Approximationsalgorithmen Einführende Beispiele*

Jochen Ott

11. Oktober 2007

1 Einleitung

Viele kombinatorische Probleme, auf die man in der Praxis durchaus stößt (etwa die Frage nach dem optimalen Stundenplan in einer Schule oder das Problem des Handlungsreisenden) sind NP-hart und besitzen — nach allem was heute bekannt ist[1] — keine gut skalierbaren Lösungsalgorithmen, d. h. die Zeit für die Berechnung der optimalen Lösung wächst exponentiell in der Eingabelänge. Dennoch benötigt man Algorithmen, die in annehmbarer Laufzeit das Problem zumindest approximativ lösen. Es wird also Qualität der Lösung gegen Geschwindigkeit getauscht. Allerdings übersteigt der Rechenaufwand selbst vermeintlich kleinerer NP-harter Probleme oft jede verfügbare Kapazität, sodass diese erst durch Approximationsalgorithmen überhaupt zugänglich werden.

Zunächst müssen noch einige Begriffe definiert werden, bevor einige konkrete, einfache Optimierungsprobleme und Approximationsalgorithmen besprochen werden.

Definition Ein *Optimierungsproblem* ist charakterisiert durch:

1. Die Menge der Probleminstanzen („Eingabemenge“) D.
2. Zu jeder Instanz $I \in D$ eine Menge der zulässigen Lösungen $S(I)$. Die Länge der Lösungen soll polynomiell in $|I|$ beschränkt sein. Außerdem

*Dieser Text ist die Ergänzung eines Vortrags, der auf der Sommerakademie in Görlitz 2007 der Studienstiftung des deutschen Volkes gehalten wurde. Für die Vortragsfolien siehe [6].

[1]Das ist die Frage nach NP=P. Ich gehe im Folgenden davon aus (wie im Allgemeinen vermutet), dass P≠NP.

soll in Polynomzeit entschieden werden könnnen, ob für ein gegebenes s gilt $s \in I$.[2]

3. Eine Bewertungsfunktion $f : S(I) \to \mathbb{Q}^+$.

4. Die Angabe, ob f minimiert oder maximiert werden soll.

Definition Ein *Approximationsalgorithmus* für ein Optimierungsproblem ist ein polynomieller Algorithmus A, der zu jeder Eingabe $I \in D$ eine Lösung $\sigma_I^A \in S(I)$ berechnet.

Zu jedem Optimierungsproblem lässt sich ganz natürlich ein Entscheidungsproblem konstruieren: Für alle $I \in D$ und $b \in \mathbb{Q}^+$ soll entschieden werden, ob eine Lösung σ exisiert, die „besser“ ist als b, für die also gilt: $f(\sigma) \geq b$ für Maximierungsprobleme bzw. $f(\sigma) \leq b$ für Minimierungsprobleme.

Falls das Optimierungsproblem optimal gelöst ist (also immer eine optimale Lösung gefunden wird), so ist natürlich auch das daraus abgeleitete Entscheidungsproblem gelöst. Insofern kann man sagen, das Optimierungsproblem ist *mindestens so schwer* wie das Entscheidungsproblem.

Interessant ist vor Allem der Fall, in dem das Optimierungsproblem NP-schwer ist. Das ist in den folgenden Beispielen immer gegeben.

2 Graphfärbungen

2.1 Knotenfärbung

Problemstellung Bestimme zu einem ungerichteten Graph $G = (V, E)$ mit Knotenmenge V und Kantenmenge E eine zulässige Knotenfärbung, d. h. eine Abbildung $g : V \to \mathbb{N}^+$, sodass für jede Kante $e = \{v_1, v_2\}$ gilt: $g(v_1) \neq g(v_2)$. Die Anzahl der dabei verwendeten Farben, also $|g(V)|$ soll dabei minimal sein.

Die „Farben“ sind also die natürlichen Zahlen. Der folgende Algorithmus ist ein typischer greedy-Algorithmus.

Algorithmus für die Knotenfärbung.

1. Markiere alle Knoten als ungefärbt.

2. Wähle einen noch ungefärbten Knoten aus und weise ihm die kleinste zulässige Farbe zu, d. h. die kleinste Farbe, die keiner seiner Nachbarn hat.

[2] Das stellt sicher, dass die Entscheidungsversion ein kurzes Zertifikat besitzt, also in NP liegt.

3. Wiederhole 2. bis alle Knoten gefärbt sind.

Bezeichnet $\Delta(G)$ den maximalen Grad der Knoten im Graphen G, so braucht der Algorithmus maximal $\Delta(G)+1$ verschiedene Farben. Die optimale Lösung ist im Allgemeinen nicht bekannt. Für die nichttrivialen Graphen mit $E \neq \emptyset$ werden jedoch mindestens 2 Farben benötigt, so dass für die Abweichung vom Optimum κ für diesen Algorithmus gilt: $\kappa \leq \Delta(G) - 1$. Man kann Beispiele finden (siehe [6]), für die diese Schranke auch angenommen wird. Die Abschätzung für die Abweichung lässt sich also nicht ohne Weiteres verbessern.

Bemerkung Im Algorithmus soll man „einen noch ungefärbten Knoten“ wählen. Um ein Beispiel zu finden, das die Schranke annimmt, genügt es, einen Graphen *und* eine Reihenfolge für die Knotenwahl zu finden, für die die Schranke angenommen wird. Man darf nicht davon ausgehen, dass der Algorithmus automatisch die optimale Reihenfolge wählt, denn das entspräche einer nichtdeterministischen Turing-Maschine. Im Fall der Knotenfärbung gibt es sogar immer eine Reihenfolge für die Knotenwahl, die zu einer optimalen Färbung führt.

2.1.1 Knotenfärbung für planare Graphen

Planare Graphen haben viel mehr Struktur und man darf erwarten, dass man diese ausnutzen kann, um bessere Ergebnisse zu erzielen. Tatsächlich findet man:

1. Es gibt einen Polynomzeit-Algorithmus, der einen planaren Graphen mit maximal 6 Farben färbt.

2. Ein einfacher Greedy-Algorithmus kann polynomiell entscheiden, ob ein gegebener Graph mit zwei Farben färbbar ist und ggf. eine Färbung finden.

Der erste Algorithmus nutzt dabei lediglich aus, dass jeder planare Graph einen Knoten mit Grad 5 oder kleiner besitzt, das zweite Ergebnis gilt natürlich für beliebige Graphen.

Durch die Kombination der beiden Algorithmen (also: versuche erst mit dem zweiten eine Färbung mit zwei Farben und wenn dies fehlschlägt verwende den ersten) erhält man einen Algorithmus, mit maximaler Abweichung 3.

Der Vierfarben-Satz kann verwendet werden, um einen polynomiellen Algorithmus zu konstruieren, der effizient jeden planaren Graphen mit maximal vier Farben färbt [2]. Damit erhält man sogar einen Algorithmus mit maximaler Abweichung 1. Die Entscheidung, ob ein planarer Graph mit drei Farben färbbar

ist, ist NP-vollständig [5]; es ist also keine Verbesserung des Algorithmus zu erwarten.

2.2 Kantenfärbung

Problemstellung Bestimme zu einem ungerichteten Graph $G = (V, E)$ eine zulässige Kantenfärbung, d. h. eine Abbildung $g : E \to \mathbb{N}^+$, sodass keine zwei Kanten mit gemeinsamen Knoten die gleiche Farbe besitzen. Die Anzahl der dabei verwendeten Farben, also $|g(E)|$ soll dabei minimal sein.

Offenbar werden mindestens $\Delta(G)$ Farben benötigt. Außerdem gibt es Graphen die $\Delta(G) + 1$ Farben benötigt werden (betrachte K_n, den vollständigen Graph mit n Knoten, mit n ungerade). Der Satz von Vizing (1964) sagt jedoch aus, dass mehr als $\Delta(G) + 1$ Farben nie benötigt werden. Es existiert auch ein polynomieller Algorithmus, der eine solche Lösung explizit konstruiert, siehe [6].

Bessere Ergebnisse kann man nicht erwarten, da das Entscheidungsproblem, ob ein Graph mit $\Delta(G)$ Farben gefärbt werden kann, NP-vollständig ist [1].

3 Ein Ergebnis zum Rucksackproblem

Bisher wurden nur Probleme und Algorithmen betrachtet, die eine maximale *absolute* Abweichung garantierten. Oft besitzen Probleme jedoch eine gewisse Skalierungseigenschaft, die dazu führen, dass es keine Algorithmen mit garantierer maximaler Abweichung geben *kann*. Das wird nun am Rucksackproblem demonstriert.

Problemstellung Gegeben sei eine Menge von Waren W sowie eine Wert-Funktion $p : W \to \mathbb{Q}^+$, eine Volumen-Funktion $v : W \to \mathbb{Q}^+$ und das Rucksackvolumen V. Gesucht ist eine Untermenge der Waren W', sodass $v(W') \leq V$ mit $p(W')$ maximal[3].

Satz Falls P≠NP, gibt keinen polynomiellen Algorithmus für das Rucksackproblem, der eine feste garantierte maximale Abweichung κ besitzt.

Beweis-Skizze Angenommen, es gibt einen solchen Algorithmus. Für eine beliebige Instanz des Rucksackproblems konstruiere nun ein skaliertes Problem, in dem alle Warenwerte so skaliert werden, dass die minimale Differenz zweier unterschiedlicher Warenwerte größer ist als κ. Wende den Algorithmus auf das

[3] p sei dabei auf Mengen folgendermaßen definiert: $p(W') := \sum_{w \in W'} p(w)$. v entsprechend.

neue Problem an. Da er eine Abweichung $\leq \kappa$ garantiert, muss die Lösung optimal sein(!). Dies liefert sofort auch eine optimale Lösung für das ursprüngliche Problem. Damit ist ein optimaler, polynomieller Algorithmus für das Rucksackproblem konstruiert. Insbesondere ist das Entscheidungsproblem gelöst und es folgt P=NP im Widerspruch zur Voraussetzung. ■

4 Knotenüberdeckung

Problemstellung Gegeben sei ein ungerichteter Graph $G = (V, E)$ und eine Kostenfunktion $c : V \to \mathbb{Q}^+$. Finde eine Knotenuntermenge $C \subseteq V$, sodass zu jeder Kante $\{u, v\} \in E$ gilt: $u \in C$ oder $v \in C$. Jede Kante soll also mit mindestens einem Knoten in C „vertreten“ sein.

Hier wird nur das Problem mit Einheitskosten, d. h. $c(u) = 1$ für alle Knoten u, betrachtet. Vor dem Algorithmus zunächst noch eine

Definition Eine *Paarung* eines Graphen $G = (V, E)$ ist eine Untermenge M der Kanten, sodass keine zwei Kanten aus M einen gemeinsamen Knoten besitzen. Eine Paarung heißt *nicht erweiterbar*, falls es *keine* Kante $e = \{u, v\} \in E$ gibt, sodass $M \cup \{e\}$ eine Paarung ist.

Algorithmus für die Knotenüberdeckung:

1. Finde durch sukzessives Hinzufügen von Kanten (die nach dem Hinzufügen einschließlich der Knoten und an diesen hängenden anderen Kanten gelöscht werden) eine nicht erweiterbare Paarung M.

2. Gib die Knoten in M aus.

Zunächst muss man sich klarmachen, dass der Algorithmus tatsächlich eine Knotenüberdeckung ausgibt, also tatsächlich zu jeder Kante einer der Knoten ausgegeben wurde. Das ist jedoch der Fall, da es sonst eine Kante gäbe, von der keiner der Knoten entfernt wurde. Das heißt jedoch, dass diese Kante der Paarung hätte hinzugefügt werden können, die Paarung im ersten Schritt wäre (entgegen der Konstruktion) erweiterbar.

Bevor das Analyse-Ergebnis formuliert wird, noch eine nützliche

Definition Ein Approximationsalgorithmus A hat bei der Eingabe $I \in D$ die *relative Güte*

$$\rho := \max \left\{ \frac{f(A(I))}{\mathrm{OPT}}, \frac{\mathrm{OPT}}{f(A(I))} \right\}$$

wobei OPT die Bewertung der optimalen Lösung ist. Die Definition stellt sicher, dass stets $\rho \geq 1$.

Satz Der angegebene Algorithmus für die Knotenüberdeckung hat im schlechtesten Fall eine relative Güte 2.

Beweis Jede Knotenüberdeckung muss nach Definition von jeder Kante mindestens einen Knoten enthalten, insbesondere für die Kanten in der Paarung M. Der Algorithmus gibt für jede Kante zwei Knoten aus und liegt damit maximal Faktor 2 neben der optimalen Lösung. Für $K_{n,n}$ wählt der Algorithmus alle Knoten aus, das Optimum wäre jedoch, nur die Hälfte zu wählen. Der Faktor 2 wird also angenommen. ■

4.1 Orakel-Algorithmus für die Knotenüberdeckung

Auf Seite 2 wurde bereits angesprochen, dass das Optimierungsproblem mindestens so schwer ist wie das Entscheidungsproblem in dem Sinn, dass eine Lösung des Optimierungsproblems auch das Entscheidungsproblem löst. In diesem Abschnitt wird gezeigt, dass im Fall der Knotenüberdeckung auch das Umgekehrte gilt: ist das Entscheidungsproblem gelöst, so kann man damit einen Algorithmus angeben, der explizit eine optimale Lösung konstruiert.

Das Entscheidungsproblem sei gelöst, d. h. es steht ein Algorithmus zur Verfügung, der für einen beliebigen Graphen G und eine ganze Zahl b ausgibt, ob eine Überdeckung existiert, die mit $\leq b$ Knoten auskommt. Da davon ausgegangen wird, dass ein NP-vollständiges Problem schnell gelöst wurde, spricht man auch von einem „Orakel“. Damit ist es nun möglich, einen konstruktiven Algorithmus anzugeben:

Algorithmus für Knotenüberdeckung mithilfe eines Orakels:

1. $C \leftarrow \emptyset$
2. Finde durch Binärsuche die Anzahl der Knoten in einer optimalen Lösung, also $\mathrm{OPT}(G)$ (das geht in polynomieller Zeit).
3. Entferne einen beliebigen Knoten v vom Graphen. Dies führt auf einen reduzierten Graphen G'. Bestimme wieder durch Binärsuche $\mathrm{OPT}(G')$. Falls $\mathrm{OPT}(G') = \mathrm{OPT}(G) - 1$, füge v zu C hinzu.
4. Fahre rekursiv mit G' fort.
5. Gib C aus.

Dass der Algorithmus tatsächlich das Gewünschte leistet, kann man sich leicht klarmachen.

Eine solche Reduktion ist sehr oft möglich. Insofern steckt der „schwierige Kern“ des Optimierungsproblems oft bereits in der (i. A. einfacheren) dazugehörigen Entscheidungsversion.

5 Maximaler Schnitt in einem Graphen

Problemstellung Bestimme zu einem ungerichteten Graph $G = (V, E)$ einen *Schnitt*, d. h. eine Zerlegung der Knotenmenge in zwei nichtleere, disjunkte Teilmengen: $V = A \dot{\cup} B$, sodass die Anzahl der zwischen A und B verlaufenden Kanten maximal ist.

Einen *minimalen* Schnitt findet man garantiert in polynomieller Zeit (mit dem Edmonds-Karp-Algorithmus, einer Variante des Ford-Fulkerson-Algorithmus').

Zunächst wird wieder ein greedy-Algorithmus betrachtet.

Algorithmus (greedy) für den maximalen Graphenschnitt.

1. Wähle zwei beliebige Startknoten v_1 und v_2 und setze $A \leftarrow \{v_1\}$, $B \leftarrow \{v_2\}$.
2. Für einen noch nicht gewählten Knoten $v \in V \setminus (A \cup B)$: Falls $\deg(v, A) > \deg(v, B)$, setze $B \leftarrow B \cup \{v\}$ sonst $A \leftarrow A \cup \{v\}$.
3. Wiederhole Schritt 2 bis alle Knoten zugeteilt sind.

Satz Der greedy-Algorithmus für den maximalen Schnitt hat eine relative Güte von 2.

Beweis Bezeichne $e(A)$ und $e(B)$ die Anzahl der Kanten innerhalb der Knotenmenge A bzw. B und $\deg(A, B)$ die Anzahl der Kanten zwischen den beiden Mengen. Nach Schritt 1 gilt $e(A) = e(B) = 0$ und $0 \leq \deg(A, B) \leq 1$, also insbesondere $e(A) + e(B) \leq \deg(A, B)$. Nach jeder Ausführung von Schritt 2 gilt nach Konstruktion immer noch $e(A) + e(B) \leq \deg(A, B)$.

Die Anzahl der Kanten zwischen A und B der optimale Lösung ist nach oben durch die Gesamtzahl der Kanten beschränkt: $\mathrm{OPT} \leq |E|$. Am Ende des Algorithmus wird jede Kante genau einmal von $e(A)$, $e(B)$ und $\deg(A, B)$ gezählt, es gilt also $e(A) + e(B) + \deg(A, B) = |E|$. Damit gilt $\deg(A, B) \geq \frac{1}{2}\mathrm{OPT}$. ∎

Ein weiterer Algorithmus mit Güte 2 verwendet das Prinzip der lokalen Suche:

Algorithmus (lokale Suche) für den maximalen Graphenschnitt.

1. Starte mit einem beliebigen Schnitt $V = A \dot{\cup} B$.
2. Suche einen Knoten v, dessen Umsortierung in die andere Menge den Schnitt verbessert (und tu dies).
3. Wiederhole Schritt 2, bis es keinen Knoten mehr gibt, auf den die Bedingung zutrifft.

Beweis Zunächst muss man sich überlegen, dass der Algorithmus terminiert. Das ist erfüllt, da der Schnitt in jeder Ausführung von Schritt 2 um mindestens 1 ansteigt und der Schnitt nach oben durch $|E|$ beschränkt ist. Der Algorithmus hat Güte 2, da nach der Ausführung jeder Knoten mindestens so viele Kanten zwischen den Mengen A, B wie innerhalb seiner eigenen Menge hat. Scharfes Hinsehen führt dann sofort auf die Behauptung. ■

Literatur

[1] Ian Holyer. *The NP-Completeness of Edge-Coloring.* SIAM Journal on Computing Vol. 10, No. 4, p. 718–720, 1981.

[2] Neil Robertson, Daniel P. Sanders, Paul Seymor, Robin Thomas. *Efficiently four-coloring planar graphs.* Proceedings of the 28th annual ACM Symposium on Theory of Computing (STOC), p. 571–575, 1996.

[3] Vijay V. Vazirani. *Approximation Algorithms.* Springer 2001, Kapitel 1 und 2 sowie Anhang A.

[4] Rolf Wanka. *Approximationsalgorithmen – Eine Einführung.* Teubner 2006, Kapitel 1 und 2.

[5] Larry Stockmeyer. *Planar 3-colorybility is polynomial complete.* ACM SIGACT News, Vol. 5, No. 3, p. 19–25, 1973.

[6] Jochen Ott. *Approximationsalgorithmen – Einführende Beispiele.* (Vortragsfolien) 2007. Verfügbar unter `http://jochenott.de/appr-v.pdf`.