Schneller Parsen

Neben allgemeinen Programmiertechniken gibt es eine Anzahl von Möglichkeiten, um Parser zu beschleunigen.

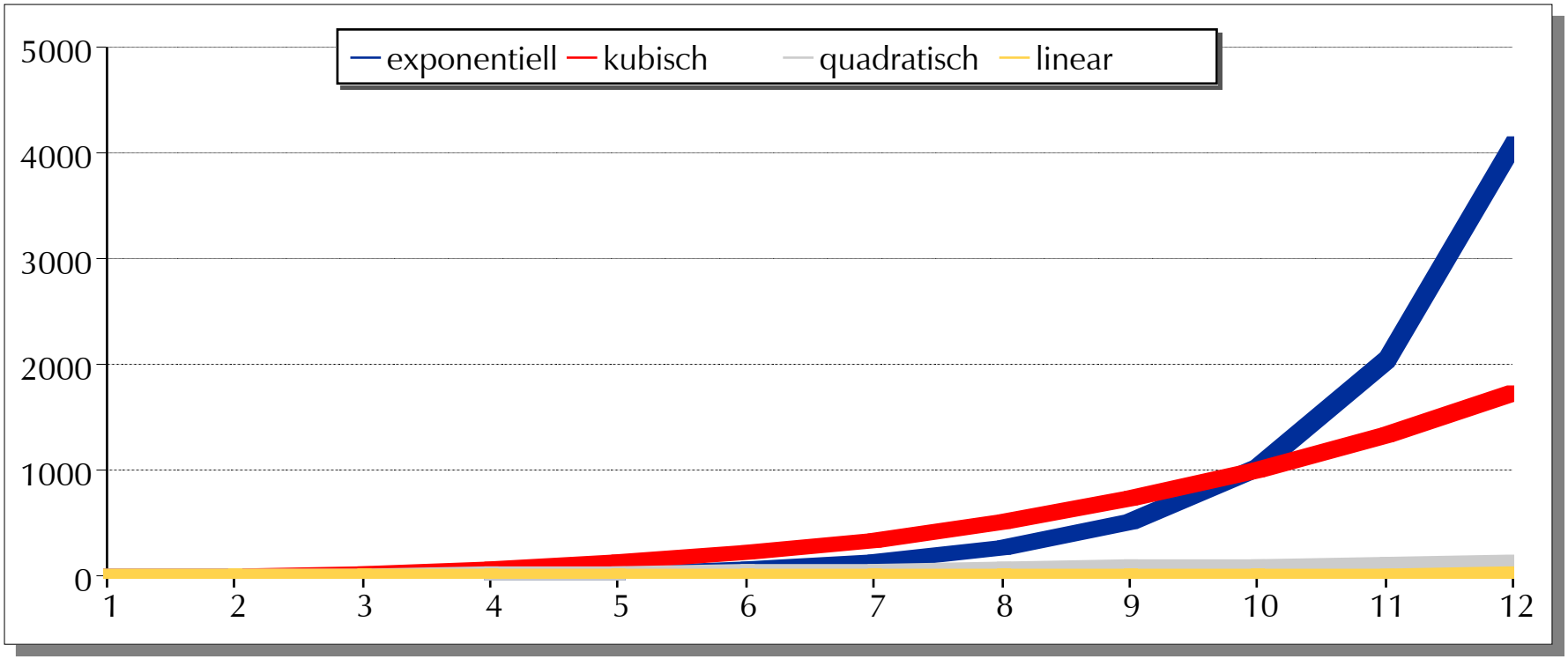
- Theorie und Praxis
  - Komplexitätseigenschaften von Parsing-Verfahren
  - Relevanz für die Praxis
  - einige Messresultate
- Grammatikflussanalyse
  - FIRST, FOLLOW, Erreichbarkeit, Produktivität
- Suchprozesse beim Chart-Parsing
- Empfehlungen
- Literatur

Theoretischer Ansatz: Komplexität

Um die Schwierigkeit von Problemen und die Effizienz von Verfahren zu beurteilen, wurde die Komplexitätstheorie entwickelt.

- Wie sehr steigt die Laufzeit/der Speicherbedarf *f* abhängig von der Länge der Eingabe *n*?
  - nachfolgend sind c_0, c_1, c_2, c_3, c_4, k irgendwelche Konstanten
- *linear* $f(n) = c_1 \cdot n + c_0$ $f = O(n)$
- *quadratisch* $f(n) = c_2 \cdot n^2 + c_1 \cdot n + c_0$ $f = O(n^2)$
- *kubisch* $f(n) = c_3 \cdot n^3 + c_2 \cdot n^2 + c_1 \cdot n + c_0$ $f = O(n^3)$
- *exponentiell* $f(n) = c \cdot k^{\text{Irgendein Polynom zur Basis n}}$ $f = O(k^n)$

Komplexität



	1	2	3	4	5	6	7	8	9	10	11	12
exponentiell	2	4	8	16	32	64	128	256	512	1024	2048	4096
kubisch	1	8	27	64	125	216	343	512	729	1000	1331	1728
quadratisch	1	4	9	16	25	36	49	64	81	100	121	144
linear	1	2	3	4	5	6	7	8	9	10	11	12

Komplexität des Parsings

Einfache Top-Down- und Bottom-Up-Verfahren besitzen exponentielle Komplexität.

- Beispiele für solche Verfahren:
  - In Prolog eingebauter DCG-Parser
  - Shift-Reduce-Verfahren

Der Earley-Algorithmus besitzt kubische Komplexität.

Theorie vs. Praxis

Wie wichtig sind für uns die Komplexitätseigenschaften von Parsing-Verfahren?

- Die Eingabelänge *n* ist selten gross
  - Es gibt kaum Sätze einer natürlichen Sprache mit 80, 800, 8000 Wörtern!
- Wenig untersucht: Wie steigt Zeit mit grösseren Grammatiken?
- Ergebnisse der Komplexitätstheorie beziehen sich fast immer auf den *schlechtesten* Fall, nicht auf den *durchschnittlichen* Fall
- sobald Nichtterminale beliebige Argumente tragen können, sind alle Verfahren exponentiell
  - allerdings: *beliebige* Argumente sind sehr selten nötig
- bis zu 90% der Zeit wird für die Unifikation, nicht für das eigentliche Parsing gebraucht

Theorie vs. Praxis

In der Komplexitätstheorie werden konstante Faktoren als vernachlässigbar angesehen.

- Weil die Eingabelängen aber bei der Verarbeitung natürlicher Sprache nicht gross variieren, spielen konstante Faktoren hier durchaus eine Rolle
- Eine gute Implementation kann um Grössenordnungen schneller sein als eine schlechte

Praktische Ergebnisse

[Covington, 1994], S. 191

- 24 kurze Sätze 100 mal mit extrem einfacher Grammatik parsen
- ***sehr* unrealistisches Szenario!**
  - Ergebnisse scheinen, verglichen mit anderen Arbeiten, bei weitem zu optimistisch

Verfahren	ALS Prolog 80386, 20 MHz	Quintus Prolog SparcStation 1+
Prolog-DCG	1.4 ms	0.1 ms
Top-Down	2.5 ms	0.5 ms
Shift-Reduce	16.0 ms	3.5 ms
Bot.-Up-Chart	29.7 ms	13.4 ms
Earley	412 ms	71.7 ms

Praktische Ergebnisse

Zusammenstellung von Parsing-Geschwindigkeiten:

- Quelle: [Abney, 1997]
- Traditionelle Chart-Parser: 1 Token/s
  - Jedoch [Carroll, 1994]: ca. 30 Token/s
- Partielle Parser, nicht-deterministisch: 20 – 50 Token/s
- Deterministische Parser: 400 – 2700 Token/s

Praktische Ergebnisse

Faktoren wie Prozessor-Caches, Befehls-Pipelines etc. können eine grosse Rolle spielen.

- nicht-deterministisches Erkennen von Nominalgruppen in Part-of-Speech-getaggten Texten durch PowerPC-Maschinencode, der von spezialisiertem Compiler erzeugt wurde (Quelle: [Brawer, 1998])
- *nicht* inbegriffen: Zeit für Lexikonzugriffe und Tagging

Anzahl Tokens	PPC 601, 66 Mhz Tokens/s	PPC 604, 150 MHz Tokens/s	PPC G3, 334 MHz Tokens/s
4092	1.9 Mio.	8.4 Mio.	21.3 Mio.
2894	2.0 Mio.	7.9 Mio.	19.0 Mio.
681	2.1 Mio.	6.7 Mio.	17.5 Mio.

Grammatikflussanalyse

Aus einer kontextfreien Grammatik können Tabellen abgeleitet werden, die später das Parsing beschleunigen.

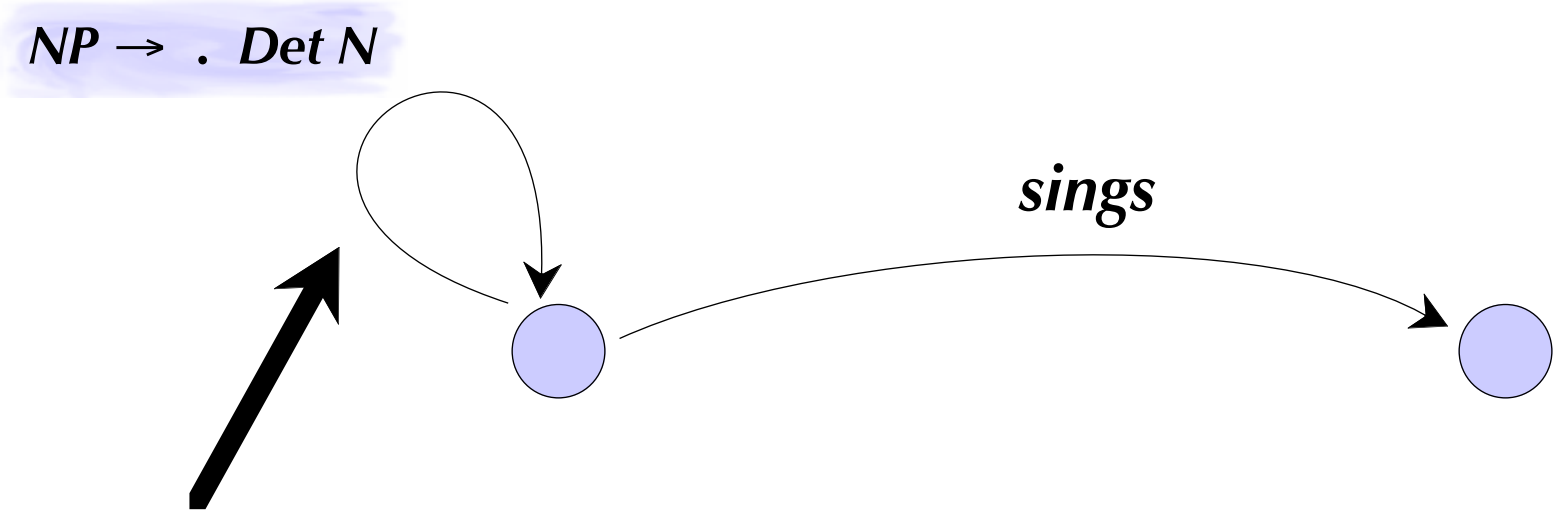
Dabei wird der »Fluss« der Grammatik vorhergesagt.

- Vermeiden überflüssiger aktiver Kanten
- Vermeiden überflüssiger inaktiver Kanten
- Verzicht auf unnötige Grammatik-Regeln

Auf diese Weise können in einem Earley-Parser etwa die Hälfte der Kanten vermieden werden.

- Quelle: [Russi, 1990]

Grammatikflussanalyse: *FIRST*



Beispiel für eine überflüssige aktive Kante:

- Annahme: *sings* ist niemals das erste Wort einer *NP*
- also könnte der Parser sich gleich sparen, eine *NP* vorauszusagen, wenn das nächste Wort *sings* ist

Grammatikflussanalyse: *FIRST*

Für jedes Nichtterminal *X* ist *FIRST*(*X*)

- die Menge aller Terminalsymbole, die an erster Stelle einer Konstituente der Kategorie *X* erscheinen kann

$FIRST(S) = \{the, a, man, woman\}$

$FIRST(NP) = \{the, a, man, woman\}$

$FIRST(VP) = \{sings\}$

$FIRST(DetP) = \{the, a\}$

$FIRST(Det) = \{the, a\}$

$FIRST(N) = \{man, woman\}$

$FIRST(V) = \{sings\}$

$S \rightarrow NP\ VP$	$Det \rightarrow the$
$NP \rightarrow DetP\ N$	$Det \rightarrow a$
$VP \rightarrow V\ NP$	$N \rightarrow man$
$DetP \rightarrow Det$	$N \rightarrow woman$
$DetP \rightarrow \varepsilon$	$V \rightarrow sings$

(linguistisch unsinnige) Grammatik

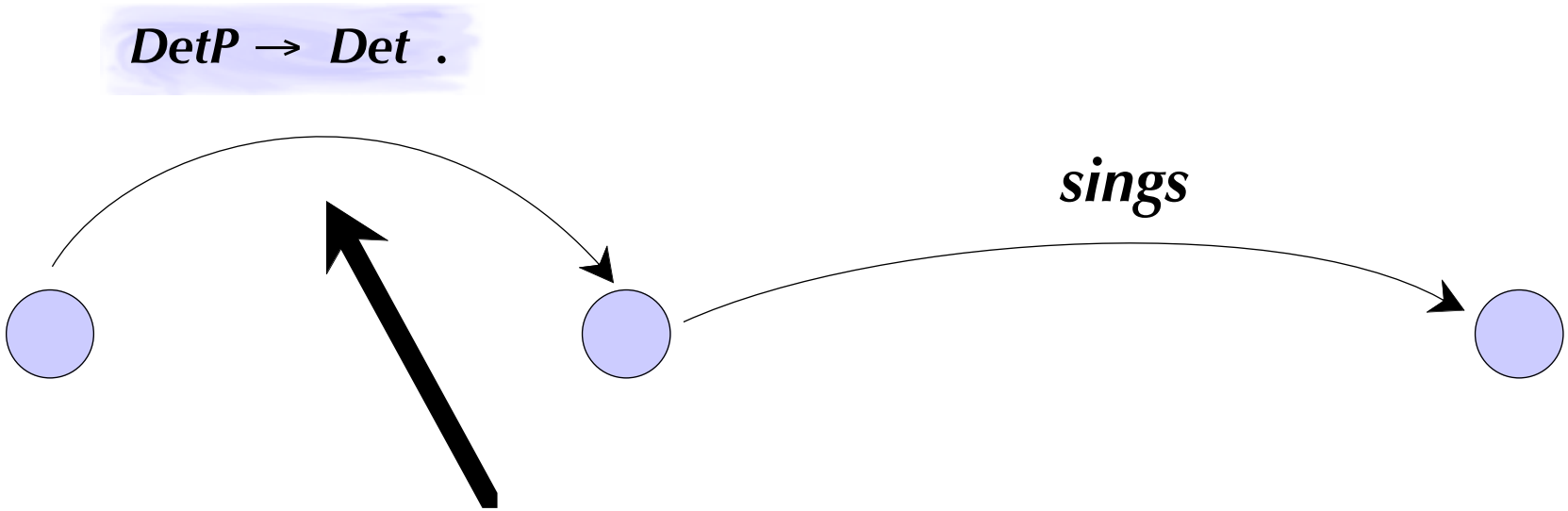
FIRST umfasst in der Literatur üblicherweise auch Nichtterminale

Grammatikflussanalyse: *FIRST*

Beim Einfügen einer aktiven Kante $X \rightarrow \alpha \,.\, B\ \gamma$:

- Ist das dem Endknoten folgende Wort Element von *FIRST*(*B*)?
- Wenn nein, kann die Kante weggelassen werden, da sie nicht zum Ziel führen würde.
- (kleinere und leicht lösbare) Komplikationen:
  - aktuelles Wort kann mehrere Lesarten haben
  - meistens sind Wörter gar nicht in der Grammatik, sondern werden durch Morphologie-Komponente bearbeitet

Grammatikflussanalyse: *FOLLOW*



Beispiel für eine überflüssige inaktive Kante:

- Annahme: *sings* steht nie direkt nach einer *DetP*
- also könnte der Parser sich gleich sparen, eine *DetP* zu »completen«, wenn das nächste Wort *sings* ist

Grammatikflussanalyse: *FOLLOW*

Für jedes Nichtterminal *X* ist *FOLLOW*(*X*)

- die Menge aller Terminalsymbole, die unmittelbar einer Konstituente der Kategorie *X* folgen können

$FOLLOW(S) = \{\ \}$

$FOLLOW(NP) = \{sings\}$

$FOLLOW(VP) = \{\ \}$

$FOLLOW(DetP) = \{man, woman\}$

$FOLLOW(Det) = \{man, woman\}$

$FOLLOW(N) = \{sings\}$

$FOLLOW(V) = \{\ \}$

$S \rightarrow NP\ VP$	$Det \rightarrow the$
$NP \rightarrow DetP\ N$	$Det \rightarrow a$
$VP \rightarrow V\ NP$	$N \rightarrow man$
$DetP \rightarrow Det$	$N \rightarrow woman$
$DetP \rightarrow \varepsilon$	$V \rightarrow sings$

(linguistisch unsinnige) Grammatik

FOLLOW umfasst in der Literatur üblicherweise auch Nichtterminale

Grammatikflussanalyse: *FOLLOW*

Beim Einfügen einer inaktiven Kante $B \rightarrow \beta$. :

- Ist das dem Endknoten folgende Wort Element von *FOLLOW*(*B*)?
- Wenn nein, kann die Kante weggelassen werden, da sie nicht zum Ziel führen würde.
- (kleinere und leicht lösbare) Komplikationen:
  - dem letzten Knoten in der Chart folgt kein Wort, daher macht der FOLLOW-Test dort keinen Sinn
  - sonst wie bei *FIRST*

GFA: Erreichbarkeit, Produktivität

In grossen Grammatiken kann es geschehen, dass bestimmte Nichtterminale unnötig sind.

- *X* ist vom Startsymbol aus nicht erreichbar *(not reachable)*
  - keine mögliche Ableitung des Startsymbols enthält *X*
- *X* ist unproduktiv *(unproductive)*
  - von *X* kann keine Kette abgeleitet werden, die nur aus Terminal-Symbolen besteht

Wenn eine dieser Bedingungen zutrifft,

- kann *X* nicht Bestandteil einer erfolgreichen Syntaxanalyse sein
- also können Regeln für *X* aus der Grammatik entfernt werden
  - Eigentlich ist dies ein Hinweis auf einen linguistischen Fehler
  - Ein System zur Grammatik-Entwicklung könnte entsprechende Warnungen ausgeben

Suchprozesse beim Chart-Parsing

Neben den Unifikationen wird beim Chart-Parsing viel Zeit für Suchvorgänge verbraucht.

- Beim Einfügen einer inaktiven Kante der Kategorie *B*:
  - Suche nach aktiver Kante, die als nächstes *B* benötigt
  - Suche nach allen Grammatik-Regel, deren rechte Seite mit *B* beginnt (bei Bottom-Up-Prediction)
- Beim Einfügen einer aktiven Kante, die nach *B* sucht:
  - Suche nach inaktiver Kante der Kategorie *B*
  - Suche nach allen Grammatik-Regeln für *B* (bei Top-Down-Prediction)

Suchprozesse beim Chart-Parsing

Empfehlungen:

- Aktive und inaktive Kanten in separaten Strukturen speichern
  - Es macht keinen Sinn, sämtliche Kanten zu durchsuchen, wenn von Anfang an klar ist, dass nur aktive (bzw. inaktive) in Frage kommen
- Kanten so speichern, dass sie schnell gefunden werden können (»indizieren«, *hashing*)
  - aktive Kanten: wenn Endknoten und gesuchtes Symbol bekannt sind
  - inaktive Kanten: wenn Anfangsknoten und Kategorie bekannt sind
- Grammatikregeln passend indizieren
  - bei Top-Down-Prediction: nach linker Regelseite
  - bei Bottom-Up-Prediction: nach erstem Symbol der rechten Regelseite
- FIRST und FOLLOW berücksichtigen

Literatur

Komplexitätstheorie allgemein

- Praktisch jede Einführung in die theoretische Informatik, z. B. Harry L. Lewis/Christos H. Papadimitriou: Elements of the Theory of Computation. London: Prentice-Hall, 1981. ISBN 0-13-273426-5. ca. CHF 77.—
  - gutes Verständnis der Berechenbarkeitstheorie ist Grundlage für Komplexitätsth.

Komplexität und die Praxis der Sprachverarbeitung

- John Carroll: Relating Complexity to Practical Performance in Parsing with Wide-Coverage Unification Grammars. In *32rd Meeting of the Association for Computational Linguistics*, 27. – 30. 6. 1994, Las Cruces, New Mexico, USA

Literatur

Effizienz partieller Parsing-Verfahren

- Steven Abney: Partial Parsing via Finite-State Cascades. In *Natural Language Engineering* **2** (4), 1997, pp. 339 – 346.
- Sascha Brawer: Patti – Compiling Unification-Based Finite-State Automata into Machine Instructions for a Superscalar Pipelined RISC Processor. März 1998.

Grammatikflussanalyse

- Reinhard Wilhelm/Dieter Maurer: Übersetzerbau – Theorie, Konstruktion, Generierung. Berlin/Heidelberg u.a.: Springer, 1992. ISBN 3-540-55704-0. S. 238 – 261
  - empfehlenswert, erfordert aber fundierte Informatik-Kenntnisse

Literatur

Grammatikflussanalyse: Empirische Versuche

- Thomas Russi: A Framework for Syntactic and Morphological Analysis and its Application in a Text-to-Speech System. Dissertation ETH Zürich Nr. 9328, Institut für Elektronik, 1990.
  - Zählt bei einer grösseren Grammatik anhand realistischer Testsätze, wie oft die FIRST- und FOLLOW-Tests fehlschlagen (und somit überflüssige Kanten verhindern)

Literatur

Effiziente Systeme der Sprachverarbeitung

- Zeitschrift »Natural Language Engineering«, Cambridge University Press, ISSN 1351-3249

Effiziente Algorithmen allgemein

- Robert Sedgewick: Algorithmen. Bonn/München u.a.: Addison-Wesley, 1991. ISBN 3-89319-301-4.
  - Algorithmen für Sortieren, Suchen, String-Verarbeitung, geometrische Probleme, Graphen, mathematische Verfahren u.a.
  - Schwerpunkt auf praktischer Anwendbarkeit (somit z.B. keine Korrektheitsbeweise)
  - sehr verständlich geschrieben
  - Programmiersprache: Pascal (es gibt auch eine Version für C)
  - 742 Seiten

Aufgaben: Schneller Parsern

Programmiertechniken der Computerlinguistik 2 · Sommersemester 1999

1. Grammatikflussanalyse

Gegeben sei folgende kontextfreie Grammatik.

S → *NP VP*	*VP* → *V*
VP → *V NP*	*NP* → *Det AdjP N*
AdjP → *Adj*	*AdjP* → *ε*
Det → *der*	*Det* → *den*
Det → *die*	*Det* → *das*
N → *Torte*	*N* → *Krümelmonster*
V → *isst*	*V* → *erblickt*
Adj → *leckeren*	*Adj* → *gierige*

Bestimme für jedes Nichtterminalsymbol die Mengen *FIRST* und *FOLLOW*.

2. Suchprozesse beim Chart-Parsing

Wie könnte die Effizienz des früher vorgestellten Bottom-Up-Chart-Parsers verbessert werden? Überlege Dir hierzu insbesondere, welche Suchprozesse stattfinden und wie diese beschleunigt werden könnten.

Diese Aufgabe ist sicher eine der schwierigsten, die in diesem Semester gestellt werden — dafür kann man gerade durch solche Überlegungen auch viel lernen … Lass Dich jedenfalls nicht abschrecken!