Algorithmen für schwierige Probleme

October 28, 2024

Contents

1 3-SAT und Vertex Cover

1.1 Vertex Cover

Definition 1.1. Ein Vertex Cover für einen Graphen G ist eine Teilmenge $C \subseteq V$ der Knoten, sodass $\forall e = (u, v) \in E$ gilt $u \in C$ oder $v \in C$.

Definition 1.2. Für zwei Probleme A, B schreiben wir $A \preceq B$, wenn man A effizient auf B transformieren kann.

Satz 1.3. *Der Greedy Algorithmus, der immer den Knoten höchsten Grades wählt, kann beliebig schlechte Ergebnisse liefern.*

Satz 1.4. *Der folgende Algorithmus liefert eine 2-Faktor-Approximation an das optimale Vertex Cover.*

1. *setze* $C = \varnothing$
2. *WHILE* $E \neq \varnothing$
3. *wähle eine Kante* (u, v)
4. *setze* $V = V \setminus \{u, v\}$, $E = e = (w, x) \in E \mid w, x \notin \{u, v\}$ *und* $C = C \cup \{u, v\}$.

Satz 1.5. *Der folgenden Ansatz benutzt außerdem einen Parameter k als Eingabe und entscheidet, ob es ein Vertex Cover der Größe $\leq k$ gibt.*

1. *FUNCTION VCover*(G, C, l)
2. *if* $E = \varnothing$ *return True*
3. *if* $l = k$ *return False*
4. *Wähle eine Kante* (u, v) *in* G
5. *if VCover*$(G \setminus \{u\}, C \cup \{u\}, l + 1)$
6. *return True*
7. *else return VCover*$(G \setminus \{v\}, C \cup \{v\}, l + 1)$

Die Laufzeit ist offensichtlich $2^k\mathcal{O}(n)$.

1.2 SAT

Eine Formel F ist in KNF gegeben, also als Konjunktion von Disjunktionen von Literalen. Eine Belegung ist eine Funktion, die jeder Variable einen Wert zuweist. Eine partielle Belegung weist nur einer Teilmenge aller Variablen Werte zu und lässt die restlichen undefiniert.

Lemma 1.6. *Reduktion von 3-Färbbarkeit auf SAT.*
Für einen Graphen G wollen wir eine Formel F konstruieren. Zunächst definieren wir die Menge der Variablen von F als

$$VAR = \{x_v^1, x_v^2, x_v^3 : \ v \in V\}$$

für drei verschiedene Farben. In einer Lösung muss jeder Knoten eine Farbe bekommen, d.h wir brauchen die $|V|$ Klauseln

$$(x_v^1 \vee x_v^2 \vee x_v^3)$$

Außerdem dürfen zwei benachbarte Knoten nicht dieselbe Farbe bekommen, das heißt, es ergeben sich die $3\,|E|$ Klauseln

$$(\neg x_v^1 \vee \neg x_w^1) \wedge (\neg x_v^2 \vee \neg x_w^2) \wedge (\neg x_v^3 \vee \neg x_w^3)$$

Satz 1.7. *Der folgende Backtracking Algorithmus bietet einen guten Ansatz, um eine Formel F auf Erfüllbarkeit zu überprüfen.*

1. *FUNCTION search(F, α partielle Belegung)*
2. *if α belegt alle Variablen: return $\alpha(F)$*
3. *if α belegt eine Klausel mit 0: return False*
4. *if search($F, \alpha 0$) = True: return True*
5. *else: return search($F, \alpha 1$)*

Dieser Algorithmus kann sogar noch weiter verbessert werden, indem zuerst nach Klauseln mit nur einem Literal gesucht wird und dann diese belegt werden. Wenn diese nicht existieren, wird nach Klauseln mit zwei Literalen gesucht und diese werden belegt. Erst dann wird eine beliebige Klausel (bzw. ein beliebiges Literal) gewählt. Die Laufzeit ist beschränkt durch $\mathcal{O}(7^{\frac{n}{3}})$.

Das Erfüllbarkeitsproblem, in dem jede Klausel aus maximal zwei Variablen besteht, wird 2-SAT genannt. Dieses Problem kann in polynomieller Zeit gelöst werden.

Satz 1.8. *Im folgenden wird ein probabilistischer Algorithmus für 2-SAT eingeführt.*

1. $\alpha := (0, 0, ..., 0)$
2. *for* $i = 0$ *to* $f(F)$
3. *if* $\alpha(F) = 1$*: return True*
4. *wähle eine zufällige Klausel* C *mit* $\alpha(C) = 0$
5. *wähle ein zufälliges Literal* x *aus* C
6. *setze* $\alpha(x) = \overline{\alpha(x)}$

Die Funktion $f : F \mapsto n \in \mathbb{N}$ *wird später definiert.*
Ist F *unerfüllbar, so funktioniert der Algorithmus korrekt. Ist* F *erfüllbar, so findet der Algorithmus eine erfüllende Belegung mit Wahrscheinlichkeit* $p \geq \frac{1}{2}$.
Ist α^* *eine erfüllende Belegung und* α *eine beliebige andere Belegung, mit Hamming-Abstand* i *zu* α^**, so definieren wir uns* $T(i)$ *als die erwartete Anzahl an Bit-Flips, die nötig sind, um* α *in eine erfüllende Belegung zu transformieren. Man kann sehen, dass* $T(n) = n^2$*. Die erwartete Anzahl an Bitflips ist also* $\mathcal{O}(n^2)$*. Wählt man nun* $f(F) = 2n^2$*, so ist die Erfolgswahrscheinlichkeit von* $\frac{1}{2}$ *garantiert.*

1.3 Min-Cut

Dieses Problem ist in P. Eine effiziente Lösung ist durch den Ford-Fulkerson Algorithmus mithilfe dem Max-Flow/Min-Cut Lemmas möglich. Im Folgenden wird ein effizienter probabilistischer Algorithmus besprochen.

1. WHILE $|V| > 2$ DO
2. wähle $(u, v) \in_R E$
3. kontrahiere (u, v)
4. ENDWHILE

Satz 1.9. *Der Algorithmus hat Laufzeit* $\mathcal{O}(n^2(\log n)^2)$*. Der Algorithmus findet immer einen Schnitt. Der Algorithmus benötigt für jede Instanz* $n - 1$ *Schritte. Für einen gegebenen minimalen Schnitt* C *ist die Wahrscheinlichkeit, dass der Algorithmus* C *findet ist* $\geq \frac{2}{n(n-1)}$*. Es genügen also* $\mathcal{O}(n^2)$ *Wiederholungen für eine konstant kleine Fehlerwahrscheinlichkeit.*

In jedem Schritt des Algorithmus wird gehofft, dass keine Kante e aus dem minimal cut C gezogen wird. Im letzten Schritt beträgt die Fehlerwahrscheinlichkeit dafür aber $\frac{2}{3}$. Um dem entgegen zu wirken, ist die Idee, den Algorithmus ein wenig umzubauen. Nach etwa $n\left(\frac{\sqrt{2}-1}{\sqrt{2}}\right)$ Schritten wird der bisher erzeugte Teilgraph H von G bestimmt. Für diesen Graphen wird der Algorithmus zwei separate Male ausgeführt und der kleinere der beiden Cuts wird gewählt. Eine Kontraktion ist in $\mathcal{O}(n)$ mithilfe der Adjazenzmatrix möglich, also wird die Laufzeit

$$T(n) = \underbrace{\left(1 - \frac{1}{\sqrt{2}}\right) n}_{\text{Anzahl Runden}} \cdot \underbrace{cn}_{\text{Laufzeit Kontraktion}} + \underbrace{2T\left(\frac{n}{\sqrt{2}}\right)}_{\text{Laufzeit Rekursion}}$$

Mittels Master-Theorem lässt sich die Laufzeit bestimmen als $T(n) = \mathcal{O}(n^2 \log n)$.
Die Wahrscheinlichkeit, dass C nun die ersten $n\left(1 - \frac{1}{\sqrt{2}}\right)$ Schritte überlebt, ist nun $\approx \frac{1}{2}$. Mittels induktivem Einsetzen kann überprüft werden, dass die Erfolgswahrscheinlichkeit $p(n) \geq \frac{1}{\log n}$ ist. Es genügen daher $\log n$ Wiederholungen für eine konstante Fehlerwahrscheinlichkeit. Die gesamte Laufzeit ist daher $\mathcal{O}(n^2(\log n)^2)$.