

2024_t2_A

⚠ Disclaimer

Alles was hier drinnen steht kann Fehler enthalten! Falls dir etwas auffällt melde dich gerne auf Discord bei mir (@xmozz)

A1: P und NP, Spezialfälle

Stoff: 9. Polynominalzeitreduktionen und sem_2/AlgoDat/vo/2. Test/10. NP-Vollständigkeit Spezialfälle

a)

- a) (8 Punkte) Seien A, B, C und D Ja/Nein-Probleme und n jeweils die Eingabegröße.
Nehmen Sie an, es gibt

- eine Reduktion von 3-SAT nach A in Zeit $O(n^5)$,
- eine Reduktion von A nach B in Zeit $O(n^2)$,
- eine Reduktion von A nach 3-SAT in Zeit $O(n^3)$,
- eine Reduktion von A nach C in Zeit $O(2^n)$,
- eine Reduktion von C nach D in Zeit $O(n^2 \log(n))$,
- einen Algorithmus für D mit Laufzeit $O(n^5)$.

(i)

- (i) Geben Sie die engste obere Schranke für die Laufzeit einer Reduktion von 3-SAT nach B in O-Notation an, die sich aus diesen Annahmen ableiten lässt. Begründen Sie Ihre Antwort kurz.

man kann von 3-Sat auf B reduzieren indem man dazwischen auf A reduziert und dann von A nach B, das nimmt eine Zeit von $O((n^5)^2) = O(n^{5 \cdot 2}) = O(n^{10})$ in Anspruch.

(ii)

- (ii) Was folgt aus den obigen Annahmen für die Komplexität der Probleme A, B und C? Beantworten Sie die Frage, indem Sie alle zutreffenden Felder in der folgenden Tabelle ankreuzen.

	in P	in NP	NP-schwer	NP-vollständig
A		X	X	X
B			X	
C	X	X		

- A
 - $3\text{-SAT} \leq_p A$ (Reduktion in $O(n^5)$) $\Rightarrow A$ ist NP-schwer
 - $A \leq_p 3\text{-SAT}$ (Reduktion in $O(n^3)$) $\Rightarrow A \in \text{NP}$
 $\Rightarrow A$ ist NP-vollständig
- B
 - $A \leq_p B$ (Reduktion in $O(n^2)$) $\Rightarrow B$ ist mindestens so schwer wie $A \Rightarrow$ NP-schwer
 - Keine Information, ob $B \in \text{NP} \Rightarrow$ nicht NP-vollständig
- C
 - $C \leq_p D$ (Reduktion in $O(n^2 \log n)$), $D \in \text{P} \Rightarrow C \in \text{P}$
 - Keine Information, dass C NP-schwer oder NP-vollständig ist

b)

- b) (8 Punkte) Wir betrachten folgendes Problem unter der Annahme $\text{P} \neq \text{NP}$.

Problem: Gegeben sei ein Graph G mit n Knoten und m Kanten und eine natürliche Zahl k . Gibt es ein Independent Set S in G mit k oder mehr Knoten?

Welche der folgenden Zertifikate sind (für einen geeigneten Zertifizierer) geeignet, um zu zeigen, dass das gegebene Problem in NP ist? Nehmen Sie dabei an, der Eingabograph G ist von der Form wie in den Spalten angegeben. Kreuzen Sie geeignete Kombinationen an und begründen Sie Zeilen mit nur einem Kreuz kurz.

	G ist schlicht	G ist ein Baum
Ein leerer String.		
Alle Independet Sets in G .		
Ein Independent Set S der Größe k .		
Ein Vertex Cover C mit $n - k$ Knoten.		

- **Leerer String:** Enthält keine Information \rightarrow nicht verifizierbar.
- **Alle Independet Sets:** Exponentiell viele \rightarrow nicht polynomiell prüfbar.
- **Independent Set der Größe k :** In polynomieller Zeit prüfbar ($\text{Größe} \geq k$, keine benachbarten Knoten).
- **Vertex Cover mit $n - k$ Knoten:** Komplement eines Independent Sets der Größe $\geq k \rightarrow$ prüfbar in polynomialem Zeit.

c)

- c) (4 Punkte) Betrachten Sie folgendes Problem und Zertifikat:

Problem: Gegeben sei ein Graph G mit n Knoten und m Kanten. Gibt es eine *ausgeglichene* 3-Färbung φ von G , das heißt, eine Färbung der Knoten von G in Rot, Gelb und Blau, sodass benachbarte Knoten nicht die gleiche Farbe besitzen und die Anzahl der rot, gelb und blau gefärbten Knoten sich jeweils um maximal eins unterscheidet?

Zertifikat: Eine ausgeglichene 3-Färbung φ von G .

(i)

- (i) Geben Sie für das gegebene Zertifikat einen geeigneten Zertifizierer an, um zu zeigen, dass das Problem in NP ist.

Gegeben: Ein Graph $G = (V, E)$ mit n Knoten und m Kanten sowie ein Zertifikat φ , das eine 3-Färbung der Knoten mit den Farben {rot, gelb, blau} ist.

Der Zertifizierer prüft:

1. Korrekte Färbung

Für jede Kante $\{u, v\} \in E$ gilt:

$$\varphi(u) \neq \varphi(v)$$

2. Ausgeglichenheit der Farbverteilung

Zähle:

- r = Anzahl rot gefärbter Knoten
- g = Anzahl gelb gefärbter Knoten
- b = Anzahl blau gefärbter Knoten

Dann prüfe:

$$\max(r, g, b) - \min(r, g, b) \leq 1$$

Beide Prüfungen sind in $\mathcal{O}(n + m)$ möglich → Das Zertifikat ist polynomiell verifizierbar.

(ii)

- (ii) Erklären Sie, welche Eigenschaften ein Zertifizierer im Allgemeinen erfüllen muss, um für ein geeignetes Zertifikat zu zeigen, dass ein Problem in NP ist.

Ein Zertifizierer muss folgende Eigenschaften erfüllen:

- **Eingabe:** Instanz x und ein Zertifikat y
- **Laufzeit:** Muss in Zeit $O(\text{poly}(|x|))$ arbeiten
- **Korrektheit:**
Akzeptiert genau dann, wenn y korrekt belegt, dass x eine "Ja"-Instanz des Problems ist

→ Damit liegt das Problem in NP: Lösungen sind effizient **verifizierbar**, auch wenn sie nicht effizient **auffindbar** sind.

A2: Branch-and-Bound

Stoff: [sem_2/AlgoDat/vo/2. Test/11. Branch and Bound](#)

Für die folgenden Teilaufgaben betrachten wir das TALENT SCHEDULING Minimierungsproblem. Dabei soll die Reihenfolge, in der Szenen für einen Film gedreht werden, bestimmt werden, sodass die Kosten minimal sind. Pro Tag kann nur eine Szene gedreht werden und alle Talente ($a_i \in A$) sind von Drehbeginn des Films durchgehend bis zum Dreh ihrer letzten Szene ($s_j \in S$) angestellt. Das Tagesgehalt ($l_i \in L$) der Talente sowie die Beteiligung an einzelnen Szenen kann tabellarisch wie in der unten stehenden Tabelle dargestellt werden.

Beispielsweise verursacht die Szenenreihenfolge [1,2,3,4] Kosten von 850, da erst nach Drehtag 3, Talente A und D mit ihrem Dreh fertig sind und für den letzten Drehtag noch Talente B und C bezahlt werden müssen.

Talent \ Szene Nr.	1	2	3	4	Gehalt
A	1	1	1	0	100
B	0	0	1	1	50
C	1	0	1	1	50
D	0	0	1	0	50

a)

- a) (4 Punkte) Bestimmen Sie eine Lösung mit einem Lösungswert von 750 oder weniger für die oben gegebene Instanz. Es muss kein Lösungsweg angegeben werden.

Lösung: Szenenreihenfolge

$$[3, 4, 1, 2]$$

Kostenberechnung:

Tag	Szene	A	B	C	D	Aktive Talente
1	3	1	1	1	1	A, B, C, D
2	4	0	1	1	0	B, C
3	1	1	0	1	0	A, C,
4	2	1	0	0	0	A

Start- und Endtage pro Talent:

- A: Tage 1 bis 4 $\rightarrow 4 \text{ Tage} \times 100 = 400$
- B: Tage 1 bis 2 $\rightarrow 2 \text{ Tage} \times 50 = 100$
- C: Tage 1 bis 3 $\rightarrow 3 \text{ Tage} \times 50 = 150$
- D: Tag 3 $\rightarrow 1 \text{ Tag} \times 50 = 50$

Gesamtkosten:

$$400 + 100 + 150 + 50 = 700$$

b)

- b) (4 Punkte) Ein Branch-and-Bound Algorithmus soll das TALENT SCHEDULING Problem lösen. Ein Teilproblem wird dabei durch ein Tupel (T, F) repräsentiert, in dem T die gegebene Probleminstanz ist und F die bisher fixierte Szenenreihenfolge. Beschreiben Sie, wie der Algorithmus das Teilproblem (T, F) in kleinere Teilprobleme zerteilen kann. Eine kurze und klare Beschreibung der Idee ohne Begründung genügt.

Hinweis: Beim Branching können auch mehr als 2 Kindknoten pro Elternknoten erzeugt werden.

Der Branch-and-Bound Algorithmus teilt das Teilproblem $((T, F))$ folgendermaßen auf:

- **Ausgehend von der bisher fixierten Szenenreihenfolge (F)** werden alle noch nicht in (F) enthaltenen Szenen betrachtet.
- Für jede dieser noch offenen Szenen wird ein neues Teilproblem erzeugt, indem diese Szene an das Ende von (F) angehängt wird.
- Somit entstehen pro Elternknoten so viele Kindknoten, wie es noch unplatzierte Szenen gibt.

c)

- c) (8 Punkte) Sei F eine Teillösung aus Unteraufgabe b). Kreuzen Sie an, ob die beschriebene Methode eine untere Schranke, obere Schranke oder keine Schranke für das TALENT SCHEDULING Problem ist.

(+1 Punkt für jede richtige, -1 Punkt für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

	untere	obere	keine
Die Gesamtkosten sind die Kosten für F , plus der Summe aller übrigen Szenenkosten. Die Kosten einer Szene sind die Summe der Gehälter aller beteiligten Talente.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Fixiere eine zufällige Reihenfolge für alle noch nicht fixierten Szenen. Berechne daraus mit F die Gesamtkosten.	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Das bisher zu zahlende Gehalt für die bereits fixierten Szenen F .	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Eine Greedy Methode, die aus einer Teillösung F eine gültige Lösung erzeugt und aus dieser die Gesamtkosten bestimmt.	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Bestimme die minimale Anzahl an übrigen Drehtagen für jedes Talent. Berechne daraus mithilfe von F die Gesamtkosten.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Zuzüglich den Kosten von F , bezahle alle Talente, die den Dreh noch nicht abgeschlossen haben bis ans Ende des Films.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Die Kosten für F plus, Restkosten als die Anzahl der noch nicht gedrehten Szenen, mal dem Durchschnittsgehalt aller Talente.	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Berechne die Gesamtkosten als die Anzahl der bereits gedrehten Szenen multipliziert mit der Summe aller Gehälter.	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

d)

Aussage	Wahr	Falsch	Begründung
(Q1) Das Branching einer Teilinstanz in einem Maximierungsproblem wird abgebrochen, wenn die lokale obere Schranke kleiner ist als die globale untere Schranke.	<input checked="" type="checkbox"/>		Ja weil es kann dann nicht mehr besser werden.
(Q2) Eine korrekte Lösung kann nur gefunden werden, wenn alle Teilprobleme betrachtet werden.		<input checked="" type="checkbox"/>	Falsch: Branch-and-Bound eliminiert korrekt viele Teilprobleme mit Schranken.
(Q3) Branch-and-Bound Methoden bringen immer eine Laufzeitverbesserung gegenüber Brute-Force Methoden.		<input checked="" type="checkbox"/>	Falsch: Im Worst Case bleibt BnB bei exponentieller Laufzeit.
(Q4) Bei der Best-first Durchmusterung in einem Maximierungsproblem wird als nächstes das Teilproblem bearbeitet, welches die geringste Differenz zwischen lokaler unterer und oberer Schranke hat.		<input checked="" type="checkbox"/>	Falsch: Best-first wählt das Teilproblem mit der besten oberen Schranke (also dem besten Potenzial).

A3: Dynamisches Programmieren

Stoff: sem_2/AlgoDat/vo/2. Test/12. Dynamische Programmierung

a)

a) (5 Punkte) Wir betrachten eine Funktion $f: \mathbb{N} \rightarrow \mathbb{Z}$, die wie folgt definiert ist:

$$f(n) = \begin{cases} 1 & \text{wenn } n = 1 \text{ oder } n = 2, \\ 3 \cdot f(n-1) - f(n-2) & \text{wenn } n > 2 \text{ und } n \text{ gerade,} \\ f(n-1) + f(n-2) & \text{wenn } n > 2 \text{ und } n \text{ ungerade.} \end{cases}$$

Bestimmen Sie die Werte $f(5)$, $f(7)$ und $f(11)$.

Hinweis: Nutzen Sie einen geeigneten Algorithmus. Notieren Sie Ihre Zwischenergebnisse.

$$\begin{aligned} f(1) &= 1 \\ f(2) &= 1 \\ f(3) &= 2 \\ f(4) &= 5 \\ f(5) &= 7 \\ f(6) &= 16 \\ f(7) &= 23 \\ f(8) &= 53 \\ f(9) &= 30 \\ f(10) &= 37 \\ f(11) &= 7 \end{aligned}$$

b)

b) (8 Punkte) Das Problem PARTITION ist wie folgt definiert:

PARTITION

Eingabe: Eine Menge $A = \{a_1, a_2, \dots, a_n\}$ mit n positiven, ganzen Zahlen, die sich auf eine gerade Zahl S aufsummieren, also $S = \sum_{i=1}^n a_i$.

Aufgabe: Entscheiden Sie, ob sich A in zwei gleich große Teile aufteilen lässt, das heißt, ob eine Teilmenge $B \subseteq A$ existiert, sodass $S/2 = \sum_{b \in B} b$.

Geben Sie einen Algorithmus an, der PARTITION löst und auf dynamischer Programmierung beruht. Seine Laufzeit muss in $O(nS)$ liegen. Die Ausgabe soll 1 sein, wenn eine Lösung existiert und 0 sonst. Vervollständigen Sie dafür den nachfolgenden Pseudocode auf der nächsten Seite.

Hinweis: Sie können das aus der Vorlesung bekannte dynamische Programm für das Rucksackproblem anpassen.

SolvePartition(Menge A):

```

 $S \leftarrow \sum_{i=1}^n a_i$ 
 $M[0, 0] \leftarrow 1$ 
for  $s \leftarrow 1$  bis  $S/2$ 
     $M[0, s] \leftarrow 0$ 

```

```

for i <- 1 bis n
    for s <- 0 bis S/2
        if s < a(i):
            M[i, s] <- M[i-1, s]
        else:
            M[i, s] <- M[i-1, s] or M[i-1, s - a(i)] + a(i)
            # wir nehmen dann entweder die option wo wirs mitnehmen oder
            # wo wirs nicht mitnehmen

```

return $M[n, S/2]$

c)

- c) (7 Punkte) Gegeben ist ein gerichteter Graph $G = (V, E)$ mit $|V| = n$ Knoten. Die Kanten von G sind mit Kantengewichten $c_{vw} \in \mathbb{R}$ für alle Kanten $(v, w) \in E$ gewichtet.

(i)

- (i) Gehen Sie davon aus, dass G keine negativen Kreise enthält. Ergänzen Sie Bellmans Rekursionsgleichungen für die Länge $OPT(i, v)$ eines kürzesten $v-t$ Pfades in G mit höchstens i Kanten, wobei $v \in V$ ein beliebiger Knoten ist und $t \in V$ ein fester Zielknoten:

$$OPT(0, t) = 0,$$

$$OPT(0, v) = \infty \text{ für } v \neq t, \text{ und}$$

$$OPT(i, v) = \min(OPT(i - 1, v), \min_{(u,v) \in E}(w(v, u) + OPT(i - 1, u)))$$

(ii)

- (ii) Geben Sie den kleinsten Wert für die Kantenanzahl i an, sodass für jeden Graphen G , der keinen negativen Kreis enthält, Folgendes gilt:

$$OPT(i, v) = OPT(i + 1, v) \text{ für alle Knoten } v \in V.$$

Kleinster Wert für i , sodass Stabilität eintritt:

$i = n - 1$

Begründung:

- Jeder kürzeste Pfad in einem Graph ohne negative Kreise besteht aus **höchstens $n - 1$ Kanten**, da ein Pfad mit n oder mehr Knoten einen Kreis enthalten müsste.
- Deshalb ändert sich der Wert von OPT ab $i = n - 1$ nicht mehr

(iii)

- (iii) Erklären Sie *kurz*, wie ein Algorithmus mithilfe der obigen Gleichungen erkennen kann, ob Graph G einen negativen Kreis enthält.

$$\exists v \in V : OPT(n, v) < OPT(n - 1, v)$$

Verändert sich nach dem **n-ten Schritt** noch ein Wert für einen Knoten, bedeutet das, dass durch das Durchlaufen eines Kreises eine Pfadkostenverbesserung möglich ist.

A4: Approximationsalgorithmen

Stoff: sem_2/AlgoDat/vo/2. Test/13. Approximation

a)

- a) (4 Punkte) Sei A ein ε -Approximationsalgorithmus für ein Minimierungsproblem, der für jede Instanz x eine gültige Lösung mit Lösungswert $c_A(x) > 0$ liefert. Sei $c_{opt}(x) > 0$ der Wert einer optimalen Lösung.

(i) Geben Sie die Ungleichung für die Gütegarantie an.

Hier ist die Ungleichung für die Gütegarantie bei einem Minimierungsproblem für einen ϵ -Approximationsalgorithmus:

$$c_A(x) \leq (1 + \epsilon)c_{opt}(x)$$

Dabei ist:

- $c_A(x)$ der Kostenwert der vom Approximationsalgorithmus gefundenen Lösung.
- $c_{opt}(x)$ der Kostenwert einer optimalen Lösung.
- ϵ der Approximationfaktor.

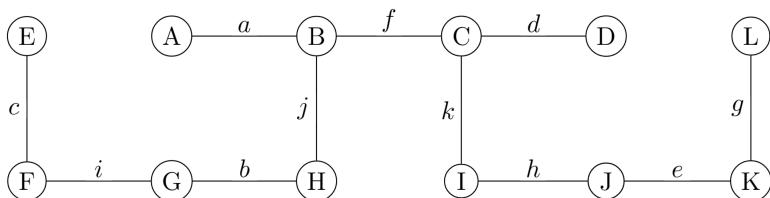
(ii) Welche Werte kann ϵ bei Minimierungsproblemen annehmen?

Für Minimierungsprobleme nimmt ϵ typischerweise Werte an, die größer oder gleich Null sind:

$$\epsilon \geq 0$$

b)

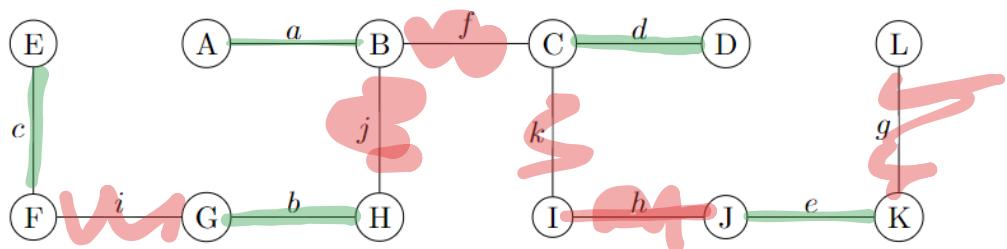
- b) (8 Punkte) Beim MINIMUM VERTEX COVER Problem geht es darum, für einen gegebenen Graphen ein Vertex Cover mit minimaler Anzahl an Knoten zu finden. Betrachten Sie dazu den aus der Vorlesung bekannten 2-Approximationsalgorithmus und den folgenden Graphen:



--> Algorithmus

(i)

- (i) (4 Punkte) Führen Sie den Algorithmus auf dem gegebenen Graphen aus und geben Sie das resultierende Vertex Cover an. Wenn Sie die Auswahl zwischen mehreren Kanten haben, nehmen Sie die alphabetisch kleinste.



$$\Rightarrow \begin{matrix} \underline{a} \\ |c| \\ \underline{e} \end{matrix} \quad \begin{matrix} \underline{d} \\ | \\ \underline{e} \end{matrix}$$

Vertex cover kanten: (a, b, c, d, e)

VC Knoten: $(A, B, C, D, E, F, G, H, J, K, L)$

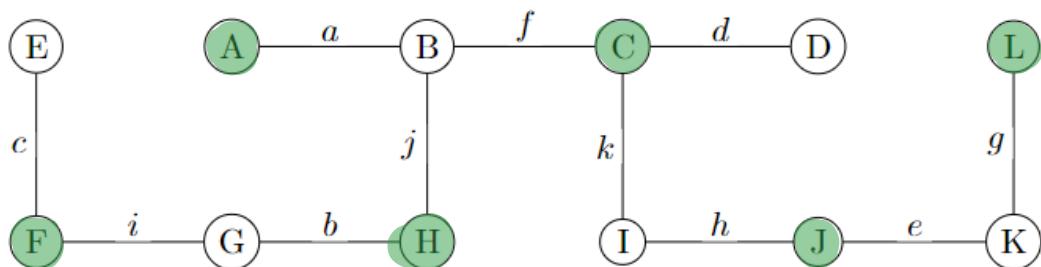
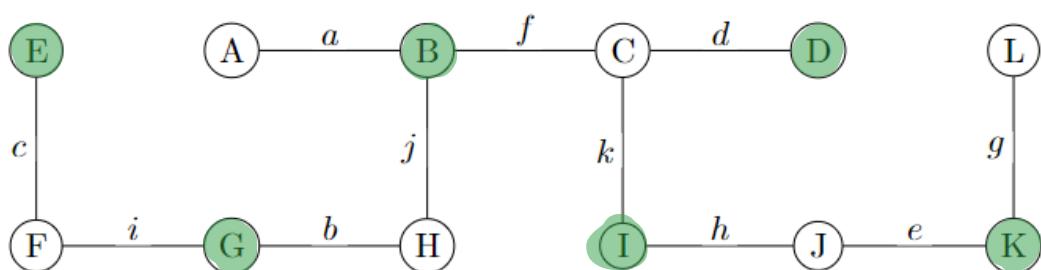
(ii)

(ii) (2 Punkte) Geben Sie ein optimales Vertex Cover an.

Der gegebene Graph ist ein Pfadgraph mit $n = 12$ Knoten. Für einen Pfadgraphen beträgt die Größe eines optimalen Vertex Covers $\lceil n/2 \rceil$. Für $n = 12$ ist die minimale Größe des Vertex Covers $\lceil 12/2 \rceil = 6$.

Ein mögliches optimales Vertex Cover (Größe 6):

$$VC_{opt} = \{B, D, K, I, G, E\}$$



(Diese Knoten decken alle Kanten ab: (A,B), (B,C), (C,D), (D,L), (L,K), (K,J), (J,I), (I,H), (H,G), (G,F), (F,E)). Ein anderes mögliches optimales Vertex Cover wäre $\{A, C, L, J, H, F\}$.

(iii)

- (iii) (2 Punkte) Welche Kanten muss der Approximationsalgorithmus auswählen, um ein optimales Vertex Cover zu finden?

Statt e sollte er h und g wähle.

c)

- c) (4 Punkte) Wir betrachten das aus der Vorlesung bekannte MAX-SAT Problem (maximiere die Anzahl der erfüllten Klauseln in einer booleschen Formel) und einen Approximationsalgorithmen A für MAX-SAT mit einer Gütegarantie von $\frac{3}{4}$.

(i)

- (i) Gegeben ist eine Formel F mit 40 Klauseln, sodass die optimale Variablenzuweisung 32 Klauseln in F erfüllt. Weiters ist I eine Variablenzuweisung die mittels A für F gefunden wurde. In welchem Intervall befindet sich die Anzahl der von I erfüllten Klauseln? Geben Sie die engstmöglichen Schranken an.

$$\frac{32}{4} \cdot 3 = 24$$

Die Anzahl der von dem Approximationsalgorithmus A erfüllten Klauseln I muss sich zwischen 24 und 32 befinden.

(ii)

- (ii) Angenommen A findet für eine andere Formel F' mit 50 Klauseln eine Variablenzuweisung die 33 Klauseln in F' erfüllt. In welchem Intervall befindet sich eine optimale Lösung für F' ? Geben Sie die engstmöglichen Schranken an.

$$33 \geq \frac{3}{4} \cdot K_{opt}$$

$$33 \cdot \frac{4}{3} \geq K_{opt}$$

$$44 \geq K_{opt}$$

zwischen 33 und 44 liegt K_{opt} .

d)

d) (4 Punkte) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

(+1 Punkt für jede richtige, -1 Punkt für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

(Q1) Ein Approximationsalgorithmus mit Gütegarantie 4 kann eine Lösung zurückliefern, deren Wert 4% über dem optimalen Lösungswert liegt.

Wahr Falsch

(Q2) Die aus der Vorlesung bekannte Spannbaumheuristik löst das *symmetrische Traveling Salesperson Problem* mit Gütegarantie 3.

Wahr Falsch

(Q3) Falls $P = NP$ gilt, dann existiert für das LASTVERTEILUNG Problem ein polynomieller 1.5-Approximationsalgorithmus.

Wahr Falsch

(Q4) Ein $\frac{1}{2}$ -Approximationsalgorithmus für ein Maximierungsproblem findet eine Lösung die mindestens halb so groß wie die optimale Lösung ist.

Wahr Falsch

Q1: nein 4 mal so groß

Q2: nein das kann man nicht approximieren, nur das metrische TSP und das mit Güte 2

Q3: Nein dann wäre alles in Polynominaler Zeit lösbar und man bräuchte keinen Approximationsalgorithmus mehr bzw. die Gütegarantie wäre 1

A5: Heuristische Verfahren

Stoff: 14. Heuristiken und Lokale Suche

a)

- a) (8 Punkte) Wir betrachten eine lokale Suche für das aus der Vorlesung bekannte MAX-SAT Problem (maximiere die Anzahl der erfüllten Klauseln in einer Boole'schen Formel). Variablenzuweisungen werden als binäre Vektoren (x_1, \dots, x_n) repräsentiert. N bezeichnet die aus der Vorlesung bekannte 1-flip Nachbarschaft für binäre Vektoren.

Gegeben ist eine Boole'sche Formel F mit Variablen x_1, x_2, x_3 :

$$(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Weiters ist die Ausgangslösung $I = (1, 0, 1)$ gegeben, es wird also x_1 auf *wahr*, x_2 auf *falsch* und x_3 auf *wahr* gesetzt.

(i)

- (i) Bestimmen Sie die Nachbarschaft $N(I)$ von I und geben Sie für jeden Vektor $J \in N(I)$ an, wie viele Klauseln in F von J erfüllt werden.

Mögliche Nachbaren:

$$\begin{aligned} I^{(1)} &= (0, 0, 1) \\ I^{(2)} &= (1, 1, 1) \\ I^{(3)} &= (1, 0, 0) \end{aligned}$$

Wie viele Klauseln stimmen:

- Für I : 3
- Für $I^{(1)}$: 4
- Für $I^{(2)}$: 4
- Für $I^{(3)}$: 5

(ii)

- (ii) Welcher Vektor in $N(I)$ wird bei einer lokalen Suche als nächstes ausgewählt, wenn Best Improvement als Schrittfunction verwendet wird?

$$I^{(3)} = (1, 0, 0)$$

(iii)

- (iii) Angenommen die Lösungen in $N(I)$ werden in lexikographisch aufsteigender Reihenfolge durchmustert. Es würde also $(0, 0, 0)$ vor $(0, 0, 1)$ vor $(0, 1, 0)$ vor $(0, 1, 1)$ usw. durchmustert werden. Welcher Vektor in $N(I)$ wird dann bei einer lokalen Suche als nächstes ausgewählt, wenn First Improvement als Schrittfunction verwendet wird?

Dann wäre die Reihenfolge:

$$\begin{aligned} I^{(1)} &= (0, 0, 1) = 4 \\ I^{(3)} &= (1, 0, 0) = 5 \\ I^{(2)} &= (1, 1, 1) = 4 \end{aligned}$$

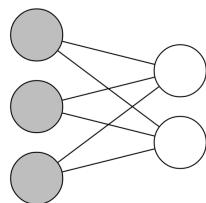
Die erste Verbesserung die dann genommen wird wäre von $I^{(1)}$ also von $(0, 0, 1)$

b)

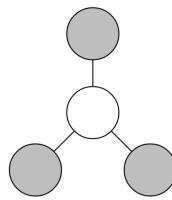
b) (8 Punkte) Im Folgenden bezeichnen N und N' die zwei aus der Vorlesung bekannten Nachbarschaftstrukturen für das MINIMUM VERTEX COVER Problem:

- $S \in N(C)$ wenn S ein Vertex Cover ist das durch Löschen eines einzigen Knotens aus C erzeugt werden kann.
- $S \in N'(C)$ wenn $S \in N(C)$ oder wenn S ein Vertex Cover ist das durch Hinzufügen eines Knotens von $V \setminus C$ und Löschen von zwei Knoten aus C gebildet werden kann.

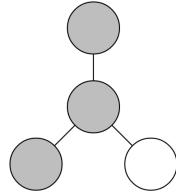
Mit welchen Nachbarschaftsstrukturen kann eine lokale Suche für MINIMUM VERTEX COVER bei den folgenden Graphen und Ausgangslösungen (grau markiert) ein globales Optimum finden? Kreuzen Sie alle zutreffenden Felder an.



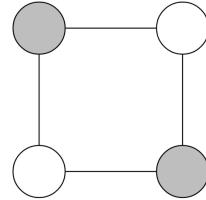
(i) N N' weder noch



(ii) N N' weder noch



(iii) N N' weder noch



(iv) N N' weder noch

Bin mir nicht ganz sicher ob ich das Beispiel richtig verstanden hab, falls jemand das anders löst bitte melden!

c)

c) (4 Punkte) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

(+1 Punkt für jede richtige, -1 Punkt für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

(Q1) Falls $P = NP$ gilt, liefert eine lokale Suche für MAX-SAT mit 1-flip Nachbarschaft garantiert eine optimale Lösung und zwar in Polynomialzeit und unabhängig von der Ausgangslösung.

Wahr Falsch

(Q2) Bei der Tabu-Suche wird das Metropolis-Kriterium verwendet, um das erneute Besuchen bereits gefundener Lösungen zu vermeiden.

Wahr Falsch

(Q3) Im Allgemeinen gilt: wenn man lokale Suche mit Schrittfunktion First Improvement lange genug ausführt, findet man immer ein lokales Optimum.

Wahr Falsch

(Q4) Bei Simulated Annealing werden immer nur Nachbarlösungen ausgewählt, welche strikt besser sind als die momentan ausgewählte Lösung.

Wahr Falsch

Q1: Selbst wenn $P=NP$ gilt, garantiert eine **lokale Suche** in der Regel **keine** optimale Lösung, da sie in lokalen Optima stecken bleiben kann. " $P=NP$ " würde bedeuten, dass es **einen** Polynomialzeit-Algorithmus gibt, der die optimale Lösung findet, aber die lokale Suche ist nicht unbedingt dieser Algorithmus.

Q2: Das Metropolis-Kriterium gehört zum Simulated Annealing, und die Tabu-Liste ist das Merkmal der Tabu-Suche, um Wiederholungen zu vermeiden.

Q3:

- **Lokale Suche** (wie wir sie bei Q3 besprochen haben) würde tatsächlich nur bessere Lösungen akzeptieren.
- **Simulated Annealing** (Simulierte Abkühlung) ist aber gerade dafür konzipiert, aus lokalen Optima auszubrechen. Es tut dies, indem es mit einer bestimmten Wahrscheinlichkeit auch **schlechtere** Nachbarlösungen akzeptiert. Diese Wahrscheinlichkeit nimmt im Laufe des Algorithmus ab ("Abkühlung"), sodass es am Ende eher wie eine lokale Suche funktioniert.