Algorithmen für NP-harte Probleme

(cc)(by) 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

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

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 Optimierungsprobleme $(\mathcal{X}, \mathcal{F}, Z, \odot)$ 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. $\mathrm{PO} = \mathrm{NPO} \implies \mathrm{P} = \mathrm{NP}$

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 $\mathrm{Opt}(x)$, 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 $Z(x, y) \leq k$?

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

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

Satz. Habe $\mathcal{P} = (\mathcal{X}, \mathcal{F}, Z, \odot) \in \mathrm{NPO}$ 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. $\mathcal{P} \in \mathrm{NPO}$, $\mathcal{P}$ 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

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

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 $\mathrm{Opt}(x)$ existiert. Der **absolute Fehler** von $y \in \mathcal{F}(x)$ ist $|Z(x, y) - \mathrm{Opt}(x)|$.

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

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

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

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

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 $\mathbb{N} \neq$ APX, 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 ungerichteter 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 $\sum\limits_{e \in E'} c(e)$.

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 MIS I in G wie folgt: Setze $H := G$ und $I := \emptyset$, 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

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

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 ∨ ··· ∨ l_k in F (mit k ∈ {0,1,2,3})
    for i := 1,...,k do
        if DECIDE(F|_{l_1=0,...,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 = ∅ then return ∅
    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} ∈ E
        return {v} ∪ COMPUTEMVC(G − {u, v})
    if G hat Dreieck mit Eckknoten u, v, w then
        C_{u,v} := {u, v} ∪ COMPUTEMVC(G − {u, v})
        C_{u,w} := {u, w} ∪ COMPUTEMVC(G − {u, w})
        C_{v,w} := {v, w} ∪ 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} ∪ COMPUTEMVC(G − {u, w})
        C_{v,w} := {v, w} ∪ COMPUTEMVC(G − {v, w})
        C_{v,z} := {v, z} ∪ 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* $A_1, \dots, A_k$, falls er den Algorithmus rekursiv mit Argumenten $G - A_1, \dots, G - A_k$ aufruft. Die Multimenge $\{|A_1|, \dots, |A_k|\}$ heißt zugehörige *Verzweigungsmultimenge.* Für eine solche Multimenge $\{a_1, \dots, a_k\}$ setzen wir

$$\alpha(a_1, \dots, a_k) \coloneqq \max\{x \geq 1 \,|\, x^{-a_1} + \ldots + x^{-a_k} \geq 1\}.$$

Schaffen wir es, einen Algorithmus mit m Fällen mit Verzweigungsmultimengen $A_1, \dots, A_m$ 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 $\max\{\alpha(A_i) \,|\, i = 1, \dots, m\}$

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.

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

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

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

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

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

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

Interpretation. p ist die Anzahl von *Arbeitern*, $l_1, \dots, l_n$ 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 $v_1, \dots, v_n$, $w_1, \dots, w_n, W \in \mathbb{R}_{>0}$ gegeben. Die Menge der möglichen Lsgn sei

$$\mathcal{F} \coloneqq \{S \subseteq \{1, \dots, n\} \,|\, \textstyle\sum_{i \in S} w_i \leq W\}.$$

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

$$\mathcal{F} \coloneqq \{\mathcal{C} \subseteq \mathcal{C}_0 \,|\, \textstyle\bigcup_{S \in \mathcal{C}} S = \bigcup_{S \in \mathcal{C}_0} S\}$$

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

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

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

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

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-Satisfiability). 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.