Algorithmen für NP-harte Probleme

Tim Baumann, http://timbaumann.info/uni-spicker

Dies ist eine Zusammenfassung zur gleichnamigen Vorlesung von Professor Dr. Torben Hagerup im Sommersemester 2017.

Def. Ein **Optimierungsproblem** ist ein Tupel $(\mathcal{X}, \mathcal{F}, Z, \odot)$ wobei

- $\mathcal{X}$ eine Menge von **Instanzen**,
- $\mathcal{F}$ eine Abbildung ist, welche jeder Instanz x eine Menge $\mathcal{F}(x)$ von **möglichen Lösungen** zuordnet,
- Z eine reellwertige Abbildung (die **Zielfunktion**) ist, die jedem $x \in \mathcal{X}$ und $y \in \mathcal{F}(x)$ einen *Zielwert* zuordnet und
- $\odot \in \{\min, \max\}$ angibt, ob die Zielfunktion minimiert oder maximiert werden soll.

Def. Eine **optimale Lösung** eines Optimierungsproblems $(\mathcal{X}, \mathcal{F}, Z, \odot)$ zu einer Instanz $x \in \mathcal{X}$ ist ein $y \in \mathcal{F}(x)$ mit

$$Z(x, y) = \odot_{y \in \mathcal{F}(x)} Z(x, y) =: \operatorname{Opt}(x).$$

Def. Ein Algorithmus **löst** ein Optimierungsproblem $(\mathcal{X}, \mathcal{F}, Z, \odot)$, falls er für jedes $x \in \mathcal{X}$

- eine optimale Lösung $y \in \mathcal{F}(x)$ berechnet, falls solch eine existiert,
- „unmöglich" ausgibt, falls keine Lösung existiert oder
- „möglich, aber keine optimale Lösung" sonst.

Def. NPO ist die Klasse aller Opt.-Probleme $(\mathcal{X}, \mathcal{F}, Z, \odot)$ mit

- $\mathcal{X} \in P$
- Es gibt ein Polynom p, sodass für alle $x \in X$
  - $|y| \leq p(|x|)$ für alle $y \in \mathcal{F}(x)$ und
  - für alle Wörter w der Länge $|w| \leq p(|x|)$ in polynomieller Zeit (in $|x|$) entscheidbar ist, ob $w \in \mathcal{F}(x)$.
- Die Funktion Z ist in polynomieller Zeit berechenbar.

Def. PO $\subseteq$ NPO ist die Subklasse für die ein Lösungsalgorithmus existiert, der in Polynomialzeit läuft.

Beob. PO $=$ NPO $\implies$ P $=$ NP

Def. Sei $\mathcal{P} = (\mathcal{X}, \mathcal{F}, Z, \min)$ ein Optimierungsproblem.

- Das zugeh. **Auswertungsproblem** $\mathcal{P}_E$ ist: Gegeben $x \in \mathcal{X}$,
  - berechne $\operatorname{Opt}(x)$, falls x eine optimale Lösung besitzt,
  - berechne $\inf \mathcal{F}(x) \in \mathbb{R} \cup \{-\infty\}$, falls es Lösungen gibt, aber keine optimale
  - oder gib „unmöglich" aus, falls keine Lösung existiert.
- Das zugeh. **Entscheidungsproblem** $\mathcal{P}_D$ ist: Gegeben $x \in \mathcal{X}$ und $k \in \mathbb{Q}$, gibt es eine Lösung $y \in \mathcal{F}(x)$ mit $Z(x, y) \leq k$?

Lem. $\mathcal{P} \in$ NPO $\implies \mathcal{P}_D \in$ NP

Def.
- Ein Entscheidungsproblem $\mathcal{P}_1$ ist (in Polynomialzeit) auf ein Entscheidungsproblem $\mathcal{P}_2$ **many-to-one-reduzierbar** (notiert $\mathcal{P}_1 \leq_m \mathcal{P}_2$) falls eine (in Polynomialzeit) berechenbare Funktion $f : \{\text{Instanzen von } \mathcal{P}_1\} \to \{\text{Instanzen von } \mathcal{P}_2\}$ existiert, sodass die Antwort auf eine Instanz x von $\mathcal{P}_1$ gleich der Antwort auf die Instanz $f(x)$ von $\mathcal{P}_2$ ist.
- Ein Problem $\mathcal{P}_1$ ist (in Polynomialzeit) auf ein Problem $\mathcal{P}_2$ **Turing-reduzierbar** (notiert $\mathcal{P}_1 \leq_T \mathcal{P}_2$) falls ein Algorithmus existiert, der unter Verwendung eines Orakels für $\mathcal{P}_2$ das Problem $\mathcal{P}_1$ (in Polynomialzeit) löst.

Beob. $\mathcal{P}_1 \leq_m \mathcal{P}_2 \implies \mathcal{P}_1 \leq_T \mathcal{P}_2$

Beob. Für $\mathcal{P} \in$ NPO gilt $\mathcal{P}_D \leq_T \mathcal{P}_E \leq_T \mathcal{P}$.

Satz. Habe $\mathcal{P} = (\mathcal{X}, \mathcal{F}, Z, \odot) \in$ NPO eine Zielfunktion mit Werten in den ganzen Zahlen.

- Es gilt $\mathcal{P}_D \equiv_T \mathcal{P}_E$.
- Angenommen, $\mathcal{P}_D$ ist NP-vollständig. Dann gilt $\mathcal{P} \equiv_T \mathcal{P}_D$.

Def. Ein Optimierungsproblem $\mathcal{P}$ heißt **NP-hart**, falls $\mathcal{P}' \leq_T \mathcal{P}$ für jedes Entscheidungsproblem $\mathcal{P}'$ in NP.

Beob. $\mathcal{P} \in$ NPO, $\mathcal{P}$ NP-vollständig $\implies \mathcal{P}$ NP-hart

Die Gierige Strategie

Problem (Cabin Manager's Problem)**.** MIS auf Intervallgraphen

Algorithmus (**Greedy MIS für Intervallgraphen**)**.** Beginne mit $C := \emptyset$, füge dann wiederholt gierig das vom aktuellen C unabhängige Intervall mit dem kleinsten Endpunkt zu C hinzu, bis es kein solches Intervall mehr gibt.

Satz. Dieser Algorithmus berechnet tatsächlich ein MIS.

Algorithmus (**Greedy Minimum Makespan Scheduling**)**.** Gehe die Jobs in nach Dauer absteigender Reihenfolge durch, weise jeden Job dem Arbeiter zu, der bisher am wenigsten ausgelastet ist.

Satz. Die Lösung, die der Alg. liefert, ist höchstens um den Faktor

$$4/3 - 1/3p$$

schlechter als eine optimale Lösung.

Beweisskizze. Sei t die Länge des letzten Jobs des am längsten beschäftigten Arbeiters und z^* die minimale Gesamtdauer.

- Falls $t > z^*/3$, so hat der Alg. sogar eine optimale Lsg gefunden.
- Falls $t \leq z^*/3$, so folgt die Behauptung durch geeign. Abschätzen.

Algorithmus (**Greedy Knapsack Packing**)**.** Gehe die Sachen absteigend nach ihrem Nutzen-Kosten-Verhältnis v_i/w_i durch und packe jede Sache ein, die noch in den Rucksack passt. Sei z der Gesamtnutzen des so zusammengestellten Sets. Falls eine Sache mit Nutzen $v_j > z$ (und $w_j \leq W$) nicht eingepackt wurde, so räume den Rucksack wieder aus und packe als einziges diese Sache ein.

Satz. Der Gesamtnutzen der durch den Algorithmus erhaltenen Lösung ist mindestens halb so groß wie der Gesamtnutzen einer optimalen Lösung.

Algorithmus (**Greedy Minimum Set Cover**)**.** Beginne mit $\mathcal{C} := \emptyset$, füge dann immer ein $T \in \mathcal{C}_0$ zu $\mathcal{C}$ hinzu, welches

$$T \cap \left(\bigcup_{S \in \mathcal{C}_0} S \setminus \bigcup_{S \in \mathcal{C}} S\right)$$

maximiert, bis $\bigcup_{S \in \mathcal{C}_0} S = \bigcup_{S \in \mathcal{C}} S$.

Satz. Sei $n := \max_{S \in \mathcal{C}_0} |S|$. Die vom Greedy-Algorithmus berechnete Lösung ist maximal um den Faktor $H_n := \sum_{j=1}^{n} 1/j$ schlechter als die optimale Lösung.

Bem. Es gibt keinen (einfachen) Greedy-Algorithmus, der das Minimum-Vertex-Coloring-Problem in guten Schranken löst.

Approximationsalgorithmen

Def. Ein **Approximationsalgorithmus** für ein Optimierungsproblem $(\mathcal{X}, \mathcal{F}, Z, \odot)$ ist ein Algorithmus, der für jedes $x \in \mathcal{X}$ eine zulässige Lösung $y \in \mathcal{F}(x)$ produziert.

Def. Sei $(\mathcal{X}, \mathcal{F}, Z, \odot)$ ein Optimierungsproblem und $x \in \mathcal{X}$ eine Instanz, für die $\operatorname{Opt}(x)$ existiert. Der **absolute Fehler** von $y \in \mathcal{F}(x)$ ist $|Z(x, y) - \operatorname{Opt}(x)|$.

Satz (**Vizings Algorithmus**, ▶)**.** Es gibt einen Algorithmus, der für jeden Graph $G = (V, E)$ eine Kantenfärbung mit höchstens $\Delta + 1$ Farben, wobei $\Delta := \max_{v \in V} \deg_G(v)$, berechnet.

Kor. Es gibt einen Polynomialzeit-Approximationsalg. für Minimum Edge Coloring mit Absolutfehler beschränkt durch 1.

Def. Sei $\mathcal{P} = (\mathcal{X}, \mathcal{F}, Z, \odot)$ ein Optimierungsproblem mit $Z \geq 0$. Der **relative Fehler** von $y \in \mathcal{F}(x)$ zu $x \in \mathcal{X}$ ist

$$\begin{cases} 0 & \text{falls } \odot = \min, Z(x,y) = \operatorname{Opt}(x) = 0, \\ (Z(x,y) - \operatorname{Opt}(x))/Z(x,y) & \text{falls } \odot = \min, Z(x,y) > 0, \\ (\operatorname{Opt}(x) - Z(x,y))/\operatorname{Opt}(x) & \text{falls } \odot = \max. \end{cases}$$

Bem. Der relative Fehler ist eine Zahl in $[0, 1]$. Eine Lösung ist genau dann optimal, falls ihr relativer Fehler $= 0$ ist.

Def. Ein **ϵ-Approximationsalgorithmus** ($\epsilon \in [0, 1]$) für $\mathcal{P}$ ist ein Algorithmus, der für jedes $x \in \mathcal{X}$ ein $y \in \mathcal{F}(x)$ mit relativem Fehler $\leq \epsilon$ berechnet. Das Problem $\mathcal{P}$ heißt **ϵ-approximierbar**, falls ein solcher Alg. mit polynomieller Laufzeit existiert.

Bspe.
- Minimum Makespan Scheduling ist (1/4)-approximierbar.
- Maximum Knapsack ist (1/2)-approximierbar.
- Minimum Set Cover ist $(\ln n/(1 + \ln n))$-approximierbar (bei Eingabegröße n).

Def. Sei $\mathcal{P} = (\mathcal{X}, \mathcal{F}, Z, \odot)$ ein Optimierungsproblem mit $Z \geq 0$. Das **Approximationsverhältnis** von $y \in \mathcal{F}(x)$ zu $x \in \mathcal{X}$ ist

$$\begin{cases} 1 & \text{falls } Z(x,y) = \operatorname{Opt}(x) = 0, \\ \operatorname{Opt}(x)/Z(x,y) \in [1, \infty] & \text{falls } \odot = \min, \operatorname{Opt}(x) > 0, \\ Z(x,y)/\operatorname{Opt}(x) \in [1, \infty] & \text{falls } \odot = \max, Z(x,y) > 0. \end{cases}$$

Bem. Das Approximationsverh. ist eine Zahl in $[1, \infty]$. Eine Lösung ist genau dann optimal, falls ihr Approximationsverh. $= 1$ ist.

Def. Ein Alg. heißt **r-Approximationsalgorithmus** ($r \in (1, \infty]$), falls er für jedes $x \in \mathcal{X}$ ein $y \in \mathcal{F}(x)$ mit Approximationsverhältnis $\leq r$ liefert. Das Problem $\mathcal{P}$ heißt **r-approximierbar**, falls ein solcher Algorithmus mit polynomieller Laufzeit existiert.

Bem. Für das Approximationsverhältnis r und den relativen Fehler ϵ von $y \in \mathcal{F}(x)$ gilt

$$r = 1/(1-\epsilon), \qquad \epsilon = 1 - 1/\epsilon.$$

Def. APX ist die Klasse aller Probleme in NPO, die r-Approximierbar für ein $r > 1$ sind.

Satz. Falls $\mathrm{P} \neq \mathrm{NP}$, so gilt Minimum TSP $\notin$ APX.

Beweisidee. Wäre Minimum TSP r-approximierbar, so könnte man diesen Algorithmus verwenden, um das NP-Problem, ob ein Graph einen Hamiltonweg besitzt, zu entscheiden.

Das Problem des Handelsreisenden

Erinnerung. Minimale Spannbäume für einen gewichteten ungerichteten Graphen können in polynomieller Zeit mit Kruskals oder mit Prims Algorithmus berechnet werden.

Satz. Minimum Δ-TSP ist 2-approximierbar.

Beweisskizze. Sei (V, c) eine Instanz und z^* die minimale Länge einer Tour. Berechne einen minimalen Spannbaum. Dessen Kanten haben eine Gesamtlänge von $\leq z^*$. Führe Tiefensuche im Spannbaum durch und liste jeden neu entdeckten Knoten auf. Die so erhaltene Tour hat (wegen der Dreiecksungleichung) Länge $\leq 2z^*$.

Def. Ein ungerichteter Multigraph heißt *Eulersch, falls er eine Eulertour besitzt, also eine Tour, die jede Kante nur ein Mal benutzt.*

Lem. Ein zshgder ungerichteter Multigraph ist genau dann Eulersch, wenn alle seine Knoten den Grad zwei haben. In dem Fall kann man eine Eulertour in Polynomialzeit finden.

Lem. Sei (V, c) eine Instanz des Minimum Δ-TSP. Aus jedem Eulerschen Multigraph auf der Knotenmenge V mit Gesamtkantengewicht C kann man eine TSP-Tour der Gesamtlänge $\leq C$ in Polynomialzeit berechnen.

Def. Sei $G = (V, E)$ ein unger. Graph. Ein **Matching** in G ist eine Teilmenge $E' \subseteq E$, sodass $e \cap e' = \emptyset$ für alle $e, e' \in E'$ mit $e \neq e'$. Ein Matching heißt **perfekt**, falls $V = \cup_{e \in E'} e$. Die *Kosten* eines Matchings bzgl. einer Kostenfunktion $c : E \to \mathbb{R}_{\geq 0}$ sind $\sum_{e \in E'} c(e)$.

Satz. Ein perfektes Matching maximaler Größe mit minimalen Kosten (unter den Matchings maximaler Größe) kann für einen Graphen G mit n Knoten in Zeit $n^{o(1)}$ berechnet werden.

Satz (Christofides). Minimum Δ-TSP ist 3/2-approximierbar.

Beweisskizze. Sei (V, c) eine Instanz und z^* die minimale Länge einer Tour. Berechne einen minimalen Spannbaum. Berechne ein perfektes Matching mit minimalen Kosten auf den Knoten des Stammbaums mit ungeradem Grad. Die Kosten dieses Matchings sind $\leq z^*/2$. Durch Hinzufügen der Kanten des Matchings zum Spannbaum erhalten wir einen Eulerschen Multigraphen mit Gesamtkosten $\leq 3/2z^*$. Aus diesem erhalten wir eine Tour der Länge $\leq 3/2z^*$.

Nochmal Minimum Vertex Coloring

Algorithmus (Greedy Vertex Coloring).
Wiederhole folgende Schritte, bis alle Knoten gefärbt sind:

1. Bestimme ein nicht-vergrößerbares IS I in G wie folgt: Setze $H := G$, $I := \emptyset$, dann führe folgende Schritte aus, solange $H \neq \emptyset$:
   (a) Wähle einen Knoten v minimalen Grades aus H aus.
   (b) Füge v zu I hinzu.
   (c) Lösche v und seine Nachbarknoten aus H.
2. Färbe alle Knoten in I in einer noch unbenutzten Farbe.
3. Lösche die Knoten in I aus G.

Satz. Greedy Vertex Coloring ist (für Graphen mit n Knoten) ein $\mathcal{O}(n/\log n)$-Approximationsalgorithmus.

Baumsuche

Strategie (Branch and Bound für Minimierungsprobleme). Organisiere den Suchraum als Baum, der zunächst nur eine Wurzel enthält, wobei mögliche Lösungen aus $\mathcal{F}(x)$ Blätter sind und „partielle Lösungen" (die zu einer möglichen Lösung erweitert werden können oder auch nicht) die Verzweigungen bilden. Es ist nicht praktikabel, den gesamten Baum zu durchsuchen. Darum mache folgendes: Beschrifte die Verzweigungen mit einer (kostengünstig) berechneten unteren Schranke für die Kosten einer Lösung, die Erweiterung der partiellen Lösung ist. Expandiere dann wiederholt eine Verzweigung im Baum, d. h. berechne seine Kindknoten und beschrifte sie mit einer unteren Schranke der Kosten. Verzweigungen, deren untere Schranke mindestens so groß ist wie die Kosten einer bisher gefundenen möglichen Lösung müssen nicht expandiert werden. Gibt es keine Verzweigung mehr, die expandiert werden muss, so ist die bisher gefundene mögliche Lösung mit minimalen Kosten eine optimale Lösung.

Bem. Der Algorithmus kann auch früher abgebrochen werden, etwa wenn die unteren Schranken nur etwas kleiner sind als die Kosten der besten bisher gefundenen Lösung und wenn man mit einer approximativen Lösung zufrieden ist.

Bsp. Für das TSP auf (V, E) sind partielle Lösungen Pfade $p = u_0 \cdots u_m$ beginnend bei einem Startknoten u_0. Eine untere Kostenschranke für Touren, die Erweiterung des Pfades p sind, ist

$$\begin{aligned} d := \; & \min\{c(u_0, v) \mid v \in V \setminus \{u_0, \cdots, u_k\}\} \\ & + \min\{c(u_k, v) \mid v \in V \setminus \{u_0, \cdots, u_k\}\} \\ & + \text{Summe der Kosten der } n-k-1 \text{ kostengünstigsten} \\ & \quad \text{Kanten zwischen Knoten in } \{u_0, \cdots, u_k\} \end{aligned}$$

Bem. In der Praxis schaffen Branch-and-Bound-Algorithmen (für NP-schwere Probleme) oft eine drastische Verkleinerung des Suchraums. Theoretisch haben sie jedoch für gewöhnlich exponentielle Laufzeit.

Notation. Für eine logische Formel F und eine Variable x sei $F|_{x=i}$ ($i \in \{0, 1\}$) die Formel, die man erhält, wenn man x durch i in F ersetzt und vereinfacht.

Satz. Eine Instanz F von 3-SAT kann in Zeit $\mathcal{O}(|F| \cdot \alpha_0^n)$ entschieden werden, wobei $\alpha_0 \approx 1.84$ und $|F|$ die Gesamtzahl der Literale in F ist.

> **function** DECIDE(F)
>   **if** F hat keine Clauses **then return** true
>   wähle eine Clause $l_1 \vee \cdots \vee l_k$ in F (mit $k \in \{0, 1, 2, 3\}$)
>   **for** $i := 1, \ldots, k$ **do**
>   **if** DECIDE($F|_{l_1=0,\ldots,l_{i-1}=0,l_i=1}$) **then return** true
>   **return** false

Satz. Instanzen von Minimum Vertex Cover mit n Knoten und m Kanten können in Zeit $\mathcal{O}(3^{n/2} \cdot m + n)$ gelöst werden.

Notation. Für einen Graphen $G = (V, E)$ und Knoten $W \subseteq V$ sei $G - W$ der Graph $(V', \{\{u, w\} \in E \mid u, w \in V'\})$ mit $V' := V \setminus W$, der durch Löschen von W entsteht.

> **function** COMPUTEMVC($G = (V, E)$)
>   **if** $G = \emptyset$ **then return** $\emptyset$
>   **if** G hat isolierten Knoten u **then**
>   **return** COMPUTEMVC($G - \{u\}$)
>   **if** G hat Knoten u vom Grad 1 **then**
>   sei v der Knoten mit $\{u, v\} \in E$
>   **return** $\{v\} \cup$ COMPUTEMVC($G - \{u, v\}$)
>   **if** G hat Dreieck mit Eckknoten u, v, w **then**
> $C_{u,v} := \{u, v\} \cup$ COMPUTEMVC($G - \{u, v\}$)
> $C_{u,w} := \{u, w\} \cup$ COMPUTEMVC($G - \{u, w\}$)
> $C_{v,w} := \{v, w\} \cup$ COMPUTEMVC($G - \{v, w\}$)
>   **return** kleinste der Überdeckungen $C_{u,v}$, $C_{u,w}$ und $C_{v,w}$
>   **if** G hat einfachen Pfad mit Knoten u, v, w und z **then**
> $C_{u,w} := \{u, w\} \cup$ COMPUTEMVC($G - \{u, w\}$)
> $C_{v,w} := \{v, w\} \cup$ COMPUTEMVC($G - \{v, w\}$)
> $C_{v,z} := \{v, z\} \cup$ COMPUTEMVC($G - \{v, z\}$)
>   **return** kleinste der Überdeckungen $C_{u,w}$, $C_{v,w}$ und $C_{v,z}$

Bem. In der Umgebung jedes Knoten eines Graphen tritt einer der vier Fälle auf. Es kann daher in konstanter Zeit einer der vier Fälle ausgewählt werden. (Es muss nicht darauf geachtet werden, die Fälle von oben nach unten durchzuarbeiten!)

Wir sagen, ein Fall *verzweigt gemäß* $A_1, \ldots, A_k$, falls er den Algorithmus rekursiv mit Argumenten $G - A_1, \ldots, G - A_k$ aufruft. Die Multimenge $\{|A_1|, \ldots, |A_k|\}$ heißt zugehörige *Verzweigungsmultimenge.* Für eine solche Multimenge $\{a_1, \ldots, a_k\}$ setzen wir

$$\alpha(a_1, \ldots, a_k) := \max\{x \geq 1 \mid x^{-a_1} + \ldots + x^{-a_k} \geq 1\}.$$

Schaffen wir es, einen Algorithmus mit m Fällen mit Verzweigungsmultimengen $A_1, \ldots, A_m$ anzugeben (sodass in konstanter Zeit entschieden werden kann, welcher Fall vorliegt), so läuft der Algorithmus in Zeit $\mathcal{O}(\alpha_0^n \cdot m + n)$, wobei $\max\{\alpha(A_i) \mid i = 1, \ldots, m\}$

Satz. Instanzen von Minimum Vertex Cover mit n Knoten und m Kanten können in Zeit $\mathcal{O}(\alpha_0^n \cdot m + n)$ gelöst werden, wobei $\alpha_0 \approx 1{,}325$ die Wurzel von $x^3 - x - 1$ ist.

Dynamische Programmierung

Satz. Sei $P = (v_1, \ldots, v_n, w_1, \ldots, w_n)$ eine Instanz von Maximum Integer Knapsack. Setze $V := v_1 + \ldots + v_n$. Dann kann P in Zeit $\mathcal{O}(nV)$ gelöst werden.

Algorithmus (Knapsack mit dynam. Programmierung). Löse mit dyn. Progr. die Unterprobleme $(P_{j,v})_{1 \le j \le n, 1 \le v \le V}$ mit

$$P_{j,v} := \text{ finde } S \subset \{1, \ldots, j\} \text{ mit } \sum_{i \in S} v_i = v \text{ und } \sum_{i \in S} w_i \text{ minimal unter allen solchen Teilmengen!}$$

Die Lösung ergibt sich aus den Lösungen von $P_{n,1}, \ldots, P_{n,V}$.

Bem. Die Laufzeit ist **pseudopolynomiell**: Sie hängt polynomiell von den in der Eingabe enthaltenen Zahlen ab.

Satz. Instanzen P von Minimum Bin Packing mit n Objekten in r versch. Größen können in Zeit $\mathcal{O}(n^{2r+2})$ gelöst werden.

Algorithmus. Seien $s_1, \ldots, s_r$ die verschiedenen Größen. Wir nennen einen Vektor $A = (a_1, \ldots, a_r) \in \mathbb{N}^r$ einen *Packungstyp*, falls $\sum_{i=1}^r a_i s_i \le 1$. Sei $\{A_1, \ldots, A_t\}$ die Menge aller Packungstypen. Löse mit dyn. Programmierung die Unterprobleme

$$P_{j,B} := \text{ finde } f : \{1, \ldots, j\} \to \mathbb{N} \text{ mit } \sum_{i=1}^j f(i) \cdot A_i = B \text{ und } \sum_{i=1}^j f(i) \text{ ist minimal unter all solchen } f.$$

mit $1 \le j \le t$ und $B \in \{1, \ldots, n\}^r$ mit $B \le P$. Das ursprüngliche Problem ist $P_{t,P}$.

Polynomialzeit-Approximationsschemata

Def. Ein **Polynomialzeit-Approximationsschema** (PTAS) für $\mathcal{P} \in \mathrm{NPO}$ ist ein Algorithmus, der für jede Instanz x von $\mathcal{P}$ und $\epsilon > 0$ eine mögliche Lösung in $\mathcal{F}(x)$ mit relativem Fehler $\le \epsilon$ liefert und dessen Laufzeit für jedes fixe $\epsilon > 0$ polynomiell in $|x|$ ist.

Def. PTAS $:= \{\mathcal{P} \in \mathrm{NPO} \mid \mathcal{P} \text{ hat ein PTAS}\}$

Bem. PO $\subseteq$ PTAS $\subseteq$ APX

Satz. Es gibt einen Algorithmus, der für jedes $\epsilon > 0$ und jede Instanz $(v_1, \ldots, v_n, w_1, \ldots, w_n, W)$ von Maximum Knapsack in Zeit $\mathcal{O}(n^3/\epsilon)$ eine ϵ-approximative Lösung liefert.

Algorithmus. 1. Setze $K := \epsilon V / n^2$, wobei $V := v_1 + \ldots + v_n$.

2. Löse die Instanz $(\lfloor v_1/K \rfloor, \ldots, \lfloor v_n/K \rfloor, w_1, \ldots, w_n, W)$ von Maximum *Integer* Knapsack mit dem Algorithmus basierend auf dynamischer Programmierung.
3. Die Lösung dieses geänderten Problems ist eine ϵ-Approximation des ursprünglichen Problems.

Lem. Es gibt einen Algorithmus, der für jede Instanz von *Minimum Bin Packing* mit n Objekten der Größe $\ge \delta$ eine Packung der Objekte in $(1 + \delta)z^* + 1$ Behälter in Zeit $\mathcal{O}(n^{2/\delta^2 + 2})$ berechnet, wobei z^* die optimale Zahl der Behälter ist.

Satz. Es gibt einen Algorithmus, der für jede Instanz von *Minimum Bin Packing* mit n Objekten eine Packung der Objekte in $(1 + \delta)z^* + 1$ Behälter in Zeit $\mathcal{O}(n^{8/\delta^2 + 2})$ berechnet.

Def. Ein **asymptot. Polynomialzeit-Approximationsschema** (APTAS) für $(\mathcal{X}, \mathcal{F}, Z, \odot)$ ist ein Algorithmus, der für alle $x \in \mathcal{X}$ und $\epsilon > 0$ eine mögliche Lösung $y \in \mathcal{F}(x)$ mit

$$|Z(x, y) - \mathrm{Opt}(x)| \le \epsilon \cdot \max\{Z(x, y), \mathrm{Opt}(x)\} + K$$

für eine Konstante K berechnet und dessen Laufzeit für alle festen ϵ polynomiell in $|x|$ ist.

Def. Ein (asympt.) **Voll-Polynomialzeit-Approx'schema** ((A)FPTAS) für $\mathcal{P}$ ist ein (A)PTAS für $\mathcal{P}$, dessen Laufzeit für die Instanz (x, ϵ) durch ein Polynom in $|x|$ und in $1/\epsilon$ beschränkt ist.

Def. (A)(F)PTAS $:= \{\mathcal{P} \in \mathrm{NPO} \mid \mathcal{P} \text{ hat ein (A)(F)PTAS}\}$

Bspe. • Maximum Knapsack $\in$ FPTAS

- Wir haben gezeigt: Minimum Bin Packing $\in$ APTAS
- Man kann zeigen: Minimum Bin Packing $\in$ AFPTAS

Parametrisierung

Vorgehen. Füge einen weiteren Parameter (zusätzlich zu den Größenparameter) für Probleminstanzen ein, suche nach einem Algorithmus, dessen Laufzeit wesentlich von diesem Parameter abhängt, sodass Instanzen, die einen kleinen Wert für den Parameter haben, in akzeptabler Zeit gelöst werden können.

Satz. Es gibt einen Algorithmus, der gegeben einen Graphen G mit n Ecken und m Kanten und ein $k \ge 0$ in Zeit $\mathcal{O}(n \cdot 1{,}325^k + m)$ ein Minimum Vertex Cover der Größe $\le k$ berechnet, falls ein solches existiert (falls nicht, so soll der Algorithmus „keine Lösung" zurückgeben).

Idee. Verwende den rekursiven Baumsuche-Algorithmus für Minimum-Vertex-Cover, aber breche die Rekursion ab, wenn das sich in Konstruktion befindende Cover Größe $> k$ erreicht. Außerdem optimiere rekursive Aufrufe dadurch, dass der Graph nicht kopiert für den Aufruf kopiert wird.

Satz. Es gibt einen Algorithmus, der gegeben einen Graphen G mit n Ecken und m Kanten und ein $k \ge 0$ in Zeit $\mathcal{O}(k^2 \cdot 1{,}325^k + n + m)$ ein Minimum Vertex Cover der Größe $\le k$ berechnet, falls ein solches existiert.

Idee. Füge jeden Knoten mit Grad $\ge k$ zum Cover hinzu (da jedes Cover der Größe $\le k$ diese enthalten muss). Wir können somit annehmen, dass der Maximalgrad in $G \le k$ ist. Lösche alle isolierten Knoten aus G. Es gilt: Falls G nun mehr als k^2 Kanten oder mehr als $k^2 + k$ Ecken hat, so besitzt G kein Vertex Cover der Größe $\le k$ und wir können „keine Lösung" ausgeben. Ansonsten verwende den Algorithmus vom letzten Satz.

Bem. Wir haben damit die Probleminstanz auf einen kleineren Kern reduziert. Diese Technik heißt *kernelization*.

Lem. Für eine Formel F in konj. NF, in der jede Variable, die in positiver wie negativer Form vorkommt, genau zwei mal vorkommt, kann in Zeit $\mathcal{O}(|F|)$ eine Zuweisung von Variablen gefunden werden, die die Anzahl der erfüllten Clauses maximiert.

Satz. Es gibt einen Algorithmus, der gegeben einer Formel in konj. NF und $k \in \mathbb{N}$ in Zeit $\mathcal{O}(k^2 \phi^k + |F|)$, wobei $\phi = (1 + \sqrt{5})/2$, eine Zuweisung der Variablen in F berechnet, sodass k Clauses erfüllt sind, oder entscheidet, dass es keine solche Zuweisung gibt.

Folgender Algorithmus entscheidet bloß, ob es eine solche Zuweisung gibt, man kann ihn aber so abändern, dass er auch eine Zuweisung berechnet:

function $\textsc{DecideMaxSAT}(F, k)$
  entferne überflüssige Literale aus allen Clauses
  $m :=$ Anzahl von Clauses in F
  if $m < k$ **then return** false
  if $k \le \lfloor m/2 \rfloor$ **then return** true
  $F_L :=$ Konjunktion der *long Clauses* in F mit $\ge k$ Literalen
  $F_S :=$ Konjunktion der *short Clauses* in F mit $< k$ Literalen
  $m_L :=$ Anzahl Clauses in F_L **return** $\textsc{Search}(F_S, k - m_L)$
function $\textsc{Search}(F', j)$

if $j \leq 0$ **then return** true
if F' hat weniger als j Clauses **then return** false
if keine Variable tritt positiv und negativ in F' auf **then**
  return true
Wähle unter den Variablen mit positiv und negativ auftreten, eine Variable x, die am öftesten auftritt
$m_0 :=$ Anzahl der negativen Vorkommen von x
$m_1 :=$ Anzahl der positiven Vorkommen von x
if $m_0 = m_1 = 0$ **then**
  $k' :=$ max. Anzahl von erfüllb. Clauses in F' (siehe Lem.)
  return $k' \geq j$
else
  if $\textsc{Search}(F'|_{x=0}, j - m_0)$ **then return** true
  return $\textsc{Seach}(F'|_{x=1}, j - m_1)$

Mit einer etwas einfacheren Search-Prozedur kann man schon zeigen:

Satz. Es gibt einen Algorithmus, der gegeben einer Formel in konj. NF und $k \in \mathbb{N}$ in Zeit $\mathcal{O}(k^2 2^k + |F|)$ eine Zuweisung der Variablen in F berechnet, sodass k Clauses erfüllt sind, oder entscheidet, dass es keine solche Zuweisung gibt.

Baumbreite

Bem. Auf Graphen, die Bäume sind, können folgende Probleme in polynomieller Zeit gelöst werden:

- Maximum Independent Set (bzw. Minimum Vertex Cover)
- Minimum Dominating Set

Def.
- Ein **k-Baum** ist ein unger. Graph, der aus einer k-Clique durch wiederholtes Anwenden der folgenden Operation entsteht: Füge einen neuen Knoten mit Kanten zu den Knoten einer bestehenden k-Clique zum Graphen hinzu.
- Ein **partieller k-Baum** ist ein Subgraph eines k-Baums.

Bspe. Ein 1-Baum ist ein gewöhnlicher Baum, ein 2-Baum ein „Baum von Dreiecken“.

Lem. Maximum Independent Set kann auf 2-Bäumen in linearer Zeit gelöst werden.

Lem. Ein k-Baum mit n Knoten enthält genau $\binom{k}{2} + (n-k)k = kn - \binom{k+1}{2}$ Kanten.

Def. Eine **Baumzerlegung** eines unger. Graphen $G = (V, E)$ ist ein Paar (T, B), wobei $T = (X, F)$ ein Baum ist und $B : X \to \mathcal{P}(V)$ jedem Knoten $x \in X$ seinen *Sack* $B(x) \subseteq V$ zuordnet, sodass

- $\cup_{x \in X} B(x) = V$
- $\forall \{u, v\} \in E : \exists x \in X : \{u, v\} \subseteq B(x)$
- Zus'hang: $\forall x, y, z \in X : y$ liegt auf Pfad zw. x und z in $T \implies B(x) \cap B(z) \subseteq B(y)$.

Die **Breite** einer Baumzerlegung (T, B) ist $\max_{x \in X} |B(x)| - 1$. Die **Baumbreite** $\mathrm{tw}(G)$ von G ist die kleinstmögliche Breite einer Baumzerlegung von G.

TODO: Wie kann man eine Baumzerlegung zur Berechnung eines MIS verwenden?

Def. Eine Baumzerlegung $((X, F), B)$ heißt **k-normal**, falls

- $|B(x)| = k + 1$ für alle $x \in X$ und
- $|B(x) \setminus B(y)| = 1$ für alle $\{x, y\} \in F$.

Lem. Jeder Graph der Breite $\leq k$ mit $\geq k + 1$ Knoten hat eine k-normale Baumzerlegung.

Lem. Sei $((X, F), B)$ eine Baumzerlegung von $G = (V, E)$ und $C \subseteq V$ die Knoten einer Clique in G. Dann $C \subseteq B(x)$ für ein $x \in X$.

Satz. Für jedes $k \in \mathbb{N}$ und jeden Graphen G sind äquivalent:

$$G \text{ ist ein } k\text{-partieller Graph} \iff G \text{ hat Baumbreite} \leq k$$

Lem. Jeder partielle k-Baum kann aus einem k-Baum durch Löschen von Kanten aus einem gewonnen werden.

Lem. Jeder unger. Graph der Baumbreite $\leq k$ hat $\leq kn$ Kanten.

Bem. Entscheiden, ob für ein Tuple (G, k) bestehend aus einem Graph G und $k \in \mathbb{N}$ der Graph G Baumbreite $\leq k$ besitzt, ist NP-vollständig.

Lem (Bodlaender und Kloks)**.** Für alle Konstanten $l \in \mathbb{N}$ und $m, n \in \mathbb{Z}$ kann das folgende Problem in Zeit $\mathcal{O}(m)$ gelöst werden: Gegeben eine Baumzerlegung mit m Knoten mit Breite $\leq l$ eines Graphen G mit n Knoten, berechne eine Baumzerlegung von G minimaler Breite mit $\leq n$ Knoten.

Lem. Der Graph G' entstehe aus G durch Zusammenziehen einer Kante. Dann gilt:

$$\mathrm{tw}(G') \leq \mathrm{tw}(G) \quad \text{und} \quad \mathrm{tw}(G) \leq \mathrm{tw}(G') + 1.$$

Lem. Der Graph G' entstehe aus G durch Zusammenziehen der Kanten eines Matchings. Dann gilt:

$$\mathrm{tw}(G') \leq \mathrm{tw}(G) \quad \text{und} \quad \mathrm{tw}(G) \leq 2\,\mathrm{tw}(G') + 1.$$

Außerdem kann für bel. $k, m \in \mathbb{N}$ aus einer Baumzerlegung von G' der Breite k eine Baumzerlegung von G der Breite $\leq 2k + 1$ in Zeit $\mathcal{O}(km)$ konstruiert werden.

Def. Ein Matching M in einem Graphen G heißt **maximal**, falls es kein Matching M' mit $M \subsetneq M'$ gibt.

Achtung (▶, ▶)**.** Jedes “maximum matching” ist ein “maximal matching”, aber nicht andersherum.

Bem (▶)*.* Maximale Matchings können gierig in linearer Zeit berechnet werden.

Satz (**Bodlaender**)**.** Sei $k \in \mathbb{N}_0$ konstant. Für jeden Graphen G der Baumbreite $\leq k$ mit n Knoten kann eine Baumzerl. minimaler Breite mit $\leq n$ Knoten in Zeit und Platz $\mathcal{O}(n)$ berechnet werden.

function $\textsc{TreeDecompose}(G = (V, E),\ k)$
  if $V = \emptyset$ **then return** (T, B) wobei $T = (\{t\}, \emptyset)$ und $B : t \mapsto \emptyset$
  $M :=$ ein maximales Matching M in G
  $G' = (U, E') := G$ mit den Kanten in M zusammengezogen, $\phi : V \to U$ die knotenidentifizierende surj. Abbildung
  $W := \{u \in U \mid |\phi^{-1}(u)| = 2\}$
  $L = (U, E_L) := G'$ mit genau so vielen Kanten bel. entfernt, dass jeder Knoten in $U \setminus W$ Grad $\leq k + 1$ hat
  $G'' := (U, E'')$ mit $E'' := E' \cup \{\{u, v\} \mid u, v \in W, |N_{u,v}| \geq k + 1\}$ wobei $N_{u,v} := \mathrm{Neighbours}_L(u) \cap \mathrm{Neighbours}_L(v) \cap (U \setminus W)$
  $A := \{u \in U \setminus W \mid \text{Nachbarn von } u \text{ in } G'' \text{ formen Clique in } G''\}$
  $G''' :=$ Subgraph von G'' mit Knotenmenge $U \setminus A$
  $(T = (X, F), B) := \textsc{TreeDecompose}(G''')$
  for $a \in A$ **do**
    finde $x \in X$ mit $\mathrm{Neighbours}_{G''}(a) \subseteq B(x)$
    füge zu T einen Blattknoten l_a mit Vaterknoten x hinzu
    setze $B(l_a) := \{a\} \cup \mathrm{Neighbours}_{G''}(a)$
  definiere $B' : X \to \mathcal{P}(V)$ durch $B'(x) := \phi^{-1}(B(x))$
  return mit der Schrumpfprozedur von Bodlaender und Kloks verkleinerte Baumzerlegung (T, B')

Bem. Man zeigt: Der Graph G''' hat $\leq (1 - a)n$ Knoten, wobei $a = 1/(k^2 + 2)$.

Planarität

Def. Eine **einfache Kurve** in $\mathbb{R}^n$ mit *Endpunkten* $a, b \in \mathbb{R}^n$ ist eine stetige Abb. $\gamma : [0,1] \to \mathbb{R}^n$ mit $\gamma(0) = a$, $\gamma(1) = b$ und $\gamma(s) \neq \gamma(t)$ für alle $s, t \in [0,1]$ mit $0 < |s-t| < 1$. Sie heißt **offen**, falls $a \neq b$, und **geschlossen**, falls $a = b$.

Def (▶). Eine **planare Einbettung** $\phi = ((p_v)_{v \in V}, (\gamma_e)_{e \in E})$ eines Graphen (V, E) ist geg. durch einen Punkt $p_v \in \mathbb{R}^2$ für jeden Knoten $v \in V$ und eine einfache Kurve $\gamma_{\{u,v\}}$ zwischn p_u und p_v für jede Kante $\{u, v\} \in E$, sodass für alle $e \neq e' \in E$ gilt:

$$\operatorname{im} \gamma_e \cap \gamma_{e'} = \{\gamma_e(0), \gamma_e(1)\} \cap \{\gamma_{e'}(0), \gamma_{e'}(1)\}.$$

Ein Graph heißt **planar**, falls er eine planare Einbettung besitzt.

Notation. $\phi(G) := \{p_v \mid v \in V\} \cup \bigcup_{e \in E} \operatorname{im} \gamma_e \subseteq \mathbb{R}^2$

Bem. Man kann zeigen (*Satz von Fáry*): Ist ein Graph (V, E) mit n Knoten planar, so gibt es eine planare Einbettung mit

- $p_v \in \{1, \ldots, n\}^2$ für alle $v \in V$ und
- $p_{\{u,v\}}(t) = (1-t)p_u + tp_v$ für alle $\{u, v\} \in E$.

Def. Die **Flächen** einer planaren Einbettung ϕ sind die Zusammenhangskomponenten von $\mathbb{R}^2 \setminus \phi(G)$. Alle Flächen bis auf eine sind dabei beschränkt. Die beschränkten Flächen heißen *innere* Flächen, die unbeschränkte *äußere* Fläche.

Satz (Euler-Formel, ▶). Sei G ein Graph mit n Knoten, m Kanten und c Zusammenhangskomponenten. Angenommen, eine planare Einbettung von G hat f Flächen. Dann gilt

$$n - m + f = c + 1.$$

Def. Der **Rand** einer Fläche F in einer Einb. ϕ von $G = (V, E)$ ist der Subgraph $G' = (V', E')$ von G mit $V' = \{v \in V \mid p_v \in \partial F\}$ und $E' = \{e \in E \mid \gamma_e \subseteq \partial F\}$, wobei $\partial F \subset \mathbb{R}^2$ der topologische Rand ist. Die Ecken in V' bzw. die Kanten in E' heißen **inzident** an F.

Lem (▶). Jeder planare Graph mit $n \geq 3$ Ecken hat höchstens $3n - 6$ Kanten.

Kor. Jeder planare Graph hat einen Knoten vom Grad ≤ 5.

Def. Ein Graph heißt **außenplanar**, falls er eine planare Einb. besitzt, bei der alle Knoten inzident zur äußeren Fläche sind.

Lem. Jeder außenplanare Graph mit ≥ 4 Knoten hat zwei nicht benachbarte Knoten mit Grad jeweils ≤ 2.

Lem. Jeder außenplanare Graph mit $n \geq 2$ Knoten hat höchstens $2n - 3$ Kanten.

Satz. Gegeben einen planaren Graphen G und ein $k \in \mathbb{N}$, kann eine unabh. Knotenmenge in Zeit $\mathcal{O}(6^k \cdot n)$ berechnet werden.

Idee. Verwende rekursive Suche und die Tatsache, dass es in einem planaren Graphen einen Knoten mit Grad ≤ 5 gibt.

Lem. Ein planarer Graph mit n Knoten, von denen $n_{\leq 2}$ Grad ≤ 2 besitzen, hat $< 3n - n_{\leq 2}$ Kanten.

Lem. Sei $G = (V, E)$ ein planarer Graph mit n Knoten und $S \subset V$ mit $S \leq n/28$. Angenommen, jeder Knoten $v \in \mathrm{Neighbours}_G(S) \setminus S$ hat mind. zwei Nachbarknoten, die nicht in $S \cup \mathrm{Neighbours}_G(S)$ liegen. Dann gibt es einen Knoten $v \in V \setminus (S \cup \mathrm{Neighbours}_G(S))$ mit $\deg(v) \leq 6$.

Satz. Für einen planaren Graph mit n Ecken und $k \in \{0, \ldots, \lfloor n/28 \rfloor\}$ kann eine dominierende Menge der Größe $\leq k$ in Zeit $\mathcal{O}(7^k \cdot n)$ berechnet werden, falls eine solche existiert.

function $\textsc{DominatingSet}(G = (V, E),\ S \subseteq V,\ k)$
  if $|S| > k$ **then return** "no solution"
  $W := S \cup \mathrm{Neighbours}_G(S)$
  $G' := (V, E')$ mit $E' := E \setminus \{\{u, v\} \mid u, v \in W\}$
  $G'' := (V'', E'')$ mit $V'' := V \setminus \{v \in W \mid \deg_G(v) \leq 1\}$
  finde $v \in V'' \setminus W$ mit $\deg_{G''}(v) \leq 6$
    (möglich dank vorhergehendem Lemma)
  $N := \{v\} \cup \mathrm{Neighbours}_{G''}(v)$
  for $w \in N$ **do** $S_w := \textsc{DominatingSet}(G'', S \cup \{w\}, k)$
  $C := \{S_w \mid w \in N\} \setminus \{\text{"no solution"}\}$
  if $C = \emptyset$ **then return** "no solution"
  return $\arg\min_{T \in C} |T|$

Probleme

Problem (Maximum Independent Set, MIS). Geg. einen unger. Graphen (V, E), berechne eine *unabh. Menge* $M \subseteq V$, d. h.

$$\forall\, v \in M \,:\, \forall\, w \in V \,:\, (v, w) \in E \implies w \notin M,$$

die maximale Größe $|M|$ unter allen unabhängigen Mengen besitzt.

Problem (Minimum Vertex Cover, MVC). Geg. einen unger. Graphen $G = (V, E)$, berechne eine *Knotenüberdeckung* C, d. h.

$$\forall\, v, w \in V \,:\, \{v, w\} \in E \implies v \in C \lor w \in C,$$

die minimale Größe $|C|$ unter allen Knotenüberdeckungen besitzt.

Bem. Für einen Graphen (V, E) und eine Teilmenge $S \subseteq V$ gilt: S ist eine unabhängige Menge $\iff$ $V \setminus S$ ist ein Vertex Cover
Die Probleme MIS und MVC sind damit äquivalent.

Def. Ein **Intervallmodell** eines Graphen $G = (V, E)$ ist eine Abbildung $\phi : E \to \{[a, b] \mid a, b \in \mathbb{Q}\}$, sodass

$$\forall\, v \neq w \in V \,:\quad (v, w) \in E \iff \phi(v) \cap \phi(w) \neq \emptyset.$$

Ein Graph heißt **Intervallgraph**, falls er ein Intervallmodell besitzt.

Problem (Minimum Makespan Scheduling). Seien $p, n \in \mathbb{N}$ und $l_1, \ldots, l_n \in \mathbb{R}_{>0}$ gegeben. Für $f : \{1, \ldots, n\} \to \{1, \ldots, p\}$ setze

$$t(f) := \max_{1 \leq i \leq p} \sum_{j \in f^{-1}(i)} l_j.$$

Berechne das f, für das $t(f)$ minimal wird!

Interpretation. p ist die Anzahl von *Arbeitern*, $l_1, \ldots, l_n$ sind die Längen von zu erledigenden *Jobs* und $t(f)$ ist die *Gesamtdauer* bei der durch f gegebenen Verteilung der Jobs auf die Arbeiter an.

Bem. MMS ist NP-hart, da das zugeh. Entscheidungsproblem Bin Packing bekannterweise NP-hart ist.

Problem (Maximum Knapsack). Seien $n \in \mathbb{N}$ und $v_1, \ldots, v_n$, $w_1, \ldots, w_n, W \in \mathbb{R}_{>0}$ gegeben. Die Menge der möglichen Lsgn sei

$$\mathcal{F} := \{S \subseteq \{1, \ldots, n\} \mid \textstyle\sum_{i \in S} w_i \leq W\}.$$

Gesucht: $\arg\max_{S \in \mathcal{F}} \sum_{i \in S} v_i$

Interpretation. Man wählt unter n Sachen mit jeweils einem *Gewicht* w_i und einem *Nutzwert* v_i diejenigen aus, die man in einen Rucksack packt, sodass das Gesamtgewicht eine festgelegte Grenze W nicht übersteigt und der Nutzen maximal wird.

Problem (Maximum Integer Knapsack). Wie Maximum Knapsack aber mit $v_1, \ldots, v_n \in \mathbb{N}_{>0}$.

Problem (Minimum Set Cover). Gegeben seien $n \in \mathbb{N}$ und $\mathcal{C}_0 \subseteq \mathcal{P}(\{1, \ldots, n\})$. Die Menge der möglichen Lösungen ist

$$\mathcal{F} := \{\mathcal{C} \subseteq \mathcal{C}_0 \mid \textstyle\bigcup_{S \in \mathcal{C}} S = \bigcup_{S \in \mathcal{C}_0} S\}$$

Aufgabe: Finde $\mathcal{C} \in \mathcal{F}$ mit minimalem $|\mathcal{C}|$!

Bem. Minimum Set Cover verallgemeinert Minimum Vertex Cover.

Problem (Minimum Vertex Coloring, ▶). Gegeben sei ein unger. Graph $G = (V, E)$. Die Menge der *Eckenfärbungen* ist

$$\mathcal{F} := \{\text{Abbildungen } c : V \to \mathbb{N} \mid \forall\, \{v, w\} \in E \,:\, c(v) \neq c(w)\}.$$

Ziel: Finde $c \in \mathcal{F}$ mit minimaler Anzahl $\max c(V)$ an Farben.

Problem (Minimum Edge Coloring). Gegeben sei ein unger. Graph $G = (V, E)$. Die Menge der *Kantenfärbungen* ist

$$\mathcal{F} := \{\text{Abb. } c : E \to \mathbb{N} \mid \forall\, e_1 \neq e_2 \in E \,:\, e_1 \cap e_2 \neq \emptyset \implies c(e_1) \neq c(e_2)\}$$

Ziel: Finde $c \in \mathcal{F}$ mit minimaler Anzahl $\max c(V)$ an Farben.

Problem (Minimum TSP). Gegeben sei ein vollständiger unger. Graph $G = (V, E)$ und eine Abb. $c : E \to \mathbb{R}_{\geq 0}$. Gesucht ist eine zyklische Permutation σ von V (eine *Tour*), sodass die *Länge* $\sum_{v \in V} c(\{v, \sigma(v)\})$ minimal wird.

Problem (Minimum Δ-TSP). Gegeben sei ein endlicher metrischer Raum (V, c). Gesucht ist eine zyklische Permutation σ von V (eine *Tour*), sodass die *Länge* $\sum_{v \in V} c(v, \sigma(v))$ minimal wird.

Problem (k-SAT(isfiability)). Gegeben sei eine Formel in konjunktiver Normalform, etwa

$$(x_1 \lor x_2 \lor \overline{x_3}) \land (\overline{x_1} \lor \overline{x_3} \lor x_4 \lor \overline{x_5}) \land (x_2 \lor x_5) \land (x_3 \lor \overline{x_4}).$$

Die Maximalzahl an *Literalen* in einer *Clause* sei dabei $\leq k$. Entscheide, ob die Formel **erfüllbar** ist,d. h. ob es eine Zuweisung der Variablen gibt, sodass die Formel wahr ist.

Problem (MaxSAT). Gegeben eine Formel in konjunktiver Normalform, finde eine Zuweisung der Variablen, die die Anzahl der gültigen Clauses maximiert.

Problem (Minimum Bin Packing). Gegeben seien *Objektgrößen* $v_1, \ldots, v_n \in [0, 1]$. *Packungen* sind Abbildungen $f : \{1, \ldots, n\} \to \mathbb{N}$, die jedem *Objekt* einen *Behälter* (mit Volumen 1) zuweisen, sodass

$$\textstyle\sum_{i \in f^{-1}(j)} v_i \leq 1 \quad \text{für alle } j \in \mathbb{N}.$$

Gesucht ist ein f mit minimaler Anzahl $\max \operatorname{im}(f)$ von Behältern.

Problem (Minimum Dominating Set). Eine **dominierende Menge** eines Graphen $G = (V, E)$ ist eine Menge $D \subseteq V$ mit $V = D \cup \mathrm{Neighbours}_G(D)$. Gegeben einen Graphen G, finde eine dominierende Menge D kleinster Größe $|D|$!

Problem (Black-and-White Dominating Set). Gegeben einen unger. Graph $G = (B \cup W, E)$ mit $B \cap W = \emptyset$, finde ein $D \subseteq B \cup W$ minimaler Größe, sodass $B \subseteq D \cup \mathrm{Neighbours}_G(D)$.