoten eines Graphen, so dass alle Kanten des Graphen inzident zu mindestens einem dieser Knoten sind. Intuitiv erkennt man, dass alle Knoten des Graphen die genannte Eigenschaft erfüllen, doch die Menge der Knoten soll weiteren Bedingungen genügen. Mit $VC(G)$ soll die Menge aller Vertex-Cover des Graphen G bezeichnet werden.

2.2.1 Minimales Vertex-Cover-Problem

Zunächst kann man die Größe des Vertex-Covers betrachten. Das führt direkt zur Formulierung des *minimalen Vertex-Cover-Problems* (mVCP). Beim *mVCP* soll ein *Vertex-Cover* mit minimaler Größe gefunden werden [PrKu92]. Formal ist ein *Vertex-Cover* $C \subseteq V$ dabei eine Knotenüberdeckung des Graphen, das heißt es gilt:

$$\forall (u, v) \in E : u \in C \vee v \in C$$

Ist $k = |C|$ die minimale Anzahl an Knoten eines *Vertex-Covers*, das heißt

$$\forall C_i \in VC(G) : |C_i| \geq |C|$$

so wird C Lösung des *mVCP*s genannt. Zur Verdeutlichung des *mVCP*s soll nun ein Beispiel eines Graphen betrachtet werden.

Beispiel — $V = \{a, b, c, d, e, f\}$, $E = \{\{a, b\}, \{a, c\}, \{c, b\}, \{c, f\}, \{c, d\}, \{d, e\}, \{e, b\}\}$

Der hier formal beschrieben Graph $G = (V, E)$ greift das Anfangsbeispiel, dargestellt in Abbildung 1(a), auf. Es ist erkennbar, dass die Knoten b, c und e ein Vertex-Cover mit minimaler Größe bilden.

Das Auffinden einer Lösung des *mVCP*s ist *NP-vollständig* (vergleiche [DiSa05]). Ein Beweis findet man in [CLRS09, Kapitel 34, Seite 1090f]. Dort wird eine polynomielle Reduktion des Cliquen-Problems (Clique ist *NP-vollständig*) auf das Vertex-Cover-Problem beschrieben, das heißt:

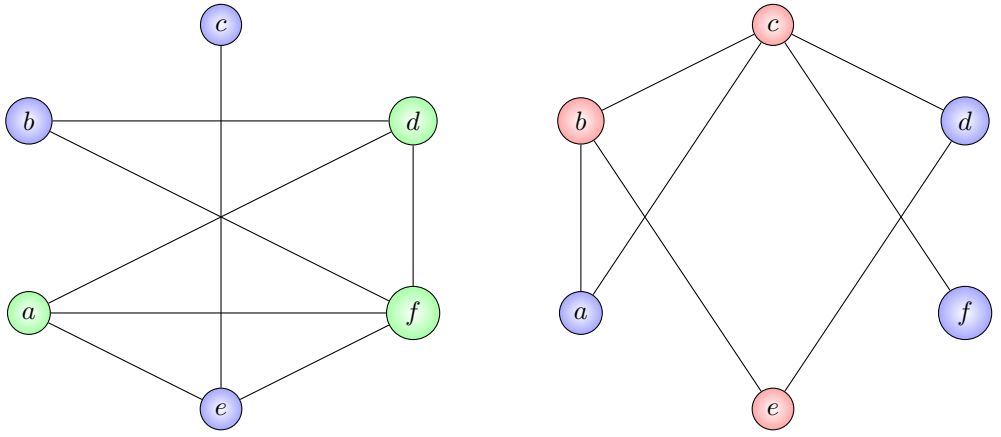
$$CLIQUE \leq_p VC$$

Beweis. Im ersten Schritt muss gezeigt werden, dass $Vertex - Cover \in NP$ gilt. Gegeben sei ein Graph $G = (V, E)$ und $k \in \mathbb{N}_+$. Rät man nun eine Lösung $C' \subseteq V$, kann ein Verifikationsalgorithmus in polynomieller Zeit überprüfen ob $|C'| = k$ und C' alle Kanten E überdeckt.

Im zweiten Schritt wird die Reduktion $CLIQUE \leq_p VC$ beschrieben. Die Grundidee der Reduktion beruht darauf, dass aus der Eingabe (G, k) für Clique eine Eingabe (G', k') für das Vertex-Cover Problem in polynomieller Zeit konstruiert wird, so dass gilt:

$$G \text{ hat } k\text{-Clique } \Leftrightarrow G' \text{ hat } k'\text{-Vertex-Cover}$$

Dabei wird als G' das Komplement von G gewählt (vergleiche Abbildung 2), also $G' = (V, E')$



(a) G, $\{a, d, f\}$ bilden 3-Clique (b) G', $\{b, c, e\}$ bilden 3-Vertex-Cover

Abbildung 2: Beispiel für Reduktion von Clique auf Vertex-Cover

mit $E' = (V \times V) \setminus E$. Die Konstruktion von G' ist in polynomieller Zeit möglich. Im nächsten Schritt muss gezeigt werden, dass gilt:

$$G \text{ hat } k\text{-Clique } \Leftrightarrow G' \text{ hat } \underbrace{(n-k)}_{=k'}\text{-Vertex-Cover}$$

Dazu sei zunächst $K \subset V$ eine k-Clique von G. Aus der Definition einer k-Clique folgt direkt für K:

$$\forall u, v \in K : (u, v) \in E, \text{ mit } |K| = k$$

In G' heißt das für K:

$$\forall u, v \in K : (u, v) \notin E', \text{ mit } |K| = k$$

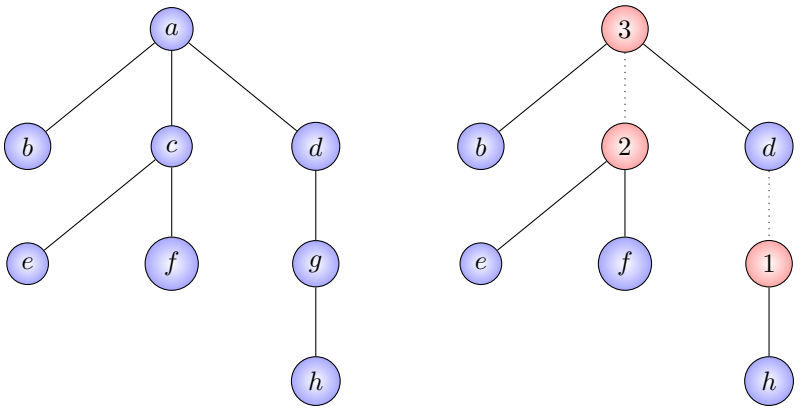
Betrachtet man nun die Menge $C = V \setminus K$, so folgt unmittelbar

$$\forall (u, v) \in E' : u \in C \vee v \in C, \text{ mit } |C| = n - k$$

was genau der Definition eines Vertex-Covers entspricht. □

Genauer gesagt sind alle in dieser Arbeit aufgeführten Varianten des Vertex-Cover-Problems *NP-vollständig*. Nur durch sehr starke Vereinfachungen im Graphen kann man ein Vertex-Cover in polynomieller Zeit exakt bestimmen.

Ein Beispiel wäre hier die Betrachtung von Vertex-Covern in Bäumen (Übungsaufgabe aus [CLRS09, Kapitel 35, Seite 1111, Aufgabe 35.1-4]). Ein möglicher Algorithmus VCT zum Auffinden eines Vertex-Covers in einem Baum würde zunächst ein Blatt betrachten. Da ein Blatt v eines Baums den Knotengrad $deg(v) = 1$ besitzt, ist es offensichtlich, dass es günstiger



(a) Baum (b) Iterationsschritte

Abbildung 3: Vertex-Cover Baum: Beispiel

ist den Elternknoten w des Blattes v in das Vertex-Cover C aufzunehmen, da w potentiell mehr Kanten überdeckt. Anschließend werden alle Kanten, die zu w führen gelöscht und das Verfahren iterativ auf dem verkleinerten Baum ausgeführt.

In den Abbildungen 3(a) und 3(b) ist ein Beispieldurchlauf des beschriebenen Algorithmus dargestellt. Im ersten Schritt wird der Knoten h als Blatt identifiziert und der Elternknoten g zum noch leeren Vertex-Cover $C = \{\}$ hinzugefügt. Darauf wird die Kante $\{d, g\}$ aus dem Baum entfernt und nur noch der verbleibende Baum, ohne die Knoten h und g, betrachtet. Der zweite Durchlauf bestimmt nun den Knoten f als Blatt. Anschließend wird der Knoten c der Menge $C = \{g\}$ hinzugefügt und die Kante $\{a, c\}$ gelöscht. Im verbleibenden letzten Schritt wird eines der Blätter b oder d gefunden und der Wurzelknoten a der Menge $C = \{g, c\}$ hinzugefügt. Der Baum wurde nun vollständig abgearbeitet und der Algorithmus hat die Menge $C = \{g, c, a\}$ als minimales Vertex-Cover berechnet. In jedem Schritt des Algorithmus wird eine Kante gelöscht, solange bis keine weiteren Kanten verbleibend sind. Die Blätter des Baums können in linearer Zeit gefunden werden. Eine Aktualisierung in jedem Schritt, in dem der Baum verändert wird, ist einfach zu implementieren. Der beschriebene Algorithmus zeigt, dass ein minimales Vertex-Cover in einem Baum in linearer Zeit $\mathcal{O}(m)$ gefunden werden kann.

Beim Betrachten anderer vereinfachter Graphen können ähnliche Algorithmen zum Finden von Vertex-Cover Lösungen bestimmt werden. Beispielsweise können Matching-Algorithmen Vertex-Cover in bipartiten Graphen in polynomieller Zeit finden. In dieser Arbeit werden jedoch Netzwerke durch Graphen repräsentiert, das heißt, dass die Graphen, mit denen man es bei der Planung eines Angriffs zu tun hat, keine Bäume oder andere vereinfachte Graphen sind. Daher sollen im weiteren Verlauf nur noch allgemeine Graphen betrachtet werden.

2.2.2 k-Vertex-Cover

Es ist nicht immer erforderlich ein minimales Vertex-Cover zu finden, so könnte eine Wunschgröße des Vertex-Covers von Bedeutung sein. Eine parametrisierte Variante des VCPs, genannt das *k-Vertex-Cover-Problem* (kVCP), beschreibt genau den genannten Fall, weil speziell ein *Vertex-Cover* C der Größe $k = |C|$ gesucht [Tail09] wird.

2.2.3 Gewichtetes Vertex-Cover

Gerade im Falle eines Angriffsszenarios könnten einige Knoten eine besondere Rolle spielen. Beispielsweise könnten es Knoten sein, die leichter anzugreifen sind, denen eine besonde-

re Bedeutung zugewiesen wird oder deren Angriff mehr Aufwand erfordern. Zur formalen Beschreibung der aufgeführten Situation können den Knoten Gewichte zugeordnet werden. Betrachtet wird dabei eine verallgemeinerte Variante des *mVCP*s, namens *gewichtetes Vertex-Cover-Problem* (gVCP). Es soll dabei ein *Vertex-Cover* C mit minimalem Gesamtgewicht gefunden werden, genauer muss C die Bedingung $\sum_{x \in C} w_v(x) \rightarrow \min$ erfüllen [GHKO03].

2.2.4 Partielles Vertex-Cover

Der Angriff, dargestellt in Abbildung 1(b), erreicht die vollständige Isolation aller Knoten des Netzwerkes. Ein Angreifer könnte ebenso gut auch als Ziel einen partiellen Ausfall des Netzwerkes haben. In der Graphentheorie wird das als *partielles Vertex-Cover-Problem* (pVCP) bezeichnet. Dabei soll ein *Vertex-Cover* C mit $|C| \leq k$ gefunden werden, welches mindestens t Kanten überdeckt [KnLR08]. Formal bedeutet das, dass die folgenden Bedingungen für C erfüllt werden müssen:

$$|\{(u,v) \in E \mid u \in C \vee v \in C\}| \geq t \text{ mit } |C| \leq k$$

2.2.5 Exaktes partielles Vertex-Cover

Beim *exakten partiellen Vertex-Cover-Problem* (epVCP) ist die Einschränkung an das Vertex-Cover noch höher als beim *pVCP*. Hierbei muss das Vertex-Cover C aus genau k Knoten bestehen.

2.2.6 Formulierung als ganzzahliges lineares Optimierungsproblem

Die aufgeführten Bedingungen des *mVCP*s und des *gVCP*s stellten Minimierungsprozesse dar. Es liegt nahe eine Lösung der Probleme in der Optimierung zu suchen. Zunächst soll das *gVCP* als Optimierungsproblem formuliert werden. Als Zielfunktion ergibt sich:

$$F(\vec{x}) = \sum_{x_i} w_v(i) \cdot x_i \rightarrow min$$

mit den Nebenbedingungen:

$$x_i + x_j \geq 1, \forall (i,j) \in E$$
$$x_i \in \{0,1\}$$

$x_i = 1$ bedeutet dabei, dass der Knoten i im Vertex-Cover enthalten ist. Es ist erkennbar, dass es sich um ein *ganzzahliges lineares Optimierungsproblem* (gLOP) handelt [Halp99]. Bei *gLOP*en besteht eine Lösungsmöglichkeit in der Betrachtung des dazugehörigen *(relaxierten) linearen Optimierungsproblems* (LOP). Das lineare Optimierungsproblem entsteht aus dem *gLOP* durch Vernachlässigung der Ganzzahligkeitsbedingung (der Vorgang wird bei binären Variablen als LP-Relaxation bezeichnet, vergleiche [DoDr, Kapitel 6, Seite 135ff]). Das relaxierte *LOP* zum *gVCP* ist:

$$F(\vec{x}) = \sum_{x_i} w_v(i) \cdot x_i \rightarrow min$$
$$x_i + x_j \geq 1, \forall (i,j) \in E$$
$$\boxed{0 \leq x_i \leq 1, x_i \in \mathbb{R}}$$

Die Formulierung als lineares Optimierungsproblem bildet die Grundlage für einige der in Abschnitt 3.1 beschriebenen Algorithmen. Üblicherweise betrachtet man zu *LOP*en auch die

dazugehörigen dualen Probleme. Das duale Problem zum *(gewichteten) Vertex-Cover-Problem* ist das *maximale Matching-Problem*. Auch die anderen Varianten des Vertex-Cover Problems können analog als *gLOP* formuliert werden.

Das relaxierte *LOP* kann genutzt werden um einen 2-Approximationsalgorithmus zu entwerfen, in dem man das *LOP* mit Verfahren aus der Optimierung löst (Simplex oder Khachian/Karmarkar).

Mittels „Branch-and-Bound“ könnte auch das ganzzahlige LOP gelöst werden, aber für Angriffe müssen nicht immer die optimalen Lösungen gefunden werden. Eine gute Annäherung ist oft ausreichend. Ein Vorteil der Approximationsalgorithmen ist, dass sie schneller und einfacher als der „Branch-and-Bound“- Ansatz sind.

2.3 Set-Cover-Problem

Bisher wurden nur einfache Graphen ohne Mehrfachkanten betrachtet. In einigen Angriffsszenarien stehen aber nicht die Informationen zur kompletten Topologie zur Verfügung, zum Beispiel wenn nur bekannt ist, dass bestimmte Knotenmengen verbunden sind, nicht aber auf welchen Wegen. Diese Formulierung führt direkt zur Abstraktion des Netzwerkes als einen Hypergraphen $H = (X, E)$ mit der Menge von Elementen X und der Menge $E \subseteq 2^X$ von Teilmengen von X. Das bedeutet, dass ein Hypergraph Kanten besitzt, die mehr als 2 Knoten verbinden können. Ein Vertex-Cover in einem Hypergraphen zu finden entspricht dem Set-Cover-Problem.

Ein Set-Cover ist eine Mengenüberdeckung. Es wird eine Auswahl von Teilmengen eines Universums gesucht, die vereinigt alle Elemente des Universums überdecken. Genau wie beim Vertex-Cover-Problem können verschiedene Varianten unterschieden werden. Zunächst soll die ungewichtete Variante und anschließend die gewichtete betrachtet werden.

2.3.1 Minimales Set-Cover-Problem

Das *minimale Set-Cover-Problem* (mSCP) ist ein kombinatorisches Problem, bei dem, ausgehend von einer Menge $U = \{u_1, ..., u_m\}$ (Universium) mit $m = |U|$ und n Teilmengen $\{V_1, ..., V_n\} = S$, $V_i \subset U$, eine Auswahl $Z \subseteq \{1, ..., n\}$ der Teilmengen V_i gefunden werden soll, die $\bigcup_{i \in Z} V_i = U$ mit minimalem $k = |Z|$ erfüllt (ähnlich [CyKW09]). Zur einfacheren Darstellung der Teilmengen wird die folgende Matrix $A \in \{0, 1\}^{n \times m}$ definiert mit

$$a_{i,j} = \begin{cases} 1 & \text{, wenn } u_j \in V_i \\ 0 & \text{, sonst} \end{cases}$$

(vergleiche [CaFT98]). Die Zeile i der Matrix A entspricht dem charakteristischen Vektor der Menge V_i bezüglich U. Zunächst soll ein Beispiel betrachtet werden.

Beispiel — $U = \{1, 2, ..., 5\}$, $V_1 = \{1\}$, $V_2 = \{1, 2\}$, $V_3 = \{3, 4, 5\}$, $S = \{V_1, V_2, V_3\}$ Die Abbilding 4 stellt das Beispiel grafisch dar. Eine mögliche Auswahl wäre $Z_1 = \{1, 2, 3\}$ $(k = 3)$. Sie erfüllt aber nicht die Minimalitätsbedingung, denn $Z_2 = \{1, 3\}$ $(k = 2)$ überdeckt U mit weniger Elementen aus S (genauer: Z_2 ist die einzige Lösung des *mSCP*s im Beispiel). Für die definierte Matrix A ergibt sich dann:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

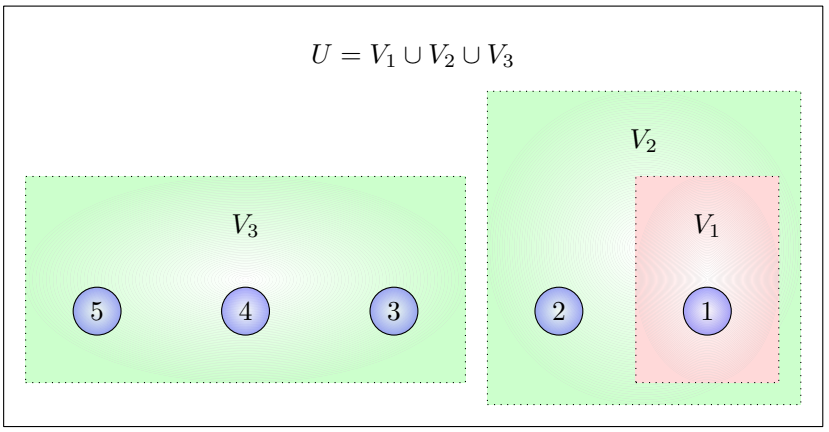


Abbildung 4: Beispiel für Set-Cover: $\{V_2, V_3\}$ bildet minimale Set-Cover Lösung

Auch wenn es für einfache Beispiele nicht schwer ist eine Lösung zu finden, gehört das *Set-Cover-Problem* auch zur Klasse der *NP-vollständigen* Probleme. Ein Reduktionsbeweis ist in [Gold10, Kapitel 4, Seite 114f] zu finden. Es wird eine Reduktion von SAT zu Set-Cover beschrieben. Da das Set-Cover-Problem auch als allgemeineres Vertex-Cover-Problem betrachtet werden kann, wird an dieser Stelle auf den Reduktionsbeweis verzichtet.

2.3.2 Gewichtetes Set-Cover

Die zweite Variante des Set-Cover-Problems ist das *gewichtete Set-Cover-Problem* (gSCP). Es soll ein Set-Cover mit minimalen Gesamtkosten gesucht werden. Eine Kostenfunktion c mit $c : S \to \mathbb{R}_+$ ordnet dabei jeder Menge $V_i \in S$ ein Gewicht $c(V_i)$ zu. Für die Auswahl Z muss dann gelten

$$\sum_{i \in Z} c(V_i) \to \min$$

(ähnlich [CaFT98]).

2.3.3 Formulierung als ganzzahliges lineares Optimierungsproblem

Ähnlich der Formulierung des Vertex-Cover-Problems als Optimierungsproblem kann auch das *gSCP* als *ganzzahliges lineares Optimierungsproblem* beschrieben werden [CaFT98], mit der Zielfunktion:

$$F(\vec{x}) = \sum_{j=1}^{n} c(V_j) \cdot x_j \to \min$$

unter den Nebenbedingungen:

$$\sum_{j=1}^{m} a_{i,j} \cdot x_j \geq 1, i = 1, ..., n$$

$$x_j \in \{0, 1\}, j = 1, ..., m$$

$x_j = 1$ heißt dabei, dass die Menge V_j in der Lösung des Set-Covers enthalten ist.

Analog zum *Vertex-Cover-Problem* kann auch das *Set-Cover-Problem* durch LP-Relaxation zu einem *relaxierten linearen Optimierungsproblem* umformuliert werden, dabei ändert sich nur die Ganzzahligkeitsbedingung in:

$$0 \leq x_j \leq 1, j = 1, ..., m$$

Einige der Approximationsalgorithmen in 3.2 benutzen das relaxierte *LOP* als Grundlage. Das duale Problem zum *(gewichteten) Set-Cover-Problem* ist das *Set-Packing-Problem.*

3 Lösungsalgorithmen

In diesem Kapitel werden zu den im Abschnitt 2 eingeführten Varianten des Vertex-Cover- und Set-Cover-Problems verschiedene Algorithmen beschrieben. Zuerst werden die Algorithmen für die Vertex-Cover-Varianten betrachtet, anschließend die Set-Cover-Probleme.

3.1 Vertex-Cover

Das Vertex-Cover-Problem wurde in Abschnitt 2.2 beschrieben. Da es ein *NP-vollständiges* Problem ist, sind keine polynomiellen exakten Lösungsalgorithmen bekannt.

3.1.1 Minimales Vertex-Cover-Problem

Beim *mVCP* bekommt der Algorithmus als Eingabe den Graphen $G = (V, E)$ und bestimmt ein minimales Vertex-Cover.

Naiver Vertex-Cover-Algorithmus

Um eine Vorstellung der Laufzeitdimensionen zu erhalten, soll zunächst ein naiver Algorithmus für das *mVCP* betrachtet werden:

```
1 function NAIVMVC(Graph G = (V, E))
2     for V_i ⊆ V ascending sorted do
3         if V_i covers all Edges E then
4             return V_i
5         end if
6     end for
7 end function
```

Die Grundidee ist, dass alle Teilmengen $V_i \subseteq V$ $(i = 1...2^n)$ aufsteigend betrachtet werden und jeweils ein Test stattfindet, ob alle Kanten E überdeckt werden. Intuitiv ist die Laufzeit $\mathcal{O}(2^n m)$.

Cormen et al

Der naive Algorithmus läuft in exponentieller Zeit, aber nicht immer ist eine optimale Lösung erforderlich. Daher wird auch in [CLRS09, Kapitel 35, Seite 1109] ein 2-Approximationsalgorithmus beschrieben.

```
1  function CORMEN(Graph G = (V, E))
2      C ← ∅
3      E' ← E
4      while E'! = ∅ do
5          let (u, v) be an arbitrary edge from E'
6          C = C ∪ {u, v}
7          remove from E' every edge incident on either u or v
8      end while
9      return C
10 end function
```

Der beschriebene Algorithmus wählt in jedem Schritt eine Kante (u, v) aus und fügt die Knoten u und v zum Vertex-Cover hinzu. Anschließend werden alle Kanten inzident zu u oder v nicht mehr betrachtet. Bei der Nutzung von Adjazenzlisten zur Darstellung der Kanten des Graphens erhält man einen Algorithmus mit der Laufzeit $\mathcal{O}(m + n)$ und Genauigkeitsfaktor 2.

Da in jeder Iteration nur Kanten entfernt werden, die die bereits hinzugefügte Knoten überdecken, ist offensichtlich, dass am Ende des Algorithmus ein Vertex-Cover zurückgegeben wird. Nicht so offensichtlich ist die Aussage, dass die zurückgegebene Lösung maximal doppelt so groß ist wie die optimale Lösung (vergleiche [CLRS09, Kapitel 35, Seite 1100]).

Beweis. Zunächst sei die in Zeile 5 gewählte Kantenmenge mit A bezeichnet. Betrachtet man nun eine optimale Lösung C^*, so muss C^* auch die Menge A überdecken. In A gibt es keine Kanten mit gemeinsamen Endpunkten, da in Zeile 7 in jeder Iteration die Kanten gelöscht werden, deren Knoten inzident zu der gewählten Kante aus A sind. Für C^* folgt, dass es mindestens einen Knoten jeder Kante aus A beinhalten muss. Somit folgt:

$$|C^*| \geq |A| \tag{1}$$

Für das berechnete Vertex-Cover C ergibt sich, da in jedem Schritt die Knoten der Kanten aus A hinzugefügt werden und sie in A nicht mehrfach vorkommen, automatisch:

$$|C| = 2|A| \tag{2}$$

Kombiniert man (1) und (2), so folgt direkt:

$$|C| \leq 2|C^*|$$

□

Monien-Speckenmeyer

Approximationsalgorithmen versuchen einen Kompromiss zwischen Genauigkeit und Laufzeit zu schaffen. Möchte man einen genaueren Algorithmus haben, so müssen weitere Einschränkungen an die Eingabe gestellt oder eine höhere Laufzeit akzeptiert werden. So wird auch in [MoSp85] ein weiterer Approximationsalgorithmus beschrieben, der eine Genauigkeit von $2 - \frac{1}{k+1}$ aufweist, wobei k die kleinste Zahl ist, die die Bedingung $|V| \leq (2k + 3)^k \cdot (2k + 2)$ erfüllt und außerdem in Zeit $\mathcal{O}(|V| \cdot |E|)$ das minimale Vertex-Cover bestimmt.

```
1  function MonienSpeckenmeyer(Graph G = (V, E))
2      //Phase (1)
3      determine smallest k , that satisfies: |V| ≤ (2k + 3)^k · (2k + 2)
4      M ← ∅
5      while there is an odd circle C of length ≤ 2k + 1 do
6          M = M ∪ V(C)
7          G = G − V(C)
8          delete all isolated vertices from G
9      end while
10     //Phase (2)
11     apply the Nemhauser-Trotter-algorithm to G yielding a partition V1, V2, V3
12     M = M ∪ V2
13     G' = G − (V1 ∪ V2)
14     //Phase (3)
15     apply the algorithm A1 to G', which computes an independent set I ⊆ V(G') for G'
16     M' = V(G') − I
17     M = M ∪ M'
```

```
18      return Vertex-Cover M
19  end function
```

Zum Verständnis des Algorithmus muss man wissen, was der Nemhauser-Trotter-Algorithmus, der in Zeile 11 benutzt wird, berechnet. Er ermittelt, mit der Laufzeit $\mathcal{O}(\sqrt{n} \cdot m)$, eine Partitionierung der Knoten mit den Eigenschaften:

a) V_2 ist ein minimales Vertex-Cover für den Graph, der aus $V_1 \cup V_2$ hervorgeht,

b) es gibt keine Kanten $\{u, v\}$ mit $u \in V_1$ und $v \in V_3$,

c) sei C^* das minimale Vertex-Cover des Graphen G', das sich aus den Knoten der Menge V_3 ergibt, dann gilt: $|C^*| \geq \frac{1}{2}|V_3|$

Grundlegend wird beim Nemhauser-Trotter-Algorithmus ein minimales Vertex-Cover in einem vorher konstruierten bipartiten Graphen bestimmt. Dabei wird automatisch die Partitionierung vorgenommen. Matching-Algorithmen für kardinalsmaximales Matching können verwendet werden, um minimale Vertex-Cover in bipartiten Graphen zu finden. Für den Algorithmus Monien-Speckenmeyer ist außerdem der benannte Algorithmus $A1$ wichtig. Er bestimmt ein Independent Set $I \subseteq V$ in Laufzeit $\mathcal{O}(m \cdot n)$ und ist ausschlaggebend für die Laufzeit des gesamten Algorithmus. Dabei ist ein Independet Set eine Menge von Knoten eines Graphens, zwischen denen es keine Kanten gibt. Auf den Nachweiß der Korrektheit und des Approximationsfaktors wird in dieser Arbeit verzichtet, da der Algorithmus auf mehreren Teilalgorithmen, die für sich betrachtet nicht trivial sind, aufbaut (für den Beweis siehe [MoSp85]).

3.1.2 k-Vertex-Cover-Problem

In diesem Abschnitt werden Algorithmen für das k-Vertex-Cover-Problem beschrieben. Sie können auch benutzt werden um Lösungen für das im vorhergehenden Abschnitt 3.1.1 beschriebene minimale VCP zu finden, indem von $k = 1, ..., n$ der k-Vertex-Cover Algorithmus angewendet wird. Falls der k-Vertex-Cover Algorithmus nun ein Vertex-Cover bestimmen kann, so wurde ein minimales Vertex-Cover gefunden. Algorithmen für das k-Vertex-Cover Problem erhalten als Eingabe den Graphen $G = (V, E)$ und zusätzlich die Größe k des zu ermittelnden Vertex-Covers.

Buss

Ein bekannter Algorithmus für das *kVCP* ist der unter anderen in [BaFR98] beschriebene Algorithmus von Buss.

```
1  function Buss(Graph G = (V, E) , k)
2      C = ∅
3      K = {v ∈ V | grad(v) ≥ k} // get all vertices with grad(v) ≥ k
4      if |K| > k then
5          // there is no vertex-cover of size k
6          return ∅ and exit
7      end if
8      C = K
9      V = V − K
10     remove all edges incident with nodes from K
11     k' = k − b
12     remove any resulting isolated vertices in G
13     if |E| > k · k' then
```

```
14          // there is no vertex-cover of size k
15          return ∅ and exit
16      end if
17      if resulting graph has a vertex cover of size k' (test with brute-force) then
18          return K and exit
19      else
20          return ∅ and exit
21      end if
22  end function
```

Grundlegend bedient sich der Algorithmus von Buss der Technik der Reduktion. Hierbei wird eine Transformation des Problems in ein anderes Problem, dessen Problemgröße durch k beschränkt ist, durchgeführt. Der Algorithmus benutzt die Beobachtung, dass jeder Knoten v eines Vertex-Covers der Größe k mindestens den $grad(v) \geq k$ besitzen muss. Da in der Zeile 17 die Brute-Force-Methode eingesetzt wird, um ein Vertex-Cover in einem veränderten Graphen zu finden, folgt unmittelbar, dass der Algorithmus exponentielle Laufzeit haben muss. In der genauen Analyse wird gezeigt, dass sich eine Laufzeit von $\mathcal{O}(kn + 2^k \cdot k^{2k+2})$ ergibt.

In [BaFR98] werden auch noch weitere Algorithmen zum Lösen des k-Vertex-Cover-Problems beschrieben, deren Laufzeiten ebenfalls exponentiell sind und sich nur gering unterscheiden.

3.1.3 Gewichtetes Vertex-Cover

Eine interessantere Problemvariante des Vertex-Cover-Problems ist das gewichtete Vertex-Cover-Problem. Jeder Knoten erhält dazu ein Gewicht und es soll ein Vertex-Cover mit minimalem Gesamtgewicht gefunden werden. Jeder der Algorithmen erhält als Eingabe den gewichteten Graphen $G = (V, E, w_v)$.

Bar-Yehuda Even

Zuerst wird der Faktor-2-Approximationsalgorithmus von Bar-Yehuda und Even für das *gV-CP* betrachtet. In [BYEv81] wird er als Algorithmus zum Lösen des Set-Cover-Problems beschrieben, aber er kann auch im Spezialfall eines gewichteten Vertex-Covers eingesetzt werden (vergleiche [BYEv85, Seite 34]) und liefert dabei gute Ergebnisse.

```
1  function BarYehudaEven(Graph G = (V, E, w_v))
2      for e ∈ E do
3          d = min{w_v(u) | u ∈ E}
4          for u ∈ e do
5              w_v(u) = w_v(u) − d
6              // Note: w_v(u) always ≥ 0
7          end for
8      end for
9      return C = {u | w_v(u) = 0}
10 end function
```

Das Grundkonzept des Algorithmus ist eine iterative Gewichtsreduktion der Knoten des Graphens. Es ist leicht erkennbar, dass sich eine Laufzeit von $\mathcal{O}(m)$ ergibt, denn die innere Schleife in Zeile 4 ist für Graphen in konstant zwei Schritten abgearbeitet. Nicht so offensichtlich ist, dass die Menge C, die in Zeile 9 berechnet wird, ein Vertex-Cover ist und zusätzlich die Eigenschaft $\sum_{c \in C} w_v(c) \leq 2 \cdot \sum_{c \in C^*} w_v(c)$ erfüllt, wobei C^* ein optimales gewichtetes Vertex-Cover sei. In [BYEv85, Seite 31ff] wird durch mehrere Schritte ein Beweis angegeben. Grundlegend baut der Beweis auf folgender Überlegung auf: Seien $W(X, w_v) = \sum_{x \in X} w_v(x)$ und w, w_1, w_2

verschiedene Gewichtsfunktionen, die $w(v) \geq w_1(v) + w_2(v)$ erfüllen. So folgt für die optimalen gewichteten Vertex-Cover C^*, C_1^* und C_2^* bezüglich der Gewichtsfunktionen w, w_1, w_2 in $G : W(C^*, w_v) \geq W(C^*, w_1) + W(C^*, w_2)$.

Halperin

Ein weiterer Approximationsalgorithmus zum Lösen des gewichteten Vertex-Cover-Problems ist der von Halperin in [Halp99] beschriebene. Der Algorithmus baut auf einer veränderten Relaxation des Optimierungsproblems auf:

$$\begin{gathered} F(\vec{v_i}) = \sum_{i \in V} w_v(i) \cdot \tfrac{1+v_0 \cdot v_i}{2} \to \min \\ v_i \in \mathbb{R}^n, ||v_i|| = 1, \forall i \in V \\ (v_j - v_0) \cdot (v_i - v_0) = 0, \forall (i,j) \in E \end{gathered} \tag{3}$$

```
1  function HALPERIN(Graph G = (V, E, w_v))
2      solve relaxation (3)
3      S_1 = {i | v_0 · v_i ≥ x}
4      S_2 = {i | v_0 · v_i ≤ x}
5      find a large independent set I in S_2:
6          Choose an n-dimensional random vector r, and let be I' = {i ∈ S_2 | v_i · r ≥ c}
7          remove vertices from I' which are non isolated in the subgraph induced by I'
8          the remaining vertices form an independent set I
9      return S_1 ∪ (S_2 \ I)
10 end function
```

Für den Algorithmus von Halperin sind zusätzlich noch die Konstanten x und c erforderlich. Sei $\Delta = \max_{v \in V}\{grad(v)\}$ der maximale Grad des Graphens, so muss x mit $x = \theta(\frac{\ln \ln \Delta}{\ln \Delta})$ und $c = \sqrt{\frac{2x}{1-x} \cdot \ln \Delta}$ gewählt werden, um eine Approximationsrate von $2 - 2x + o(x)$ zu erhalten. Die Laufzeit ist hauptsächlich durch das Lösen der Relaxation bestimmt und hängt vom verwendeten Verfahren ab. Benutzt man den Algorithmus von Khachian/Karmarkar, so ergibt sich eine polynomielle Laufzeit.

3.1.4 Partielles Vertex-Cover

In diesem Abschnitt sollen Algorithmen zum Lösen des partiellen Vertex-Cover-Problems vorgestellt werden. Die jeweiligen Algorithmen erhalten als Eingabe den Graphen $G = (V, E)$ und die ganzen Zahlen t und k. Es soll ein partielles Vertex-Cover der Größe $\leq k$ gefunden werden, welches mindestens t Kanten überdeckt.

Kneis et al - deterministisch

Zuerst soll der deterministische Algorithmus, der in [KnLR08] beschrieben wird, betrachtet werden. Er testet, ob ein partielles Vertex-Cover im Graphen G mit den Parametern t und k enthalten ist.

```
1  function KNEISDET(Graph G = (V, E), t, k)
2      select a node v of maximum degree d
3       N(v) = {v_1, ..., v_d} and deg(v_1) ≥ ... ≥ deg(v_d)
4      if deg(v) ≤ 3 then
5          apply branching rules for graphs of maximum degree three
```

```
6     else
7         for i = 1, ..., d − 1 do
8             if i ≥ deg(v_i) then
9                 return No
10            else
11                if i < d − 1 ∧ KneisDet(G − v, k − 1, t − d) then
12                    return Yes
13                else
14                    return KneisDet(G − v_{d−1}, v_d, k − 2, t − deg(v_{d−1}) − deg(v_d))
15                end if
16            end if
17        end for
18    end if
19 end function
```

Jeder Knoten v, der den Test in Zeile 11 erfolgreich besteht, gehört zu dem partiellen (t, k)-Vertex-Cover. Demzufolge kann während des Tests eine Lösung konstruiert werden. Als Laufzeit wird $\mathcal{O}^*(1.26^t)$ angegeben.

Kneis et al - randomisiert

Weiterhin wird in [KnLR08] noch ein randomisierter Algorithmus (KneisRand) mit einseitigem Fehler beschrieben, der eine Laufzeit von $\mathcal{O}^*(1.2993^t)$ aufweist. Grundsätzlich wählt er zufällig einen Knoten und testet, ob er zu einem (t, k)-Vertex-Cover gehören kann. Die Laufzeitschranke entsteht dabei durch Derandomisierung.

Kneis beschreibt unter anderem in [KnLR08] einen Algorithmus zum Lösen des *pVCPs* in Graphen mit maximalen Knotengrad von drei, der in dieser Arbeit nicht betrachtet werden soll, da die Einschränkungen zu stark sind. Er weist eine Laufzeit von $\mathcal{O}^*(1.26^t)$ auf.

Mestre

Auch für das *partielle Vertex-Cover-Problem* gibt es Approximationsalgorithmen. In [Mest05] findet sich ein Algorithmus mit einem Genauigkeitsfaktor von 2 und Laufzeit $\mathcal{O}(n \cdot \log n + m)$. Der Algorithmus von Mestre bedient sich des Konzepts der Primal-Dual-Approximation. Das heißt, er benutzt primale und duale Optimierungsprobleme zum Lösen des partiellen Vertex-Cover-Problems. Iterativ wird sich der Lösung genähert. In jeder Iteration findet dabei ein sogenannter „pruning-step“ und ein „dual-update-step“ statt. Im „pruning-step“ wird überprüft, ob ein neuer Knoten v eine zulässige Lösung des primalen Optimierungsproblems ist. Falls der Fall eintritt wird der Knoten in die Lösung aufgenommen und für weitere Betrachtungen gesperrt. Wenn die aktuelle Lösung bereits t Kanten überdeckt, terminiert der Algorithmus. Falls noch keine Lösung gefunden wurde, folgt der „dual-update-step“. Dort wird ein Knoten v bestimmt, der am Rand des dualen Optimierungsproblems liegt. Der Vorgang wird wiederholt, bis eine Lösung gefunden wurde.

3.1.5 Exaktes partielles Vertex-Cover

Beim exakten partiellen Vertex-Cover soll ein partielles Vertex-Cover mit Größe $\leq k$, welches genau t Kanten überdeckt, gefunden werden. Kneist beschreibt in [KnLR08] auch für diese Problemvariante einen Algorithmus.

```
1 function KneisExact(Graph G = (V, E), t, k)
2     choose random orientation for each edge {u, v}:
3         v → u (to u),
```

```
4           v ← u (to v) or
5           v − u (undirected)
6       compute all inner components CP
7       for comp ∈ CP do
8           if comp has ≤ k nodes and covers t edges  then
9               return comp is (t, k)-vertex-cover
10          end if
11      end for
12  end function
```

Der Algorithmus KneisExact bedient sich dem Konzept der zufälligen Orientierung von Kanten. Als innerer Knoten v wird ein Knoten bezeichnet, dessen inzidente Kanten entweder ungerichtete sind oder in Richtung v führen. Eine innere Komponente U ist eine minimale Menge von inneren Knoten, so dass $u \leftarrow v$ für jede Kante $\{u, v\} \in E$ mit $u \in U$ und $v \notin U$ gilt. Die Erfolgswahrscheinlichkeit ist nur abhängig von der zufälligen Wahl der Richtungen der t Kanten des Vertex-Covers, also $\geq 3^{-t}$. Durch Derandomisierung erhält man somit einen Algorithmus mit Laufzeit $\mathcal{O}^*(3^t)$ und konstanter Fehlerwahrscheinlichkeit.

3.2 Set-Cover

Nachdem unterschiedliche Algorithmen für die Vertex-Cover-Probleme betrachtet wurden, sollen nun Lösungen für das Set-Cover-Problem betrachtet werden. Zuerst wird das *minimale* und dann das *gewichtete Set-Cover-Problem* besprochen.

3.2.1 Minimales Set-Cover-Problem

Ziel des *minimalem Set-Cover-Problems* ist es, ein Set-Cover minimaler Größe zu finden. Als Eingabe bekommen die Algorithmen das Universum U und die Menge der Teilmengen S.

Johnson

Der Algorithmus von Johnson berechnet mit einer Genauigkeit von $\ln k$ ein *minimales Set-Cover* [John74], wobei $k = \max_{V \in S}\{|V|\} \leq m$ ist.

```
1  function JOHNSON(subsets S = {V_1, ..., V_n} , universe U )
2      Z = ∅
3      uncovered = U
4      for i = 1, ..., n do
5          set(i) = V_i
6      end for
7      while uncovered != ∅ do
8          choose j, with j ≤ n ∧ |set(j)| → max
9          Z = Z ∪ j
10         uncovered = uncovered − set(j)
11         for i = 1, ..., n do
12             set(i) = set(i) − set(j)
13         end for
14     end while
15     return Z
16 end function
```

In jedem Schritt des Algorithmus wird die größte Teilmenge V_i zum Set-Cover hinzugefügt. Anschließend werden überdeckte Elemente aus allen Teilmengen V_i entfernt. Der Vorgang wird solange wiederholt bis alle Elemente aus U überdeckt werden. Der von Johnson beschriebene Algorithmus läuft in polynomieller Zeit.

3.2.2 Gewichtetes Set-Cover Problem

Nun folgen Algorithmen für das gewichtete Set-Cover-Problem. Sie erhalten als Eingabe das Universum U, die Menge der Teilmengen S und die Gewichtsfunktion c.

Bar-Yehuda Even SC

Der in Abschnitt 3.1.3 eingeführte Algorithmus von Bar-Yehuda Even zum Lösen des gewichteten Vertex-Cover-Problems kann analog auch für Hypergraphen angwendet werden. Somit kann ohne Veränderung auch das gewichtete Set-Cover-Problem gelöst werden. Dadurch, dass Kanten eines Hypergraphen aus mehr als zwei Knoten bestehen können, ergibt sich eine veränderte Laufzeit von $\mathcal{O}(\sum_{i \in Z} |V_i|)$ und einen Genauigkeitsfaktor von f mit $f = \max_{u \in U}\{|\underbrace{\{i \mid u \in V_i\}}_{=F}|\}$

(F bestimmt die Anzahl an Teilmengen V_i, die u enthalten).

Young

Young beschreibt in [Youn] einen greedy Approximationsalgorithmus für das gewichtete Set-Cover Problem mit einer Approxmationsrate von $\ln n - \ln \ln n + \mathcal{O}(1)$. Der Algorithmus ist eine Erweiterung des Johnson Algorithmus und unterscheidet sich nur geringfügig.

```
1 function YOUNG(subsets S = {V_1, ..., V_n} ,universe U (with m elements), weights c)
2     Z = ∅
3     // Note f(X) = | ⋃_{i∈X} V_i|
4     while f(Z) != m do
5         choose j, with price c(V_j) / (f(Z∪j) − f(Z)) → min
6         Z = Z ∪ j
7     end while
8     return Z
9 end function
```

Durch Hinzufügen einer der Mengen V_i, die das Gewicht pro Anzahl der Elemente, die noch nicht überdeckt sind, minimiert, wird ein Set-Cover konstruiert. Der Algorithmus läuft in polynomieller Zeit mit einer Genauigkeit von $\ln m$.

Abschließend sei noch erwähnt, dass in [CyKW09] mehrere exponentielle Approximationsalgorithmen für das gewichtete Set-Cover-Problem beschrieben werden, welche nicht angesprochen wurden, weil die Laufzeiten zu schlecht sind.

4 Auswertung

Vertex-Cover-Problem

Algorithmus	Laufzeit	Art*	Genauigkeit**	Code***
MINIMALES VERTEX-COVER-PROBLEM				
Naiver VC	$\mathcal{O}(2^n m)$	BF		
Cormen	$\mathcal{O}(n+m)$	AP	2	
Monien-Speckenmeyer	$\mathcal{O}(m \cdot n)$	AP	$2 - \frac{1}{k+1}$	
k-VERTEX-COVER-PROBLEM				
Buss	$\mathcal{O}(kn + 2^k \cdot k^{2k+2})$	BF		
GEWICHTETES VERTEX-COVER-PROBLEM				
Bar-Yehuda Even	$\mathcal{O}(m)$	AP	2	
Halperin	polyomiell	AP	$2 - \epsilon$	
PARTIELLES VERTEX-COVER-PROBLEM				
KneisDet	$\mathcal{O}^*(1.396^t)$	D		
KneisRand	$\mathcal{O}^*(1.2993^t)$	R		
Mestre	$\mathcal{O}(n \cdot \log n + m)$	AP	2	
EXAKTES PARTIELLES VERTEX-COVER-PROBLEM				
KneisExact	$\mathcal{O}^*(3^t)$	R		

Set-Cover-Problem

Algorithmus	Laufzeit	Art*	Genauigkeit**	Code***
MINIMALES SET-COVER-PROBLEM				
Johnson	polyomiell	AP	$\log m$	
GEWICHTETES SET-COVER-PROBLEM				
Bar-Yehuda Even	$\mathcal{O}(\sum_{i \in Z} \lvert V_i \rvert)$	AP	$\max_{u \in U}\{\lvert\{i \mid u \in V_i\}\rvert\}$	
Young	polynomiell	AP	$\ln m$	

* Art: BF = Brute-Force, AP = Approximation, R = randomisiert, D = deterministisch

** Genauigkeit: Approximationsfaktor des Algorithmus

*** Code(komplexität): Algorithmus kann  = gut,  = mittel, = schlecht umgesetzt werden

Erkennbar beim Vergleich der Vertex-Cover und Set-Cover-Algorithmen ist, dass die Set-Cover-Algorithmen meist eine höhere Laufzeit oder einen höheren Approximationsfaktor aufweisen. Begründet wird der Fakt durch den bereits genannten Zusammenhang, dass das Set-Cover-Problem eine Verallgemeinerung des Vertex-Cover-Problems darstellt. Einzig der Algorithmus von Bar-Yehuda Even kann in beiden Problemfällen angewendet werden. Zum Planen von Angriffen eignen sich eher Algorithmen der Vertex-Cover-Varianten. Werden die gewichteten und ungewichteten Problemvarianten verglichen, ist auffällig, dass sie sich sehr ähnlich sind und daher beide gut geeignet sind. Auffällig sind außerdem die exponentiellen Laufzeiten bei einigen Algorithmen. Das begründet sich auf der *NP-Vollständigkeit* der

genannten Probleme und dadurch, dass die Algorithmen optimale Lösungen ermitteln. Allgemein kann festgestellt werden, dass die aufgeführten Approximationsalgorithmen grundsätzlich in polynomieller (maximal mit $\mathcal{O}(x^2)$) Zeit laufen. Die Approximationsalgorithmen des Vertex-Covers bieten maximal den Approximationsfaktor von 2, das heißt, die berechneten Lösungen sind maximal doppelt so groß, wie die optimalen. Es liegt in der Natur der Approximationsalgorithmen, dass sie eine bessere Laufzeit haben als exakte Algorithmen.

Auch wenn die partiellen Vertex-Cover-Problemvarianten durchaus interessant wirken, stellt sich bei der Betrachtung der Algorithmen heraus, dass sie auch bei Approximationen weniger gut umsetzbar sind.

Die „Codekomplexität“ der Set-Cover Algorithmen wurde mit gut bewertet, aber trotzdem überwiegt der Genauigkeitsnachteil.

Anmerkend sei noch erwähnt, dass die Algorithmen, deren „Codekomplexität“ mit schlecht bewertet wurde, meist Lösungen durch komplexe Optimierungsansätze oder mit komplizierten Verfahren ermitteln. Im Rahmen dieses Hauptseminars konnten nicht alle Grundlagen zu solchen Verfahren besprochen werden.

Abschließend soll noch genannt werden, dass in diesem Hauptseminar auf Algorithmen, die auf neuronale Netze (siehe [CTXL+04]), evolutionäre beziehungsweise genetische Algorithmen (siehe [OlHY07a], [OlHY09], [OlHY07b]) oder Auktionen ([KeSM10]) aufbauen, nicht besprochen wurden, weil die genannten Techniken meist keine genauen Aussagen über Laufzeit und Genauigkeit bieten. Weiterhin gibt es noch weitere nicht aufgeführte Problemvarianten, zum Beispiel das k-Set-Cover-Problem oder partielle Set-Cover-Varianten.

5 Fazit

Das Vertex-Cover- sowie Set-Cover-Problem sind klassische und häufig diskutierte Probleme in der Informatik. Bei Weitem konnte in dieser Arbeit nur ein kleiner Einstieg in die Problematik geboten werden. Dennoch wurden einige Algorithmen vorgestellt, die sich zum Planen von Angriffen auf Netzwerke eignen. Es ist abhängig von den gewählten Rahmenbedingungen, welcher Algorithmus am sinnvollsten ist. Besonders günstig scheinen gewichtete und ungewichtete Vertex-Cover-Algorithmen zu sein. Es ist auch denkbar, eine Kombination von mehreren Algorithmen zu verwenden. Beispielsweise könnten zwei Approximationsalgorithmen ausgeführt werden und man verwendet das kleinere Cover.

„The Answer to the Great Question...
Of Life, the Universe and Everything...
Is... Forty-two..“

Douglas Adams, The Hitchhiker's Guide to the Galaxy

Literatur

[AuKS09] P. Austrin, S. Khot und M. Safra. Inapproximability of Vertex Cover and Independent Set in Bounded Degree Graphs. In *Computational Complexity, 2009. CCC '09. 24th Annual IEEE Conference on*, july 2009, S. 74 –80.

[AvIm] David Avis und Tomokazu Imamura. A list heuristic for vertex cover. *Operations Research Letters* Band 35, S. 2007.

[BaFR98] R. Balasubramanian, Michael R. Fellows und Venkatesh Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters* 65(3), 1998, S. 163 – 168.

[BuGo91] J. Buss und J. Goldsmith. Nondeterminism within P. *STACS 91*, 1991, S. 348–359.

[BYEv81] R. Bar-Yehuda und S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms* 2(2), 1981, S. 198–203.

[BYEv85] R. Bar-Yehuda und S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics* Band 25, 1985, S. 27–46.

[CaFT98] Alberto Caprara, Matteo Fischetti und Paolo Toth. Algorithms for the Set Covering Problem. *Annals of Operations Research* Band 98, 1998, S. 2000.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein. *Introduction to algorithms*. The MIT press. 3. Auflage, 2009.

[CTXL+04] Xiaoming Chen, Zheng Tang, Xinshun Xu, Songsong Li, Guangpu Xia, Ziliang Zong und Jiahai Wang. An Hopfield network learning for minimum vertex cover problem. In *SICE 2004 Annual Conference*, Band 2, aug. 2004, S. 1150 –1155 vol. 2.

[CyKW09] M. Cygan, Ł. Kowalik und M. Wykurz. Exponential-time approximation of weighted set cover. *Information Processing Letters* 109(16), 2009, S. 957–961.

[DiSa05] Irit Dinur und Samuel Safra. On the hardness of approximating minimum vertex cover, 2005.

[DoDr] W. Domschke und A. Drexl. *Einführung in Operations Research*, Band 7.

[GHKO03] Sudipto Guha, Refael Hassin, Samir Khuller und Einat Or. Capacitated vertex covering. *JOURNAL OF ALGORITHMS* Band 48, 2003, S. 257–270.

[Gold10] O. Goldreich. *P, Np, and Np-completeness: The Basics of Computational Complexity*. Cambridge University Press. 2010.

[GuNW06] Jiong Guo, Rolf Niedermeier und Sebastian Wernicke. Parameterized Complexity of Vertex Cover Variants, 2006.

[Halp99] Eran Halperin. Improved Approximation Algorithms for the Vertex Cover Problem in Graphs and Hypergraphs, 1999.

[John74] D.S. Johnson. Approximation algorithms for combinatorial problems*. *Journal of computer and system sciences* 9(3), 1974, S. 256–278.

[KeSM10] D. Kempe, M. Salek und C. Moore. Frugal and Truthful Auctions for Vertex Covers, Flows and Cuts. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, oct. 2010, S. 745 –754.

[KnLR08] Joachim Kneis, Alexander Langer und Peter Rossmanith. Improved Upper Bounds for Partial Vertex Cover, 2008.

[Mest05] J. Mestre. A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, 2005, S. 610–610.

[MoAA11] S.V.R. Mohan, B.D. Acharya und M. Acharya. A Sufficiency Condition for Graphs to Admit Greedy Algorithm in Solving the Minimum Sum Vertex Cover Problem. In *Process Automation, Control and Computing (PACC), 2011 International Conference on*, july 2011, S. 1 –5.

[MoSp85] B. Monien und E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica* 22(1), 1985, S. 115–123.

[OlHY07a] P. S. Oliveto, J. He und X. Yao. Evolutionary Algorithms and the Vertex Cover Problem, 2007.

[OlHY07b] P.S. Oliveto, J. He und X. Yao. Evolutionary algorithms and the Vertex Cover problem. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, sept. 2007, S. 1870 –1877.

[OlHY09] P.S. Oliveto, Jun He und Xin Yao. Analysis of the -EA for Finding Approximate Solutions to Vertex Cover Problems. *Evolutionary Computation, IEEE Transactions on* 13(5), oct. 2009, S. 1006 –1029.

[PrKu92] I. Pramanick und J.G. Kuhl. A practical method for computing vertex covers for large graphs. In *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, Band 4, may 1992, S. 1859 –1862 vol.1.

[Tail09] P.J. Taillon. Parameterized VERTEX COVER in Graphs of Small Degree. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, Band 1, 31 2009-april 2 2009, S. 728 –732.

[WaKO91] T. Watanabe, S. Kajita und K. Onaga. Vertex covers and connected vertex covers in 3-connected graphs. In *Circuits and Systems, 1991., IEEE International Sympoisum on*, jun 1991, S. 1017 –1020 vol.2.

[XU] Ke XU. Benchmarks with Hidden Optimum Solutions for Graph Problems. `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`. Stand: 29.06.2012.

[Youn] Neal E. Young. Greedy Set-Cover Algorithms.

[Zeit] Zeit. "Unknowns" hacken Nasa und andere Websites. `http://www.zeit.de/digital/datenschutz/2012-05/unknown-nasa-hack`. Stand: 21.05.2012.

[ZWLX09] Yongguo Zeng, Dezheng Wang, Wei Liu und Ao Xiong. An approximation algorithm for weak vertex cover problem in IP network traffic measurement. In *Network Infrastructure and Digital Content, 2009. IC-NIDC 2009. IEEE International Conference on*, nov. 2009, S. 182 –186.