Das Lastverteilungsproblem

Multiprocessor Scheduling

Franz Brauße

26. März 2012

Proseminar Theoretische Informatik
bei Prof. Dr. H. Fernau
FB IV, Universität Trier

Inhaltsverzeichnis

1 Einführung

1.1 Approximation

Um die Komplexität von Algorithmen zur Lösung „schwieriger“ Probleme zu senken und sie für größere Probleminstanzen praktisch einsetzbar zu machen, kann versucht werden, die Qualität der Lösung zu reduzieren, also eine Annäherung eines optimalen Ergebnisses zu berechnen, die dennoch nicht zu stark davon abweicht. Die Abweichung wird mit einem Gütebegriff beschrieben, auf den im Abschnitt 1.2 eingegangen wird. Allgemein fällt die Vorgehensweise unter die Bezeichnung Optimierung.

Das Entscheidungsproblem zum hier Untersuchten ist **NP**-vollständig, siehe Abschnitt 2.2, weshalb **P**-Algorithmen betrachtet werden, die es approximieren.

1.2 Güte

Es wird zwischen absoluter und relativer Güte von Algorithmen unterschieden [Wan06, 2, 3]. Diese Größen werden zurückgeführt auf die Bewertung von Lösungen durch eine Gütefunktion $v : W \to \mathbb{R}_{\geq}$ über dem Lösungsraum W des Algorithmus. Existiert eine Metrik d auf W, so wird die Güte einer Lösung $w \in W$ durch $w \mapsto v(w) = d(0, w)$ bestimmt. Auf Vektorräumen $W = V^k$ kann d auch durch eine p-Norm $\|\cdot\|_p$ über $d(x, y) = \|x - y\|_p$ induziert werden, bspw. $\|\cdot\|_\infty$ bei [BCJG+76, 1.2.4]. Dann gilt

$$v(w) = d(0, w) = \|0 - w\|_\infty = \max_{i=1,\ldots,k} \{|w_i|\},$$

wie v als Gütefunktion ab Abschnitt 2.1.1 hier auch verwandt wird.

Zu einer Probleminstanz I seien w_I^* eine exakte Lösung und w_I die vom zu bewertenden Algorithmus errechnete Lösung. Sodann wird die absolute Güte $\kappa(I)$ der Lösung w_I und ihre relative Güte $\varrho(I)$ bestimmt [Wan06] durch

$$\begin{aligned} \kappa(I) &= |v(w_I) - v(w_I)| \\ \varrho(I) &= \max\left\{\frac{v(w_I)}{v(w_I^*)}, \frac{v(w_I^*)}{v(w_I)}\right\} \geq 1. \end{aligned}$$

Die entsprechende Güte des Algorithmus ergibt sich nun als obere Schranke [GGL95, 28.2] auf $\kappa(I)$ bzw. $\varrho(I)$ über alle Eingaben I. Im Folgenden wird die relative Güte des jeweilig untersuchten Algorithmus mit ϱ bezeichnet.

2 Lastverteilung

2.1 Überblick

Mit Lastverteilung oder Scheduling wird eine ganze Klasse von Problemen bezeichnet, die gemein haben, dass eine begrenzte Zahl von Ressourcen optimal über Zeit aufgeteilt wird [LLRKS89]. Üblich sind bspw. Verteilungen von wartenden Kunden auf Verkäufer, Threads auf Prozessoren oder von Jobs auf Server. Es lassen sich viele weitere Beispiele finden.

Je nach Art der Anwendung haben sich im Laufe der Untersuchung des Problems verschiedene Bedingungen an die Jobs und Server ergeben [BCJG+76, 1.2], [LLRKS89, I.3], [GGL95, 29.3.1, 35.9, 35.10]. Hier sei ein Überblick dieser Varianten versucht. In den Abschnitten 2.3.1 und 2.3.2 werden zwei Algorithmen für die unterstrichenen Eigenschaften untersucht.

- Jobs setzen zusätzlich zu Prozessoren eine Menge von Ressourcen voraus (hier keine).
- Jobs benötigen unterschiedliche Prozessoren gleichzeitig oder nacheinander (*job-shop scheduling*) zur Ausführung (hier einen).
- Jobs können gleichzeitig auf mehreren Prozessoren ausgeführt werden (hier auf einem).
- Alle Prozessoren sind gleich oder unterscheiden sich in Merkmalen wie Ausführungsgeschwindigkeit (pro Job) oder der Möglichkeit einen Task überhaupt auszuführen.
- Jobs sind voneinander unabhängig oder setzen den Abschluss der Bearbeitung anderer Jobs voraus.
- Alle Informationen über Job-Dauern und Abhängigkeiten sind im Voraus bekannt (*online*) oder nicht (*offline*).
- (Nicht präemptive) Jobs liegen in vorgegebener (*list-scheduling*) oder einer beliebigen Reihenfolge vor.
- Jobs sind unterbrechbar (*preemptive*) oder nicht.
- Es entstehen Kosten $w_i t$ zur Ausführung eines Jobs i zum globalen Zeitpunkt t ($w_i = 1$).

Ziel ist eine Bestimmung der Verteilung S der Jobs auf die Prozessoren P_i ($i = 1, \ldots, n$). Im Folgenden werden die Jobs mit fester, bekannter Ausführungszeit $t_j > 0$ in der globalen Zeit t lückenlos auf die Prozessoren verteilt, d.h. ist P_i zu Zeitpunkt t_0 kein Job zugeteilt, so wird sich P_i auch für alle $t' > t_0$ im Leer-

lauf befinden. Damit werden bloß die Gesamtbelegzeiträume T_i der Prozessoren interessant, d.h.

$$T_i := \min\{t : P_i \text{ nicht belegt}\}, \qquad i = 1, \ldots, m.$$

Da Jobs in dieser Untersuchung nicht präemptiv zerteilt werden können, entspricht die Nichtleerlaufzeit T_i von P_i gerade der Gesamtzeit aller zugewiesen Jobs aus S_i, deren Reihenfolge in der Summe dann ebenfalls keine Rolle mehr spielt.

$$S_i = \{j : \text{Job } j \in \{1, \ldots, m\} \text{ ist } P_i \text{ zugewiesen}\}$$

$$T_i = \sum_{j \in S_i} t_j$$

Zu jeder der obigen Varianten können verschiedene Metriken zur Bestimmung der Güte eines Scheduling-Ergebnisses $S = \{S_i : 1 \leq i \leq m\}$ herangezogen werden. Dabei ergeben sich zwei mögliche Eigenschaften auf natürliche Weise durch leichte Variation der Formulierung der Frage.

2.1.1 Schnelle Bearbeitung aller Jobs

Als *makespan* wird in der englischsprachigen Literatur die Gesamtzeit $\hat{T}$ bezeichnet, die Prozessoren P_i $(i = 1, \ldots, m)$ an den Jobs arbeiten.

$$\hat{T} = \max_{i=1,\ldots,m} \{T_i\}$$

Ziel ist hier also die Minimierung von $\hat{T}$.

2.1.2 Hohe Auslastung aller Maschinen

Hierbei wird versucht die Gesamtauslastung der Maschinen zu maximieren. In obigem speziellen Modell entspricht dies dem Versuch, der Durchschnittlichszeit

$$\overline{T} := \frac{1}{m} \sum_{j=1}^{n} t_j$$

mit allen Prozessoren nahe zu kommen, also einer Minimierung von $\max_{i=1,\ldots,m} |\overline{T} - T_i|$.

Eng damit verbunden ist auch die in [BCJG+76, 1.2.4] vorgestellte durchschnittliche Anzahl der im System verbliebenen Jobs

$$\bar{N} = \frac{1}{\hat{T}} \int_0^{\hat{T}} N(t) \, \mathrm{d}t,$$

für $N(t)$ Anzahl nicht fertiggestellter Jobs zum globalen Zeitpunkt t, welche in praktischer Anwendung einer durchschnittlich vorzuhaltenden Lagerkapazität entspricht. Ebd. wird für den hier betrachteten Fall festgestellt, dass sich dieser Wert zu $\bar{N} = n \cdot \frac{\overline{T}}{\hat{T}}$ ergibt.

2.2 Entscheidungsproblem

Als Entscheidungsproblem lässt sich das hier untersuchte Problem folgenderweise darstellen.

Gegeben sind die Zahl der Prozessoren $m \in \mathbb{N}$, die Zeiten t_j $(1 \leq j \leq n)$ der Jobs und ein Zeitlimit T, je binär. Gefragt ist, ob die maximale Wartezeit $\hat{T}$ die Zeitschranke T einhält, d.h. ob $\hat{T} \leq T$ für diese Instanz gilt.

Dieses Problem ist **NP**-vollständig [GJ79, A5.2]; eine einfache Reduktion um die Härte zu zeigen geht von PARTITION (ebd.) aus, wofür bloß pseudopolynomielle Algorithmen bekannt sind [LLRKS89, I.2]: Wähle $m = 2$ als Zahl der Partitionen, $t_j = s_j$ und als Deadline $T = \frac{1}{2} \sum s_j$. Wegen $T = \overline{T} \leq \hat{T}$ folgt aus der Existenz eines Lastverteilungsplans S, der T erfüllt, dass sich die Menge der Jobs in zwei Partitionen aufteilen lässt, deren summierte Zeiten sich gleichen und PARTITION damit erfüllt ist:

$$T_1 \leq T,\ T_2 \leq T \text{ und } 2T \geq T_1 + T_2 = \sum_{j=1}^{n} t_j = 2T \implies T_1 = T_2 = T.$$

Die Mitgliedschaft in **NP** lässt sich vermittels *Guess-and-Check* nachweisen, d.h. eine Lösung $S = \{S_i : 1 \leq i \leq m\}$ wird nichtdeterministisch bestimmt und im Anschluss wird überprüft, ob sie eine korrekte Scheduling-Lösung darstellt und für die resultierenden Zeiten je $T_i \leq T$ gilt.

2.3 Optimierungsproblem

Das zugehörige und hier untersuchte Optimierungsproblem beinhaltet die Zeitschranke T des Entscheidungsproblems nicht, sondern minimiert den *makespan* $\hat{T}$ [KT06, 11.1].

Bezeichnet T^* den Optimalwert für $\hat{T}$ einer gegebenen Instanz, so muss er einerseits mindestens die Größe des längsten Jobs umfassen, andererseits darf T^* – aufgrund der Einschränkung, dass sich kein Prozessor temporär im Leerlauf befinden darf – $\sum_{j=1}^{n} t_j$ nicht überschreiten, d.h.

$$\max_{j=1,\ldots,n} \{t_j\} \leq T^* \leq \sum_{j=1}^{n} t_j. \tag{1}$$

```
1  m' ← min{m, n}
2
3  for i = 1 to m' do
4      T_i ← 0
5      S_i ← ∅                          /* jobs assigned to P_i */
6  end
7
8  for j = 1 to n do
9      i ∈ arg min_{k=1,...,m'} T_k     /* lowest workload so far */
10     S_i ← S_i ∪ {j}                  /* assign job j to P_i */
11     T_i ← T_i + t_j
12 end
```

Abbildung 1: *Online Greedy*-Scheduling-Algorithmus 1.

2.3.1 *Online Greedy*-Algorithmus

Der in Abbildung 1 vorgestellte Algorithmus weist jedem neuen Job den zu diesem Zeitpunkt am wenigsten belasteten Prozessor zu, d.h. er arbeitet lokal optimal und fällt damit in die Klasse der als *greedy* bezeichneten Algorithmen [GGL95, 28.2].

Er durchläuft eine Initialisierungsschleife über alle m Prozessoren und besucht dann in der vorgegebenen Reihenfolge alle n Jobs einmal, weist sie zu und zählt die Zeit, die der je gewählte Prozessor nun beschäftigt ist. Damit handelt es sich um einen *online*-Algorithmus, da er die vollständige Eingabe nicht vorab benötigt.

Da die Zahl der Prozessoren binär gegeben ist, wird, um sicherzustellen, dass der Algorithmus keine pseudopolynomielle Laufzeit wegen der Initialisierungsschleife aufweist, in Zeile 1 die Anzahl betrachteter Prozessoren eingeschränkt. Sollten $m > n$ Prozessoren verfügbar sein, so werden wegen der Unteilbarkeit der Jobs mindestens $m - n$ Prozessoren nicht belastet, weshalb sie auch keine Betrachtung im Algorithmus finden müssen.

Weil die Zeiten t_i für jedes $i = 1, \ldots, n$ separat in der Eingabe vorliegen, ist n implizit unär gegeben und es ergibt sich eine in einem Polynom in n beschränkte Laufzeit bei geeigneter Implementierung von Zeile 9, bspw. durch ein Vorhalten der T_k ($k = 1, \ldots, m'$, $m' \leq n$) in einem sortierten Array und binäre Suche nach der Einfügeposition von T_i dem Update in Zeile 11. Dies würde zu einem ähnlichen Algorithmus wie Heapsort mit einer oberen Laufzeitschranke ebenfalls in $\mathcal{O}(n \log n)$ führen[1]. Die Mengen S_i müssen nicht explizit gespeichert werden, wenn Elemente der Form (i, j) für das Ergebnis „Job j wird Prozessor P_i zugewiesen“ die Ausgabe bilden können, da S_i vom Algorithmus selbst nicht benötigt wird.

[1] $m' \leq n \implies n \cdot \log_2 m' \in \mathcal{O}(n \log n)$. Diese Abschätzungen benutzen das Einheitskostenmaß.

2.3.1.1 Güteabschätzung Betrachte eine Eingabe mit $m = 4$ Prozessoren und $n = 13$ Jobs mit Zeiten

$$t_1 = t_2 = \ldots = t_{n-1} = 1, \qquad t_n = 4.$$

Abbildung 2 stellt die Verteilung wie sie der skizzierte Algorithmus erzeugt einer optimalen Verteilung gegenüber. Es ist zu erkennen, dass der letzte Job mit Zeit $t_n = 4$ deutlich dazu beiträgt, dass das optimale Ergebnis von $T^* = 4$ verfehlt wird. Der Algorithmus findet hier nur eine Verteilung mit $\hat{T} = 7$.

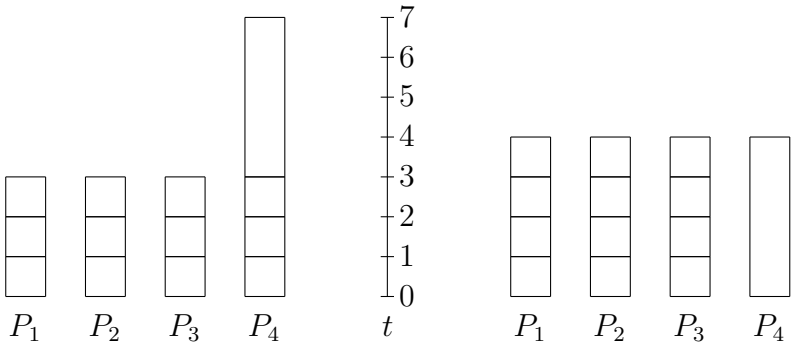

Abbildung 2: Links: Greedy ($\hat{T} = 7$); rechts: optimal ($\hat{T} = T^* = 4$).

Satz 2.1 Der *online* Greedy-Algorithmus 1 hat die relative Güte $\varrho = 2$.

Beweis Jeder Job muss von einem Prozessor bearbeitet werden, d.h. die Zeiten t_j aller Jobs j fließen in die Optimalzeit ein. Da jeder Prozessor maximal einen Job zu jedem Zeitpunkt bearbeitet, gilt für den Optimalwert T^* die Einschränkung

$$\frac{1}{m} \sum_{j=1}^{n} t_j \leq T^*. \tag{2}$$

Außerdem gilt nach (1) $t_j \leq T^*$ $(j = 1, \ldots, n)$.

Betrachte nun einen Prozessor P_i mit $T_i = T^*$ in der vom Algorithmus bestimmten Verteilung und den letzten ihm zugewiesenen Job j.

P_i wurde gewählt, da er vor Zuweisung von j minimale Last $T_i - t_j$ besaß, d.h.

$$T_k \geq T_i - t_j \qquad k = 1, \ldots, m.$$

Durch Aufsummieren der $(m-1)$ Summanden T_k $(k \neq i)$ sowie T_i selbst erhält man die Abschätzung

$$\begin{aligned} \sum_k T_k &\geq (m-1)(T_i - t_j) + (T_i - t_j) + t_j \\ &\geq m(T_i - t_j) \\ \iff \qquad T_i - t_j &\leq \frac{1}{m} \sum_k T_k \qquad (3) \\ &\overset{(2)}{\leq} T^*. \end{aligned}$$

Mit (1) folgt daraus durch Zuweisung des letzten Jobs an P_i mit $t_j \leq T^*$

$$T_i = (T_i - t_j) + t_j \leq T^* + T^* = 2T^*.$$

Zuletzt ergibt sich die relative Güte zu $\varrho = \max\{\frac{2T^*}{T^*}, \frac{T^*}{2T^*}\} = \frac{2}{1} = 2$. □

Diese Abschätzung kann beliebig angenähert werden, wie durch natürliche Erweiterung des in Abbildung 2 dargestellten Beispiels auf höhere m und n ersichtlich ist, denn dort ist $t_j = 1$ $(j < n)$, $t_n = \left\lceil \frac{n-1}{m-1} \right\rceil = T^*$ und $\hat{T} = \left\lfloor \frac{n-1}{m} \right\rfloor + t_n$. Es folgt

$$\varrho = \frac{\hat{T}}{T^*} = 1 + \delta \qquad \text{mit} \qquad \delta = \left\lfloor \frac{n-1}{m} \right\rfloor \Big/ \left\lceil \frac{n-1}{m-1} \right\rceil \overset{m \to \infty}{\to} 1 \qquad \text{für } n > m.$$

Die Erkenntnis, dass diese „Ausreißer“ am Ende der Liste der bisher als *list-scheduling* behandelten, d.h. in der Reihenfolge fest vorgegebenen, Eingabe die Abschätzung, die bis zum vorletzten Job des Prozessors P_i, der am längsten benötigt, noch unter dem Optimalwert T^* lag, so massiv stört, führt zu dem im Folgenden beschriebenen Ansatz.

2.3.2 *Offline Greedy*-Algorithmus

Jobs mit vergleichsweise großer Dauer am Ende lassen sich ausschließen, indem die gegebene Liste der Zeiten t_i zunächst absteigend vorsortiert wird. Dies setzt jedoch voraus, dass die Eigenschaft eines *online*-Algorithmus aufgegeben wird, denn zur Sortierung müssen alle Elemente mindestens einmal betrachtet werden, bevor der eigentliche Scheduling-Vorgang begonnen werden kann. Abbildung 3 zeigt den aus dieser Änderung resultierenden Algorithmus.

2.3.2.1 Güteabschätzung Da sich, abgesehen vom Zwischenschritt der Sortierung der Eingabe in Zeile 8, die – falls Speicherplatzbetrachtungen einbezogen werden – auch über eine Maximumssuche in der Eingabe ohne explizite Kopie der

```
1  m' ← min{m, n}
2
3  for i = 1 to m' do
4      T_i ← 0
5      S_i ← ∅                          /* jobs assigned to P_i */
6  end
7
8  (t'_1, ..., t'_n) ← sort {t_1, ..., t_n}    /* decreasing order */
9
10 for j = 1 to n do
11     i ∈ arg min_{k=1,...,m'} T_k     /* lowest workload so far */
12     S_i ← S_i ∪ {j}                  /* assign job j to P_i */
13     T_i ← T_i + t'_j
14 end
```

Abbildung 3: *Offline Greedy*-Scheduling-Algorithmus 2.

Eingabewerte t_j in quadratischer Zeit und logarithmischem Platz emuliert werden kann, im Vergleich zum Algorithmus 1 nichts ändert und die Sortierung bloß eine Spezialisierung der allgemeinen Eingabeliste darstellt, gilt die gleiche Abschätzung von oben auch hier. Sie lässt sich nun jedoch verbessern.

Lemma 2.2 Es gilt nach der absteigenden Sortierung $(t'_j)_j := (t_{j_k})_k$ von $(t_j)_j$

$$n > m \implies T^* \geq 2t'_{m+1}.$$

Beweis Sei $n > m$, dann wird t_m wegen $t'_j > 0 \ \forall j$ dem letzten Prozessor, o.E. P_m, zugewiesen, womit dieser die Zeit t_m erhält. Sei P_i der Prozessor, dem Job $(m+1)$ zugewiesen wird. Wegen der Sortierung gilt $t'_j \geq t'_m$ $(j = 1, \ldots, m-1)$, sowie $t'_m \geq t'_{m+1}$ und damit folgt

$$T^* \geq t'_i + t'_{m+1} \geq t'_m + t'_{m+1} \geq 2t'_{m+1}.$$

□

Eine Veranschaulichung dieses Sachverhalts bietet Abbildung 4.

Satz 2.3 Der *offline* Greedy-Algorithmus 3 hat die relative Güte $\varrho = \frac{3}{2}$.

Beweis Um die vollständige Aussage zu zeigen folgen zwei Fallunterscheidungen, zunächst nach der Anzahl n der Jobs relativ zu der der Prozessoren m.

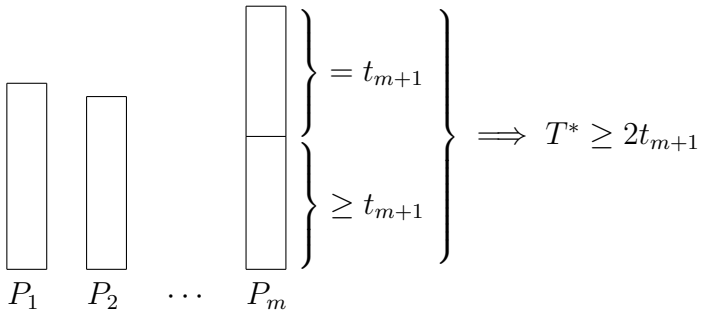

Abbildung 4: Zuweisung der ersten $m+1$ nach Dauer absteigend sortierten Jobs.

- Fall 1: $n \leq m$, dann können alle Jobs einem eigenen Prozessor zugeteilt werden und es gilt

$$\begin{aligned} \hat{T} &= \max_{i=1,\ldots,m}\{T_i\} = \max_{j=1,\ldots,n}\{t'_j\} = t'_1 \overset{(1)}{\leq} T^* \\ \Longrightarrow \qquad \hat{T} &= T^* \leq \frac{3}{2}T^*. \end{aligned}$$

- Fall 2: $n > m$, so betrachte wie im Beweis zu Satz 2.1 den Prozessor P_i mit $T_i = \hat{T}$ mit seinem zuletzt zugeteilten Job j, für den in (3) gezeigt wurde, dass

$$T_i - t'_j \leq T^*$$

gilt.

Eine Fallunterscheidung nach j bzgl. m ermöglicht die Anwendung des vorangegangenen Lemmas.

  - Fall 1: $j \leq m$. Wie im Beweis zu Lemma 2.2 erläutert, werden die ersten m Jobs vom Algorithmus disjunkt verteilt. Damit gilt $\hat{T} = T_i = t'_j$ und es folgt, dass keine Kombination von Jobs das Resultat $\hat{T}$ bestimmt, sondern ein Job alle anderen dominiert. In diesem Fall gilt wie oben $t'_j = t'_1 = \hat{T} = T^* \leq \frac{3}{2}T^*$.
  - Fall 2: $j \geq m+1$. Mit der Aussage des Lemmas 2.2, äquivalent zu

$$t'_j \leq t'_{m+1} \leq \frac{1}{2}T^*,$$

ist nun eine Abschätzung besserer Güte möglich:

$$T_i = (T_i - t'_j) + t'_j \leq T^* + \frac{1}{2}T^* = \frac{3}{2}T^*.$$

Insgesamt ergibt sich die relative Güte als obere Schranke für jede Eingabe, also hier zu $\varrho = \frac{3}{2}T^* \,/\, T^* = \frac{3}{2}$. □

3 Fazit

Es wurden zwei Approximationsalgorithmen für die spezielle Variante des Multiprocessor Scheduling aus der Familie der Scheduling-Probleme untersucht. Vergleichend lässt sich feststellen, dass eine einfache Änderung wie das Sortieren der Eingabe bereits eine Halbierung des relativen Fehlers bewirkt. Gezeigt wurde, dass dieser Gewinn jedoch durch eine Aufgabe der *online*-Eigenschaft und einen Tradeoff zwischen zusätzlichem Speicher und zusätzlicher Laufzeit erkauft wird, was für praktische Belange durchaus eine Rolle spielen kann.

Literatur

[BCJG+76] J.L. Bruno, E.G. Coffman Jr., R.L. Graham, W.H. Kohler, R. Sethi, K. Steiglitz, and J.D. Ullman. *Computer and Job-Shop Scheduling Theory.* Wiley–Interscience, 1976.

[GGL95] R.L. Graham, M. Grötschel, and L. Lovász, editors. *Handbook of Combinatorics*, volume 2. Elsevier Science, 1995.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractiability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[KT06] J.M. Kleinberg and É. Tardos. *Algorithm Design.* Addison-Wesley, 2006.

[LLRKS89] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. Technical Report BS-R8909, Centre for Mathematics and Computer Science, 1009 AB Amsterdam, June 1989.

[Wan06] R. Wanka. *Approximationsalgorithmen.* B.G. Teubner Verlag, first edition, October 2006.