Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Magnus Bender (Übungen)

sowie viele Tutoren

Hm... NP-schwere Probleme sind trickreich

- Was machen wir, wenn Heuristiken und Pruning nicht greifen und unserer Algorithmus sich im Backtracking verirrt?
- Approximation der Lösung?
  - Vielleicht in nebenläufiger Berechnung?
  - Wenn die Bestimmung der optimalen Lösung zu lange dauert, nimm approximative Lösung, sofern sie bereitsteht
- Können wir bei Approximation Garantien für die Güte geben?

Approximationsalgorithmen

Frage: Ich will ein NP-schweres Problem lösen. Was muss ich tun?

Antwort: Polynomialzeitalgorithmus dafür wohl nicht möglich.

Annahme:

Entwurfsmuster zur Aufwandsreduktion (siehe SAT) aufwendig bzw. Reduktion auf SAT (o.ä.) nicht offensichtlich,

Eine der drei Eigenschaften muss aufgegeben werden:

- Löse das Problem optimal.
- Löse das Problem in polynomieller Zeit
- Löse beliebige Instanzen des Problems

ρ-Approximationsalgorithmus:

- Läuft in polynomieller Zeit.
- Löst beliebige Instanzen des Problems.
- Findet eine Lösung, die höchstens Faktor ρ weg von Optimum ist (Beispiel $\rho=2$: Lösung 2x so schlecht wie Optimum)

Herausforderung: Lösung sollte möglichst nah an Optimum sein.

Anwendungsproblem: Lastbalancierung

Eingabe: m identische Maschinen, n Jobs. Job i hat Laufzeit t_i.

Einschränkungen:

- Ein einmal ausgeführter Job muss bis zum Ende auf derselben Maschine ausgeführt werden.
- Jede Maschine kann höchstens einen Job gleichzeitig bearbeiten.

Definition: Sei J[i] die Teilmenge der Jobs, die Maschine i zugewiesen werden. Dann ist $L[i] = \sum_{j \in J[i]} t[j]$ die Last der Maschine i.

Definition: Der Makespan L ist die maximale Last aller Maschinen, d.h. $L = \max_i L[i]$

Lastbalancierung: finde Zuweisung, die Makespan minimiert

Lastbalancierung: List Scheduling

List-Scheduling Algorithmus:

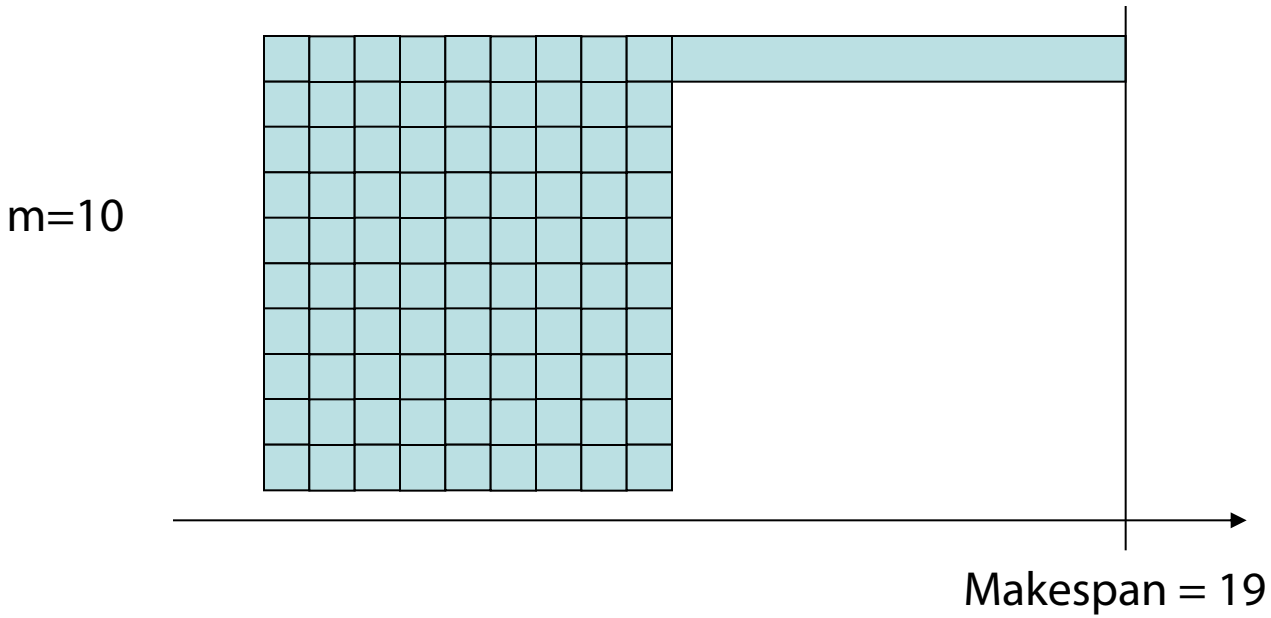
- Betrachte n Jobs in einer festen Reihenfolge und m Maschinen
- Weise Job j der Maschine mit z.Zt. geringster Last zu

```
function list_scheduling(m::Int, n::Int, t::Vector{Int})
  L = Vector(undef, m); J = Vector(undef, m)
  for i = 1:m      L[i] = 0; J[i] = Int[] end
  for j = 1:n
    i = findmin(L).    # naive Implementierung
    push!(J[i], j); L[i] = L[i] + t[j]
  end
  return J
end
```

Da $n \gg m$ Laufzeit: $O(n \log m)$ unter Verwendung einer Priority Queue zum Lösen des `findmin`-Problems

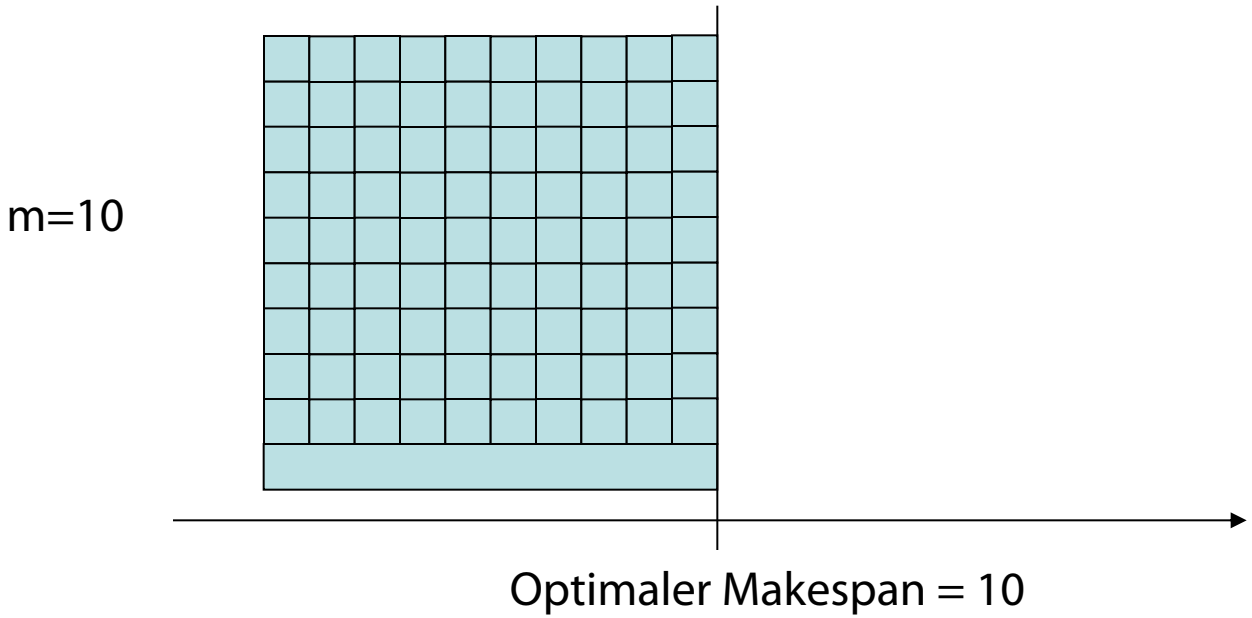
Lastbalancierung: List Scheduling

Beispiel: m Maschinen, m(m-1) Jobs der Länge 1,
ein Job der Länge m



Lastbalancierung: List Scheduling

Beispiel: m Maschinen, m(m-1) Jobs der Länge 1,
ein Job der Länge m



Lastbalancierung: List Scheduling

Theorem (Graham): Der Greedy Algorithmus ist 2-approximativ, die Lösung also kann eine Gesamtmakespan suggerieren, die doppelt so groß ist, wie bei der optimalen Lösung

Ohne Beweis

Zusammenfassung

- P und NP
- Gierige Algorithmen zur Approximation
  - Güte der Approximation sollte abschätzbar sein
- Anwendungsbeispiel: Lastbalancierungsproblem