Approximationsalgorithmen

Einführende Beispiele

Jochen Ott

3. September 2007

Übersicht

Outline

Approximationsalgorithmen

Problemstellung

Für viele wichtige (und interessante) Probleme existieren vermutlich keine effizienten Lösungsalgorithmen. Es kann aber durchaus gute *Approximationen* geben.

Bemerkungen

- Es geht um $\mathcal{NP}$-vollständige Probleme.
- Wir interessieren uns für Algorithmen in $\mathcal{P}$.
- Ich gehe im Folgenden davon aus, dass $\mathcal{P} \neq \mathcal{NP}$.
- Wie gut eine Approximation ist, wird mit einer (jeweils zu spezifizierenden) Bewertungsfunktion gemessen.

Approximationsalgorithmen

Problemstellung

Für viele wichtige (und interessante) Probleme existieren vermutlich keine effizienten Lösungsalgorithmen. Es kann aber durchaus gute *Approximationen* geben.

Bemerkungen

- Es geht um $\mathcal{NP}$-vollständige Probleme.
- Wir interessieren uns für Algorithmen in $\mathcal{P}$.
- Ich gehe im Folgenden davon aus, dass $\mathcal{P} \neq \mathcal{NP}$.
- Wie gut eine Approximation ist, wird mit einer (jeweils zu spezifizierenden) Bewertungsfunktion gemessen.

Approximationsalgorithmen

Problemstellung

Für viele wichtige (und interessante) Probleme existieren vermutlich keine effizienten Lösungsalgorithmen. Es kann aber durchaus gute *Approximationen* geben.

Bemerkungen

- Es geht um $\mathcal{NP}$-vollständige Probleme.
- Wir interessieren uns für Algorithmen in $\mathcal{P}$.
- Ich gehe im Folgenden davon aus, dass $\mathcal{P} \neq \mathcal{NP}$.
- Wie gut eine Approximation ist, wird mit einer (jeweils zu spezifizierenden) Bewertungsfunktion gemessen.

Approximationsalgorithmen

Problemstellung

Für viele wichtige (und interessante) Probleme existieren vermutlich keine effizienten Lösungsalgorithmen. Es kann aber durchaus gute *Approximationen* geben.

Bemerkungen

- Es geht um $\mathcal{NP}$-vollständige Probleme.
- Wir interessieren uns für Algorithmen in $\mathcal{P}$.
- Ich gehe im Folgenden davon aus, dass $\mathcal{P} \neq \mathcal{NP}$.
- Wie gut eine Approximation ist, wird mit einer (jeweils zu spezifizierenden) Bewertungsfunktion gemessen.

Approximationsalgorithmen

Problemstellung

Für viele wichtige (und interessante) Probleme existieren vermutlich keine effizienten Lösungsalgorithmen. Es kann aber durchaus gute *Approximationen* geben.

Bemerkungen

- Es geht um $\mathcal{NP}$-vollständige Probleme.
- Wir interessieren uns für Algorithmen in $\mathcal{P}$.
- Ich gehe im Folgenden davon aus, dass $\mathcal{P} \neq \mathcal{NP}$.
- Wie gut eine Approximation ist, wird mit einer (jeweils zu spezifizierenden) Bewertungsfunktion gemessen.

Optimierungsproblem – formale Definition

Definition *Optimierungsproblem*

- die Probleminstanzen D
- zu $I \in D$ je eine Menge der zulässigen Lösungen $S(I)$
- eine Bewertungsfunktion $f : S(I) \to \mathbb{Q}^+$
- die Angabe, ob f maximiert oder minimiert werden soll.

Definition *Approximationsalgorithmus*

Ein Algorithmus in $\mathcal{P}$, der zu jeder Eingabe $I \in D$ eine Lösung $\sigma_I^A \in S(I)$ berechnet.

Optimierungsproblem – formale Definition

Definition *Optimierungsproblem*

- die Probleminstanzen D
- zu $I \in D$ je eine Menge der zulässigen Lösungen $S(I)$
- eine Bewertungsfunktion $f : S(I) \to \mathbb{Q}^+$
- die Angabe, ob f maximiert oder minimiert werden soll.

Definition *Approximationsalgorithmus*

Ein Algorithmus in $\mathcal{P}$, der zu jeder Eingabe $I \in D$ eine Lösung $\sigma_I^A \in S(I)$ berechnet.

Entscheidungsversion eines Optimierungsproblems

Entscheidungsversion

Aus jedem Optimierungsproblem lässt sich ein Entscheidungsproblem ableiten:
Gegeben sei $I \in D$, $B \in \mathbb{Q}^+$. Existiert eine Lösung zu I, die besser ist als B?

Bemerkungen

- Falls sich das Optimierungsproblem in $\mathcal{P}$ perfekt lösen lässt, ist auch die Entscheidungsversion in $\mathcal{P}$.
- Insofern überträgt sich die „Schwierigkeit" des Entscheidungsproblems auf das Optimierungsproblem.

Definition

Ein Optimierungsproblem heißt $\mathcal{NP}$-hart genau dann, falls seine Entscheidungsversion $\mathcal{NP}$-hart ist.

Entscheidungsversion eines Optimierungsproblems

Entscheidungsversion

Aus jedem Optimierungsproblem lässt sich ein Entscheidungsproblem ableiten:
Gegeben sei $I \in D$, $B \in \mathbb{Q}^+$. Existiert eine Lösung zu I, die besser ist als B?

Bemerkungen

- Falls sich das Optimierungsproblem in $\mathcal{P}$ perfekt lösen lässt, ist auch die Entscheidungsversion in $\mathcal{P}$.
- Insofern überträgt sich die „Schwierigkeit“ des Entscheidungsproblems auf das Optimierungsproblem.

Definition

Ein Optimierungsproblem heißt $\mathcal{NP}$-hart genau dann, falls seine Entscheidungsversion $\mathcal{NP}$-hart ist.

Entscheidungsversion eines Optimierungsproblems

Entscheidungsversion

Aus jedem Optimierungsproblem lässt sich ein Entscheidungsproblem ableiten:
Gegeben sei $I \in D$, $B \in \mathbb{Q}^+$. Existiert eine Lösung zu I, die besser ist als B?

Bemerkungen

- Falls sich das Optimierungsproblem in $\mathcal{P}$ perfekt lösen lässt, ist auch die Entscheidungsversion in $\mathcal{P}$.
- Insofern überträgt sich die „Schwierigkeit" des Entscheidungsproblems auf das Optimierungsproblem.

Definition

Ein Optimierungsproblem heißt $\mathcal{NP}$-hart genau dann, falls seine Entscheidungsversion $\mathcal{NP}$-hart ist.

Entscheidungsversion eines Optimierungsproblems

Entscheidungsversion

Aus jedem Optimierungsproblem lässt sich ein Entscheidungsproblem ableiten:
Gegeben sei $I \in D$, $B \in \mathbb{Q}^+$. Existiert eine Lösung zu I, die besser ist als B?

Bemerkungen

- Falls sich das Optimierungsproblem in $\mathcal{P}$ perfekt lösen lässt, ist auch die Entscheidungsversion in $\mathcal{P}$.
- Insofern überträgt sich die „Schwierigkeit“ des Entscheidungsproblems auf das Optimierungsproblem.

Definition

Ein Optimierungsproblem heißt $\mathcal{NP}$-hart genau dann, falls seine Entscheidungsversion $\mathcal{NP}$-hart ist.

Outline

Knotenfärbung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Knotenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- *Graph* (V, E) mit V: Knotenmenge, $E \neq \emptyset$: Kantenmenge (eine Untermenge der Menge der zweielementigen Knotenuntermengen)
- *zulässige Knotenfärbung*: Abbildung $g : V \to \mathbb{N}$ mit $g(u) \neq g(v)$, falls $\{u, v\} \in E$.
- D: die ungerichteten Graphen; $S(I)$: Färbungen g; f: die Anzahl der Farben $f(g) = |g(V)|$; minimiere.

Knotenfärbung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Knotenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- *Graph* (V, E) mit V: Knotenmenge, $E \neq \emptyset$: Kantenmenge (eine Untermenge der Menge der zweielementigen Knotenuntermengen)
- *zulässige Knotenfärbung*: Abbildung $g : V \to \mathbb{N}$ mit $g(u) \neq g(v)$, falls $\{u, v\} \in E$.
- D: die ungerichteten Graphen; $S(I)$: Färbungen g; f: die Anzahl der Farben $f(g) = |g(V)|$; minimiere.

Knotenfärbung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Knotenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- *Graph* (V, E) mit V: Knotenmenge, $E \neq \emptyset$: Kantenmenge (eine Untermenge der Menge der zweielementigen Knotenuntermengen)
- *zulässige Knotenfärbung*: Abbildung $g : V \to \mathbb{N}$ mit $g(u) \neq g(v)$, falls $\{u, v\} \in E$.
- D: die ungerichteten Graphen; $S(I)$: Färbungen g; f: die Anzahl der Farben $f(g) = |g(V)|$; minimiere.

Knotenfärbung

Optimierungsproblem

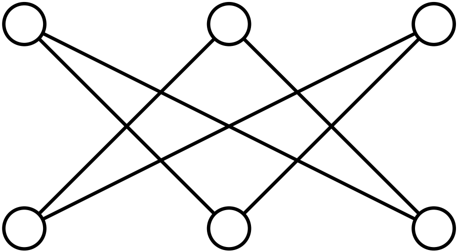
Bestimme zu einem ungerichteten Graph eine zulässige Knotenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- *Graph* (V, E) mit V: Knotenmenge, $E \neq \emptyset$: Kantenmenge (eine Untermenge der Menge der zweielementigen Knotenuntermengen)
- *zulässige Knotenfärbung*: Abbildung $g : V \to \mathbb{N}$ mit $g(u) \neq g(v)$, falls $\{u, v\} \in E$.
- D: die ungerichteten Graphen; $S(I)$: Färbungen g; f: die Anzahl der Farben $f(g) = |g(V)|$; minimiere.

Knotenfärbung: Beispiel

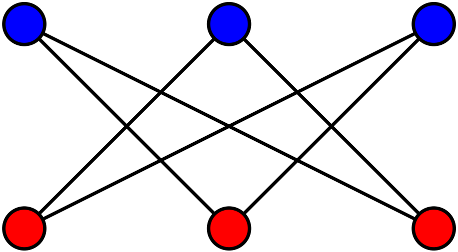
Graph G mit maximalem Grad $\Delta(G) = 2$:



Eine mögliche Färbung g mit $f(g) = 2$.

Knotenfärbung: Beispiel

Graph G mit maximalem Grad $\Delta(G) = 2$:



Eine mögliche Färbung g mit $f(g) = 2$.

Knotenfärbung: Algorithmus

Algorithmus

1. Markiere alle Knoten als ungefärbt (bzw. Farbe = ∞)
2. Wähle einen ungefärbten Knoten aus und gib ihm die kleinste zulässige Farbe (also eine Farbe, die keiner seiner Nachbarn hat), bis alle Knoten gefärbt sind.

Analyse

- Braucht höchstens $\Delta(G) + 1$ Farben.
- Optimaler Wert von f ist unbekannt, jedoch mindestens 2.
- Somit Abschätzung für maximale Abweichung κ: $\kappa \leq \Delta(G) - 1$.

Knotenfärbung: Algorithmus

Algorithmus

1. Markiere alle Knoten als ungefärbt (bzw. Farbe = ∞)
2. Wähle einen ungefärbten Knoten aus und gib ihm die kleinste zulässige Farbe (also eine Farbe, die keiner seiner Nachbarn hat), bis alle Knoten gefärbt sind.

Analyse

- Braucht höchstens $\Delta(G) + 1$ Farben.
- Optimaler Wert von f ist unbekannt, jedoch mindestens 2.
- Somit Abschätzung für maximale Abweichung κ: $\kappa \leq \Delta(G) - 1$.

Knotenfärbung: Algorithmus

Algorithmus

1. Markiere alle Knoten als ungefärbt (bzw. Farbe = ∞)
2. Wähle einen ungefärbten Knoten aus und gib ihm die kleinste zulässige Farbe (also eine Farbe, die keiner seiner Nachbarn hat), bis alle Knoten gefärbt sind.

Analyse

- Braucht höchstens $\Delta(G) + 1$ Farben.
- Optimaler Wert von f ist unbekannt, jedoch mindestens 2.
- Somit Abschätzung für maximale Abweichung κ: $\kappa \leq \Delta(G) - 1$.

Knotenfärbung: Algorithmus

Algorithmus

1. Markiere alle Knoten als ungefärbt (bzw. Farbe = ∞)
2. Wähle einen ungefärbten Knoten aus und gib ihm die kleinste zulässige Farbe (also eine Farbe, die keiner seiner Nachbarn hat), bis alle Knoten gefärbt sind.

Analyse

- Braucht höchstens $\Delta(G) + 1$ Farben.
- Optimaler Wert von f ist unbekannt, jedoch mindestens 2.
- Somit Abschätzung für maximale Abweichung κ: $\kappa \leq \Delta(G) - 1$.

Knotenfärbung: Algorithmus

Algorithmus

1. Markiere alle Knoten als ungefärbt (bzw. Farbe = ∞)
2. Wähle einen ungefärbten Knoten aus und gib ihm die kleinste zulässige Farbe (also eine Farbe, die keiner seiner Nachbarn hat), bis alle Knoten gefärbt sind.

Analyse

- Braucht höchstens $\Delta(G) + 1$ Farben.
- Optimaler Wert von f ist unbekannt, jedoch mindestens 2.
- Somit Abschätzung für maximale Abweichung κ: $\kappa \leq \Delta(G) - 1$.

Knotenfärbung: Algorithmusanalyse

Analyse – Fortsetzung

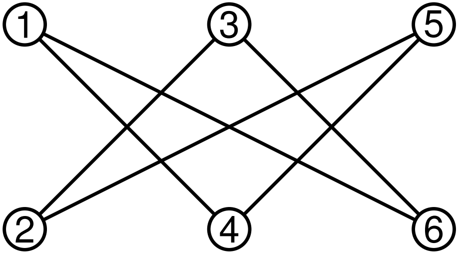
Bisher nur Abschätzung nach *oben* für κ. Wird dieser Wert auch angenommen oder war die Abschätzung schlecht?

Im Folgenden: Beispiel mit $\kappa = \Delta(G) - 1 = 2 - 1 = 1$.

Knotenfärbung: Algorithmusanalyse

Analyse – Fortsetzung

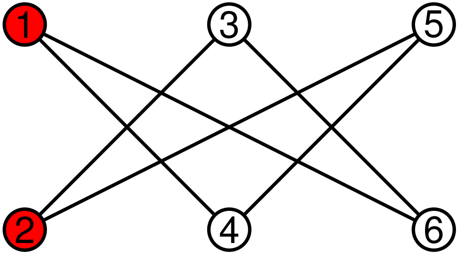
Bisher nur Abschätzung nach *oben* für κ. Wird dieser Wert auch angenommen oder war die Abschätzung schlecht?
Im Folgenden: Beispiel mit $\kappa = \Delta(G) - 1 = 2 - 1 = 1$.



Knotenfärbung: Algorithmusanalyse

Analyse – Fortsetzung

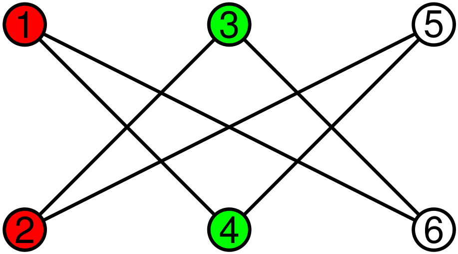
Bisher nur Abschätzung nach *oben* für κ. Wird dieser Wert auch angenommen oder war die Abschätzung schlecht?
Im Folgenden: Beispiel mit $\kappa = \Delta(G) - 1 = 2 - 1 = 1$.



Knotenfärbung: Algorithmusanalyse

Analyse – Fortsetzung

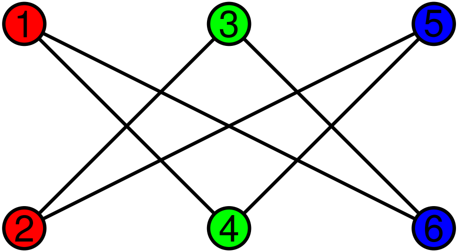
Bisher nur Abschätzung nach *oben* für κ. Wird dieser Wert auch angenommen oder war die Abschätzung schlecht?
Im Folgenden: Beispiel mit $\kappa = \Delta(G) - 1 = 2 - 1 = 1$.



Knotenfärbung: Algorithmusanalyse

Analyse – Fortsetzung

Bisher nur Abschätzung nach *oben* für κ. Wird dieser Wert auch angenommen oder war die Abschätzung schlecht?
Im Folgenden: Beispiel mit $\kappa = \Delta(G) - 1 = 2 - 1 = 1$.



3 Farben benutzt, aber nur 2 notwendig: Abweichung $\kappa = 1$ (Beispiel problemlos auf bel. $\Delta(G)$ erweiterbar).

Knotenfärbung: Algorithmenanalyse

Allgemeines zur Abschätzung der Abweichung

- Abschätzung, um obere Schranke der Abweichung zu finden.
- Beispiele, dass die gefundene Schranke auch angenommen wird („Zeuge“). Liefert oft Verständnis oder Verbesserungsansätze für den Algorithmus.

Zur Beliebigkeit der „Wahl“

- Im Algorithmus: „wähle einen ungefärbten Knoten“ ist unscharf.
- Hier: es gibt immer eine Reihenfolge, die optimal färbt(!)
- Aber das zeigt nur: nichtdeterministisch polynomiell.

Knotenfärbung: Algorithmenanalyse

Allgemeines zur Abschätzung der Abweichung

- Abschätzung, um obere Schranke der Abweichung zu finden.
- Beispiele, dass die gefundene Schranke auch angenommen wird („Zeuge“). Liefert oft Verständnis oder Verbesserungsansätze für den Algorithmus.

Zur Beliebigkeit der „Wahl“

- Im Algorithmus: „wähle einen ungefärbten Knoten“ ist unscharf.
- Hier: es gibt immer eine Reihenfolge, die optimal färbt(!)
- Aber das zeigt nur: nichtdeterministisch polynomiell.

Knotenfärbung: Algorithmenanalyse

Allgemeines zur Abschätzung der Abweichung

- Abschätzung, um obere Schranke der Abweichung zu finden.
- Beispiele, dass die gefundene Schranke auch angenommen wird („Zeuge“). Liefert oft Verständnis oder Verbesserungsansätze für den Algorithmus.

Zur Beliebigkeit der „Wahl“

- Im Algorithmus: „wähle einen ungefärbten Knoten“ ist unscharf.
- Hier: es gibt immer eine Reihenfolge, die optimal färbt(!)
- Aber das zeigt nur: nichtdeterministisch polynomiell.

Knotenfärbung: Algorithmenanalyse

Allgemeines zur Abschätzung der Abweichung

- Abschätzung, um obere Schranke der Abweichung zu finden.
- Beispiele, dass die gefundene Schranke auch angenommen wird („Zeuge“). Liefert oft Verständnis oder Verbesserungsansätze für den Algorithmus.

Zur Beliebigkeit der „Wahl“

- Im Algorithmus: „wähle einen ungefärbten Knoten“ ist unscharf.
- Hier: es gibt immer eine Reihenfolge, die optimal färbt(!)
- Aber das zeigt nur: nichtdeterministisch polynomiell.

Knotenfärbung: Algorithmenanalyse

Allgemeines zur Abschätzung der Abweichung

- Abschätzung, um obere Schranke der Abweichung zu finden.
- Beispiele, dass die gefundene Schranke auch angenommen wird („Zeuge“). Liefert oft Verständnis oder Verbesserungsansätze für den Algorithmus.

Zur Beliebigkeit der „Wahl“

- Im Algorithmus: „wähle einen ungefärbten Knoten“ ist unscharf.
- Hier: es gibt immer eine Reihenfolge, die optimal färbt(!)
- Aber das zeigt nur: nichtdeterministisch polynomiell.

Knotenfärbung für planare Graphen

Ergebnisse für planare Graphen

- Knotenfärbung mit 6 Farben in Polynomzeit ist möglich.
- Algorithmus für 2 Farben: Fange mit Farbe f_1 irgendwo an und färbe soweit möglich ($\neg f_2 = f_1$).

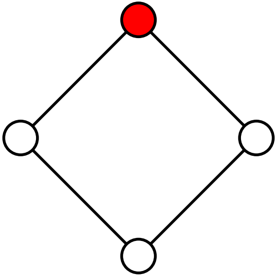
Durch Kombination erhält man einen Algorithmus mit Abweichung $\kappa \leq 3$.

Knotenfärbung für planare Graphen

Ergebnisse für planare Graphen

- Knotenfärbung mit 6 Farben in Polynomzeit ist möglich.
- Algorithmus für 2 Farben: Fange mit Farbe f_1 irgendwo an und färbe soweit möglich ($\neg f_2 = f_1$).

Durch Kombination erhält man einen Algorithmus mit Abweichung $\kappa \leq 3$.

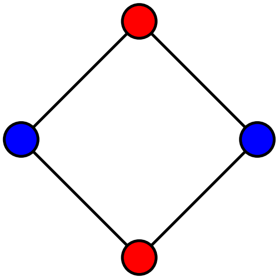


Knotenfärbung für planare Graphen

Ergebnisse für planare Graphen

- Knotenfärbung mit 6 Farben in Polynomzeit ist möglich.
- Algorithmus für 2 Farben: Fange mit Farbe f_1 irgendwo an und färbe soweit möglich ($\neg f_2 = f_1$).

Durch Kombination erhält man einen Algorithmus mit Abweichung $\kappa \leq 3$.



Knotenfärbung für planare Graphen

Ergebnisse für planare Graphen

- Knotenfärbung mit 6 Farben in Polynomzeit ist möglich.
- Algorithmus für 2 Farben: Fange mit Farbe f_1 irgendwo an und färbe soweit möglich ($\neg f_2 = f_1$).

Durch Kombination erhält man einen Algorithmus mit Abweichung $\kappa \leq 3$.

Knotenfärbung für planare Graphen

Satz

Jeder planare Graph enthält mindestens einen Knoten vom Grad ≤ 5.

Algorithmus

Eingabe: teilgefärbter planarer Graph G.

1. Wähle einen Knoten vom Grad ≤ 5 und färbe ihn und seine Nachbarn (benötigt max. 6 Farben).
2. Entferne den gefärbten Knoten $\rightsquigarrow$ teilgefärbter, planarer Graph G'. Fahre damit rekursiv fort.

Knotenfärbung für planare Graphen

Satz

Jeder planare Graph enthält mindestens einen Knoten vom Grad ≤ 5.

Algorithmus

Eingabe: teilgefärbter planarer Graph G.

1. Wähle einen Knoten vom Grad ≤ 5 und färbe ihn und seine Nachbarn (benötigt max. 6 Farben).
2. Entferne den gefärbten Knoten $\rightsquigarrow$ teilgefärbter, planarer Graph G'. Fahre damit rekursiv fort.

Knotenfärbung für planare Graphen

Satz

Jeder planare Graph enthält mindestens einen Knoten vom Grad ≤ 5.

Algorithmus

Eingabe: teilgefärbter planarer Graph G.

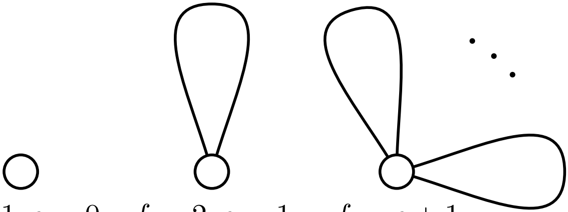
1. Wähle einen Knoten vom Grad ≤ 5 und färbe ihn und seine Nachbarn (benötigt max. 6 Farben).
2. Entferne den gefärbten Knoten $\rightsquigarrow$ teilgefärbter, planarer Graph G'. Fahre damit rekursiv fort.

Knotenfärbung für planare Graphen

Beweis

- Eulerscher Polyedersatz für planare Graphen:
  $v + f - e = 2$ bei e Kanten, v Knoten und f Flächen
  (Beweis: Induktion über v mit Multigraphen).

Induktionsanfang, $v = 1$:



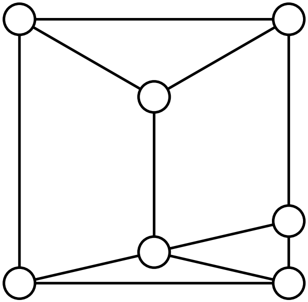
$f = 1, e = 0$ $\quad f = 2, e = 1$ $\quad f = e + 1$

Knotenfärbung für planare Graphen

Beweis

- Eulerscher Polyedersatz für planare Graphen:
  $v + f - e = 2$ bei e Kanten, v Knoten und f Flächen
  (Beweis: Induktion über v mit Multigraphen).

Induktionsschritt in v:

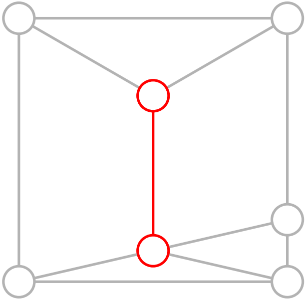


Knotenfärbung für planare Graphen

Beweis

- Eulerscher Polyedersatz für planare Graphen:
  $v + f - e = 2$ bei e Kanten, v Knoten und f Flächen
  (Beweis: Induktion über v mit Multigraphen).

Induktionsschritt in v:

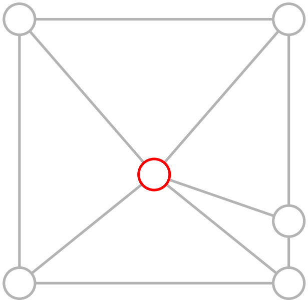


Knotenfärbung für planare Graphen

Beweis

- Eulerscher Polyedersatz für planare Graphen:
  $v + f - e = 2$ bei e Kanten, v Knoten und f Flächen
  (Beweis: Induktion über v mit Multigraphen).

Induktionsschritt in v:



Knotenfärbung für planare Graphen

Beweis

- Eulerscher Polyedersatz für planare Graphen: $v + f - e = 2$ bei e Kanten, v Knoten und f Flächen (Beweis: Induktion über v mit Multigraphen).
- Sei $v \geq 3$. Jede Fläche wird von ≥ 3 Kanten begrenzt; jede Kante dient als Grenze für maximal 2 Flächen. Also $f \leq \frac{2e}{3}$.

Knotenfärbung für planare Graphen

Beweis

- Eulerscher Polyedersatz für planare Graphen: $v + f - e = 2$ bei e Kanten, v Knoten und f Flächen (Beweis: Induktion über v mit Multigraphen).
- Sei $v \geq 3$. Jede Fläche wird von ≥ 3 Kanten begrenzt; jede Kante dient als Grenze für maximal 2 Flächen. Also $f \leq \frac{2e}{3}$.
- Damit: $e \leq 3v - 6$.

Knotenfärbung für planare Graphen

Beweis

- Eulerscher Polyedersatz für planare Graphen: $v + f - e = 2$ bei e Kanten, v Knoten und f Flächen (Beweis: Induktion über v mit Multigraphen).
- Sei $v \geq 3$. Jede Fläche wird von ≥ 3 Kanten begrenzt; jede Kante dient als Grenze für maximal 2 Flächen. Also $f \leq \frac{2e}{3}$.
- Damit: $e \leq 3v - 6$.
- Annahme: Alle Knoten sind vom Grad ≥ 6, also

$$e \geq \frac{6v}{2} = 3v.$$

Widerspruch. $\square$

Kantenfärbung: Einführung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Kantenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- Offenbar braucht man mindestens $\Delta(G)$ Farben.
- Es gibt Graphen, bei denen $\Delta(G)+1$ Farben notwendig sind.
- $\Delta(G)+1$ Farben sind immer ausreichend. Es gibt dazu einen Algorithmus in $\mathcal{P}$.
- Die Entscheidung, ob ein 3-regulärer Graph mit 3 Farben Kanten-färbbar ist, ist $\mathcal{NP}$-vollständig.

Kantenfärbung: Einführung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Kantenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- Offenbar braucht man mindestens $\Delta(G)$ Farben.
- Es gibt Graphen, bei denen $\Delta(G) + 1$ Farben notwendig sind.
- $\Delta(G) + 1$ Farben sind immer ausreichend. Es gibt dazu einen Algorithmus in $\mathcal{P}$.
- Die Entscheidung, ob ein 3-regulärer Graph mit 3 Farben Kanten-färbbar ist, ist $\mathcal{NP}$-vollständig.

Kantenfärbung: Einführung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Kantenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- Offenbar braucht man mindestens $\Delta(G)$ Farben.
- Es gibt Graphen, bei denen $\Delta(G) + 1$ Farben notwendig sind.
- $\Delta(G) + 1$ Farben sind immer ausreichend. Es gibt dazu einen Algorithmus in $\mathcal{P}$.
- Die Entscheidung, ob ein 3-regulärer Graph mit 3 Farben Kanten-färbbar ist, ist $\mathcal{NP}$-vollständig.

Kantenfärbung: Einführung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Kantenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- Offenbar braucht man mindestens $\Delta(G)$ Farben.
- Es gibt Graphen, bei denen $\Delta(G) + 1$ Farben notwendig sind.
- $\Delta(G) + 1$ Farben sind immer ausreichend. Es gibt dazu einen Algorithmus in $\mathcal{P}$.
- Die Entscheidung, ob ein 3-regulärer Graph mit 3 Farben Kanten-färbbar ist, ist $\mathcal{NP}$-vollständig.

Kantenfärbung: Einführung

Optimierungsproblem

Bestimme zu einem ungerichteten Graph eine zulässige Kantenfärbung, so dass die Anzahl der verwendeten Farben minimal ist.

Bemerkungen

- Offenbar braucht man mindestens $\Delta(G)$ Farben.
- Es gibt Graphen, bei denen $\Delta(G) + 1$ Farben notwendig sind.
- $\Delta(G) + 1$ Farben sind immer ausreichend. Es gibt dazu einen Algorithmus in $\mathcal{P}$.
- Die Entscheidung, ob ein 3-regulärer Graph mit 3 Farben Kanten-färbbar ist, ist $\mathcal{NP}$-vollständig.

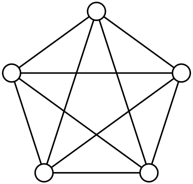
Kantenfärbung: Beispiel für $\Delta(G) + 1$

Satz

Eine Kantenfärbung von K_n mit n ungerade braucht (mindestens) $\Delta(G) + 1 = n$ Farben.

Beweis

Durch Widerspruch. Jeder Knoten hat $\Delta(G) = n - 1$ Kanten, daher hat jeder der n Knoten genau eine „rote“ Kante. Andererseits ist die Anzahl Knoten mit roter Kante gerade. Widerspruch. $\square$



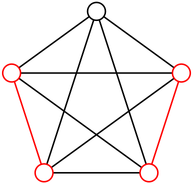
Kantenfärbung: Beispiel für $\Delta(G) + 1$

Satz

Eine Kantenfärbung von K_n mit n ungerade braucht (mindestens) $\Delta(G) + 1 = n$ Farben.

Beweis

Durch Widerspruch. Jeder Knoten hat $\Delta(G) = n - 1$ Kanten, daher hat jeder der n Knoten genau eine „rote“ Kante. Andererseits ist die Anzahl Knoten mit roter Kante gerade. Widerspruch. $\square$



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Satz

Für eine gegebene Kantenfärbung von G und zwei Knoten u, v mit $\deg u, \deg v < \Delta(G)$ kann G so umgefärbt werden, dass an u und v dieselbe Farbe fehlt (alle Färbungen haben dabei maximal $\Delta(G) + 1$ Farben).

Algorithmus

1. Entferne eine Kante $\{u, v\}$ von G.
2. Färbe den Restgraphen so, dass an u und v dieselbe Farbe fehlt.
3. Füge $\{u, v\}$ wieder ein und färbe mit der fehlenden Farbe.

Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Satz

Für eine gegebene Kantenfärbung von G und zwei Knoten u, v mit $\deg u, \deg v < \Delta(G)$ kann G so umgefärbt werden, dass an u und v dieselbe Farbe fehlt (alle Färbungen haben dabei maximal $\Delta(G) + 1$ Farben).

Algorithmus

1. Entferne eine Kante $\{u, v\}$ von G.
2. Färbe den Restgraphen so, dass an u und v dieselbe Farbe fehlt.
3. Füge $\{u, v\}$ wieder ein und färbe mit der fehlenden Farbe.

Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Satz

Für eine gegebene Kantenfärbung von G und zwei Knoten u, v mit $\deg u, \deg v < \Delta(G)$ kann G so umgefärbt werden, dass an u und v dieselbe Farbe fehlt (alle Färbungen haben dabei maximal $\Delta(G) + 1$ Farben).

Algorithmus

1. Entferne eine Kante $\{u, v\}$ von G.
2. Färbe den Restgraphen so, dass an u und v dieselbe Farbe fehlt.
3. Füge $\{u, v\}$ wieder ein und färbe mit der fehlenden Farbe.

Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Satz

Für eine gegebene Kantenfärbung von G und zwei Knoten u, v mit $\deg u, \deg v < \Delta(G)$ kann G so umgefärbt werden, dass an u und v dieselbe Farbe fehlt (alle Färbungen haben dabei maximal $\Delta(G) + 1$ Farben).

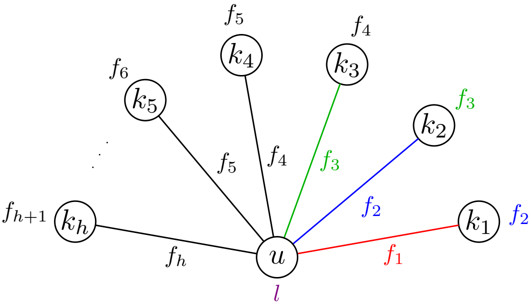
Algorithmus

1. Entferne eine Kante $\{u, v\}$ von G.
2. Färbe den Restgraphen so, dass an u und v dieselbe Farbe fehlt.
3. Füge $\{u, v\}$ wieder ein und färbe mit der fehlenden Farbe.

Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Beweis

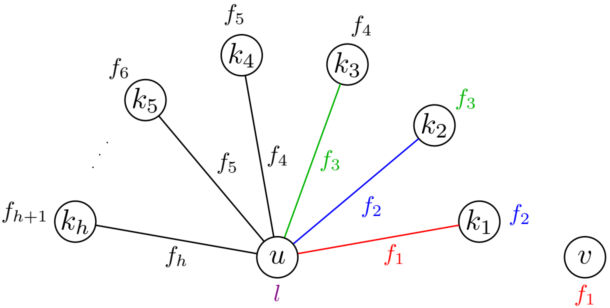
Konstruktiv. Konstruiere Sequenz von Nachbarknoten von u: $(k_1, \ldots k_h)$. Die dazugehörigen Farben seien $(f_1, \ldots f_{h+1})$. Es soll gelten: am Knoten k_i fehlt die Farbe f_{i+1}.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 1

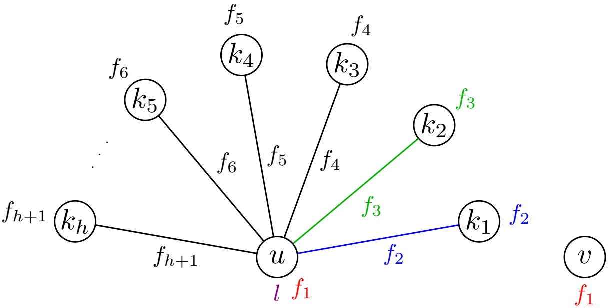
Man findet keine Kante an u mit der Farbe f_{h+1}. In diesem Fall umfärben.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 1

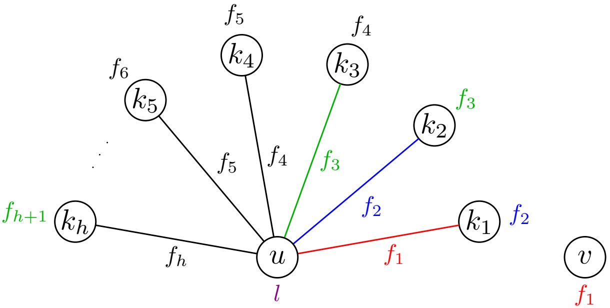
Man findet keine Kante an u mit der Farbe f_{h+1}. In diesem Fall umfärben.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2

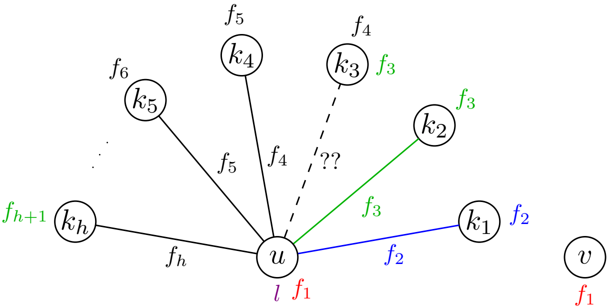
Die Farbe f_{h+1} ist bereits in der Sequenz (bei $\{u, k_j\}$).
Verschiebe zunächst die Farben von f_1 bis f_j wie in Fall 1,
entfärbe $\{u, k_j\}$.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2

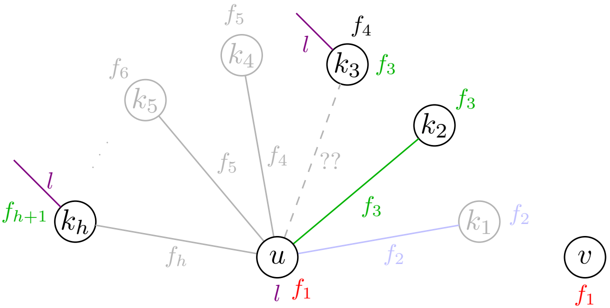
Die Farbe f_{h+1} ist bereits in der Sequenz (bei $\{u, k_j\}$). Verschiebe zunächst die Farben von f_1 bis f_j wie in Fall 1, entfärbe $\{u, k_j\}$.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2 – Fortsetzung

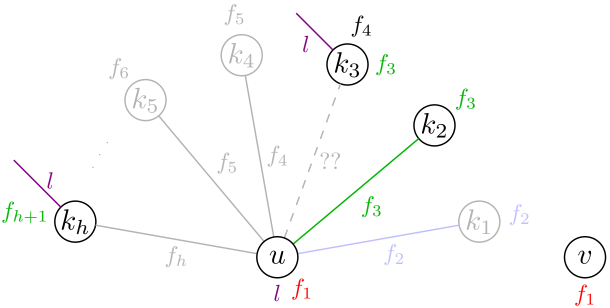
Betrachte Teilgraphen mit den Farben f_{h+1} und s. Da jeder Knoten höchstens zwei Kanten besitzt, besteht er nur aus Kreisen und Pfaden. u, k_j und k_h sind Endpunkte von Pfaden, liegen daher nicht alle drei in derselben Zsh-komponente.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2a

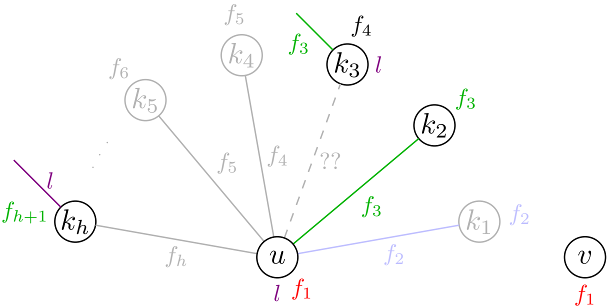
u und k_j liegen in verschiedenen Zsh-komponenten.
Vertausche die Farben s und f_{h+1} in der Komponente von k_j. s fehlt an u und k_j, färbe $\{u, k_j\}$ mit s.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2a

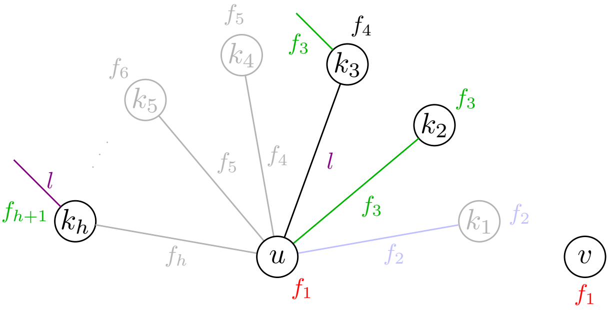
u und k_j liegen in verschiedenen Zsh-komponenten.
Vertausche die Farben s und f_{h+1} in der Komponente von k_j. s fehlt an u und k_j, färbe $\{u, k_j\}$ mit s.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2a

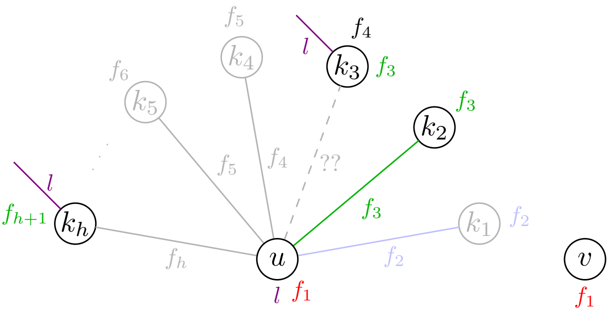
u und k_j liegen in verschiedenen Zsh-komponenten.
Vertausche die Farben s und f_{h+1} in der Komponente von k_j. s fehlt an u und k_j, färbe $\{u, k_j\}$ mit s.



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2b

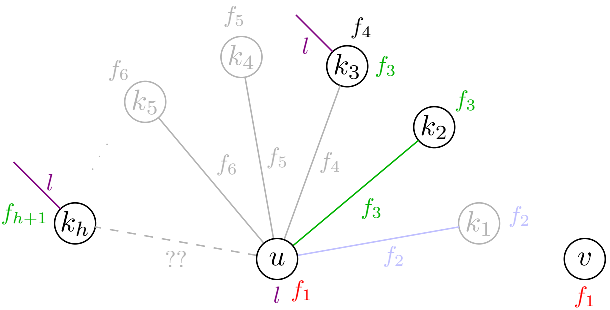
u und k_h liegen in verschiedenen Zsh-komponenten. Tausche Farben $f_j, \ldots f_h$, entfärbe $\{u, k_h\}$. Somit liegt Fall 2a, vor: wieder Farben f_{h+1} und s vertauschen in der k_h-Komponente, $\{u, k_h\}$ mit s einfärben. □



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2b

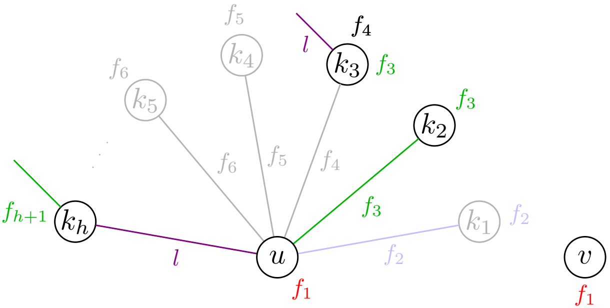
u und k_h liegen in verschiedenen Zsh-komponenten. Tausche Farben $f_j, \dots f_h$, entfärbe $\{u, k_h\}$. Somit liegt Fall 2a, vor: wieder Farben f_{h+1} und s vertauschen in der k_h-Komponente, $\{u, k_h\}$ mit s einfärben. □



Kantenfärbung: $\Delta(G) + 1$-Algorithmus

Fall 2b

u und k_h liegen in verschiedenen Zsh-komponenten. Tausche Farben $f_j, \ldots f_h$, entfärbe $\{u, k_h\}$. Somit liegt Fall 2a, vor: wieder Farben f_{h+1} und s vertauschen in der k_h-Komponente, $\{u, k_h\}$ mit s einfärben. $\square$



Outline

Rucksackproblem: Einführung

Bemerkungen

- Bisher nur *absolute* Abweichung zum Optimum betrachtet.
- Oft sind Probleme jedoch „skaleninvariant“, sodass effiziente Algorithmen mit maximaler absoluter Abweichung nicht möglich sind.

Rucksackproblem

Gegeben sei eine Menge mit Waren W, deren Elemente je ein Wert $p(w) \in \mathbb{Q}^+$ und ein Volumen $v(w) \in \mathbb{Q}^+$ zugeordnet sei. Der Rucksack habe das Volumen V.
Wähle eine Untermenge der Waren so aus, dass sie in den Rucksack passen und ihr Wert möglichst groß ist.

Rucksackproblem: Einführung

Bemerkungen

- Bisher nur *absolute* Abweichung zum Optimum betrachtet.
- Oft sind Probleme jedoch „skaleninvariant“, sodass effiziente Algorithmen mit maximaler absoluter Abweichung nicht möglich sind.

Rucksackproblem

Gegeben sei eine Menge mit Waren W, deren Elemente je ein Wert $p(w) \in \mathbb{Q}^+$ und ein Volumen $v(w) \in \mathbb{Q}^+$ zugeordnet sei. Der Rucksack habe das Volumen V.
Wähle eine Untermenge der Waren so aus, dass sie in den Rucksack passen und ihr Wert möglichst groß ist.

Rucksackproblem: Einführung

Bemerkungen

- Bisher nur *absolute* Abweichung zum Optimum betrachtet.
- Oft sind Probleme jedoch „skaleninvariant“, sodass effiziente Algorithmen mit maximaler absoluter Abweichung nicht möglich sind.

Rucksackproblem

Gegeben sei eine Menge mit Waren W, deren Elemente je ein Wert $p(w) \in \mathbb{Q}^+$ und ein Volumen $v(w) \in \mathbb{Q}^+$ zugeordnet sei.
Der Rucksack habe das Volumen V.
Wähle eine Untermenge der Waren so aus, dass sie in den Rucksack passen und ihr Wert möglichst groß ist.

Rucksackproblem: Beispiel

Rucksack-Volumen: 5.

Wert	1	1	1	4
Volumen	1	1	1	5

Ausgabe eines Greedy-Algorithmus

Wert	1	1	1	4
Volumen	1	1	1	5

Abweichung vom Optimum: 1

Rucksackproblem: Beispiel

Rucksack-Volumen: 5. Optimale Lösung:

Wert	1	1	1	4
Volumen	1	1	1	5

Ausgabe eines Greedy-Algorithmus

Wert	1	1	1	4
Volumen	1	1	1	5

Abweichung vom Optimum: 1

Rucksackproblem: Beispiel

Rucksack-Volumen: 5. Optimale Lösung:

Wert	1	1	1	4
Volumen	1	1	1	5

Ausgabe eines Greedy-Algorithmus

Wert	1	1	1	4
Volumen	1	1	1	5

Abweichung vom Optimum: 1

Rucksackproblem: Beispiel

Rucksack-Volumen: 5. Optimale Lösung:

Wert	1	1	1	4
Volumen	1	1	1	5

Ausgabe eines Greedy-Algorithmus

Wert	10	10	10	40
Volumen	1	1	1	5

Abweichung vom Optimum: 10

Rucksackproblem: Ein Unmöglichkeitsergebnis

Satz

Es gibt keinen effizienten Algorithmus für das Rucksackproblem, der eine garantierte maximale Abweichung κ hat.

Bemerkungen

- *effizienter Algorithmus*: Algorithmus soll in $\mathcal{P}$ liegen.
- Es wird $\mathcal{P} \neq \mathcal{NP}$ vorausgesetzt.
- Der Satz gilt unabhängig vom Algorithmus.

Rucksackproblem: Ein Unmöglichkeitsergebnis

Satz

Es gibt keinen effizienten Algorithmus für das Rucksackproblem, der eine garantierte maximale Abweichung κ hat.

Bemerkungen

- *effizienter Algorithmus*: Algorithmus soll in $\mathcal{P}$ liegen.
- Es wird $\mathcal{P} \neq \mathcal{NP}$ vorausgesetzt.
- Der Satz gilt unabhängig vom Algorithmus.

Rucksackproblem: Ein Unmöglichkeitsergebnis

Satz

Es gibt keinen effizienten Algorithmus für das Rucksackproblem, der eine garantierte maximale Abweichung κ hat.

Bemerkungen

- *effizienter Algorithmus*: Algorithmus soll in $\mathcal{P}$ liegen.
- Es wird $\mathcal{P} \neq \mathcal{NP}$ vorausgesetzt.
- Der Satz gilt unabhängig vom Algorithmus.

Rucksackproblem: Ein Unmöglichkeitsergebnis

Satz

Es gibt keinen effizienten Algorithmus für das Rucksackproblem, der eine garantierte maximale Abweichung κ hat.

Bemerkungen

- *effizienter Algorithmus*: Algorithmus soll in $\mathcal{P}$ liegen.
- Es wird $\mathcal{P} \neq \mathcal{NP}$ vorausgesetzt.
- Der Satz gilt unabhängig vom Algorithmus.

Rucksackproblem: Ein Unmöglichkeitsergebnis

Beweis

Durch Widerspruch. Angenommen es gibt ein $\mathcal{P}$-Algorithmus A mit maximaler absoluter Abweichung κ.

- Skaliere alle Werte so, dass die minimale Wertdifferenz zweier Waren von unterschiedlichem Wert κ übersteigt und wende A auf das neue Problem an.
- Die Warenauswahl, die A ausgibt, löst das ursprüngliche Problem mit einer Abweichung, die kleiner ist als die kleinste Differenz zweier Warenwerte. Damit ist die Lösung optimal.
- Somit folgt $\mathcal{P} = \mathcal{NP}$. Widerspruch. $\square$

Rucksackproblem: Ein Unmöglichkeitsergebnis

Beweis

Durch Widerspruch. Angenommen es gibt ein $\mathcal{P}$-Algorithmus A mit maximaler absoluter Abweichung κ.

- Skaliere alle Werte so, dass die minimale Wertdifferenz zweier Waren von unterschiedlichem Wert κ übersteigt und wende A auf das neue Problem an.
- Die Warenauswahl, die A ausgibt, löst das ursprüngliche Problem mit einer Abweichung, die kleiner ist als die kleinste Differenz zweier Warenwerte. Damit ist die Lösung optimal.
- Somit folgt $\mathcal{P} = \mathcal{NP}$. Widerspruch. $\square$

Rucksackproblem: Ein Unmöglichkeitsergebnis

Beweis

Durch Widerspruch. Angenommen es gibt ein $\mathcal{P}$-Algorithmus A mit maximaler absoluter Abweichung κ.

- Skaliere alle Werte so, dass die minimale Wertdifferenz zweier Waren von unterschiedlichem Wert κ übersteigt und wende A auf das neue Problem an.
- Die Warenauswahl, die A ausgibt, löst das ursprüngliche Problem mit einer Abweichung, die kleiner ist als die kleinste Differenz zweier Warenwerte. Damit ist die Lösung optimal.
- Somit folgt $\mathcal{P} = \mathcal{NP}$. Widerspruch. $\square$

Rucksackproblem: Ein Unmöglichkeitsergebnis

Beweis

Durch Widerspruch. Angenommen es gibt ein $\mathcal{P}$-Algorithmus A mit maximaler absoluter Abweichung κ.

- Skaliere alle Werte so, dass die minimale Wertdifferenz zweier Waren von unterschiedlichem Wert κ übersteigt und wende A auf das neue Problem an.
- Die Warenauswahl, die A ausgibt, löst das ursprüngliche Problem mit einer Abweichung, die kleiner ist als die kleinste Differenz zweier Warenwerte. Damit ist die Lösung optimal.
- Somit folgt $\mathcal{P} = \mathcal{NP}$. Widerspruch. $\square$

Outline

Knotenüberdeckung: Einführung

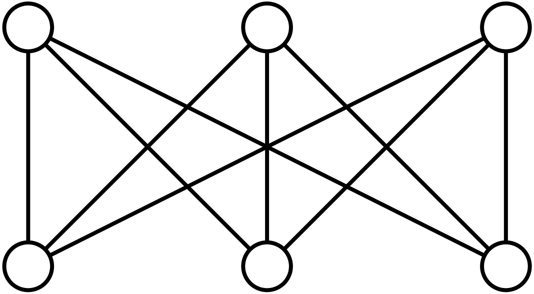
Optimierungsproblem

Gegeben sei ein (ungerichteter) Graph (V, E) und eine Kostenfunktion für die Knoten $c : V \to \mathbb{Q}^+$. Finde eine möglichst günstige Knotenmenge $C \subseteq V$, die zu jeder Kante $\{u, v\}$ des Graphen mindestens einen der Knoten u oder v enthält.

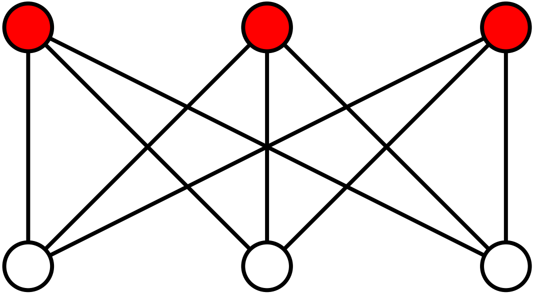
Bemerkung

Wir untersuchen den Fall $c(u) = 1$ für alle Knoten u.

Knotenüberdeckung: Beispielproblem



Knotenüberdeckung: Beispielproblem

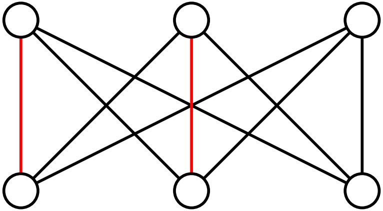


Mögliche Lösung: Jede Kante hat einen gewählten Knoten.

Knotenüberdeckung: Definition

Definition

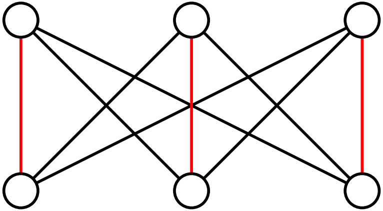
Eine *Paarung* eines (ungerichteten) Graphen (V, E) ist eine Untermenge M der Kanten, sodass keine zwei Kanten aus M einen gemeinsamen Knoten besitzen. Eine Paarung heißt *nicht erweiterbar*, falls es keine Kante $\{u, v\} \in E$ gibt, sodass $M \cup \{u, v\}$ eine Paarung ist.



Knotenüberdeckung: Definition

Definition

Eine *Paarung* eines (ungerichteten) Graphen (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 $\{u, v\} \in E$ gibt, sodass $M \cup \{u, v\}$ eine Paarung ist.



Knotenüberdeckung: Algorithmus

Algorithmus

1. Finde durch sukzessives Hinzufügen von Kanten (und Löschen der jeweiligen Knoten) eine nicht erweiterbare Paarung M.
2. Gib die Knoten von M aus.

Analyse

- Die Ausgabe ist eine Knotenüberdeckung, sonst wäre die Paarung erweiterbar.
- Jede Knotenüberdeckung muss mindestens 1 Knoten für jede Kante in M beisteuern. Der Algorithmus gibt 2 Knoten für jede Kante aus, liegt also maximal Faktor 2 daneben.
- Für $K_{n,n}$ wählt der Algorithmus alle Knoten aus, statt der Hälfte, Faktor 2 wird also angenommen.

Knotenüberdeckung: Algorithmus

Algorithmus

1. Finde durch sukzessives Hinzufügen von Kanten (und Löschen der jeweiligen Knoten) eine nicht erweiterbare Paarung M.
2. Gib die Knoten von M aus.

Analyse

- Die Ausgabe ist eine Knotenüberdeckung, sonst wäre die Paarung erweiterbar.
- Jede Knotenüberdeckung muss mindestens 1 Knoten für jede Kante in M beisteuern. Der Algorithmus gibt 2 Knoten für jede Kante aus, liegt also maximal Faktor 2 daneben.
- Für $K_{n,n}$ wählt der Algorithmus alle Knoten aus, statt der Hälfte, Faktor 2 wird also angenommen.

Knotenüberdeckung: Algorithmus

Algorithmus

1. Finde durch sukzessives Hinzufügen von Kanten (und Löschen der jeweiligen Knoten) eine nicht erweiterbare Paarung M.
2. Gib die Knoten von M aus.

Analyse

- Die Ausgabe ist eine Knotenüberdeckung, sonst wäre die Paarung erweiterbar.
- Jede Knotenüberdeckung muss mindestens 1 Knoten für jede Kante in M beisteuern. Der Algorithmus gibt 2 Knoten für jede Kante aus, liegt also maximal Faktor 2 daneben.
- Für $K_{n,n}$ wählt der Algorithmus alle Knoten aus, statt der Hälfte, Faktor 2 wird also angenommen.

Knotenüberdeckung: Algorithmus

Algorithmus

1. Finde durch sukzessives Hinzufügen von Kanten (und Löschen der jeweiligen Knoten) eine nicht erweiterbare Paarung M.
2. Gib die Knoten von M aus.

Analyse

- Die Ausgabe ist eine Knotenüberdeckung, sonst wäre die Paarung erweiterbar.
- Jede Knotenüberdeckung muss mindestens 1 Knoten für jede Kante in M beisteuern. Der Algorithmus gibt 2 Knoten für jede Kante aus, liegt also maximal Faktor 2 daneben.
- Für $K_{n,n}$ wählt der Algorithmus alle Knoten aus, statt der Hälfte, Faktor 2 wird also angenommen.

Knotenüberdeckung: Algorithmusanalyse

Daraus folgt:

Satz

Der vorgestellte Algorithmus zur Knotenüberdeckung hat im schlechtesten Fall eine relative Güte von 2.

Definition

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

$$\rho := \frac{f(A(I))}{\text{OPT}}$$

wobei OPT der Funktionswert der Bewertungsfunktion f der optimalen Lösung von I ist.
(Für Maximierungs-Algorithmen definiert man $\rho := \text{OPT}/f(A(I))$, sodass stets $\rho \geq 1$).

Knotenüberdeckung: Algorithmusanalyse

Daraus folgt:

Satz

Der vorgestellte Algorithmus zur Knotenüberdeckung hat im schlechtesten Fall eine relative Güte von 2.

Definition

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

$$\rho := \frac{f(A(I))}{\mathrm{OPT}}$$

wobei OPT der Funktionswert der Bewertungsfunktion f der optimalen Lösung von I ist.
(Für Maximierungs-Algorithmen definiert man $\rho := \mathrm{OPT}/f(A(I))$, sodass stets $\rho \geq 1$).

Knotenüberdeckung: Selbstreduktion

- Hatten bereits: Optimierungsproblem ist mindestens so schwer wie das zugehörige Entscheidungsproblem.
- Oft gilt umgekehrt: Ist das Entscheidungsproblem gelöst, lässt sich ein $\mathcal{P}$-Optimierungsalgorithmus angeben.

Bemerkung

Annahme: das Entscheidungsproblem ist (in $\mathcal{P}$) gelöst, wir haben ein „Orakel“.

Knotenüberdeckung: Selbstreduktion

- Hatten bereits: Optimierungsproblem ist mindestens so schwer wie das zugehörige Entscheidungsproblem.
- Oft gilt umgekehrt: Ist das Entscheidungsproblem gelöst, lässt sich ein $\mathcal{P}$-Optimierungsalgorithmus angeben.

Bemerkung

Annahme: das Entscheidungsproblem ist (in $\mathcal{P}$) gelöst, wir haben ein „Orakel“.

Knotenüberdeckung: Selbstreduktion

- Hatten bereits: Optimierungsproblem ist mindestens so schwer wie das zugehörige Entscheidungsproblem.
- Oft gilt umgekehrt: Ist das Entscheidungsproblem gelöst, lässt sich ein $\mathcal{P}$-Optimierungsalgorithmus angeben.

Bemerkung

Annahme: das Entscheidungsproblem ist (in $\mathcal{P}$) gelöst, wir haben ein „Orakel".

Knotenüberdeckung: Selbstreduktion

Algorithmus

Eingabe: Graph $G = (V, E)$.

1. Bestimme $\mathrm{OPT}(G)$ durch Binärsuche mithilfe des Orakels.
2. Entferne einen beliebigen Knoten $v \rightsquigarrow$ reduzierter Graph G'. Falls $\mathrm{OPT}(G') = \mathrm{OPT}(G) - 1$, gehört v zur optimalen Lösung, andernfalls alle Nachbarn von v.
3. Verfahre genauso mit G'.

Anmerkung und Fazit

- Selbstreduktion hier sehr einfach. Aber: Ähnliches ist oft möglich.
- Insofern steckt der „schwierige Kern“ oft bereits in der Entscheidungsversion.

Knotenüberdeckung: Selbstreduktion

Algorithmus

Eingabe: Graph $G = (V, E)$.

1. Bestimme $\mathrm{OPT}(G)$ durch Binärsuche mithilfe des Orakels.
2. Entferne einen beliebigen Knoten $v \rightsquigarrow$ reduzierter Graph G'. Falls $\mathrm{OPT}(G') = \mathrm{OPT}(G) - 1$, gehört v zur optimalen Lösung, andernfalls alle Nachbarn von v.
3. Verfahre genauso mit G'.

Anmerkung und Fazit

- Selbstreduktion hier sehr einfach. Aber: Ähnliches ist oft möglich.
- Insofern steckt der „schwierige Kern“ oft bereits in der Entscheidungsversion.

Knotenüberdeckung: Selbstreduktion

Algorithmus

Eingabe: Graph $G = (V, E)$.

1. Bestimme $\mathrm{OPT}(G)$ durch Binärsuche mithilfe des Orakels.
2. Entferne einen beliebigen Knoten $v \rightsquigarrow$ reduzierter Graph G'. Falls $\mathrm{OPT}(G') = \mathrm{OPT}(G) - 1$, gehört v zur optimalen Lösung, andernfalls alle Nachbarn von v.
3. Verfahre genauso mit G'.

Anmerkung und Fazit

- Selbstreduktion hier sehr einfach. Aber: Ähnliches ist oft möglich.
- Insofern steckt der „schwierige Kern“ oft bereits in der Entscheidungsversion.

Knotenüberdeckung: Selbstreduktion

Algorithmus

Eingabe: Graph $G = (V, E)$.

1. Bestimme $\mathrm{OPT}(G)$ durch Binärsuche mithilfe des Orakels.
2. Entferne einen beliebigen Knoten $v \rightsquigarrow$ reduzierter Graph G'. Falls $\mathrm{OPT}(G') = \mathrm{OPT}(G) - 1$, gehört v zur optimalen Lösung, andernfalls alle Nachbarn von v.
3. Verfahre genauso mit G'.

Anmerkung und Fazit

- Selbstreduktion hier sehr einfach. Aber: Ähnliches ist oft möglich.
- Insofern steckt der „schwierige Kern“ oft bereits in der Entscheidungsversion.

Knotenüberdeckung: Selbstreduktion

Algorithmus

Eingabe: Graph $G = (V, E)$.

1. Bestimme $\mathrm{OPT}(G)$ durch Binärsuche mithilfe des Orakels.
2. Entferne einen beliebigen Knoten $v \rightsquigarrow$ reduzierter Graph G'. Falls $\mathrm{OPT}(G') = \mathrm{OPT}(G) - 1$, gehört v zur optimalen Lösung, andernfalls alle Nachbarn von v.
3. Verfahre genauso mit G'.

Anmerkung und Fazit

- Selbstreduktion hier sehr einfach. Aber: Ähnliches ist oft möglich.
- Insofern steckt der „schwierige Kern“ oft bereits in der Entscheidungsversion.

Mengenüberdeckung: Einführung

Optimierungsproblem

Gegeben seien:

- Eine Menge U mit n Elementen
- Eine Menge S von Untermengen von U: $S = \{S_1, \ldots S_k\}$ mit $S_i \subseteq U$ für $i = 1 \ldots k$
- Eine Kostenfunktion $c : S \to \mathbb{Q}^+$

Finde eine Überdeckung von U mit möglichst geringen Kosten.

Definition und Bemerkung

- Eine *Überdeckung* ist eine Untermenge $T = \{T_1, \ldots T_l\}$ von S, sodass $\bigcup_i T_i = U$.
- Knotenüberdeckung ist der Spezialfall $|S_i| = 2 \forall_i$.

Mengenüberdeckung: Einführung

Optimierungsproblem

Gegeben seien:

- Eine Menge U mit n Elementen
- Eine Menge S von Untermengen von U: $S = \{S_1, \ldots S_k\}$ mit $S_i \subseteq U$ für $i = 1 \ldots k$
- Eine Kostenfunktion $c : S \to \mathbb{Q}^+$

Finde eine Überdeckung von U mit möglichst geringen Kosten.

Definition und Bemerkung

- Eine *Überdeckung* ist eine Untermenge $T = \{T_1, \ldots T_l\}$ von S, sodass $\bigcup_i T_i = U$.
- Knotenüberdeckung ist der Spezialfall $|S_i| = 2 \forall_i$.

Mengenüberdeckung: Einführung

Optimierungsproblem

Gegeben seien:

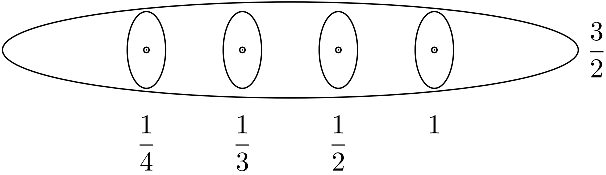
- Eine Menge U mit n Elementen
- Eine Menge S von Untermengen von U: $S = \{S_1, \ldots S_k\}$ mit $S_i \subseteq U$ für $i = 1 \ldots k$
- Eine Kostenfunktion $c : S \rightarrow \mathbb{Q}^+$

Finde eine Überdeckung von U mit möglichst geringen Kosten.

Definition und Bemerkung

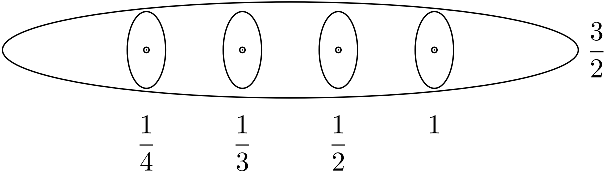
- Eine *Überdeckung* ist eine Untermenge $T = \{T_1, \ldots T_l\}$ von S, sodass $\bigcup_i T_i = U$.
- Knotenüberdeckung ist der Spezialfall $|S_i| = 2 \forall_i$.

Mengenüberdeckung: Beispielproblem

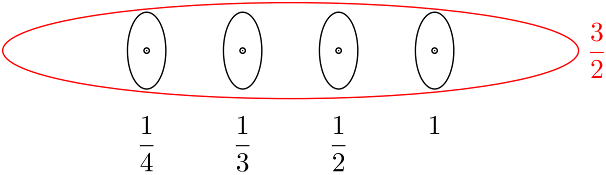


Optimale Lösung:

Mengenüberdeckung: Beispielproblem



Optimale Lösung:



Mengenüberdeckung: Algorithmus

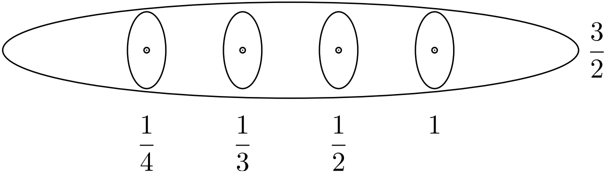
Greedy-Algorithmus

- $C \leftarrow \emptyset$
- Berechne die Kosteneffektivität α für jede der verbleibenden Mengen S_i:

$$\alpha = \frac{c(S_i)}{|S_i \setminus C|}.$$

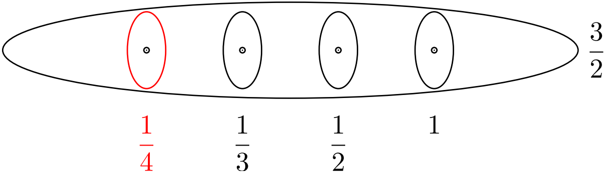
  Wähle die kosteneffektivste Menge: $C \leftarrow C \cup S$.
- Gib die gewählten Mengen aus.

Mengenüberdeckung: Algorithmusanwendung



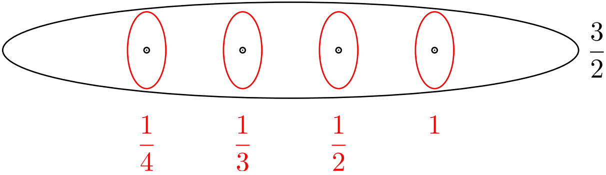
Kosten: $\frac{25}{12}$.

Mengenüberdeckung: Algorithmusanwendung



Kosten: $\frac{25}{12}$.

Mengenüberdeckung: Algorithmusanwendung



Kosten: $\frac{25}{12}$.

Mengenüberdeckung: Algorithmusanalyse

Satz

Der Greedy-Algorithmus hat im schlimmsten Fall eine relative Güte von H_n mit $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$.

Mengenüberdeckung: Algorithmusanalyse

Beweis

- Betrachte die Situation in einer beliebigen Iteration des Algorithmus. Die optimale Lösung kann die übrigen Elemente mit Kosten $\leq$ OPT überdecken.
- Es gibt also eine Menge mit Kosteneffektivität $\alpha \leq \mathrm{OPT}/|U \setminus C|$.
- Im schlimmsten Fall: je (o. E. einelementige) Mengen mit $\alpha = \mathrm{OPT}/|U \setminus C|$.
- Damit Kosten *maximal*

$$\sum_{i=1}^{n} \mathrm{OPT} \frac{1}{n-i+1} = \mathrm{OPT} \cdot H_n.$$

Mengenüberdeckung: Algorithmusanalyse

Beweis

- Betrachte die Situation in einer beliebigen Iteration des Algorithmus. Die optimale Lösung kann die übrigen Elemente mit Kosten $\leq$ OPT überdecken.
- Es gibt also eine Menge mit Kosteneffektivität $\alpha \leq \mathrm{OPT}/|U \setminus C|$.
- Im schlimmsten Fall: je (o. E. einelementige) Mengen mit $\alpha = \mathrm{OPT}/|U \setminus C|$.
- Damit Kosten *maximal*

$$\sum_{i=1}^{n} \mathrm{OPT} \frac{1}{n-i+1} = \mathrm{OPT} \cdot H_n.$$

Mengenüberdeckung: Algorithmusanalyse

Beweis

- Betrachte die Situation in einer beliebigen Iteration des Algorithmus. Die optimale Lösung kann die übrigen Elemente mit Kosten $\leq$ OPT überdecken.
- Es gibt also eine Menge mit Kosteneffektivität $\alpha \leq \mathrm{OPT}/|U \setminus C|$.
- Im schlimmsten Fall: je (o. E. einelementige) Mengen mit $\alpha = \mathrm{OPT}/|U \setminus C|$.
- Damit Kosten *maximal*

$$\sum_{i=1}^{n} \mathrm{OPT}\frac{1}{n-i+1} = \mathrm{OPT} \cdot H_n.$$

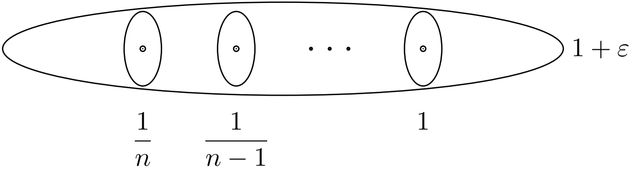
Mengenüberdeckung: Algorithmusanalyse

Beweis

- Betrachte die Situation in einer beliebigen Iteration des Algorithmus. Die optimale Lösung kann die übrigen Elemente mit Kosten $\leq$ OPT überdecken.
- Es gibt also eine Menge mit Kosteneffektivität $\alpha \leq \mathrm{OPT}/|U \setminus C|$.
- Im schlimmsten Fall: je (o. E. einelementige) Mengen mit $\alpha = \mathrm{OPT}/|U \setminus C|$.
- Damit Kosten *maximal*

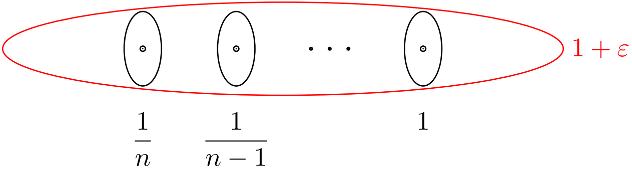
$$\sum_{i=1}^{n} \mathrm{OPT} \frac{1}{n-i+1} = \mathrm{OPT} \cdot H_n.$$

Mengenüberdeckung: Algorithmusanalyse



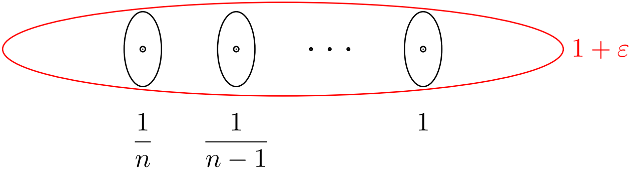
Optimal: $1+\varepsilon$.

Mengenüberdeckung: Algorithmusanalyse

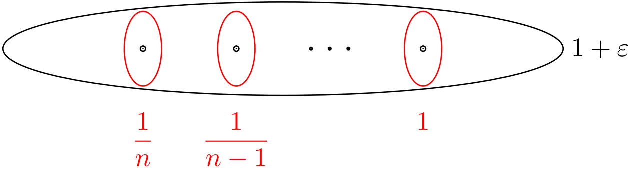


Optimal: $1 + \varepsilon$.

Mengenüberdeckung: Algorithmusanalyse



Optimal: $1+\varepsilon$.



Greedy: H_n. $\varepsilon \to 0$ liefert untere Abschätzung für relative Güte.

Outline

Graphenschnitte: Einführung

Optimierungsproblem

Gegeben sei ein ungerichteter Graph $G = (V, E)$. Finde einen maximalen Schnitt, d. h. eine Zerlegung der Knotenmenge in zwei disjunkte Teilmengen A und B mit $V = A \dot{\cup} B$, sodass die Anzahl der Kanten, die zwischen den Mengen verlaufen, maximal ist.

Bemerkung

Einen *minimalen* Schnitt kann man in polynomieller Zeit finden (Max-Flow Min-Cut Theorem, Ford-Fulkerson).

Graphenschnitte: Einführung

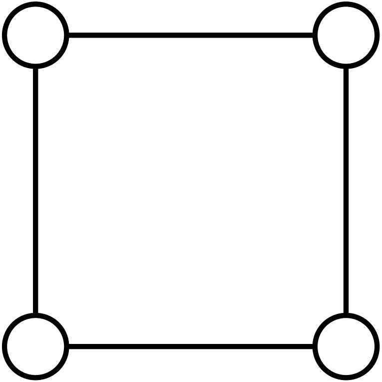
Optimierungsproblem

Gegeben sei ein ungerichteter Graph $G = (V, E)$. Finde einen maximalen Schnitt, d. h. eine Zerlegung der Knotenmenge in zwei disjunkte Teilmengen A und B mit $V = A \dot{\cup} B$, sodass die Anzahl der Kanten, die zwischen den Mengen verlaufen, maximal ist.

Bemerkung

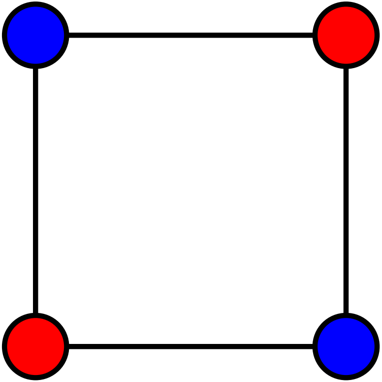
Einen *minimalen* Schnitt kann man in polynomieller Zeit finden (Max-Flow Min-Cut Theorem, Ford-Fulkerson).

Graphenschnitte: Beispiel



Optimale Lösung: Alle 4 Kanten verlaufen zwischen A und B.

Graphenschnitte: Beispiel



Optimale Lösung: Alle 4 Kanten verlaufen zwischen *A* und *B*.

Graphenschnitte: Greedy-Algorithmus

Algorithmus

1. Wähle zwei Start-Knoten: $A \leftarrow \{v_1\}$ und $B \leftarrow \{v_2\}$.
2. Für einen nicht zugeteilte Knoten $v \in V \setminus (A \cup B)$: Falls $\deg(v, A) > \deg(v, B)$: Füge v zu B hinzu, sonst zu A.

Satz

Der Greedy-Algorithmus hat eine relative Güte von 2.

Graphenschnitte: Greedy-Algorithmus

Algorithmus

1. Wähle zwei Start-Knoten: $A \leftarrow \{v_1\}$ und $B \leftarrow \{v_2\}$.
2. Für einen nicht zugeteilte Knoten $v \in V \setminus (A \cup B)$: Falls $\deg(v, A) > \deg(v, B)$: Füge v zu B hinzu, sonst zu A.

Satz

Der Greedy-Algorithmus hat eine relative Güte von 2.

Graphenschnitte: Greedy-Algorithmus

Algorithmus

1. Wähle zwei Start-Knoten: $A \leftarrow \{v_1\}$ und $B \leftarrow \{v_2\}$.
2. Für einen nicht zugeteilte Knoten $v \in V \setminus (A \cup B)$: Falls $\deg(v, A) > \deg(v, B)$: Füge v zu B hinzu, sonst zu A.

Satz

Der Greedy-Algorithmus hat eine relative Güte von 2.

Graphenschnitte: Analyse von Greedy

Beweis

- Seien $e(A)$, $e(B)$ die Anzahl der Kanten innerhalb von A bzw. B und $\deg(A, B)$ die Kanten zwischen A und B.
- Zu Beginn gilt: $e(A) = e(B) = 0$ und $\deg(A, B) = 0$ oder 1, also $e(A) + e(B) \leq \deg(A, B)$.
- Nach jedem Schritt gilt nach Wahl der Menge für v: $e(A) + e(B) \leq \deg(A, B)$
- Es ist OPT $\leq |E|$ und $e(A) + e(B) + \deg(A, B) = |E|$.
- Somit ist $\deg(A, B) \geq \frac{1}{2}$OPT.

Graphenschnitte: Analyse von Greedy

Beweis

- Seien $e(A)$, $e(B)$ die Anzahl der Kanten innerhalb von A bzw. B und $\deg(A, B)$ die Kanten zwischen A und B.
- Zu Beginn gilt: $e(A) = e(B) = 0$ und $\deg(A, B) = 0$ oder 1, also $e(A) + e(B) \leq \deg(A, B)$.
- Nach jedem Schritt gilt nach Wahl der Menge für v: $e(A) + e(B) \leq \deg(A, B)$
- Es ist OPT $\leq |E|$ und $e(A) + e(B) + \deg(A, B) = |E|$.
- Somit ist $\deg(A, B) \geq \frac{1}{2}$OPT.

Graphenschnitte: Analyse von Greedy

Beweis

- Seien $e(A)$, $e(B)$ die Anzahl der Kanten innerhalb von A bzw. B und $\deg(A, B)$ die Kanten zwischen A und B.
- Zu Beginn gilt: $e(A) = e(B) = 0$ und $\deg(A, B) = 0$ oder 1, also $e(A) + e(B) \leq \deg(A, B)$.
- Nach jedem Schritt gilt nach Wahl der Menge für v: $e(A) + e(B) \leq \deg(A, B)$
- Es ist $\mathrm{OPT} \leq |E|$ und $e(A) + e(B) + \deg(A, B) = |E|$.
- Somit ist $\deg(A, B) \geq \frac{1}{2}\mathrm{OPT}$.

Graphenschnitte: Analyse von Greedy

Beweis

- Seien $e(A)$, $e(B)$ die Anzahl der Kanten innerhalb von A bzw. B und $\deg(A, B)$ die Kanten zwischen A und B.
- Zu Beginn gilt: $e(A) = e(B) = 0$ und $\deg(A, B) = 0$ oder 1, also $e(A) + e(B) \leq \deg(A, B)$.
- Nach jedem Schritt gilt nach Wahl der Menge für v: $e(A) + e(B) \leq \deg(A, B)$
- Es ist OPT $\leq |E|$ und $e(A) + e(B) + \deg(A, B) = |E|$.
- Somit ist $\deg(A, B) \geq \frac{1}{2}$OPT.

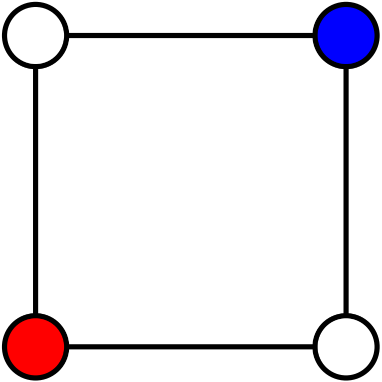
Graphenschnitte: Analyse von Greedy

Beweis

- Seien $e(A)$, $e(B)$ die Anzahl der Kanten innerhalb von A bzw. B und $\deg(A, B)$ die Kanten zwischen A und B.
- Zu Beginn gilt: $e(A) = e(B) = 0$ und $\deg(A, B) = 0$ oder 1, also $e(A) + e(B) \leq \deg(A, B)$.
- Nach jedem Schritt gilt nach Wahl der Menge für v: $e(A) + e(B) \leq \deg(A, B)$
- Es ist $\mathrm{OPT} \leq |E|$ und $e(A) + e(B) + \deg(A, B) = |E|$.
- Somit ist $\deg(A, B) \geq \frac{1}{2}\mathrm{OPT}$.

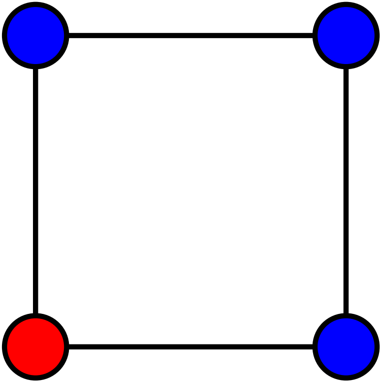
Graphenschnitte: Analyse von Greedy

Nach Wahl von v_1 und v_2:



Am Ende: Schnitt hat Größe 2, also wird relative Güte angenommen.

Graphenschnitte: Analyse von Greedy



Am Ende: Schnitt hat Größe 2, also wird relative Güte angenommen. $\square$

Graphenschnitte: Lokale Suche

Algorithmus

1. Starte mit einem beliebigen Schnitt.
2. Sortiere Knoten um, falls dies den Schnitt verbessert.

Analyse

- Terminiert, da in jedem Schritt der Schnitt um mindestens 1 steigt (und Schnitt beschränkt).
- Hat ebenfalls Güte 2, da jeder Knoten mindestens soviele Kanten zwischen den Mengen A, B wie innerhalb „seiner“ Menge hat.

Graphenschnitte: Lokale Suche

Algorithmus

1. Starte mit einem beliebigen Schnitt.
2. Sortiere Knoten um, falls dies den Schnitt verbessert.

Analyse

- Terminiert, da in jedem Schritt der Schnitt um mindestens 1 steigt (und Schnitt beschränkt).
- Hat ebenfalls Güte 2, da jeder Knoten mindestens soviele Kanten zwischen den Mengen A, B wie innerhalb „seiner“ Menge hat.

Graphenschnitte: Lokale Suche

Algorithmus

1. Starte mit einem beliebigen Schnitt.
2. Sortiere Knoten um, falls dies den Schnitt verbessert.

Analyse

- Terminiert, da in jedem Schritt der Schnitt um mindestens 1 steigt (und Schnitt beschränkt).
- Hat ebenfalls Güte 2, da jeder Knoten mindestens soviele Kanten zwischen den Mengen A, B wie innerhalb „seiner“ Menge hat.

Graphenschnitte: Lokale Suche

Algorithmus

1. Starte mit einem beliebigen Schnitt.
2. Sortiere Knoten um, falls dies den Schnitt verbessert.

Analyse

- Terminiert, da in jedem Schritt der Schnitt um mindestens 1 steigt (und Schnitt beschränkt).
- Hat ebenfalls Güte 2, da jeder Knoten mindestens soviele Kanten zwischen den Mengen A, B wie innerhalb „seiner“ Menge hat.

Graphenschnitte: Anmerkungen

Anmerkungen

- Anwendung in der Datenanalyse: Daten so Clustern, dass „Abstand“ der Cluster möglichst groß.
- Lange (bis Mitte/Ende 90er) kein besserer Algorithmus bekannt.
- Approximierbar mit Güte von 1.1383 (Vortrag 14 von Daniel).
- Es gibt einen exakten $\mathcal{P}$-Algorithmus für planare Graphen.

Graphenschnitte: Anmerkungen

Anmerkungen

- Anwendung in der Datenanalyse: Daten so Clustern, dass „Abstand“ der Cluster möglichst groß.
- Lange (bis Mitte/Ende 90er) kein besserer Algorithmus bekannt.
- Approximierbar mit Güte von 1.1383 (Vortrag 14 von Daniel).
- Es gibt einen exakten $\mathcal{P}$-Algorithmus für planare Graphen.

Graphenschnitte: Anmerkungen

Anmerkungen

- Anwendung in der Datenanalyse: Daten so Clustern, dass „Abstand" der Cluster möglichst groß.
- Lange (bis Mitte/Ende 90er) kein besserer Algorithmus bekannt.
- Approximierbar mit Güte von 1.1383 (Vortrag 14 von Daniel).
- Es gibt einen exakten $\mathcal{P}$-Algorithmus für planare Graphen.

Graphenschnitte: Anmerkungen

Anmerkungen

- Anwendung in der Datenanalyse: Daten so Clustern, dass „Abstand“ der Cluster möglichst groß.
- Lange (bis Mitte/Ende 90er) kein besserer Algorithmus bekannt.
- Approximierbar mit Güte von 1.1383 (Vortrag 14 von Daniel).
- Es gibt einen exakten $\mathcal{P}$-Algorithmus für planare Graphen.