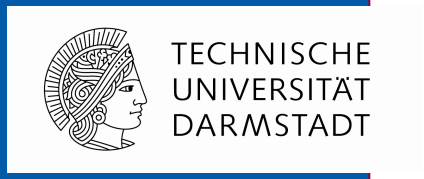
Optimierung in dynamischer Umgebung



(Dozent: PD Dr. Ulf Lorenz)

Literatur und Danksagung

TECHNISCHE UNIVERSITÄT DARMSTADT

Literatur:

s. Webseiten der Veranstaltung

Dank

Für Anregungen und die Erlaubnis Unterlagen nutzen zu dürfen, möchte ich mich bedanken bei:
Prof. Dr. Meyer auf der Heide, Uni Paderborn,
Prof. Dr. Schindelhauer von der Uni Freiburg,
Prof. Dr. Ziegler, TU Darmstadt

Übersicht II: Ausgangssituation Optimierung

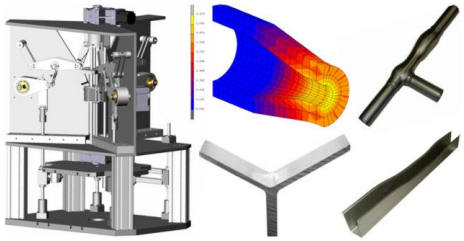
TECHNISCHE UNIVERSITÄT DARMSTADT

Häufige Annahme für Planung und Entscheidungsfindungen in logistischen Prozessen und in Herstellungsprozessen:

- im vorhinein bestimmbare Daten

Beobachtende und auf Planabweichungen fexibel reagierende Kontrollstrategie

- existiert entweder nicht oder
- wird von der Planung getrennt betrachtet.



Folgen der Unsicherheit

- große Planabweichungen

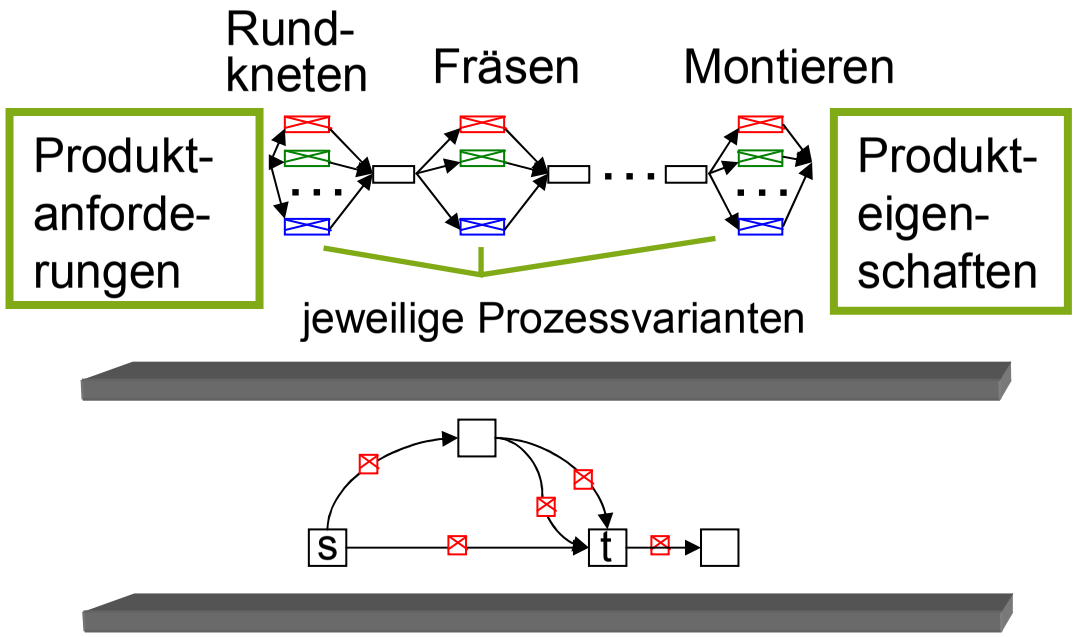
oder

- hohe Sicherheitsbeiwerte
- Überdimensionierung
- große Pufferbildung

Übersicht II: Arbeitsprogramm im SFB 805

TECHNISCHE UNIVERSITÄT DARMSTADT

Modellierung und Optimierung



$$\max_{y_1^{(u,v)} \in \square^{n_1}} c_1^T y_1 + \mathrm{Erw}[Q_2(y_1^{(u,v)}; \overline{\eta}_2)]$$

$$\text{s.t.} \sum_{v \in \delta^+(s)} y_1^{(s,v)} = 1$$

$$\text{mit } Q_t\left(y_{t-1}^{(u,v)}; \overline{\eta}_t(\omega)\right) := \max_{y_t^{(u,v)} \in \mathbb{R}^{n_t}} c_t(\omega)^T y_t^{(u,v)} + \mathrm{Erw}[Q_{t+1}\left(y_t^{(u,v)}; \overline{\eta_{t+1}}\right)]$$

$$\text{s.t.} \sum_{v \in \left(\delta^+(u)(\omega)\right)} y_t^{(u,v)} - \sum_{u \in \left(\delta^-(v)(\omega)\right)} y_{t-1}^{(u,v)} = 0$$

n-stufige, modulare Prozesskette mit verschiedenen Entscheidungs-alternativen

Netz von Entscheidungs-alternativen

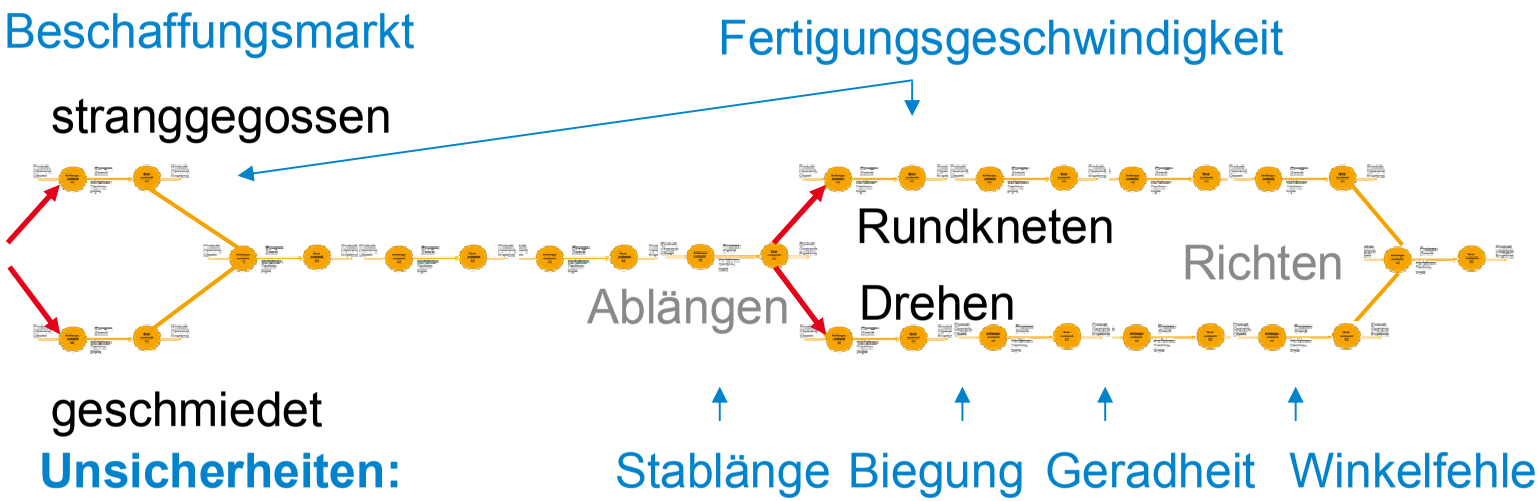
mehrstufiges stochastisches Optimierungsmodell

Übersicht II: Ziel

TECHNISCHE UNIVERSITÄT DARMSTADT

Entscheidungsoptimierung unter Unsicherheit

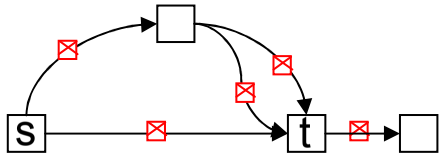
Durch nicht vorhersagbare Eingabedaten entstandene
Unsicherheiten
mit Hilfe mathematischer Modelle und Optimierungsverfahren
beherrschen und ihre Auswirkungen minimieren.



Übersicht II: Andere Modellierungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Modellierung und Optimierung, Alternativen



$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \exists x_5 \ldots$

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \leq b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{24}x_4 \leq b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{35}x_5 \leq b_3$$

$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \exists x_5 \ldots$

$$a_{11}x_1x_2 + a_{13}x_3 \leq b_1$$
$$a_{21}x_1 + a_{22}x_2x_4 \leq b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{35}x_5 \leq b_3$$

Algorithmen:

- ganzzahlig (random / worst case):
  „von Aussen nach Innen“
  mittels backtracking

- kontinuierlich (random / worst case):
  „von Innen nach Aussen“
  mittels Variablemelemination

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einführung in Komplexitätstheorie

- Dynamic Graph Reliability Probleme
- Schach: Lösungsalgorithmen und Näherungsideen
- Go: UCT Lösungsverfahren
- Sokoban, Rushhour und Stackingprobleme

- Satz von Savich, speziell: NPSPACE = DPSPACE

- Stochastic Programming
- Quantifizierte Lineare Programme

Übersicht I

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einführung in Komplexitätstheorie
  - Unentscheidbarkeit

    Welche Probleme können mit einem Algorithmus gelöst werden? Was ist überhaupt ein “Algorithmus”, was ist ein “Problem”?

  - verschiedene Maschinenmodelle und formale Sprachen
  - Algorithmen, Komplexitätsklassen P, NP, PSPACE
  - Reduktionstechnik

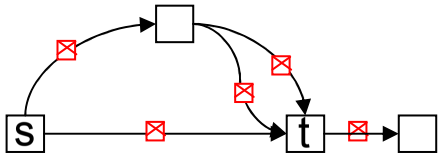
Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Das Dynamic Graph Reliability Problem

  geg.: ein DAG (gerichteter Graph ohne Kreise); Regeln, nach denen Kanten im Graphen ausfallen; Startknoten, Endknoten

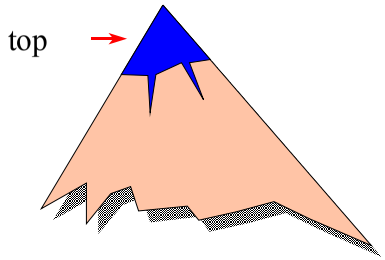
  ges.: Gibt es eine Gewinnstrategie, die einem Walker im Startknoten erlaubt, an den Endknoten zu gelangen?



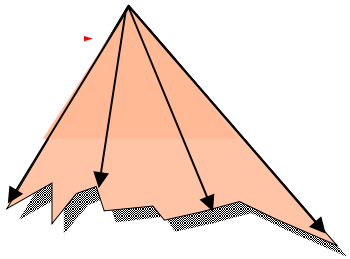
Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schach: Lösungsalgorithmen und Näherungsideen



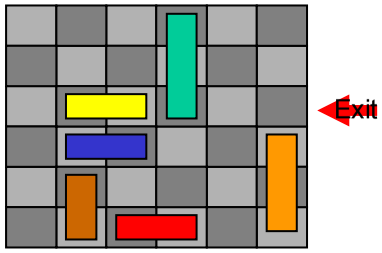
- Go: UCT Lösungsverfahren

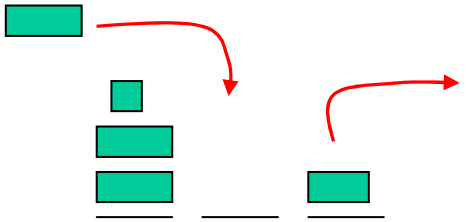


Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sokoban, Rushhour und Stackingprobleme





- Satz von Savich, speziell: NPSPACE = DPSPACE
- Stochastic Programming

$$\max_{y_1^{(u,v)} \in \square^{n_1}} c_1^T y_1 + \text{Erw}[Q_2(y_1^{(u,v)}; \overline{\eta}_2)]$$

$$\text{s.t.} \sum_{v \in \delta^+(s)} y_1^{(s,v)} = 1$$

$$\text{mit } Q_t\left(y_{t-1}^{(u,v)}; \overline{\eta}_t(\omega)\right) := \max_{y_t^{(u,v)} \in \mathbb{R}^{n_t}} c_t(\omega)^T y_t^{(u,v)} + \text{Erw}[Q_{t+1}\left(y_t^{(u,v)}; \overline{\eta_{t+1}}\right)]$$

$$\text{s.t.} \sum_{v \in \left(\delta^+(u)(\omega)\right)} y_t^{(u,v)} - \sum_{u \in \left(\delta^-(v)(\omega)\right)} y_{t-1}^{(u,v)} = 0$$

- Quantifizierte Lineare Programme

Es fängt ganz harmlos an

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ein Alphabet Σ besteht aus einer endlichen Menge von Zeichen, z.B.
  - Σ_1 = {a,b,c}
  - Σ_2 = {0,1}
  - Γ = {0,1,x,y,z}
- Eine Zeichenkette (String/Wort) ist eine endliche Folge (Tupel) von Zeichen, z.B.
  - w = abba
    - Notation: w_1=a, w_2=b, w_3=b, w_4=a
    - Die Länge eines Worts wird mit |w| beschrieben: |w| = 4
- Σ^* bezeichnet die Menge aller Zeichenketten über Alphabet Σ
  - z.B.: “abba“ $\in$ {a,b}*
  - Die leere Zeichenkette wird mit ε bezeichnet.
    - Es gilt: $|\varepsilon| = 0$
- Eine Teilmenge von Σ^* wird als Sprache bezeichnet

Sprachbeschreibungen und Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

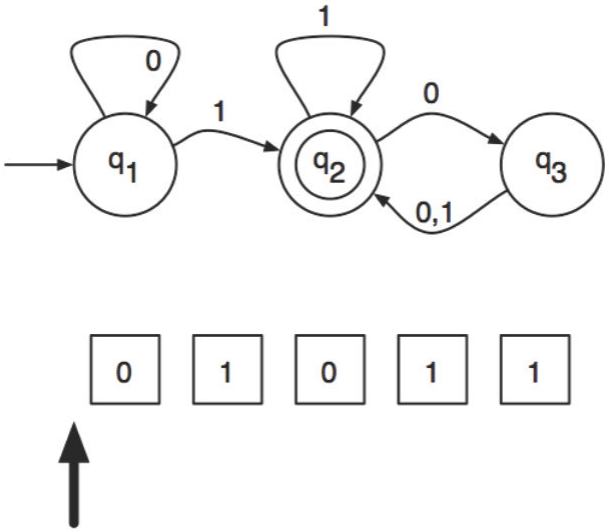
- Eine Sprache $L \subseteq \Sigma^*$ muss nun irgendwie beschrieben werden.
  - z.B. durch einen *regulären Ausdruck*: (0*10*)
    - $\varnothing$ ist ein regulärer Ausdruck.
    - ε ist ein regulärer Ausdruck.
    - $\forall\ a_i \in \Sigma$ ist a_i ein regulärer Ausdruck.
    - Sind *x* und *y* reguläre Ausdrücke, so auch $x \cup y$, (*xy*) und *x* *.
    - Es gibt keine weiteren regulären Ausdrücke.

  - z.B. durch eine **Problembeschreibung**:
    - **Definition**: Ein *Entscheidungsproblem* ist ein input-output Tupel mit
      geg.: Kodierung eines Inputs einer Instanz, mittels Alphabet ∑
      ges.: ja/nein
    - Die Teilmenge aller Inputs, für die die Antwort “ja” ist, ist offenbar eine Sprache

Sprachbeschreibungen und Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Frage, ob ein $w \in \Sigma^*$ ein Wort aus einer Sprache $L \subseteq \Sigma^*$ ist, kann unterschiedlich schwierig zu lösen sein
  - Bsp. 1: In einem sehr einfachen Fall durch einen endlichen Automaten:
    0*1 (1 | 0(0|1))*



Sprachbeschreibungen und Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Formal ist ein deterministischer endlicher Automat (DEA) ein 5-Tupel

Def: Ein (deterministischer) endlicher Automat (**DFA**) ist ein 5-Tupel (Q,Σ,δ,q_0,F), wobei

- Q eine endliche Menge von Zuständen ist,
- Σ ein endliches Alphabet
- δ: $Q\times\Sigma\rightarrow Q$ die Übergangsfunktion,
- q_0 der Startzustand und
- $F\subseteq Q$ die Menge akzeptierender Endzustände.

Sprachbeschreibungen und Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

$A := (Q, \Sigma, \delta, q_0, F)$,

$Q := \{s, q_1, q_2, r_1, r_2\}$,

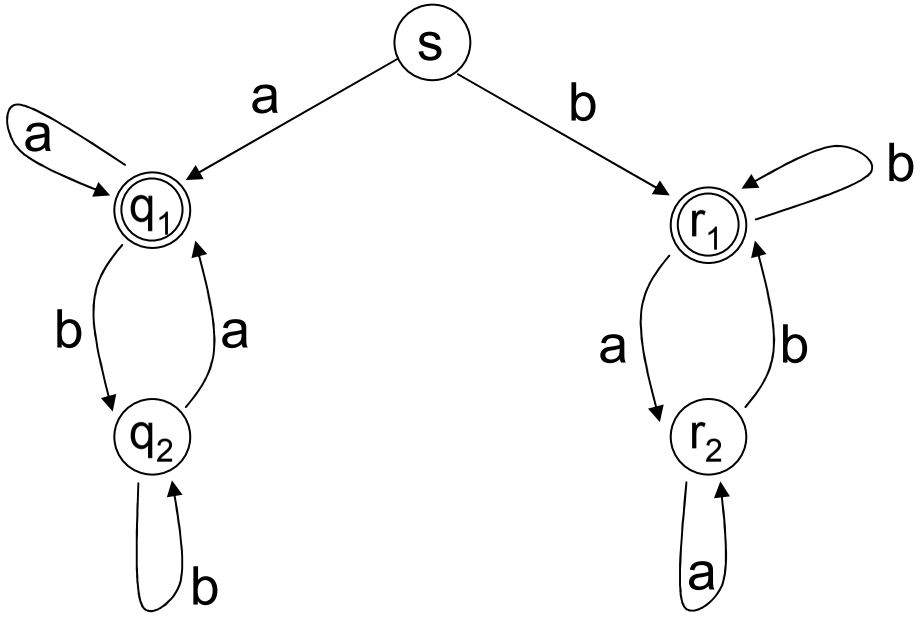
$\Sigma := \{a,b\}$,

$F := \{q_1,q_2\}$

$q_0 = s$

$L(A) = \{w_1w_2 \ldots w_n : w_i \in \{a,b\}, w_1 = w_n\}$

δ	s	q_1	q_2	r_1	r_2
a	q_1	q_1	q_1	r_2	r_2
b	r_1	q_2	q_2	r_1	r_1



Sprachbeschreibungen und Maschinen

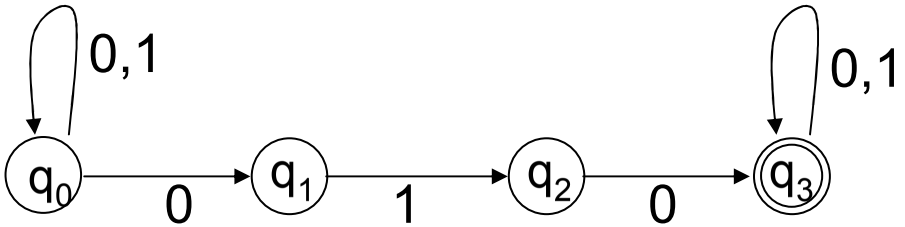
TECHNISCHE UNIVERSITÄT DARMSTADT

Formal ist ein nichtdeterministischer endlicher Automat (NEA) (ohne ε-Übergänge) ein 5-Tupel

Def: Ein (nichtdeterministischer) endlicher Automat (**NFA**) ist ein 5-Tupel (Q,Σ,δ,q_0,F), wobei

- Q eine endliche Menge von Zuständen ist,
- Σ ein endliches Alphabet
- δ: $Q\times\Sigma\rightarrow 2^Q$ die Übergangsfunktion,
- q_0 der Startzustand und
- $F\subseteq Q$ die Menge akzeptierender Endzustände.

Bsp.:



	0	1
q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	-	$\{q_2\}$
q_2	$\{q_3\}$	-
q_3	$\{q_3\}$	$\{q_3\}$

Sprachbeschreibungen und Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz: Sei N ein NFA und L = L(N). Dann gibt es einen DFA A mit L(A) = L

Beweis: Sei $N = (Q,\Sigma,\delta,q_0,F)$. Die folgende Konstruktion heißt auch "Potenzmengenkonstruktion". Um $A = (Q',\Sigma,\delta',q'_0,F')$ zu definieren setzen wir:

- $Q' = 2^Q$
- $q'_0 = \{q_0\}$
- $F' = \{R \in Q' \mid R \cap F \neq \{\}\}$
- $\delta'(R,a) = \bigcup_{r\in R} \delta(r,a) = \{q \in Q \mid \text{es gibt ein } r \in R \text{ mit } q \in \delta(r,a)\}$

Dann gilt:
$$w \in L(N) \Leftrightarrow \delta(q_0,w) \cap F \neq \{\}$$
$$\Leftrightarrow \delta(q'_0,w) \in F'$$
$$\Leftrightarrow w \in L(A)$$

Hierbei bedeutet $\delta(q,w)$, dass die Übergangsfunktion δ mehrfach auf das Wort w angwendet wird, Buchstabe für Buchstabe und startend bei Zustand q.

Sprachbeschreibungen und Maschinen

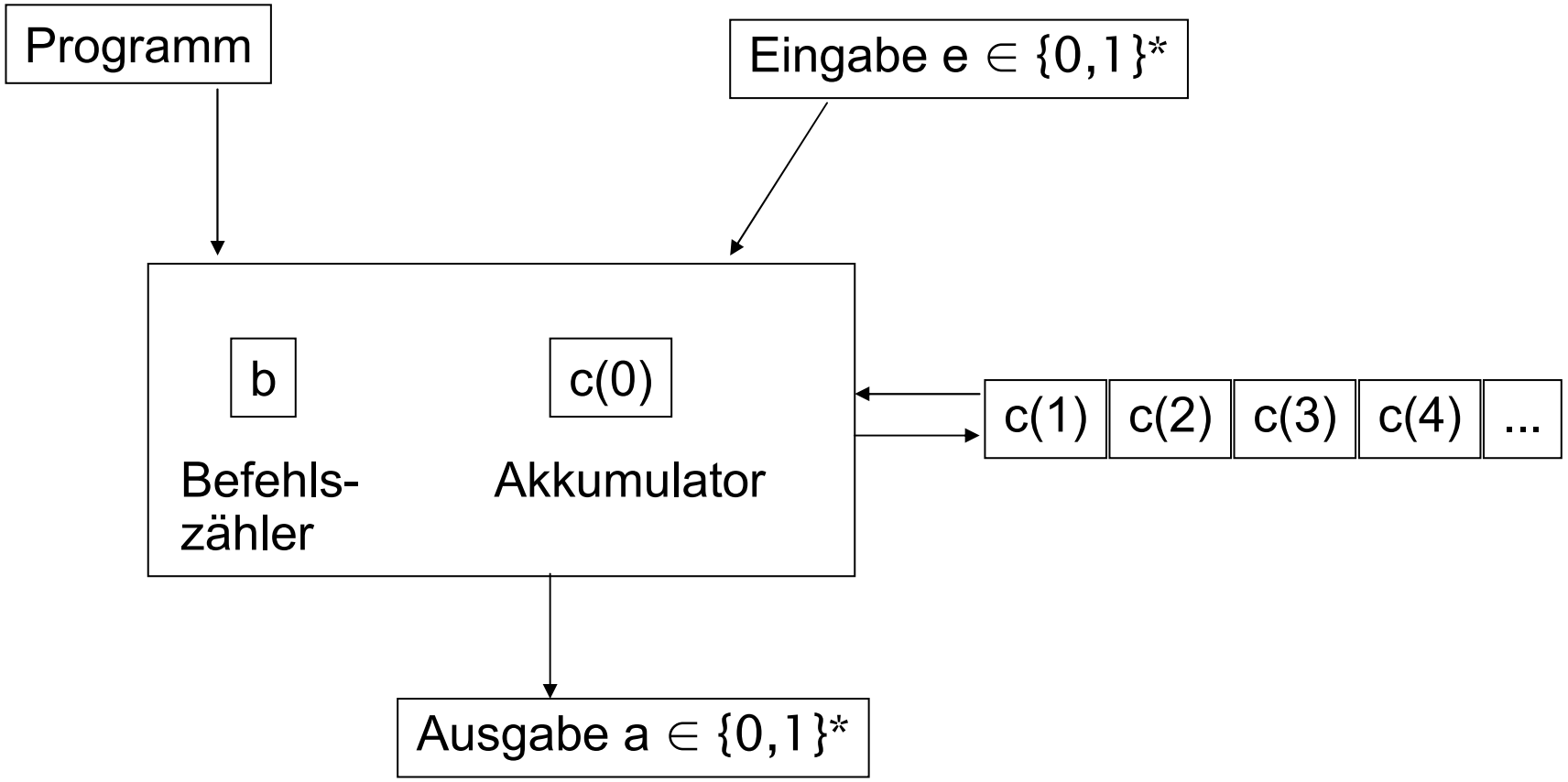
TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Frage, ob ein $w \in \Sigma^*$ ein Wort aus einer Sprache $L \subseteq \Sigma^*$ ist, kann unterschiedlich schwierig zu lösen sein
  - **Bsp 2.: In einem sehr komplizierten Fall ist sie nicht entscheidbar**:
    - **geg**: Codierung einer Random Access Machine (RAM, das entspricht in etwa einem herkömmlicher Computer mit unendlich viel Speicher), sowie ein $w \in \Sigma^*$
      Frage: Hält die RAM bei Eingabe w?

"nicht entscheidbar" heisst: es gibt keinen Algorithmus, der alle Instanzen das Problem lösen kann. (faszinierende Nebeneffekte, Busy Beaver)

Random Access Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT



Random Access Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein/Ausgabe

read	c(0) := head(e); e := e \ head(e); b := b+1; falls |e| > 0
	c(0) := EOF; b := b + 1; sonst
write	a := a c(0); b := b + 1;

Arithmetik

add x	c(0) := c(0) + c(x); b := b + 1;
sub x	c(0) := c(0) – c(x); b := b + 1; falls c(x) < c(0)
	c(0) := 0; b := b + 1; sonst
c *add* x	c(0) := c(0) + x; b := b + 1;
c *sub* x	analog: Operation mit Konstante c

Random Access Maschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Sprünge

goto j	b := j;
if (c(0) R i) *then goto* j;	b := $\begin{cases} j & \text{falls } c(0)\ R\ i \\ b+1 & \text{sonst} \end{cases}$, $R \in \{<, >, =, \leq, \geq\}$
end	Programm hält

Speicherzugriffe

direkt

load x	c(0) := c(x); b := b + 1;
store x	c(i) := c(0); b := b + 1;

indirekt

iload x	c(0) := c(c(x)); b := b + 1;
istore x	c(c(i)) := c(0); b := b + 1;

Turingmaschinen

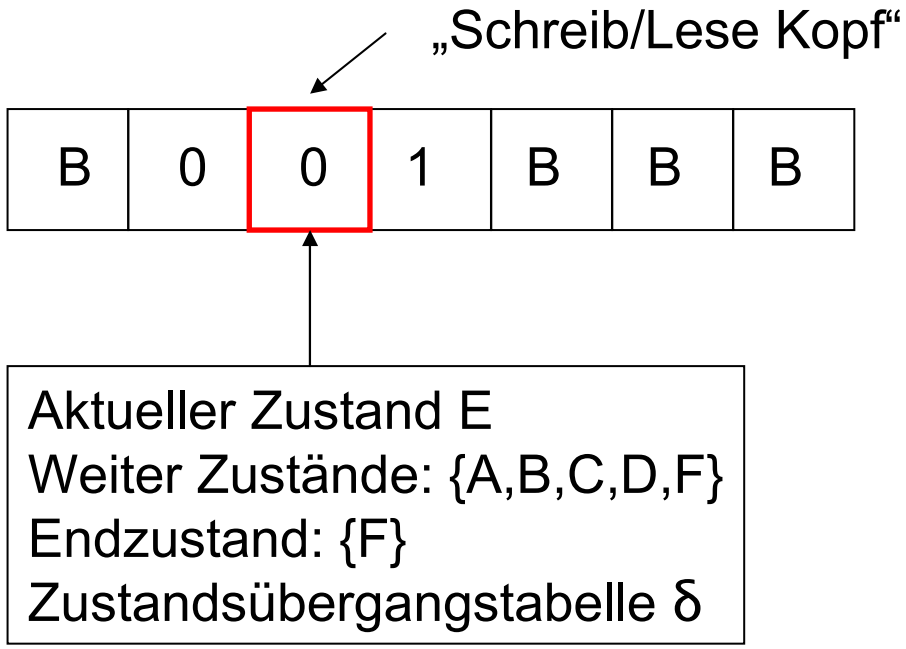
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Formal ist eine (1-Band) Turingmaschine ein 6-Tupel**

- **Def:** Eine (deterministischer 1-Band) Turingmaschine ist ein 6-Tupel $(Q,\Sigma,\Gamma,\delta,q_0,F)$, wobei
- Q eine endliche Menge von Zuständen ist,
- Σ ein endliches Alphabet
- $\Gamma := \Sigma \cup \{B\}$, B das so genannte Blank-Symbol
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{R,N,L\}$ die (partielle) Übergangsfunktion,
- q_0 der Startzustand und
- $F \subseteq Q$ die Menge akzeptierender Endzustände.

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT



„Programm“: Falls die Turingmaschine im Zustand q ist und das Zeichen a liest,
dann gehe in den Zustand q‘, überschreibe a durch a‘, und bewege den Kopf nach rechts, links oder gar nicht.

Schreibweise: $\delta(q,a) = (q',a',R)$

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Eine Mehrband- Turingmaschine ist ein 6-Tupel**

- **Def:** Eine (deterministischer 1-Band) Turingmaschine ist ein 6-Tupel $(Q,\Sigma,\Gamma,\delta,q_0,F)$, wobei
- Q eine endliche Menge von Zuständen ist,
- Σ ein endliches Alphabet
- $\Gamma := \Sigma \cup \{B\}$, B das so genannte Blank-Symbol
- $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{R,N,L\}^k$ die (partielle) Übergangsfunktion,
- q_0 der Startzustand und
- $F \subseteq Q$ die Menge akzeptierender Endzustände.

Theorem 1: RAM und Turingmaschine können sich gegenseitig simulieren.

Theorem 2:

Church-Turing Hypothese:

Die von jeglicher Maschine berechenbaren Funktionen sind genau die, die von Turingmaschinen berechenbar sind.

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Def.: Zeit- und Platzkomplexität

Sei M eine determistische Turingmaschine (DTM), die für jede Eingabe hält.
Zeitkomplexität: $T_M(x)$:= #Schritte, die M mit Eingabe x ausführt.
Platzkomplexität: $S_M(x)$:= # verschiedener Speicherzellen, die der Kopf von M bei Eingabe x besucht.

$T_M(n) = \max\{T_M(x) \mid x \in \Sigma^{\leq n}\}$

Seien $t,s: \mathbb{N} \rightarrow \mathbb{N}$ Abbildungen. Dann ist M
t(n)-zeitbeschränkt und s(n)-platzbeschränkt, falls
$T_M(n) \leq t(n)$ und $S_M(n) \leq s(n)$ für alle $n \in \mathbb{N}$.

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir sagen:
„asympototisch wächst f nicht stärker als g“, genau dann, wenn
$\exists\ k > 0, n_0 > 0\ \forall n > n_0 : f(n) \leq k*g(n)$

Man schreibt zudem auch $f(n) \in O(g(n))$, d.h.
$f(n) \in \{ h: \mathbb{N} \rightarrow \mathbb{N} \mid \exists\ k > 0, n_0 > 0\ \forall n > n_0 : f(n) \leq k*g(n) \}$

Satz: Jede t(n)-zeit, und s(n)-platzbeschränkte k-Band-DTM kann durch eine O(t(n)*s(n)) zeit- und O(s(n)) platzbeschränkte 1-Band DTM simuliert werden.
Satz: Jede t(n)-zeitbeschränkte RAM kann durch eine $O(t(n)^3)$ zeitbeschränkte DTM simuliert werden.

(ohne Beweis)

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Eine Turingmaschine heißt Zähler, wenn sie, gestartet mit bin(p), $p \in \mathbb{N}$,

bin(p-1), bin(p-2),...bin(1),bin(0)

hintereinander, immer auf dem gleichen Bandbereich erzeugt
und dann stoppt. Sei n = | bin(p) | die Länge der Binärdarstellung von p.

Satz: Es gibt einen O(n) platz- und $O(2^n)$ zeitbeschränkten Zähler.

Beweis: gemeinsam in Übung

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

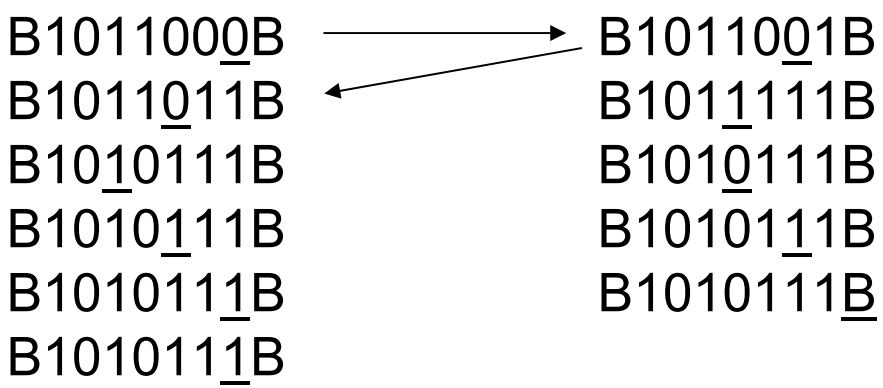
Eingabe bin(p) = $x_{n-1}...x_0$

s = p;
while s <> 0 do
  Berechne bin(s-1) aus bin(s) wie folgt:
  - gehe zum rechten Rand der Eingabe
  - solange eine 0 gelesen wird, überschreibe sie mit einer 1 und gehe nach links. Sobald eine 1 gelesen wird ersetze diese durch eine 0.
  - gehe mit dem Kopf eine Stelle weiter nach links. Falls ein B gelesen wird, gehe nach rechts, ersetze die 0 durch ein B und gehe an den rechten Rand. Sonst gehe, ohne ein Zeichen zu ändern, an den rechten Rand.
  - s = s-1;

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel: Eine 1 von 1011000 subtrahieren



- gehe zum rechten Rand der Eingabe
- solange eine 0 gelesen wird, überschreibe sie mit einer 1 und gehe nach links. Sobald eine 1 gelesen wird ersetze diese durch eine 0.

10 Schritte bzw, wenn a Nullen rechts stehen:
2a+4 Schritte

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

$\delta(q_0,0) = (q_0,B,R)$
$\delta(q_0,B) = (q_f,B,N)$
$\delta(q_0,1) = (q_1,1,R)$

ersetze führende Nullen duch Bs
falls nur Nullen: fertig
sonst:

$\delta(q_1,0) = (q_1,0,R)$
$\delta(q_1,1) = (q_1,1,R)$
$\delta(q_1,B) = (q_2,B,L)$

gehe an den rechten Rand der Eingabe,
und dann vor das letzte B

$\delta(q_2,0) = (q_2,1,L)$
$\delta(q_2,1) = (q_3,0,L)$

solange eine 0 gelesen wird, wird 1 geschrieben
wenn eine 1 gelesen wird, wird 0 geschrieben und ...

$\delta(q_3,B) = (q_0,B,R)$
$\delta(q_3,0) = (q_1,0,R)$
$\delta(q_3,1) = (q_1,1,R)$

falls dort ein B steht, prüfe, ob fertig.
sonst gehe zum rechten Rand

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Beachte: mit n = | bin(p) | gilt: $2^{n-1} \leq p < 2^n$, bzw. $n-1 \leq \log(p) < n$

$S_M(n) = n+2 = O(n)$

1x dekrementieren: falls rechts a Nullen stehen, $\leq 2a+4$ Schritte
-> p-mal dekrementieren bei Zeit höchstens 2n+4

Also: $T_M(n) = O(n\ 2^n)$

Frage: dauert das wirklich so lange?

Antwort: nein, denn die meisten Dekrements sind viel schneller.

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

In 50% aller Fälle (beim runterzählen) steht eine 1 am Schluß. D.h. a=0
In 25% aller Fälle steht eine 10 am Schluß. D.h. a=1
In 12,5% aller Fälle steht eine 100 am Schluß. D.h. a=2
In 6,25% aller Fälle steht eine 1000 am Schluß. D.h. a=3
.....

Im Durchschnitt sind das nicht mehr als $\sum_{a\geq 0} (2a+4)*2^{-a-1}$

$= \sum_{a\geq 0} 2a*2^{-a-1} + 4*2^{-a-1}$ viele Schritte.

$= \sum_{a\geq 0} a*2^{-a-1} + \sum_{a\geq 0} 2^{-a+1}$

$= \quad 2 \quad + \quad 4$

Also ein Dekrement in durchschnittlich 6 Schritten.
Laufzeit $O(2^n)$

Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

Def.:
Eine Sprache L heißt entscheidbar, wenn es eine Turingmaschine gibt, die zu jeder Eingabe w $\in \Sigma^*$ nach endlicher Zeit anhält, und genau dann in einem akzeptierenden Zustand endet, wenn w $\in$ L gilt.

Eine Sprache L heißt semi-entscheidbar, wenn es eine Turingmaschine gibt, die zu jeder Eingabe w $\in$ L nach endlicher Zeit in einem akzeptierenden Endzustand anhält.

Eine Funktion f heißt berechenbar, wenn es eine Turingmaschine gibt, die für alle Eingaben x, die aus dem Definitionsbereich von f stammen nach endlich vielen Schritten anhält und f(x) auf das Band schreibt.

Unendscheidbarkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

Gibt es unendscheidbare Sprachen?

Ja, denn es gibt nur abzählbar unendlich viele Turingmaschinen, aber überabzählbar viele Sprachen $L \subseteq \{0,1\}^*$

Begründung mit Hilfe des Cantor‘schen Diagonalisierungsverfahrens:

	M_1	M_2	M_3	M_4	... →
0	n	j	j	n	
1	n	n	n	j	
01	j	j	j	n	
...					
x_i ↓					

Eintrag (M_i,x_k)=„j“ bedeutet, dass x_k aus der Sprache $L(M_i)$ ist. Sei nun L die Sprache, die genau aus den Wörtern besteht, bei denen beim Eintrag (M_i,x_i) „n“ steht. L gehört zu keiner der aufgeführten TMs.

Berechenbarkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

Gibt es Funktionen, die nicht von einer Turingmaschine berechnet werden können?
Ja.

Die Busy-Beaver Funktion $\Sigma(n)$ ist definiert als die Anzahl der Einsen, die eine Champion-Turingmaschine auf ein zu Beginn leeres Band ausgibt, wobei n die Anzahl der erlaubten Zustände darstellt. Die TM muss irgendwann halten. Wir gehen weiterhin davon aus, dass diese Einsen alle zusammenhängend sein müssen.

Beweis:

Berechenbarkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: Die Busy-Beaver Funktion $\Sigma(n)$ ist berechnbar, und EVALΣ ist die TM, die $\Sigma(n)$ berechnet. Bei einer Eingabe von n Einsen schreibt sie $\Sigma(n)$ Einsen auf das Band und hält dann an.

Im folgenden definieren wir 4 Hilfs-TMs.

Sei INC eine TM, die bis zum ersten B nach rechts läuft, dort eine 1 schreibt und dann hält.

DOUBLE sein eine andere TM, die die Anzahl Einsen, die sich auf dem Band befinden verdoppelt. DOUBLE berechnet also zu Eingabe n n+n.

Wir bilden nun eine neue TM: DOUBLE | EVALΣ | INC
Die Anzahl der Zustände dieser Maschine sei n_0

Berechenbarkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

Sei CREATE_n_0 eine weitere TM, welche n_0 Einsen auf ein leeres Band schreibt. Diese TM gibt es, trivialerweise eine mit n_0 vielen Zuständen.

Sei nun $N := n_0 + n_0$

Das Finale: Sei BADΣ folgende TM:

$$\underbrace{\text{CREATE\_}n_0}_{n_0} \mid \underbrace{\text{DOUBLE} \mid \text{EVAL}\Sigma(N) \mid \text{INC}}_{n_0}$$

Diese Maschine hat N Zustände. Sie startet auf leerem Band, schreibt n_0 Einsen, verdoppelt diese, berechnet $\Sigma(N)$ und schreibt eine weitere 1.

BADΣ hat also eine 1 mehr als $\Sigma(N)$ geschrieben! Es folgt, dass die Annahme falsch war.

Busy Beaver

TECHNISCHE UNIVERSITÄT DARMSTADT

Interessanterweise sind einige Busy-Beaverwerte bekannt. Z.B. für TMs mit 2 Symbolen :

#Zustände	Anzahl Einsen des Siegers
1	1
2	4
3	6
4	13
5	>= 4098
6	>= 95.524.079

Schönheit in der Mathematik: (nach Prof. Hesse, Universität Stuttgart Fakultät für Mathematik und Physik, Dresden 2008)

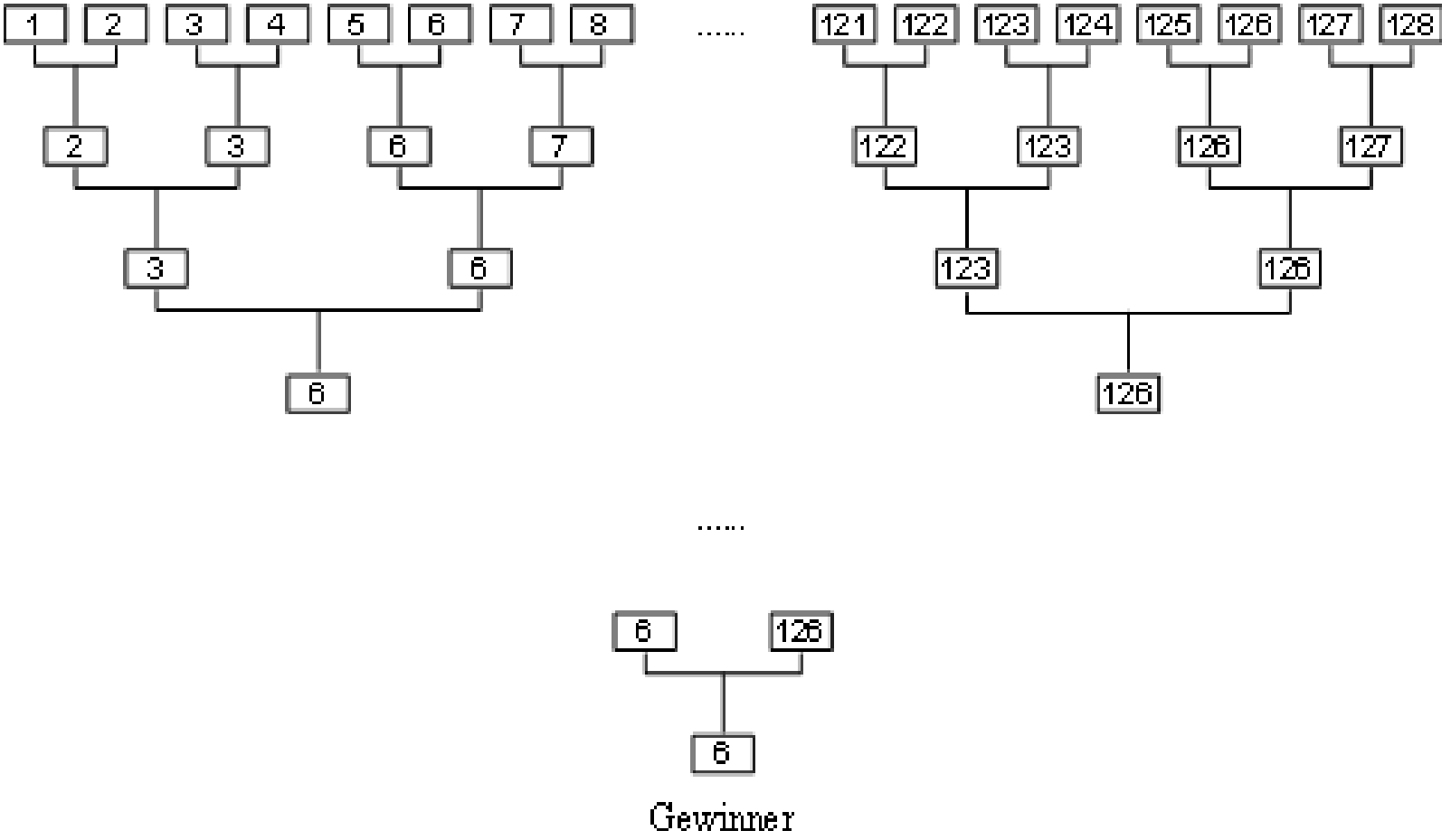
TECHNISCHE UNIVERSITÄT DARMSTADT

- Präzision
- Klarheit
- Eleganz

Beispiel: Turnierproblem

Tennisturnier mit 128 Spielern nach K.O.-System.
Wie viele Begegnungen werden ausgetragen?

Turnierverlauf



Turnierproblem: Lösung 1

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Runde: 128 Spieler, 64 Paare, **64** Begegnungen
2. Runde: 64 Spieler, 32 Paare, **32** Begegnungen
3. Runde: 32 Spieler, 16 Paare, **16** Begegnungen
4. Runde: 16 Spieler, 8 Paare, **8** Begegnungen
5. Runde: 8 Spieler, 4 Paare, **4** Begegnungen
6. Runde: 4 Spieler, 2 Paare, **2** Begegnungen
7. Runde: 2 Spieler, 1 Paar, **1** Begegnung

Insgesamt: **1** + **2** + **4** + **8** + **16** + **32** + **64** = **127** Begegnungen

Turnierproblem: Lösung 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zahl der Spieler: Zweierpotenz, $128 = 2^7$
- Begegnungen je Runde: fortgesetzte Halbierung

Begegnungen gesamt

= Summe von Zweierpotenzen

$= 1 + 2 + 2^2 + ... + 2^6$

$= 2^7 - 1 = 127$

Allgemeiner

TECHNISCHE UNIVERSITÄT DARMSTADT

Für 2^n Spieler, 2^{n-1} Erstrunden-Begegnungen
2^{n-2} Zweitrunden-Begegnungen, etc.

$$1 + 2 + 2^2 + \ldots + 2^{n-1}$$
$$= (1 + 2 + 2^2 + \ldots + \ldots 2^{n-1})(2 - 1)$$
$$= 2 + 2^2 + \ldots + 2^{n-1} + 2^n - 1 - 2 - 2^2 - \ldots - 2^{n-1}$$
$$= 2^n - 1$$

Fortschritt: Tiefe, Verallgemeinerung, Verständnis

Turnierproblem: Lösung 3

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Jede Begegnung hat einen Sieger und einen Verlierer.

b) Jeder Spieler spielt so lange, bis er verliert.

Also: es gibt genauso viele Begegnungen, wie es Verlierer gibt.

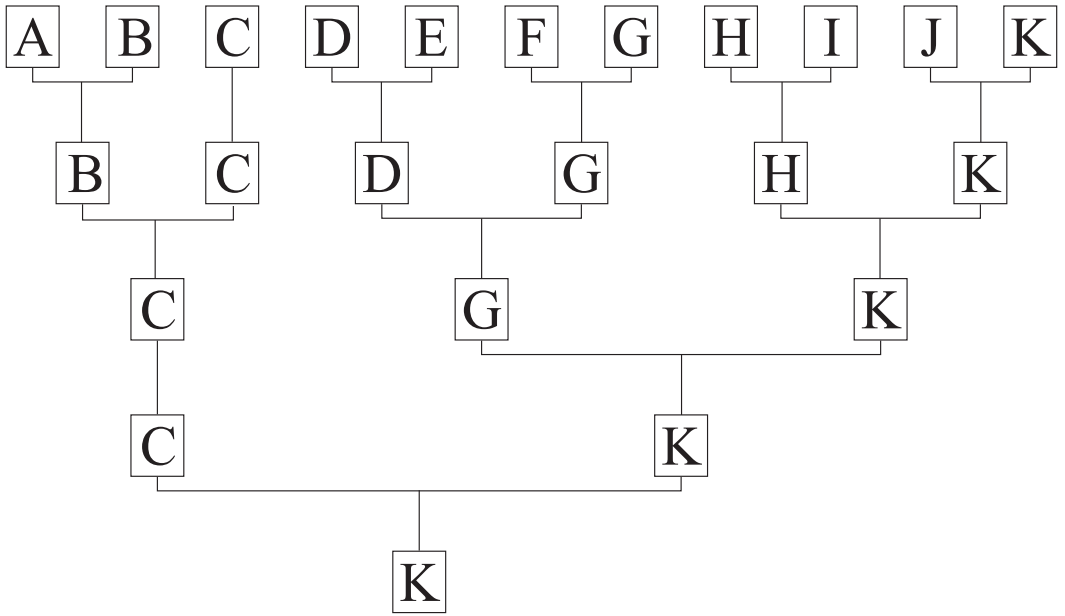
Jeder Spieler außer dem Champion ist ein Verlierer.

Also: Anzahl Begegnungen = Anzahl Verlierer
= Anzahl Spieler – 1.

Fortschritt: Vereinfachung, Tiefe, Verallgemeinerung , Ästhetik

Tennisturnier mit 11-er Feld

TECHNISCHE UNIVERSITÄT DARMSTADT

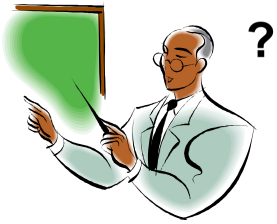


Probleme des täglichen Lebens

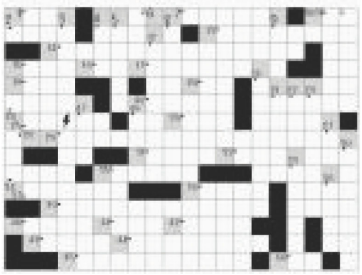
TECHNISCHE UNIVERSITÄT DARMSTADT

Im folgenden sind die Probleme lösbar. Die Frage ist nur in welcher Zeit und mit wieviel Speicherplatz.

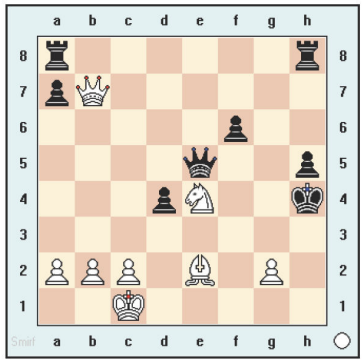
- Was ist schwieriger?



??



- Kopfrechnen
- Kreuzworträtsel
- Schach



- Sokoban
- Puzzle

Probleme und Problembeschreibungen, Wdh

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine Sprache $L \subseteq \Sigma^*$ muss nun irgendwie beschrieben werden.
  - z.B. durch einen *regulären Ausdruck*: (0*10*)
    - $\emptyset$ ist ein regulärer Ausdruck.
    - ε ist ein regulärer Ausdruck.
    - $\forall\ a_i \in \Sigma$ ist a_i ein regulärer Ausdruck.
    - Sind *x* und *y* reguläre Ausdrücke, so auch $x \cup y$, (*xy*) und *x* *.
    - Es gibt keine weiteren regulären Ausdrücke.
  - z.B. durch eine **Problembeschreibung**:
    - **Definition**: Ein *Entscheidungsproblem* ist ein input-output Tupel mit

      geg.: Kodierung eines Inputs einer Instanz, mittels Alphabet $\sum$

      ges.: ja/nein
    - Die Teilmenge aller Inputs, für die die Antwort “ja” ist, ist offenbar eine Sprache

Ein Zeit-Komplexitätsmaß

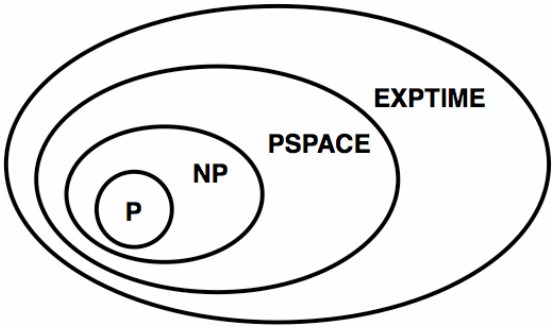
TECHNISCHE UNIVERSITÄT DARMSTADT

- Definition: Komplexität eines Algorithmus
  - Sei A ein deterministischer (RAM-)Algorithmus, der auf allen Eingaben hält.
  - Die **Laufzeit (Zeitkomplexität)** von A ist eine Funktion f: N→N,
    - wobei f(n) die maximale Anzahl von Schritten von A beschreibt **auf einer Eingabe der Länge n**.
    - Linear-Zeit-Algorithmus: $f(n) \leq c\, n$ für eine Konstante c
    - Polynom-Zeit-Algorithmus: $f(n) \leq c\, n^k$ für Konstanten c und k
- Definition: Komplexität eines Problems
  - Die Zeit- (Platz-) Komplexität eines Problems p ist die Laufzeit des schnellsten (am wenigsten Speicherplatz benötigenden) Algorithmus, der Problem p löst.
  - Ein Problem p ist "in Polynomzeit lösbar", wenn es Algorithmus A, Polynom Π und $n_0 \in \mathbb{N}$ gibt, so dass für alle $n > n_0$ **gilt : f(n) <= Π(n)**

P, NP, PSPACE

TECHNISCHE UNIVERSITÄT DARMSTADT

- **P:** Klasse aller Probleme, die von einer deterministischen RAM in Polynomzeit gelöst werden können
- **NP:** Klasse aller Probleme, die von einer nichtdeterministischen TM in Polynomzeit gelöst werden können.
- **PSPACE :** Klasse aller Probleme, die von einer deterministischen RAM mit polynomiell viel Platz gelöst werden können
- Man weiß nur, dass P ≠ EXPTIME und $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$
- EXPTIME = $\bigcup_k \mathbf{TIME}(2^{n^k})$
- Allgemein wird aber vermutet, dass alle Inklusionen echt sind, d.h.



Nichtdeterministische Turingmaschinen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine nichtdeterministische Turingmaschine (NTM) ist definiert, wie eine deterministische Turingmaschine, nur dass δ eine Übergangsrelation und keine Funktion ist.

- $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{R,N,L\}}$ ist die Übergangsrelation.

- Bsp.: Wenn die TM in Zustand q ist, und ein a liest, und $\delta(q,a) = \{(q', b, R), (q'', a, L)\}$ ist, dann ist die nichtdeterministische TM im nächsten Schritt entweder in Zustand q‘, nachdem sie ein b geschrieben hat, oder sie ist in Zustand q‘‘ nachdem sie ein a schrieb.

- Die Laufzeit einer NTM ist definiert als die Länge des kürzesten Berechnungsweges, der in einem akzeptierenden Endzustand endet.

Nichtdeterministische Turingmaschinen und Verifizierer

TECHNISCHE UNIVERSITÄT DARMSTADT

Def. Es sei L eine Sprache. Ein Verifizierer für L ist ein deterministischer Algorithmus A, mit L = {w | es gibt ein c mit A akzeptiert wc}

Der Zeitaufwand für einen Verifizierer wird abhängig von der Länge von w gemessen. L ist polynomiell prüfbar, wenn es einen Verifizierer mit polynomiellem Zeitaufwand gibt.

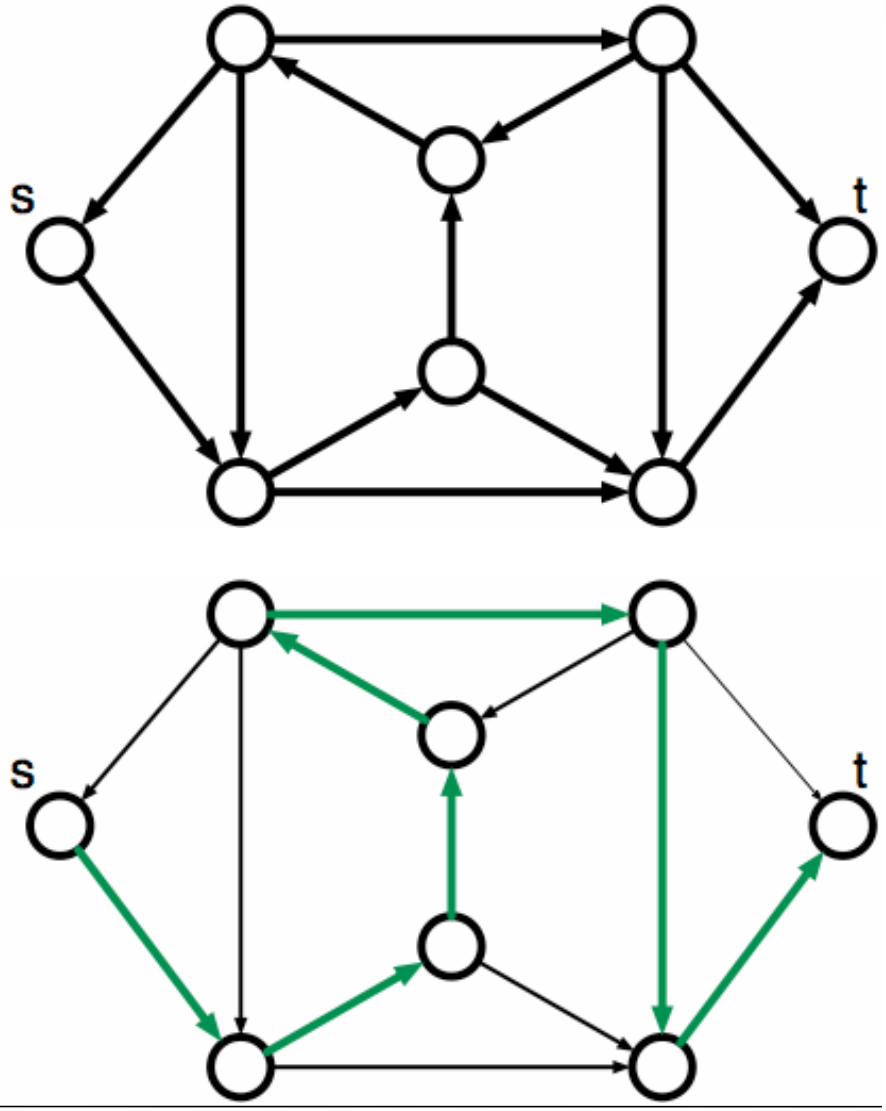
Satz: NP ist die Menge aller Probleme, für die es einen Verifzierer mit polynomiellem Zeitaufwand gibt.

(ohne Beweis)

Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Definition: *HAMPATH***
  - Das Hamiltonsche Pfadproblem
    - Geg.:
      - ein gerichteter Graph
      - Zwei Knoten s,t
    - Ges.: existiert ein Hamitonscher Pfad von s nach t
      - d.h. ein gerichteter Pfad, der alle Knoten besucht, aber keine Kante zweimal benutzt
- **Algorithmus für Hamiltonscher Pfad:**
  - Rate eine Permutation $(s,v_1,v_2,...,v_{n-2},t)$
  - Teste, ob Permutation ein Pfad ist
    - falls ja, akzeptiere
    - falls nein, verwerfe
- **Also: HamPath ∈ NP**



Das SAT Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine Boolesche Funktion $f(x_1,x_2,..,x_n)$ ist erfüllbar, wenn es eine Wertebelegung für $x_1,x_2,..,x_n$ gibt, so dass $f(x_1,x_2,..,x_n) = 1$
  - $(x \vee y) \wedge (z \vee \neg x \vee \neg y) \wedge (x \vee \neg z)$ ist erfüllbar, da
    - die Belegung $x = 1$, $y = 0$, $z = 0$
  - $(1 \vee 0) \wedge (0 \vee 0 \vee 1) \wedge (1 \vee 1) = 1 \wedge 1 \wedge 1 = 1$ liefert.
- Definition (SAT Problem, die Mutter aller NPc Probleme)
  - **Gegeben**:
    - Boolesche Funktion ϕ
  - **Gesucht**:
    - Gibt es $x_1,x_2,..,x_n$ so dass $\phi(x_1,x_2,..,x_n) = 1$
- SAT ist in NP. Man vermutet, dass SAT nicht in P ist.

Das QSAT Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine quantifizierte Boolesche Formel (QBF) besteht aus
  - Einer Folge von Quantoren $\exists x$, $\forall y$ mit daran gebundenen Variablen; obdA seien genau alle x_i mit ungeradem i existenzquantifiziert
  - Einer Booelschen Funktion $F(x_1,x_2,...,x_m)$
  - Jede Variable der Funktion ist genau einmal an einem Quantor gebunden
- Die quantifizierte Boolesche Formel ist erfüllbar falls
  - Im Falle eines Existenzquantors: $\exists x\ F(x) \Leftrightarrow F(0) \vee F(1)$
  - Im Falle eines Allquantors: $\forall x\ F(x) \Leftrightarrow F(0) \wedge F(1)$
- Definition (QSAT Problem, die Mutter aller PSPACEc Probleme)
  - **Gegeben**: Quantifizierte Boolesche Funktion ϕ
  - **Frage**: Gibt es x_1, so dass es für alle x_2 ein x_3 gibt, so dass ... so dass $\phi(x_1,x_2,..,x_n) = 1$
- QSAT ist in PSPACE. Man vermutet, dass QSAT nicht in NP ist.

Beispiele:

TECHNISCHE UNIVERSITÄT DARMSTADT

- $\exists x\ \forall y\ (x \wedge y) \vee (\neg x \wedge \neg y)$

  $= (\forall y\ (0 \wedge y) \vee (\neg 0 \wedge \neg y)) \quad \vee (\forall y\ (1 \wedge y) \vee (\neg 1 \wedge \neg y))$

  $= (\forall y: \neg y) \quad \vee (\forall y: y)$

  $= (\neg 0 \wedge \neg 1) \quad \vee (0 \wedge 1)$

  $= 0 \vee 0$

  $= 0$

- $\forall y\ \exists x\ (x \wedge y) \vee (\neg x \wedge \neg y)$

  $= (\exists x: (x \wedge 0) \vee (\neg x \wedge \neg 0)) \wedge (\exists x: (x \wedge 1) \vee (\neg x \wedge \neg 1))$

  $= (\exists x: : \neg x) \quad \wedge (\exists x: x)$

  $= (\neg 0 \vee \neg 1) \quad \wedge (0 \vee 1)$

  $= 1 \wedge 1$

  $= 1$

Einordnung von Problemen in P, NP, PSPACE

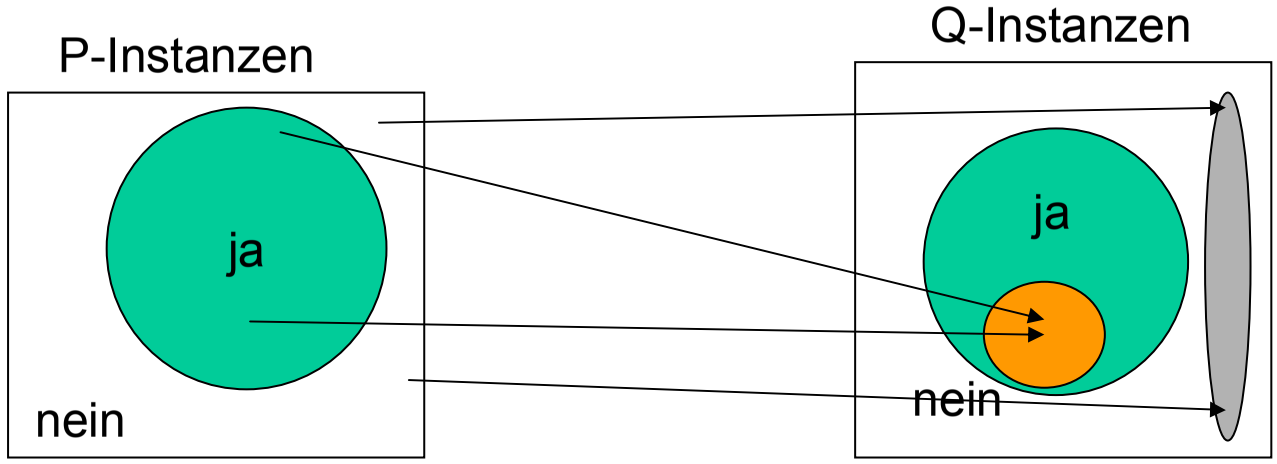
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Angabe eines Algorithmus für Problem**
- **Reduktionstechniken u.a.**

Definition: Seien P,Q Probleme. Sei L_P (L_Q) die Menge der Instanzen des Problems P (Q), für die die Antwort „ja“ ist. P heißt auf Q polynomiell reduzierbar ($P \leq_p Q$), wenn es eine von einem deterministischen Algorithmus in Polynomzeit berechenbare Funktion f: $\Sigma^* \rightarrow \Sigma^*$ gibt, so dass

$$x \in L_P \Leftrightarrow f(x) \in L_Q$$

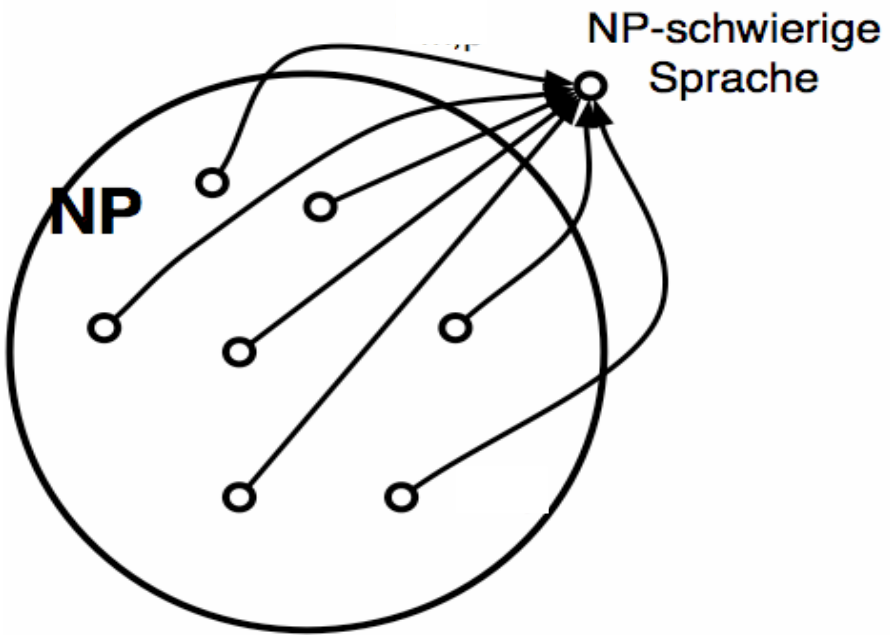
Bsp.:



NP-Schwierig

TECHNISCHE UNIVERSITÄT DARMSTADT

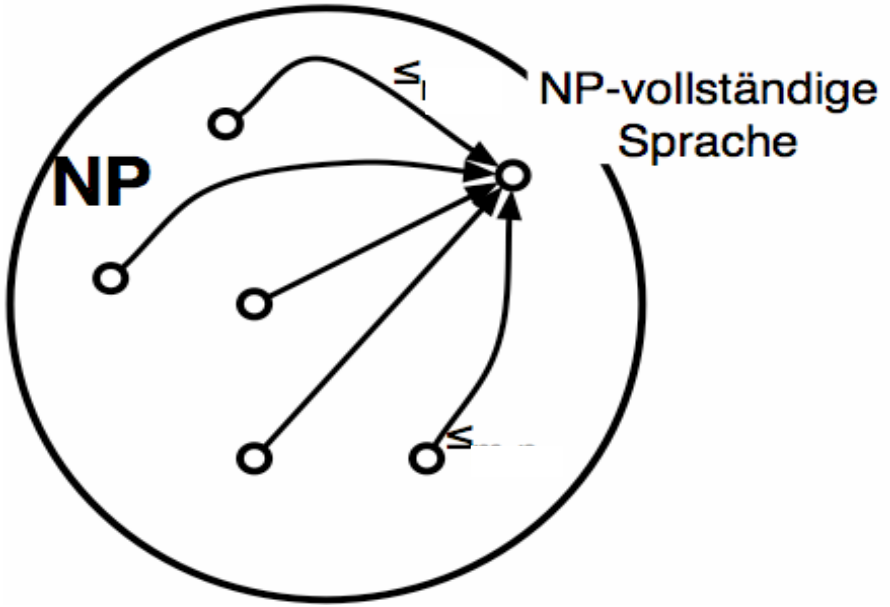
- Definition:
  - Eine Sprache S ist **NP-schwierig** (NP-hard) wenn:
    - jede Sprache aus NP mit einer Polynom-Zeit-Abbildungsreduktion auf S reduziert werden kann, d.h.
    - für alle $L \in NP$: $L \leq_p S$
- Theorem
  - Falls eine NP-schwierige Sprache in P ist, ist P=NP
- Beweis
  - Falls $S \in P$ und $L \leq_p S$ gilt $L \in P$.



NP-Vollständigkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

- Definition:
  - Eine Sprache S ist **NP-vollständig** (NP-complete) wenn:
    - S $\in$ NP
    - S ist NP-schwierig
- Korollar:
  - Ist eine NP-vollständige Sprache in P, dann ist P=NP
- Beweis:
  - folgt aus der NP-Schwierigkeit der NP-vollständigen Sprache.

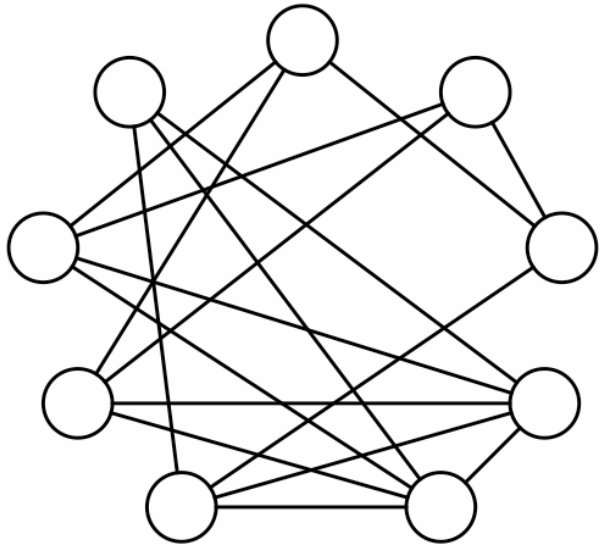


Das 3-SAT-Problem und das Clique-Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

- 3-SAT:
  - **Gegeben**:
    - Eine Boolesche Formel in 3-CNF
  - **Gesucht**:
    - Gibt es eine erfüllende Belegung
- Definition k-Clique
  - Ein ungerichteter Graph G=(V,E) hat eine k-Clique,
    - falls es k verschiedene Knoten gibt,
    - so dass jeder mit jedem anderen eine Kante in G verbindet
- CLIQUE:
  - **Gegeben**:
    - Ein ungerichteter Graph G
    - Eine Zahl k
  - **Gesucht**:
    - Hat der Graph G eine Clique der Größe k?

$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

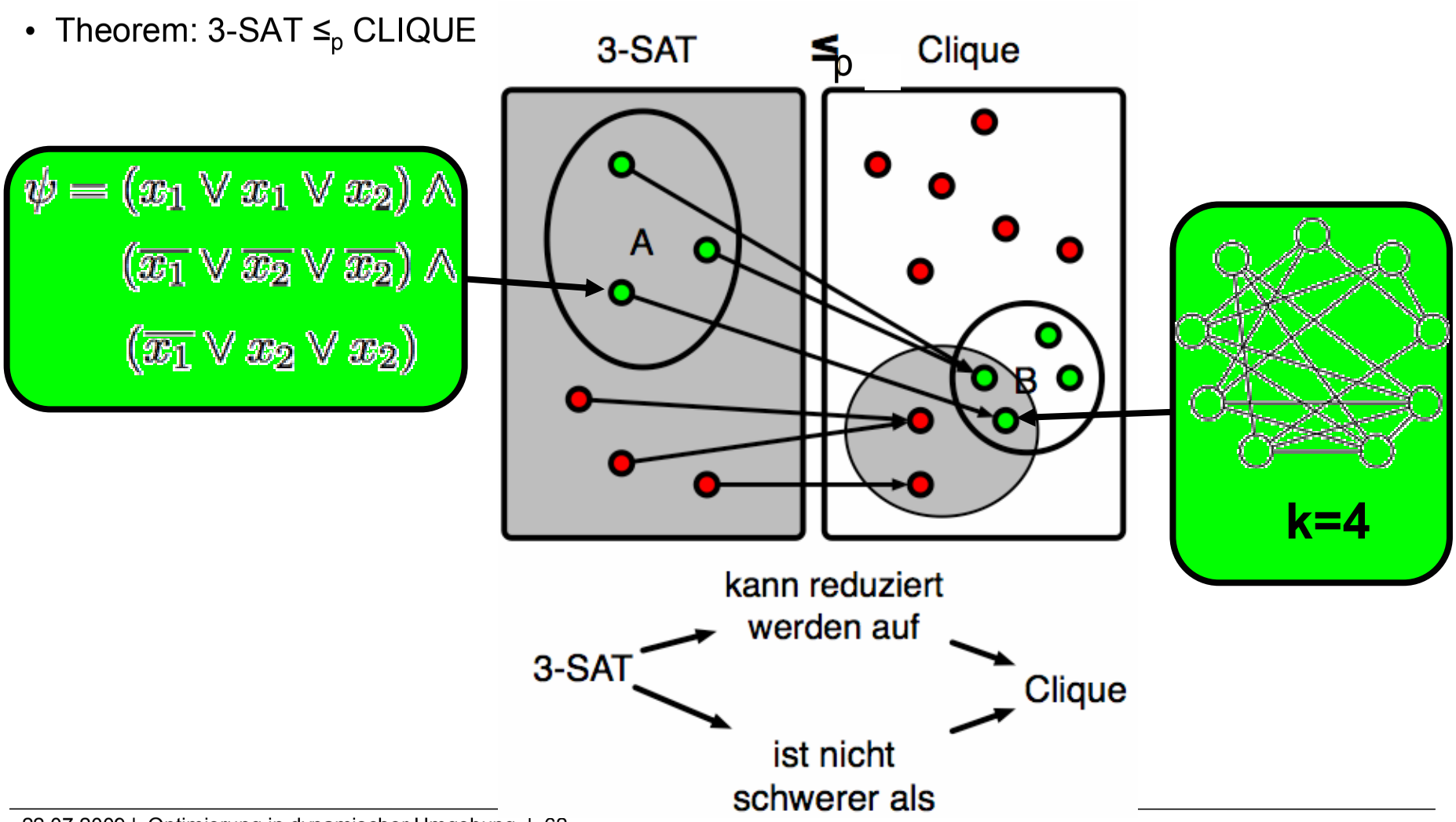


k=4

3-SAT läßt sich auf Clique reduzieren

TECHNISCHE UNIVERSITÄT DARMSTADT

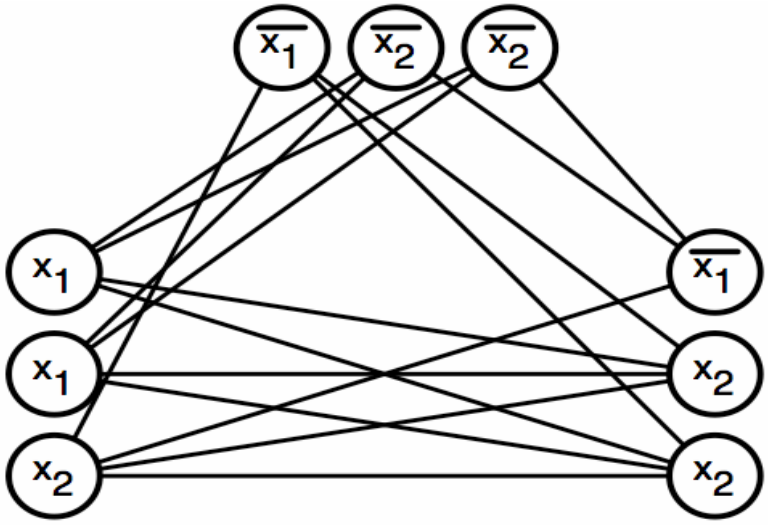
- Theorem: 3-SAT $\leq_p$ CLIQUE



3-SAT läßt sich auf Clique reduzieren

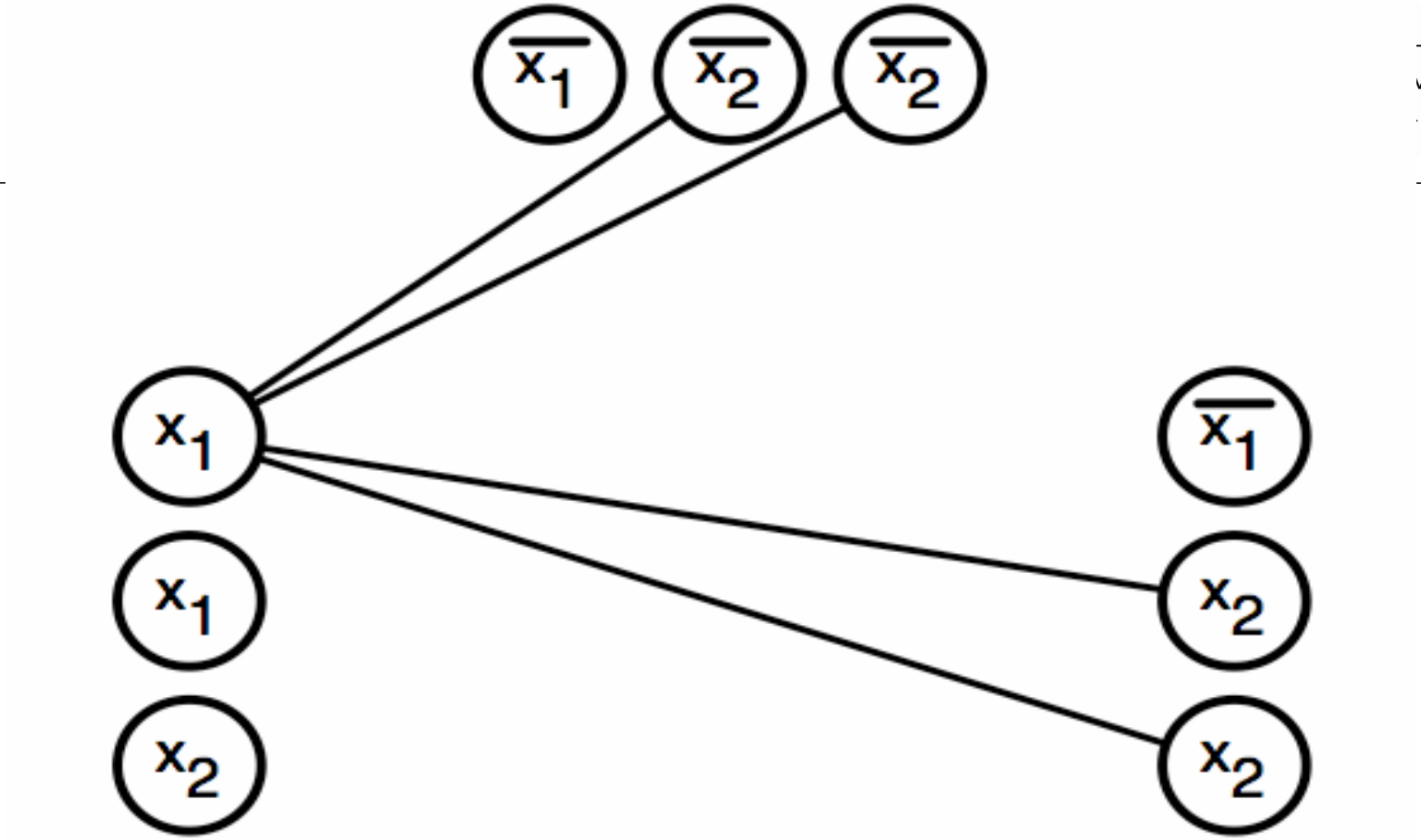
TECHNISCHE UNIVERSITÄT DARMSTADT

- Theorem: 3-SAT $\leq_p$ CLIQUE
- Beweis
  - Konstruiere Reduktionsfunktion f wie folgt:
  - f(ϕ) = <G,k>
  - k = Anzahl der Klauseln
  - Für jede Klausel C in ϕ werden drei Knoten angelegt, die mit den Literalen der Klausel bezeichnet werden
  - Füge Kante zwischen zwei Knoten ein, gdw.
    - die beiden Knoten nicht zur selben Klausel gehören und
    - die beiden Knoten nicht einer Variable und der selben negierten Variable entsprechen.

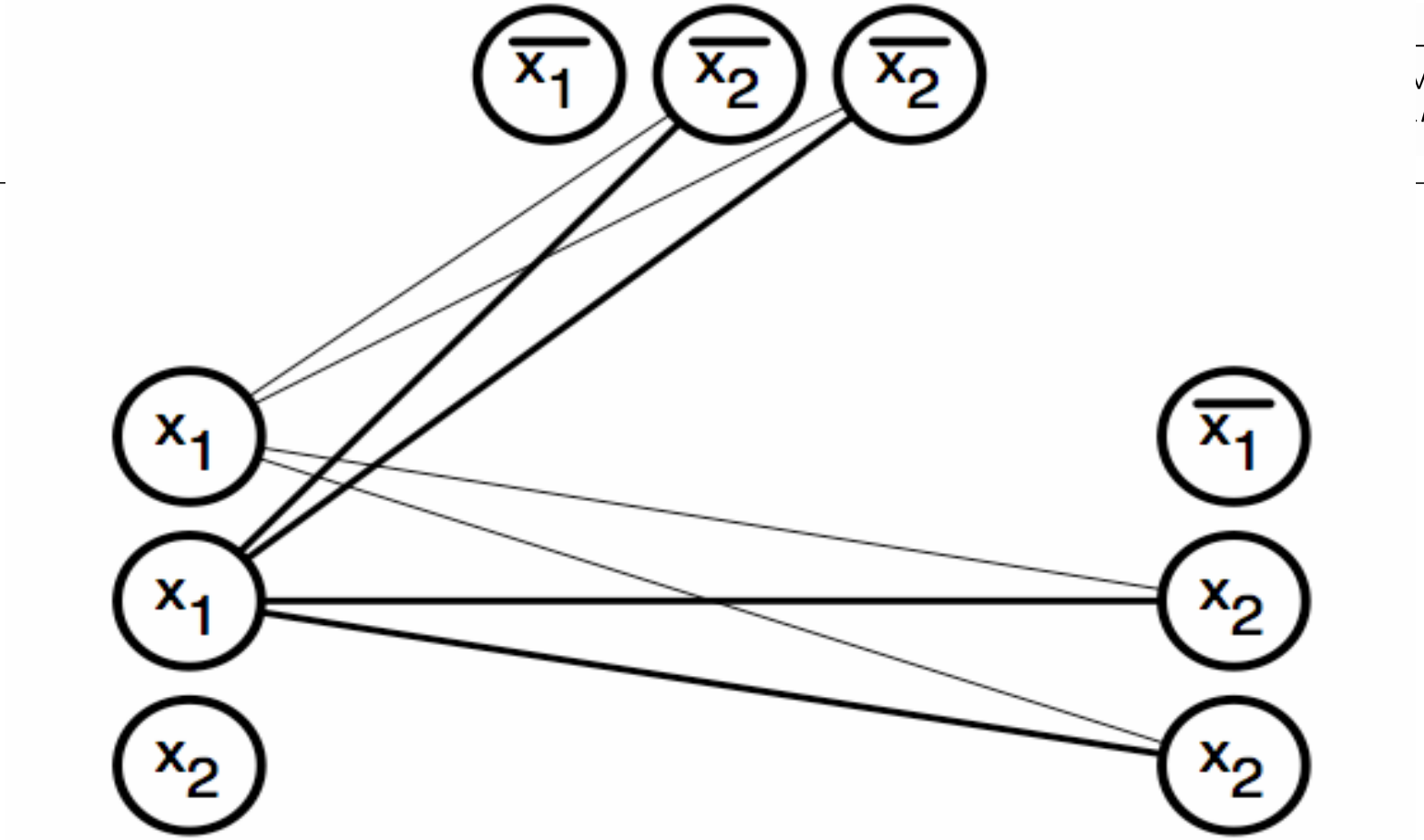


$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

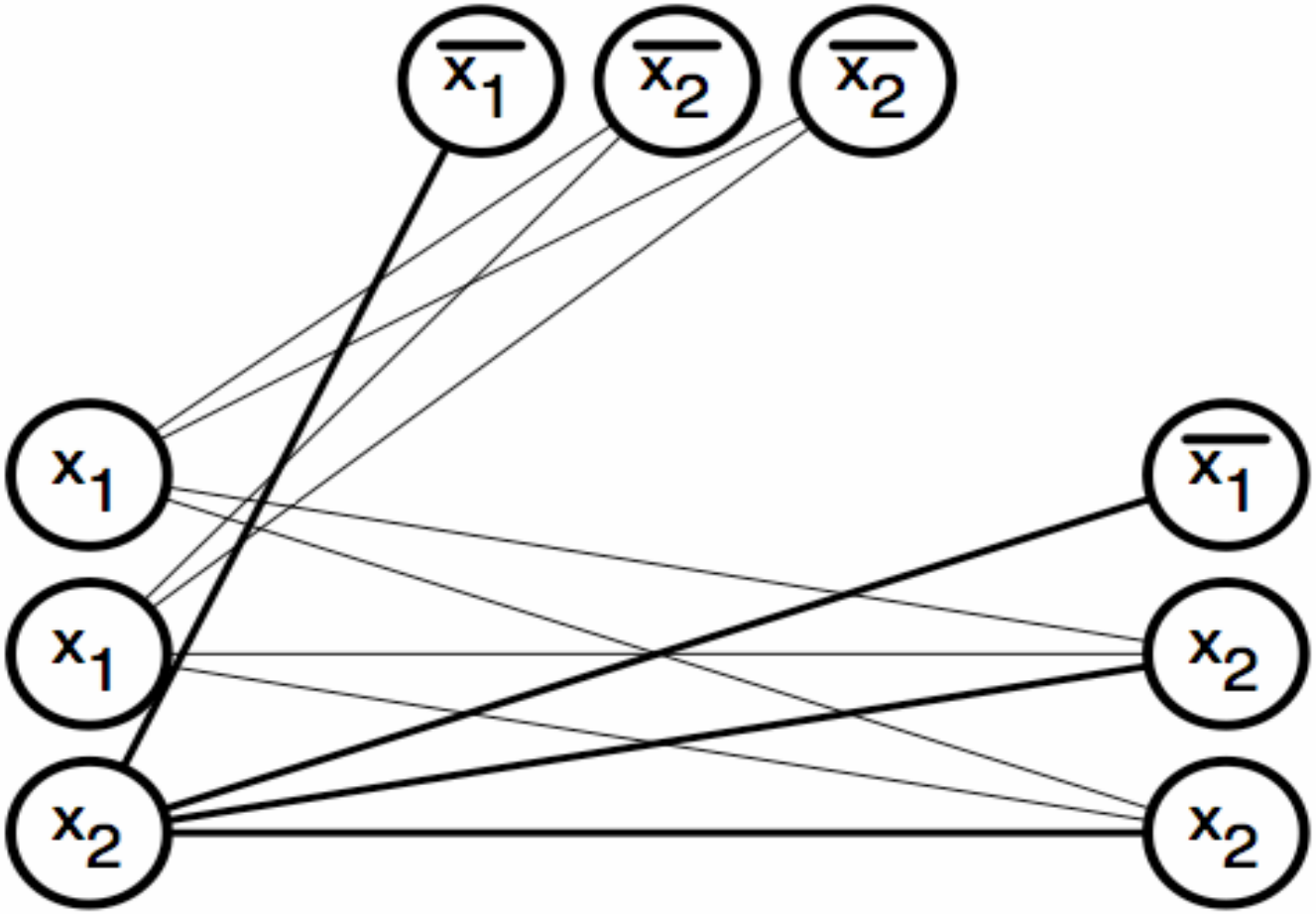
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



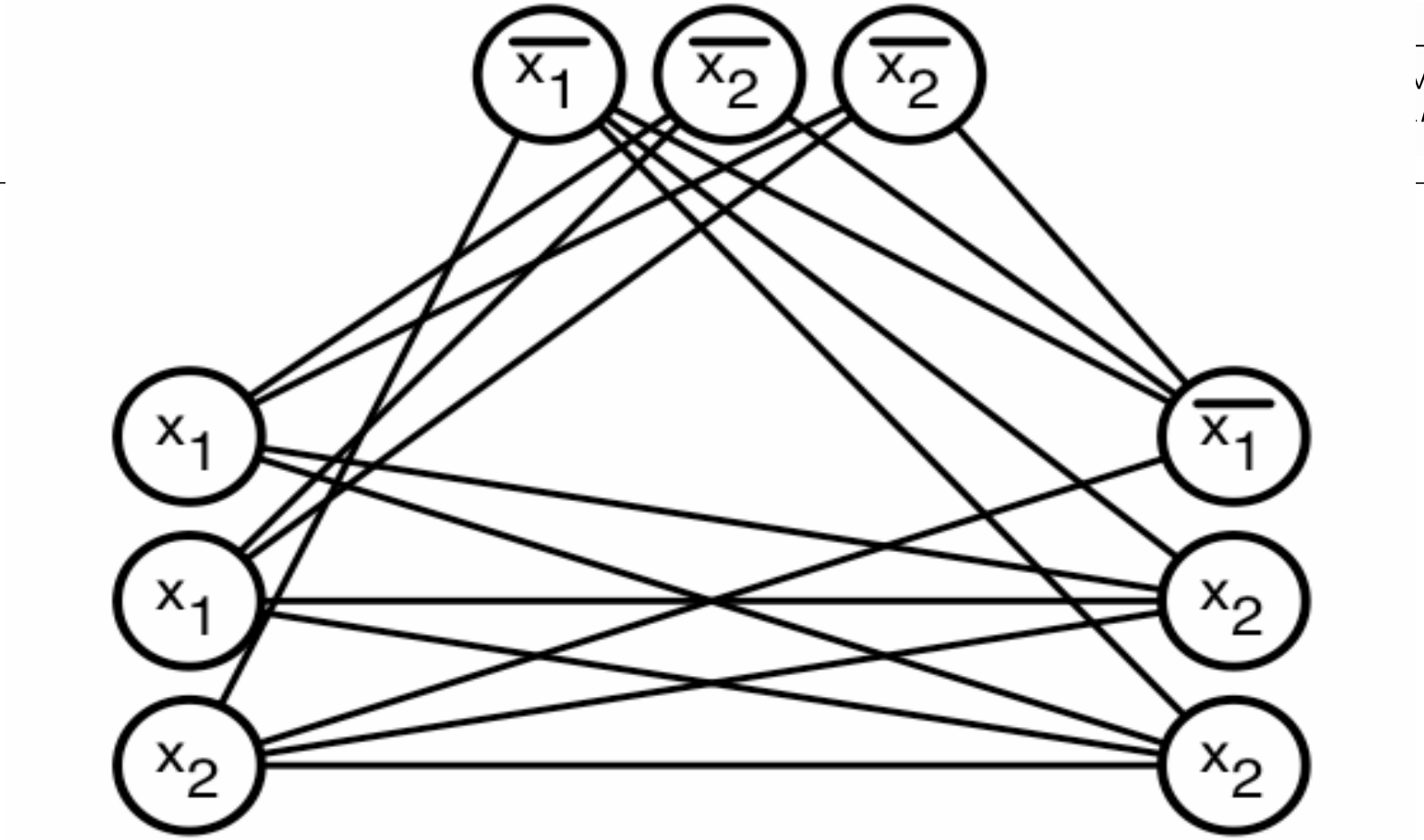
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



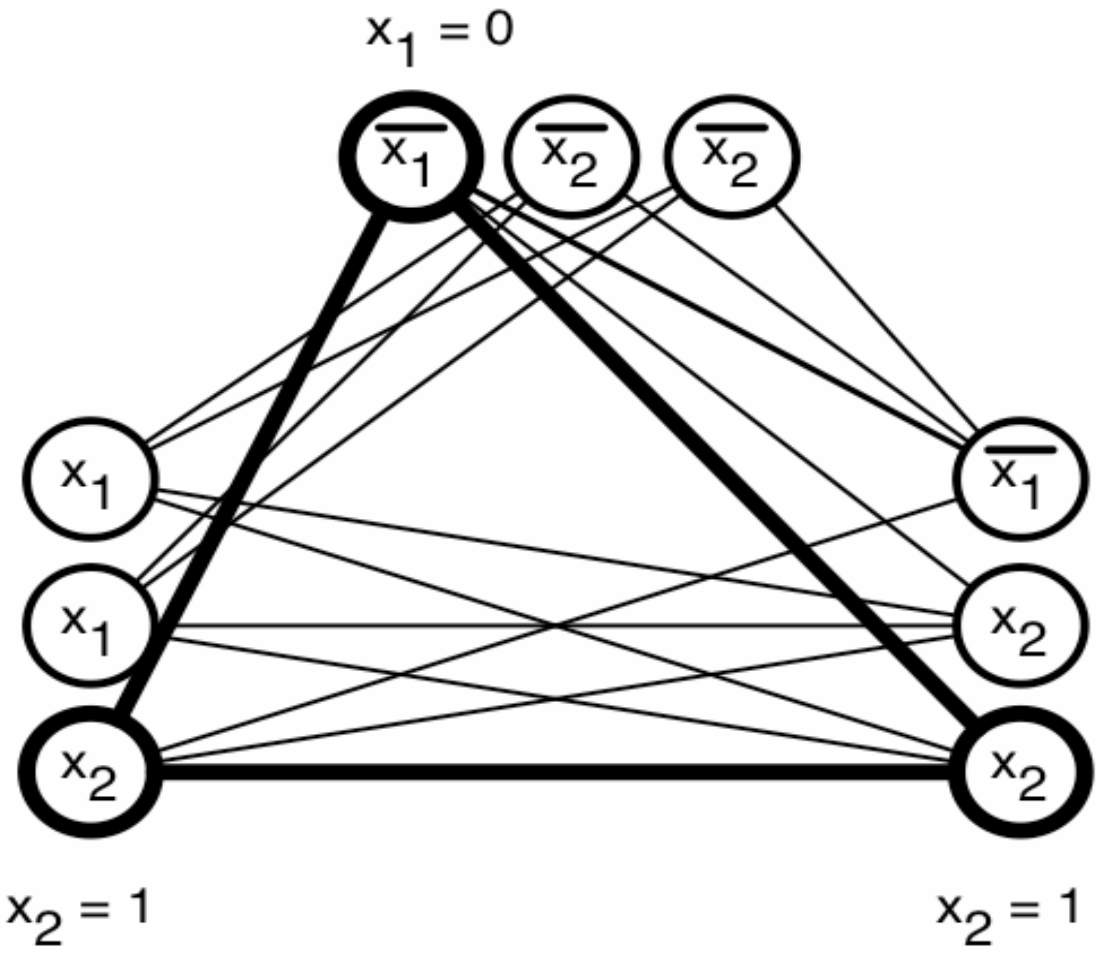
$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$$\psi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

0	0	**1**	**1**	0	0	**1**	**1**	**1**

Beweis der Korrektheit der Reduktionsfunktion

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Reduktionsfunktion ist korrekt:
- Behauptung;
  - Eine erfüllende Belegung in ϕ existert gdw. eine k-Clique in G existiert
- 1. Fall: eine erfüllende Belegung existiert in ϕ
  - Dann liefert die Belegung in jeder Klausel mindestens ein Literal mit Wert 1
  - Wähle aus der Knotenmenge einer Klausel ein beliebiges solches Literal
  - Die gewählte Knotenmenge besteht dann aus k Knoten
  - Zwischen allen Knoten existiert eine Kante, da Variable und negierte Variable nicht gleichzeitig 1 sein können
- 2. Fall: eine k-Clique existiert in G
  - Jeder der Knoten der Clique gehört zu einer anderen Klausel
  - Setze die entsprechenden Literale auf 1
  - Bestimmte daraus die Variablen-Belegung
  - Das führt zu keinem Widerspruch, da keine Kanten zwischen einem Literal und seiner negierten Version existieren
- Laufzeit:
  - Konstruktion des Graphens und der Kanten benötigt höchstens quadratische Zeit.

Limits

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Komplexität wird gemessen mit Hilfe des worst-case,**
  - In der realen Welt treten aber oft „gutmütige“ Instanzen auf
  - andererseits: Haben wir dann das richtige Problem formuliert?

- **Die Eingabegröße**

  Sei ein Problem p gegeben. Wenn wir nun die Kodierung der Eingabe signifikant kleiner bekommen, wird das resultierende aus p abgeleitete Problem p‘ möglicherweise schwerer, obwohl die Laufzeit möglicherweise sinkt.

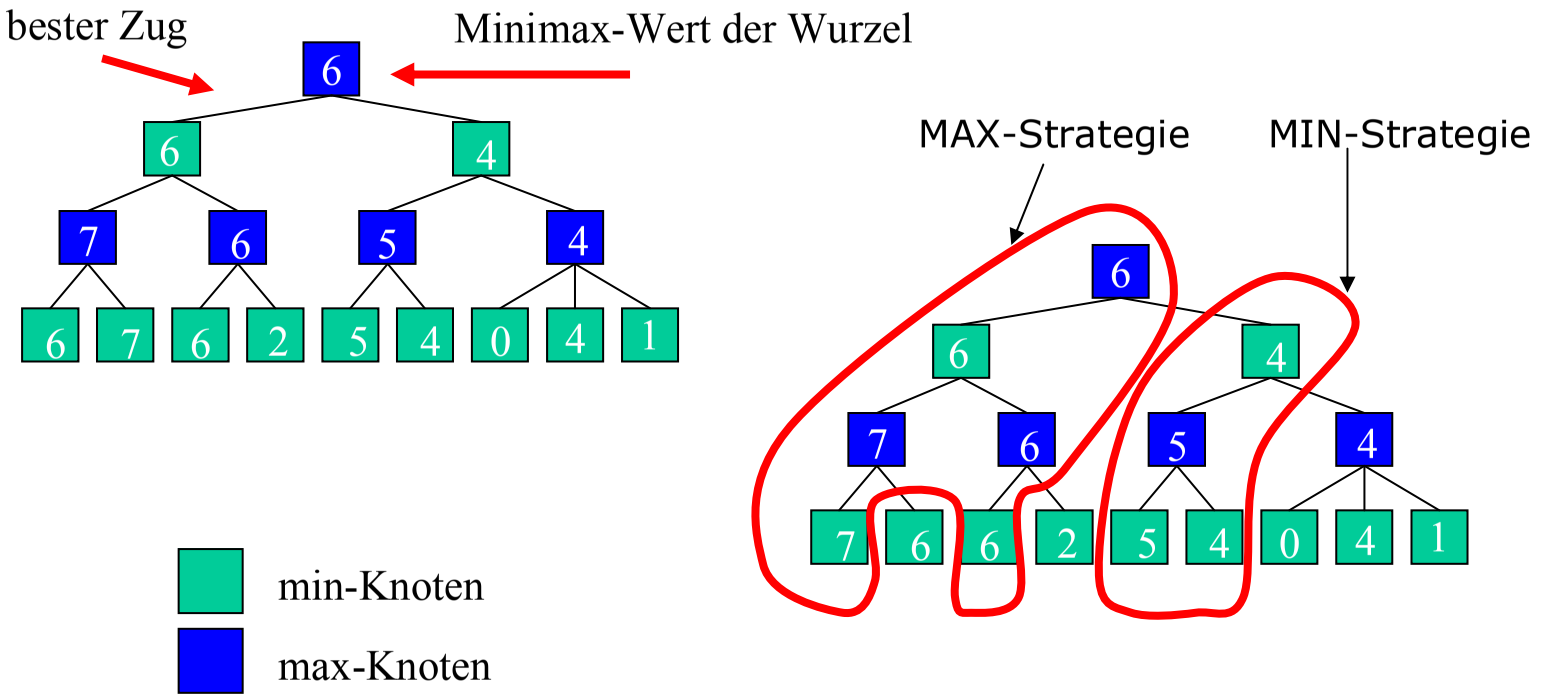
Fazit: Die Lücke zwischen der Komplexitätstheorie und der Wirklichkeit ist größer als wir es von der Physik gewohnt sind. Trotzdem ist die Komplexitätstheorie eine großartige Theorie.

Computerschach

Grundlagen I

Was macht ein Schachprogramm eigentlich?

(II) Vorausschau



Minimax-Prinzip

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition: Im folgenden ist G = (T,h) ein **Spielbaum**, wenn T = (V,E) ein Baum ist und h : L(G) → {0,1} eine Funktion ist, die Blattknoten des Spielbaums G auf Zahlen abbildet. L(G) bezeichnet hier die Menge der Blattknoten von G.

Definition: Es gebe 2 Spieler MIN und MAX. MAX besitzt das Zugrecht auf den geraden Ebenen des Spielbaums, MIN auf den anderen. Hierdurch wird eine Spielerfunktion definiert: **p : V → {MAX, MIN}**

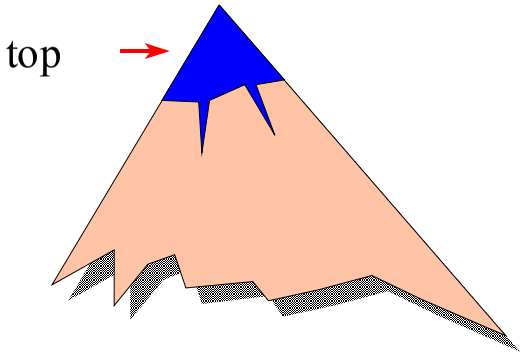
Definition: Sei G = ((V,E),h) ein Spielbaum. Sei $v \in V$ ein Knoten des Spielbaums G. Die Funktion **minimax: V → {0,1}** ist induktiv wie folgt definiert:

$$\text{minimax}(v) := \begin{cases} h(v), & \text{falls } v \in L(G) \\ \max\{\text{minimax}(v' \mid (v,v') \in E\}, & \text{falls } p(v) = \text{MAX} \\ \min\{\text{minimax}(v') \mid (v,v') \in E\}, & \text{falls } p(v) = \text{MIN} \end{cases}$$

Der Minimaxwert des Knotens v ist minimax(v). Der Minimaxwert der Wurzel eines Spielbaums G wird mit minimax(G) bezeichnet.

Schnell, schnell, schnell!

Der kritische Punkt ist die intelligente Vorausschau.



- **Man wähle einen geeigneten Teilbaum, der die aktuelle Stellung enthält. So groß wie möglich.**
- **Weise den künstlichen Endstellungen heuristische Werte zu!**
- **Werte aus!**

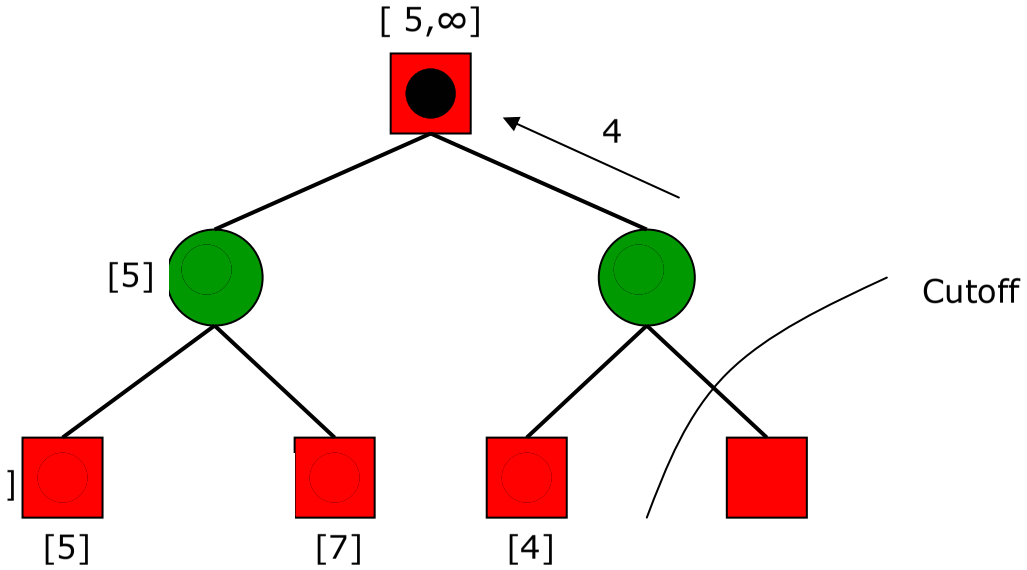
50 Jahre währende Erfahrung zeigt:

Der Spielbaum verhält sich wie ein Fehlerfilter!

*Je größer der Baum, desto besser der Filter. Der **Alphabeta-Algorithmus** spielt dabei eine wichtige Rolle.*

Alphabeta-Algorithmus

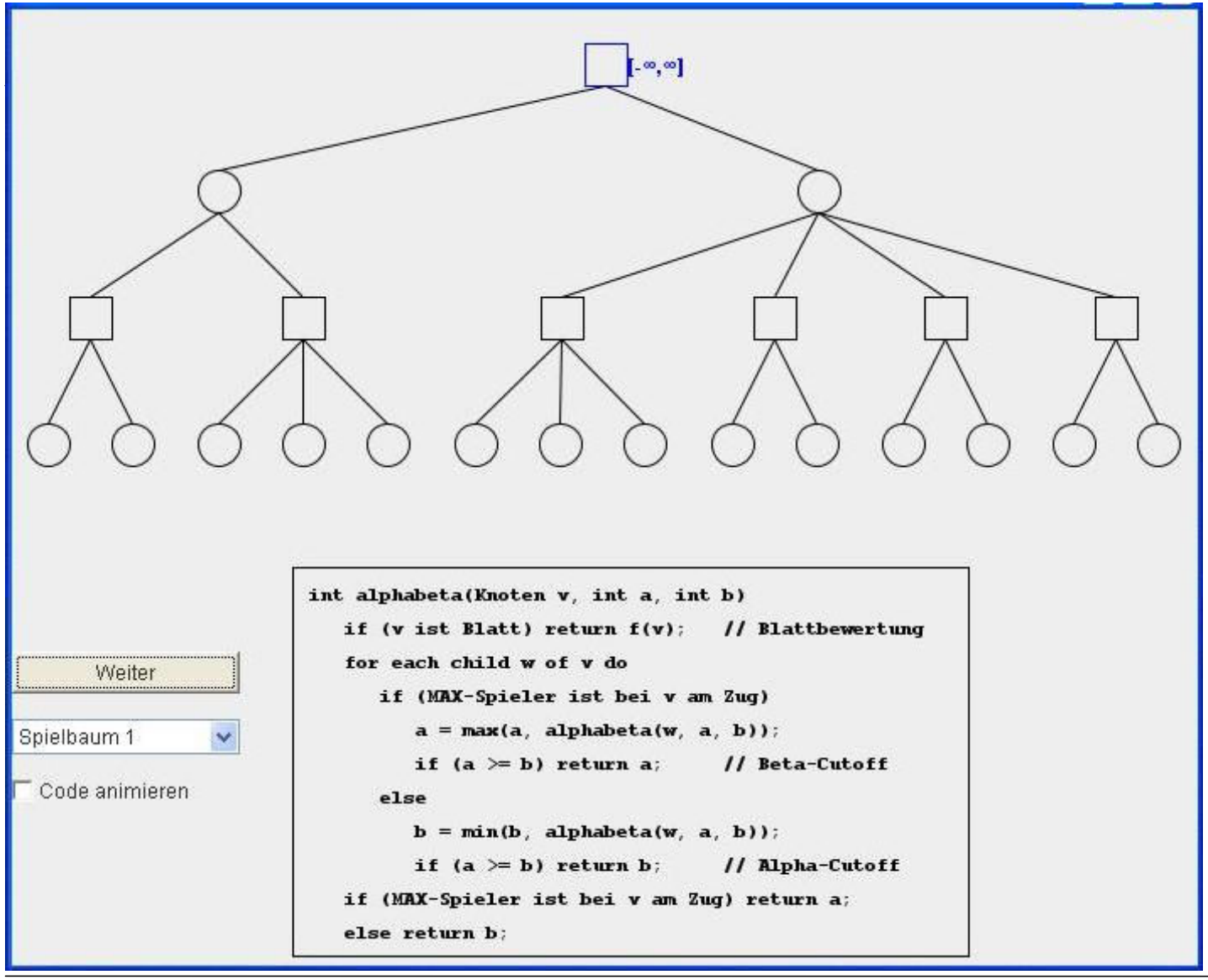
TECHNISCHE UNIVERSITÄT DARMSTADT



Miniaturbeispiel

Alphabeta-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT



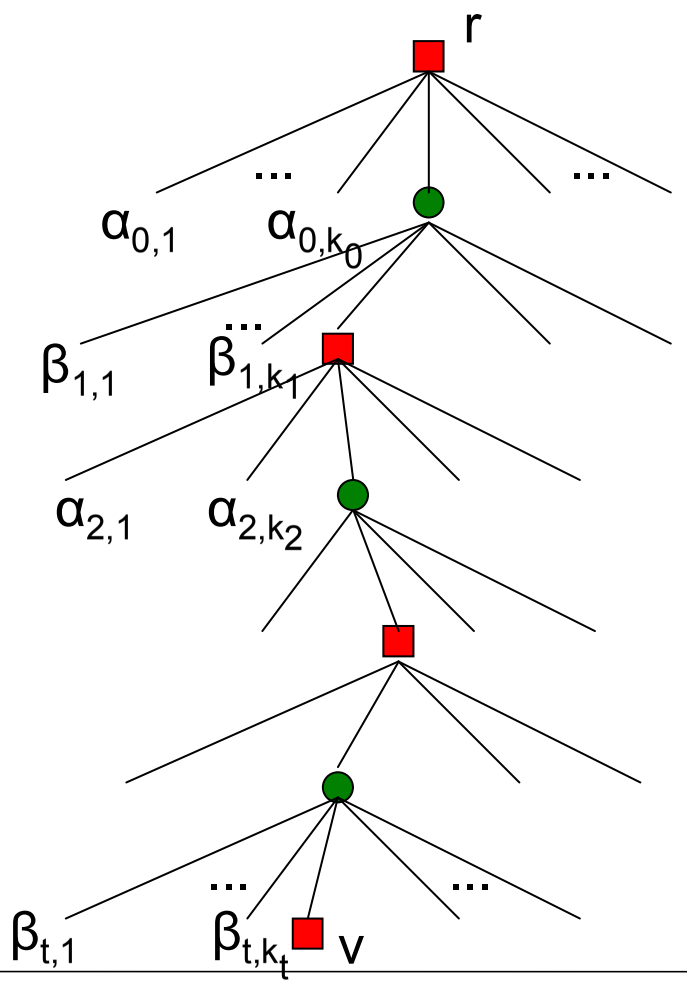
Alphabeta-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Der Alphabetaalgorithmus:

•Tiefensuche

• wenn die Suche an Knoten v angelangt, können wir uns vorstellen, dass
  - der durchsuchte Teil des Suchbaums „links“ von dem Weg von der Wurzel r zu v liegt, und
  - der noch nicht durchsuchte Teil „rechts“ des Weges liegt.



Alphabeta-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Es seien $\alpha_{i,j}$ ($\beta_{i,j}$) die Werte der linken Nachbarknoten von MIN- (bzw. MAX)- Knoten auf dem Weg von r nach v.

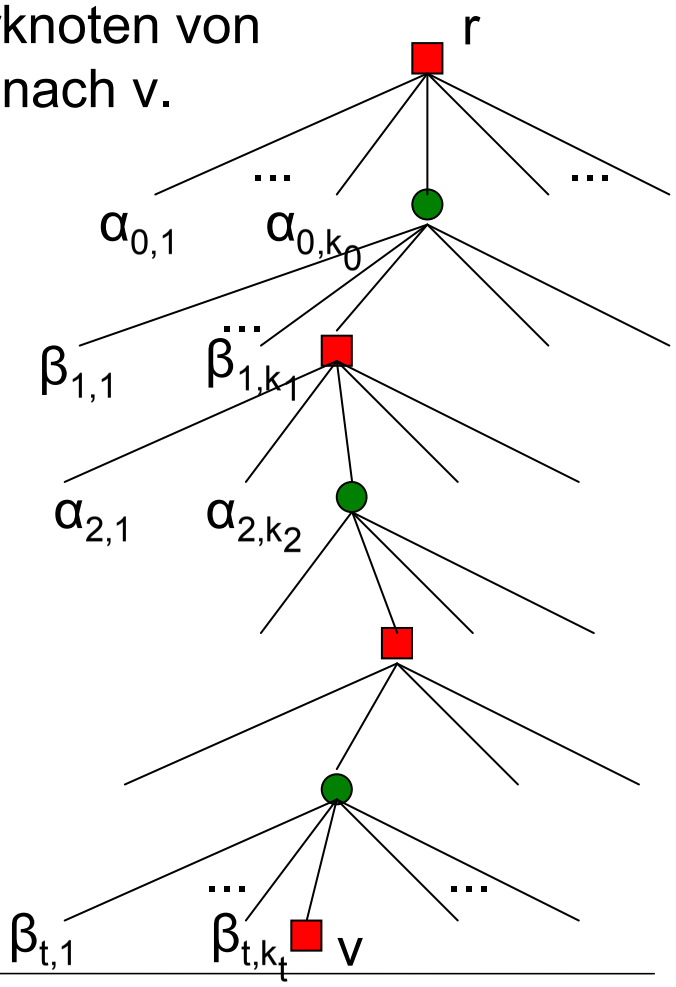
Sei

$\alpha(v) := \max_{i=0,...,t;\ i \text{ gerade}} \max_{j=1,...,k_i} \alpha_{i,j}$

und

$\beta(v) := \min_{i=0,...,t;\ i \text{ ungerade}} \min_{j=1,...,k_i} \beta_{i,j}$

Dann kann MAX bereits α(v) erreichen, MIN bereits β(v). F(v) ist also nur noch von Interesse, wenn $\alpha(v) < F(v) < \beta(v)$. Wurde Alphabeta(r,-∞,∞) aufgerufen, wird der Alphabeta-Algorithmus für v mit Alphabeta(v,α(v),β(v)) aufgerufen.



Alphabeta-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Beobachtung: α(v) und β(v) sind abhängig von dem bereits durchsuchten Teil des Suchbaums und damit von der Reihenfolger von Nachfolge-knoten.

Satz: Es sei G=(V,E,f) ein Spielbaum, v ein Knoten aus V, α(v) und β(v) die Schranken für v. Der Alphabeta-Algorithmus untersucht v genau dann, wenn α(v) < β(v).

Beweis: klar mit Überlegung von vorher und Cutoff-Bedingung im Algorithmus.

Alphabeta-Algorithmus, Negamax-Variante

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int negamax(node v, int a, b)
{
  if (v ist Blatt) return f(v);      // f bewerte v jetzt aus Sicht des Ziehenden!
  erzeuge alle Nachfolger v_0 ... v_b-1 von v
  for (i = 0; i < b; i++) {
        a = max(a, -negamax(v_i, -b, -a))
        if (a ≥ b) return a;        // cutoff
  }
  return a;
}
```

Lemma: Sei x = negamax(v,α,β) und F beschreibe den Minimax-Wert aus Sicht des Spielers, der bei v am Zug ist. Dann gilt:

$x \leq \alpha$	falls $F(v) \leq \alpha$
$x = F(v)$	falls $\alpha < F(v) < \beta$
$x \geq \beta$	falls $F(v) \geq \beta$

Beweis: Induktion über Höhe h des Spielbaums → Übung

Alphabeta-Algorithmus, Aufwand

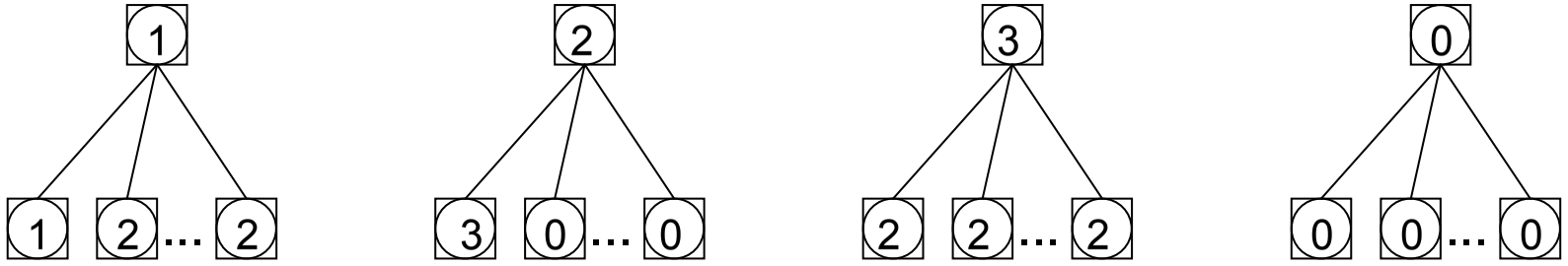
TECHNISCHE UNIVERSITÄT DARMSTADT

Sei G = (V,E,f) ein b/t-uniformer Spielbaum, d.h., alle inneren Knoten besitzen b Nachfolger und alle Blätter liegen in Tiefe t. Sei r seine Wurzel.

Beobachtung: Im schlimsten Fall besucht der Alphabeta-Algorithmus alle Knoten des Spielbaums.

Definition (Knotentyp, kritischer Knoten)

a) Die Abbildung *Typ* : V → {0,1,2,3} ist definiert durch Typ(r) = 1 und



b) ein Knoten v heißt *kritisch*, wenn Typ(v) ≠ 0.

Alphabeta-Algorithmus, Aufwand

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz: Es sei A ein beliebiger Algorithmus, der zu einem Spielbaum G = (V,E,f) mit Wurzel r dem Minimaxwert F(r) berechnet.

Dann kann man G durch Permutationen der Navchfolger der inneren Knoten so zu G‘ verändern, dass jeder Knoten, den der Alphabeta-Algorithmus auf G‘ besucht auch von A auf G besucht wird. Der Alphabeta-Algorithmus besucht gerade die kritischen Knoten.

Alphabeta-Algorithmus, Aufwand

TECHNISCHE UNIVERSITÄT DARMSTADT

Beweisskizze:

Für Knoten vom Typ 1,2,3 gelten folgende Aussagen, welche durch eine gemeinsame Induktion über die Tiefe von G bewiesen werden können:

A1: Sei v ein Knoten mit Typ(v) = 1. Dann gilt:

- A berechnet für v den Minimaxwert F(v).
- ist v ein Blatt, so wird von A f(v) berechnet,
- sonst:
  - muss A einen Nachfolger v_i von v gefunden haben, dessen untere Schranke den Wert F(v) besitzt. Ausserdem müssen für alle Nachfolger obere Schranken ≤ F(v) nachgewiesen worden sein. Insbesondere muss der Minimaxwert von v_i bestimmt worden sein. Permutiere die Nachfolger von v in G‘ nun so, dass v_i in G‘ der erste Nachfolger ist. In G weisen wir dem Knoten v_i den Typ 1 zu, allem anderen Nachfolgern von v den Typ 2.
- v wird mit negamax(v,-∞,∞) untersucht
- Typ(v_i) = 1 in G‘ Typ(v_j) = 2 für j ≠ i in G‘ und v_i wird in G‘ mit negamax(v_i,-∞,∞) untersucht, und alle anderen Nachfolger von v mit negamax(v_j,-∞,-F(v_i)).
  An den v_j kommt es zu Cutoffs, weil der erste Nachfolger von v in G‘ den größten Wert besitzt.

Alphabeta-Algorithmus, Aufwand

TECHNISCHE UNIVERSITÄT DARMSTADT

A2: Sei v ein Knoten mit Typ(v) = 2. Dann gilt:

- A berechnet für v zumindest eine untere Schranke $\beta > -\infty$.
- ist v ein Blatt, so wird von A f(v) berechnet,
- sonst:
  - muss A einen Nachfolger v_i von v gefunden haben, dessen untere Schranke $\beta > -\infty$ ist. Wir weisen v_i den Typ 3 zu, allen anderen Nachfolgern von v den Typ 0. Permutiere die Nachfolger von v in G‘ nun so, dass v_i in G‘ der erste Nachfolger ist.
- v wird mit negamax(v,$-\infty$,β) untersucht
- $Typ(v_i) = 3$ in G‘ $Typ(v_j) = 0$ für $j \neq i$ in G‘ und v_i wird in G‘ mit negamax(v_i,$-\beta$,∞) untersucht. negamax(v_i,$-\beta$,∞) liefert einen Wert $x \leq -\beta$ und es erfolgt ein Cutoff an v.

A3: Sei v ein Knoten mit Typ(v) = 3. Dann gilt:

- A berechnet für v eine obere Schranke für F(v). Er muss also alle Nachfolger betrachten. Alle Nachfolger bekommen den Typ 2.
- ist v ein Blatt, so wird von A f(v) berechnet,
- sonst: v wird in G‘ mit negamax(v,α,∞) untersucht, und $Typ(v_j) = 2$ für alle Nachfolger v_j von v. Die Nachfolger werden gesucht mit dem Aufruf negamax(v_j,$-\infty$,α).

Alphabeta-Algorithmus, Aufwand

TECHNISCHE UNIVERSITÄT DARMSTADT

Folgerung: Es sei G = (V,E,f) ein b/t-uniformer Spielbaum mit Wurzel r. Jeder Algorithmus zur Berechnung von F(r) besucht mindestens $b^{\lfloor t/2 \rfloor} + b^{\lceil t/2 \rceil} - 1$ Blätter von G.

Beweis:
Es sei $k_i(j)$ die Anzahl der Knoten v in Ebene j von G mit Typ(v) = i. Dann ist $k_1(0) = 1$, $k_2(0) = 0$ und $k_3(0) = 0$. Weiter ist

$k_1(t) = k_1(t-1)$
$k_2(t) = (b-1) \cdot k_1(t-1) + b \cdot k_3(t-1)$
$k_3(t) = k_2(t-1)$

Lösung des Systems ergibt:

$k_1(t) = 1$
$k_2(t) = b^{\lceil t/2 \rceil} - 1$
$k_3(t) = b^{\lfloor t/2 \rfloor} - 1$

Computerschach

Die letzte Runde im Mensch-Maschine Schachwettkampf ist beendet

Anmerkung: Dieses Kapitel enthält Aussagen, die man zu Recht als „nicht-wissenschaftlich“ bezeichnen sollte. Es handelt sich dabei um reine Spekulation, und sie wurden deshalb als solche gekennzeichnet. Dies gilt insbesondere für die Folien 109 und 110.

Computerschach hat eine atemberaubende Geschichte

TECHNISCHE UNIVERSITÄT DARMSTADT

1940-1970: Versuche, menschliches Schachspiel nachzuahmen, versanden

1. 1970s: Chess 4.5 ist das erste 'starke' Programm. Es legt Wert auf Sucheffizienz. Erster Computersieg eines Menschenturniers, Minnesota Open 1977.
2. 1983: Belle wurde 'National Master', 2100 ELO.
3. 1988: Hitec erringt einen ersten Sieg gegen einen Großmeister
4. 1988: Deep Thought auf Großmeisterniveau
5. 1992: Die ChessMachine gewinnt die offene Computerschachweltmeisterschaft, ein konventionelles PC-Programm von Ed Schröder.
6. 1997: Deep Blue schlägt Kasparov in einem 6-Spiele Wettkampf.
7. Von der Zeit an, dominieren PC-Programme die Welt, mit einer jährliche Spielstärke-steigerung von ca. 30 ELO. Im Internet gibt es virtuelle Räume, in denen Turniere abgehalten werde, in denen man jederzeit gegen Großmeister oder starke Maschinen spielen kann.

ELO: statistisches Maß; 100 Pkte Differenz entsprechen einer 64% Gewinnchance

Anfänger 1000ELO
Internationaler Meister > 2400
Großmeister > 2500
Menschlicher Weltmeister > 2800

Das Ziel

Nur ein BIG POINT fehlte noch:
stärker spielen als die besten Menschen.

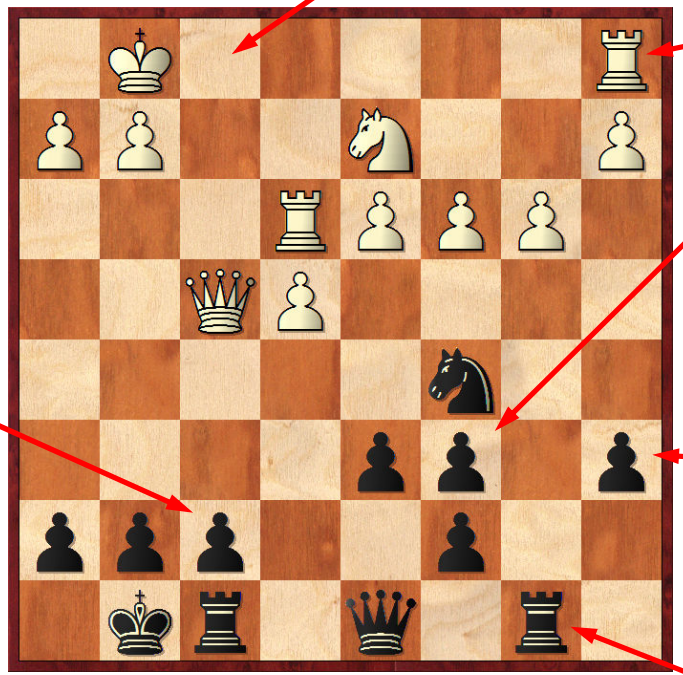
4 Programme lieferten sich bis 2005 ein Rennen:

- **Shredder** von Stefan Meyer-Kahlen,
- **Fritz** von Frans Morsch,
- **Junior** von Amir Ban und Shay Bushinsky
- **Brutus/Hydra** von Chrilly Donninger, Ulf Lorenz (2003-2006), Christopher Lutz, Nasir Ali
- Rybka, Fruit seit 2005

Die oberen vier Programme holten z.B. auf der Computerschach-Weltmeisterschaft in Graz, 2003, mehr als 95% der Punkte gegen das restliche Feld.

Was macht ein Schachprogramm eigentlich?

(I) Stellungsbewertung



London: Die Kontrahenten

TECHNISCHE UNIVERSITÄT DARMSTADT

Adams:

geboren 17.11.1971, lebt in London.

- Wurde 1989 Großmeister, mit dem Gewinn der Britischen Meisterschaften, im Alter von 17 Jahren.
- 1997 britischer Meister.
- 1990 und 2005 zum Spieler des Jahres gekürt und
- gewann zwischen 1993 und 2002 rekordverdächtig viele Turniere.

Zu der Zeit Nr. 1 in England, Nr. 7 der Welt, mit einer Spielstärke von 2741 Elo.

London: Die Kontrahenten

TECHNISCHE UNIVERSITÄT DARMSTADT

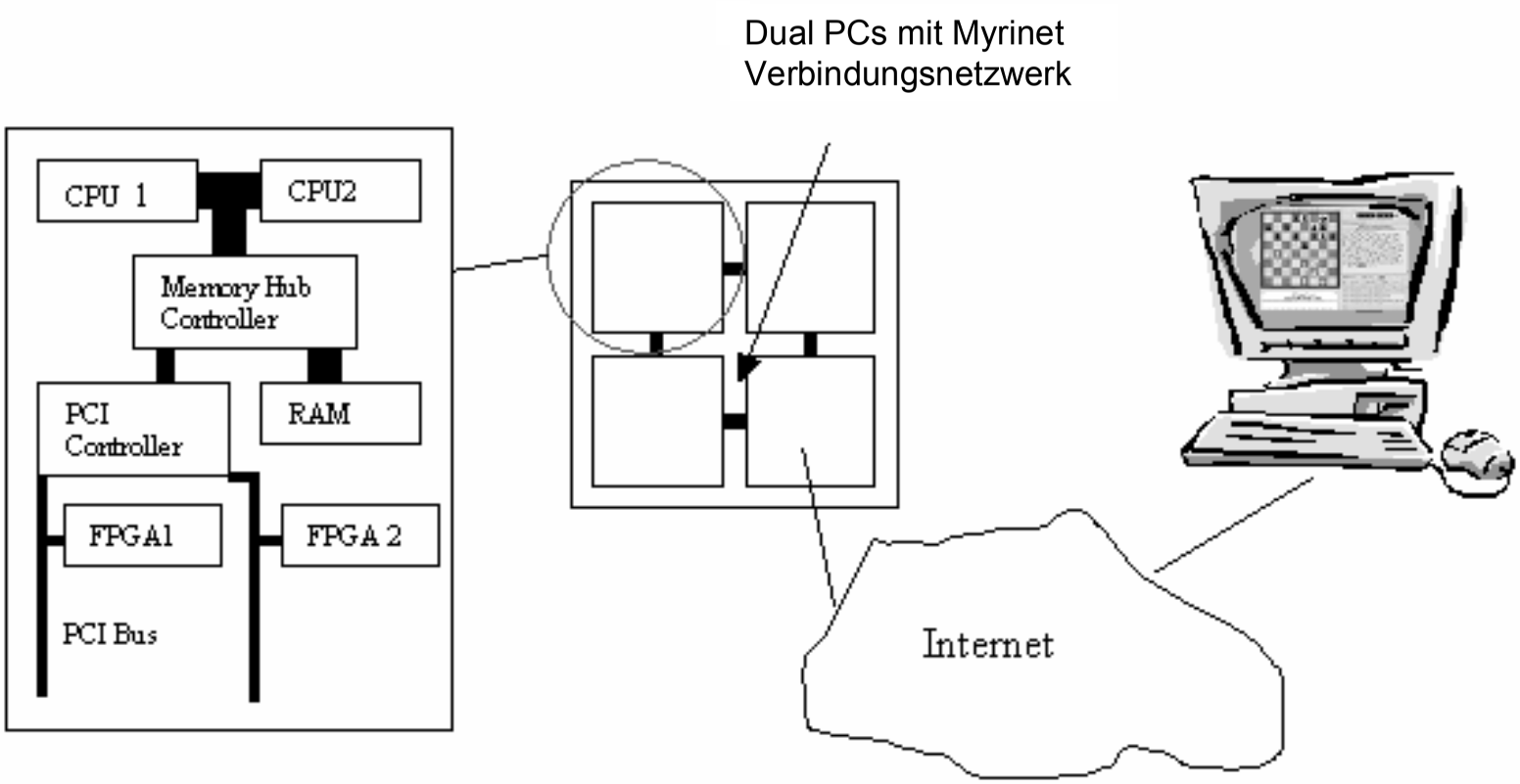
Hydra:

- Die einzige 'Schacheinheit', die jemals (bis 2009) deutlich über 3000 Elo spielt.
- 150 Millionen Schachstellungen pro Sekunde
- typische Rechentiefe nach der Eröffnung: 19 bis 20 Halbzüge
- Cluster mit 32 Pentium Prozessoren, jeweils mit einem XilinX Virtex Pro II 7000 als Schach-Koprozessor

(finanziert und durchgeführt von PAL Computer Systems, VAE, Abu Dhabi)

- Gewinner des 13. IPCCC 2004 mit 6.5 aus 7
- Gewinner gegen Shredder in Abu Dhabi mit 5.5 : 2.5
- Gewinner gegen GM Vladimirov (2630 Elo) mit 3.5 : 0.5
- Mensch-Maschine Mannschaftsweltmeister mit 3.5 : 0.5 gegen Schnitt von 2690 ELO
- Gewinner des 14. IPCCC 2005 mit 8 aus 9

Hydras globale Architektur

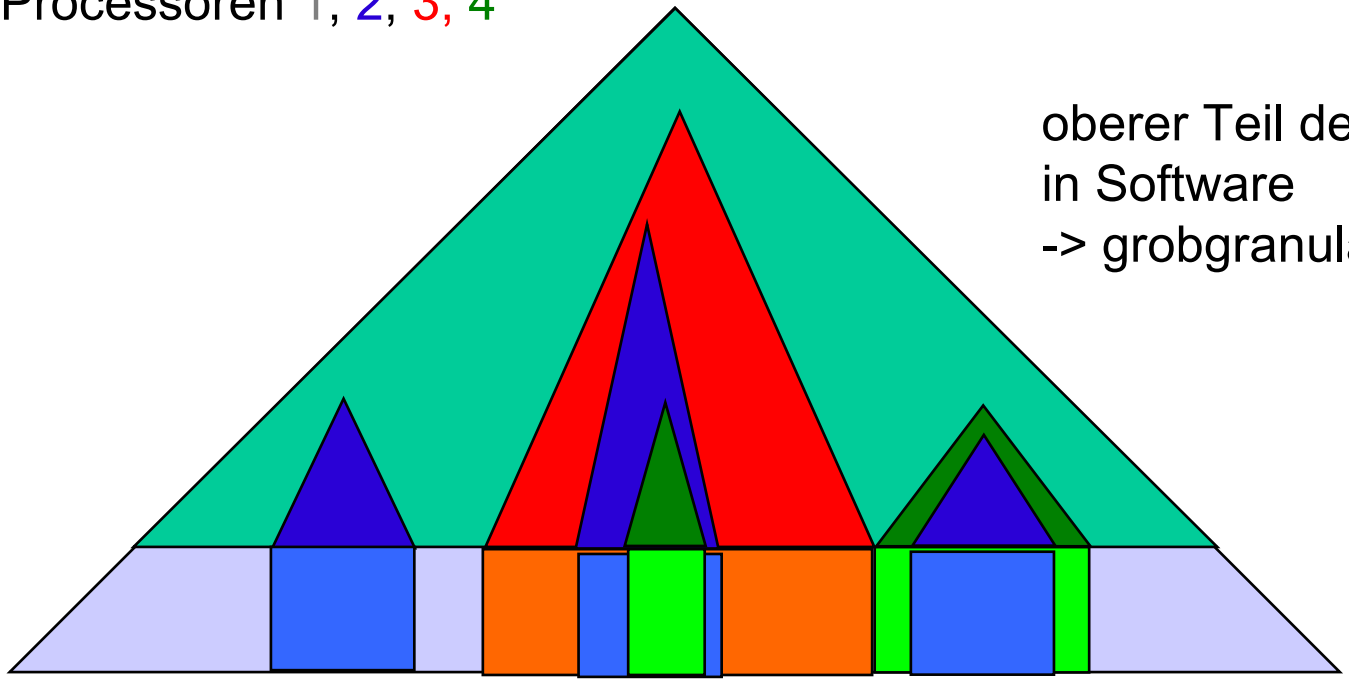


FPGA: Virtex II: VP70 -> ca. 5,4 Millionen Suchknoten pro Sekunde pro Prozessor
bis zu 32 CPUs: 3.0 GHz -> Speedup 17 und somit 100 Millionen Suchknoten pro Sekunde

Zerlegung des Suchbaums

TECHNISCHE UNIVERSITÄT DARMSTADT

Processoren 1, 2, 3, 4



oberer Teil des Baums
in Software
-> grobgranulare Parallelität

die letzten 4 Ebenen in FPGA -> feingranulare Parallelität,
FPGA Karte wird ca. 100000 mal pro Sekunde angestoßen

Paralleler Suchalgorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zerlegung des Suchbaums
- ‘Work Stealing‘: Prozessoren ohne Arbeit senden randomisiert Anfragen
  in den Cluster;
  ein spezieller Prozessor beginnt die Berechnungen wie im sequentiellen Fall
- ein Prozessor, der Arbeit hat und eine Arbeitsanfrage einfängt, gibt ein Teilproblem
  an den Sender der Anfrage ab,
  der Anfragende wird 'worker‘, der andere ‘master‘
- master/worker Beziehungen sind dynamisch, oft geknüpft und wieder gelöst

- Nachrichten: REQUEST, WORK, NO-WORK, WINDOW_UPDATE,
  CUTOFF, RESULT

- Problemauswahl für Abgaben: nah an der Wurzel, YBWC

Speedups

	Zeit(s)	SPE	SO %
1	24213	1	0
2	12139	1.99	0
4	6888	3.5	3.6
8	3488	6.5	-1
16	2011	12	9.3
32	1424 *	17	80 *

* geschätzt, da andere Versionen von Hydra zugrunde liegen

FPGA (Field Programmable Gate Array)

TECHNISCHE UNIVERSITÄT DARMSTADT

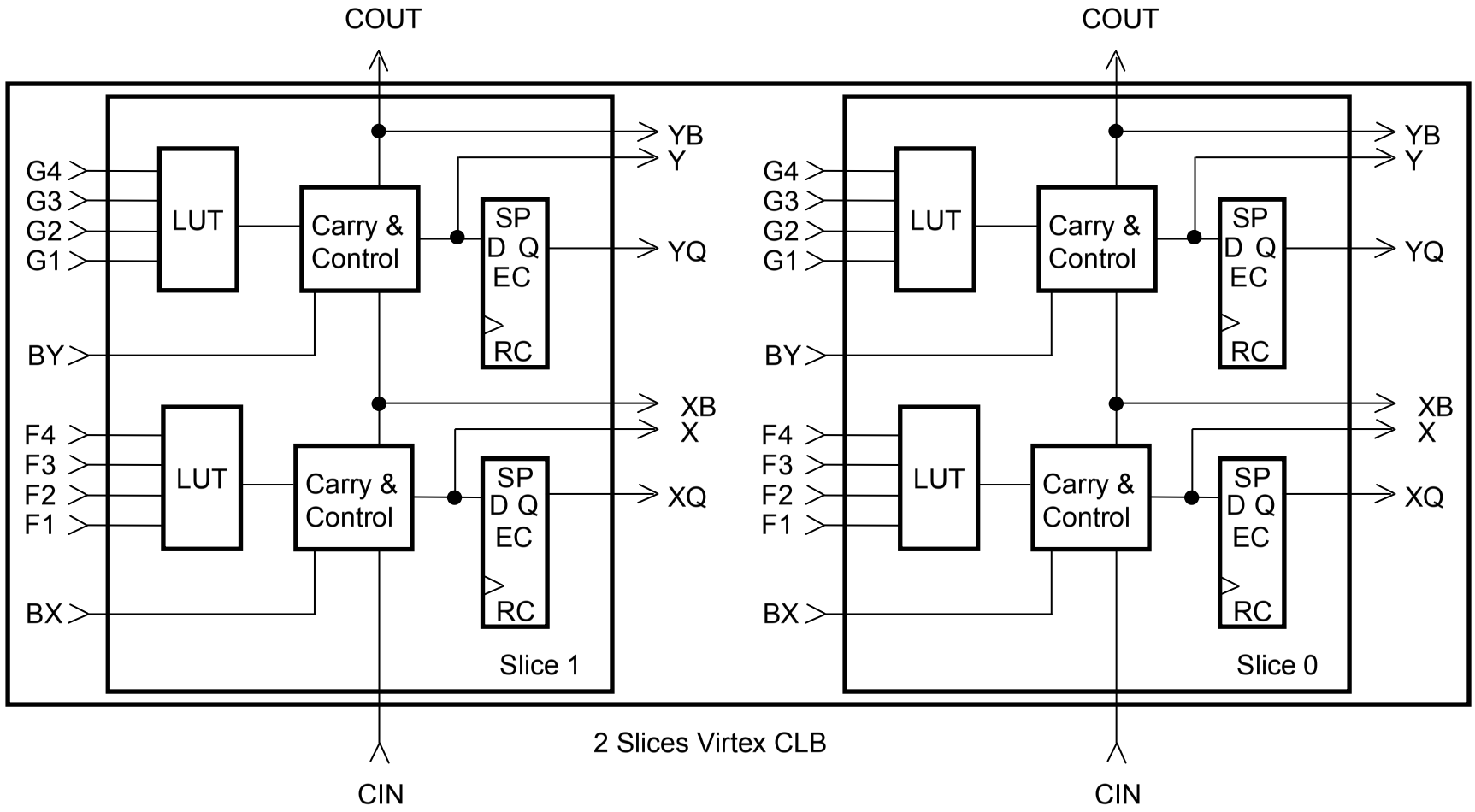
- **ist rekonfigurierbare Logic für rapid prototyping und für die Implementierung digitaler Systeme**
- **ist eine Art in RAM simulierter Hardware, die fein-granulare Parallelität erhaltend**
- **ist eine Möglichkeit Logik zu realisieren**

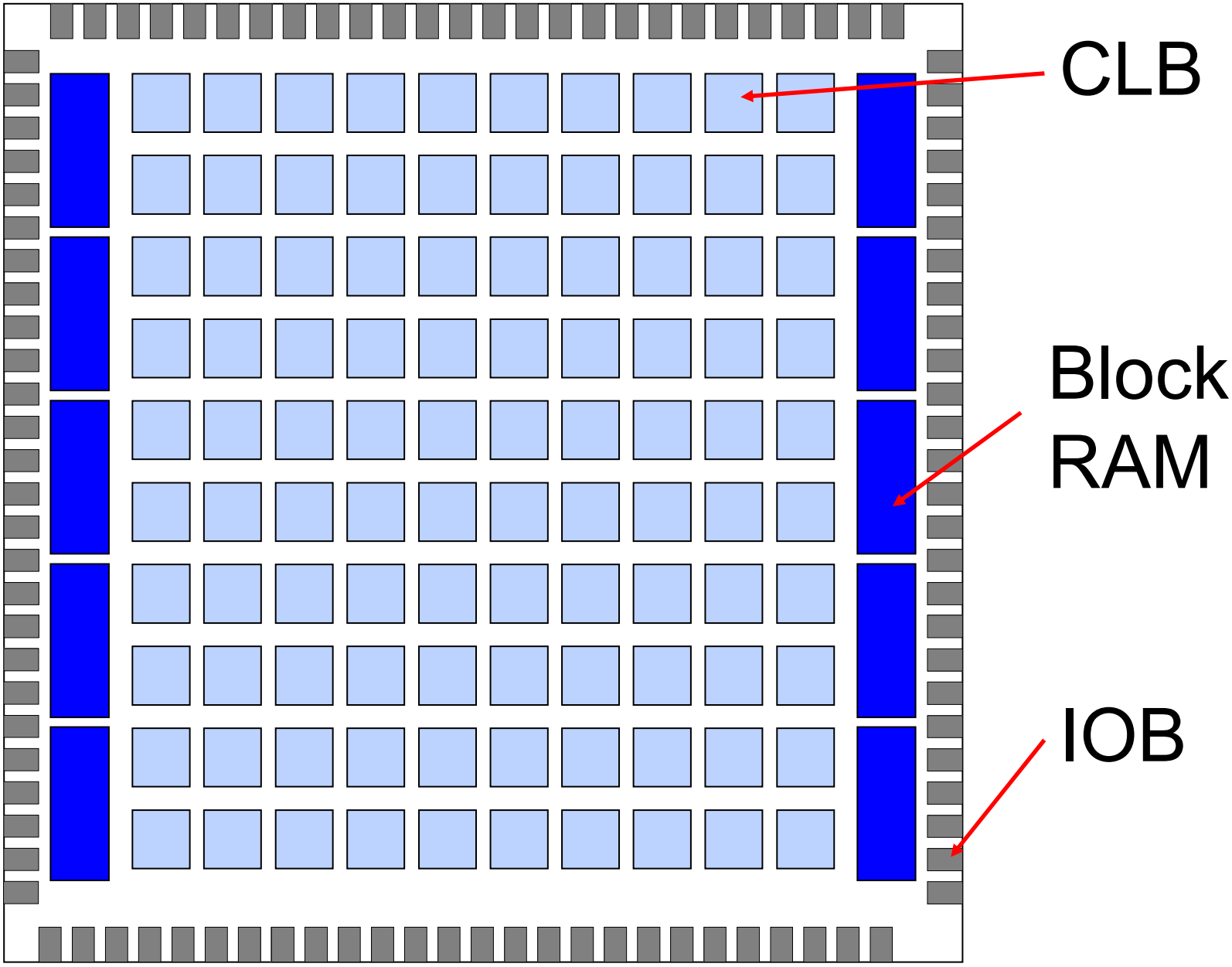
Hauptvorteile (für Hydra):

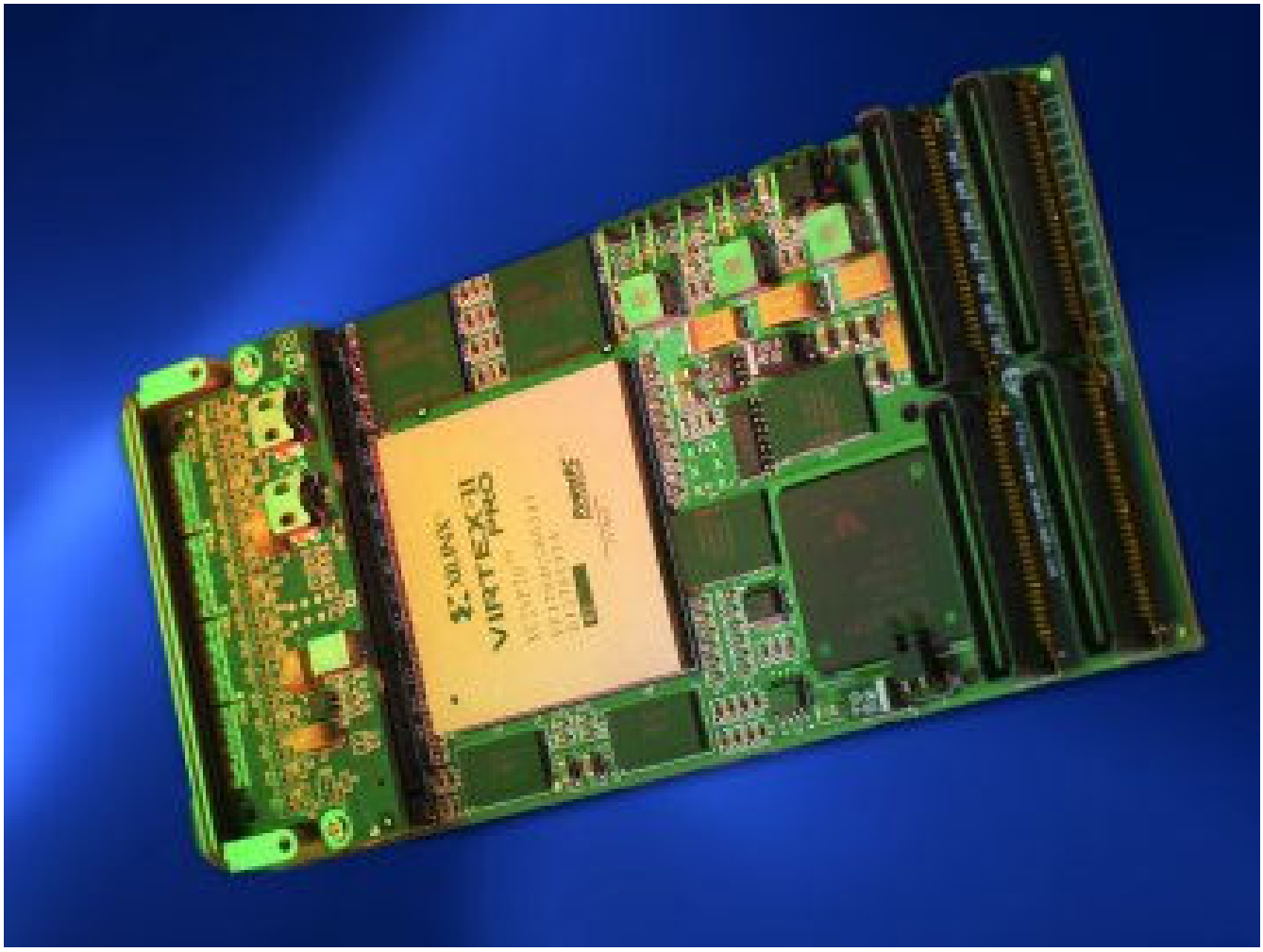
1. Das Hinzufügen von mehr Schachwissen erfordert zusätzlichen Platz, kostet aber nahezu keine zusätzliche Berechnungszeit

2. FPGA Code kann ge-debugged und schnell wie in Software geändert werden, ohne langwierige ASIC Entwicklungszeiten

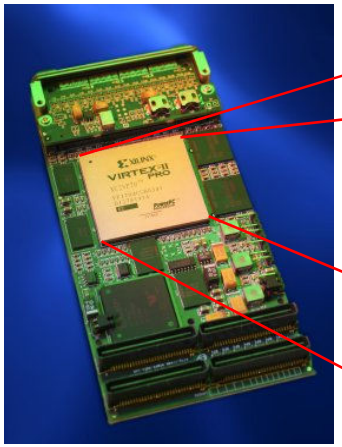
3. feingranulare Parallelität kann genutzt werden -> genMove, doMove, eval, undoMove in 9 Taktzyklen, bei einer clock-rate von 50MHz

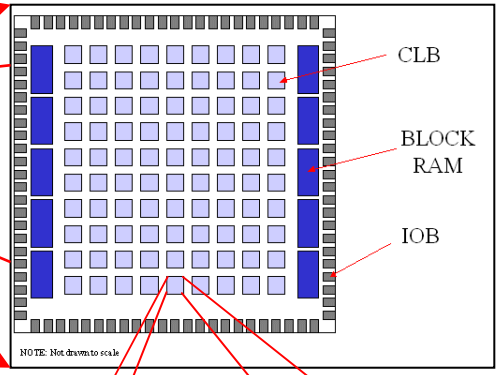

COUT
COUT
YB
Y
YQ
XB
X
XQ
G4
G3
G2
G1
BY
F4
F3
F2
F1
BX
LUT
Carry &
Control
SP
D Q
EC
RC
Slice 1
Slice 0
2 Slices Virtex CLB
CIN
CIN

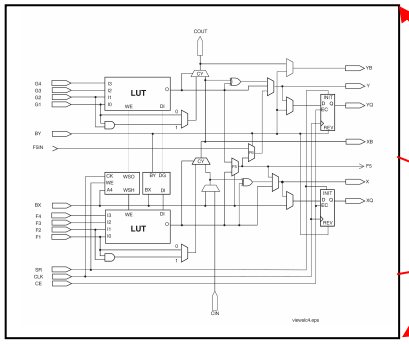

CLB
Block
RAM
IOB


VIRTEX-II
PRO

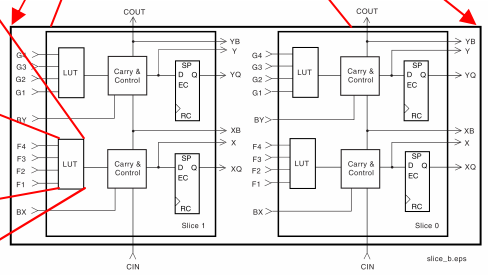
HPRCC
High Performance Reconfigurable Cluster-Computing







Detailed View of Virtex Slice

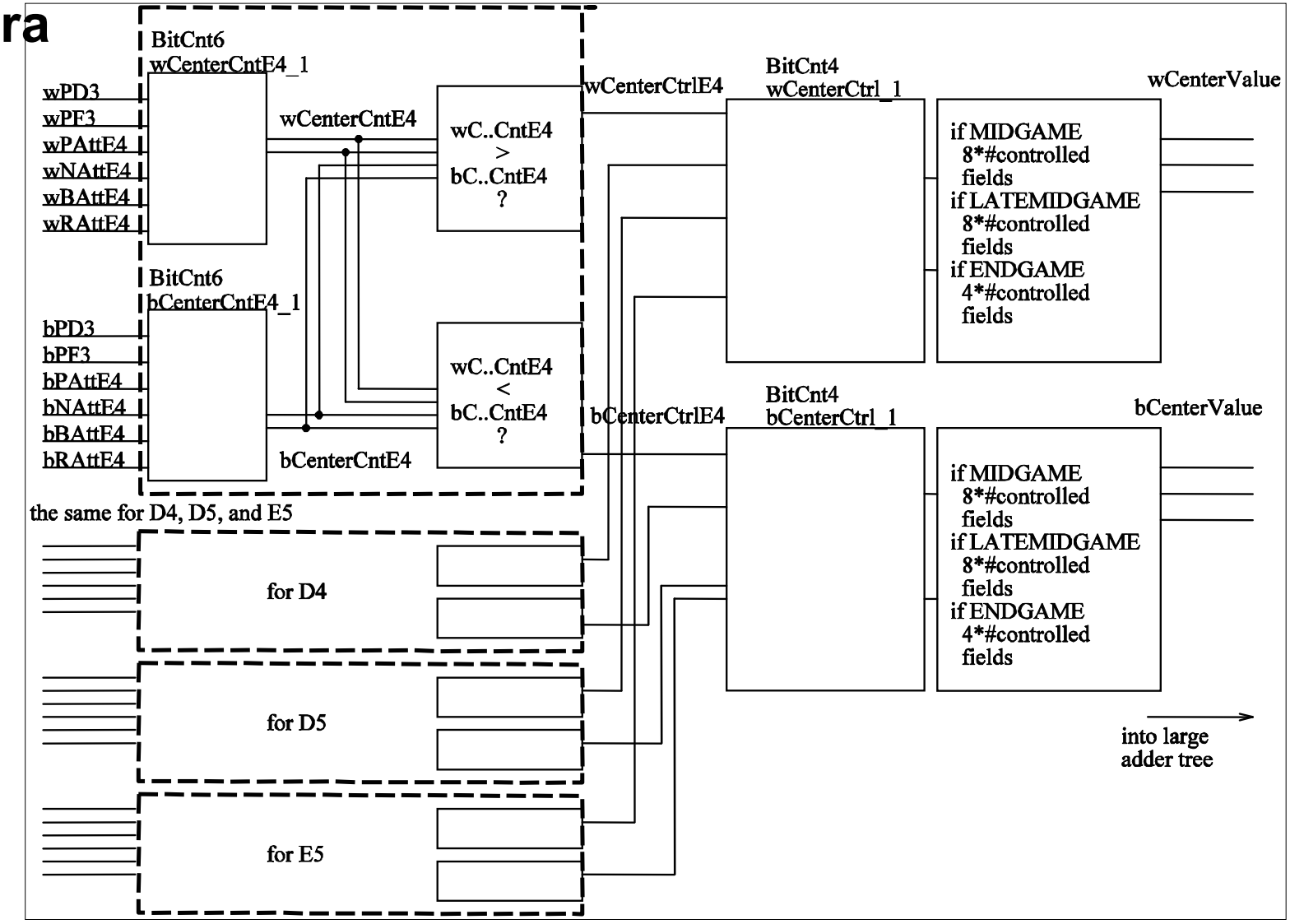


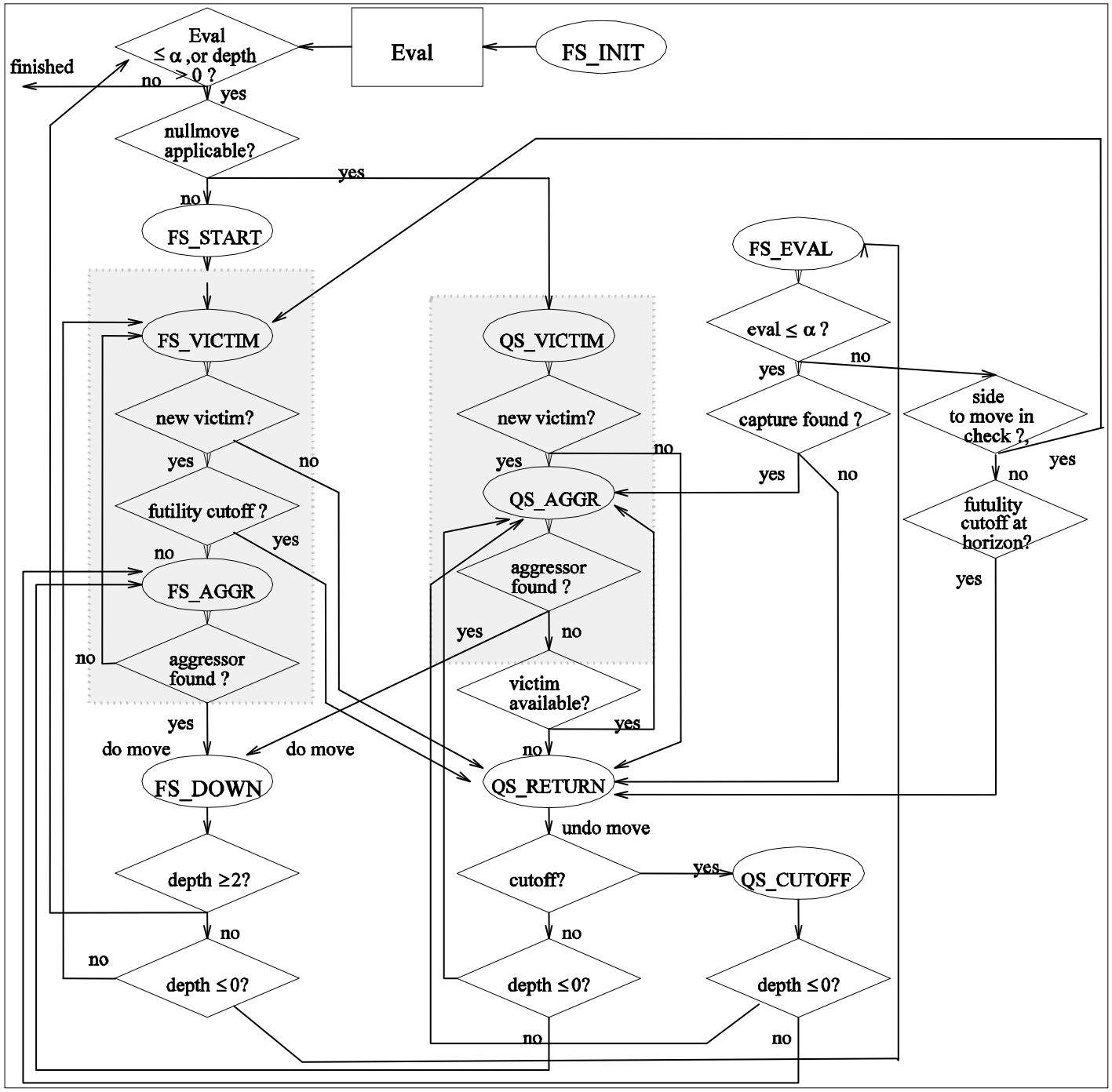
2 Slices Virtex CLB

FPGA, Verilog

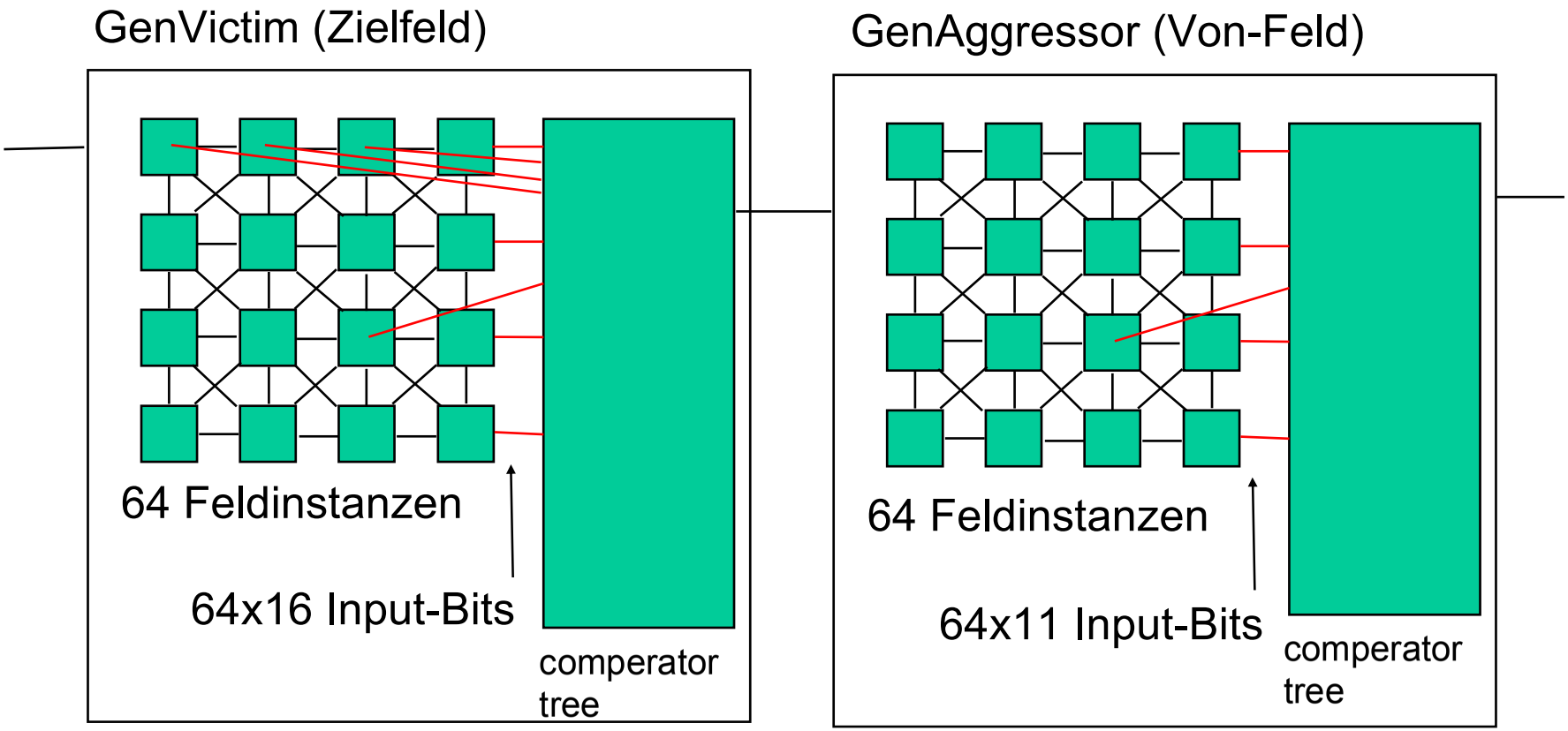
```
module Search();
  ControlLogic;
          // Enables/disables the modules and converts data to the correct format.
          // E.g. the module Board does not know whether DoMove or UndoMove is
          // done. The input is always piece-from, piece-to. The
          // ControlLogic converts the move
          // information in the correct format for the Board module.
  AlphaBetaFSM;   // Alphabeta search with Finite State Machine.
          // The FSM sets a few signals like "DoMove", "UndoMove",
          // "Next-Victim". The next state is determined from the
          // output of the modules and the current state. The signals are used by the
          // ControlLogic to control the operation of the modules.
          // The FSM has 54-States, but some states are simply waitstates. It changes no
          // signals. Only a transition to the next
          // state can be done.
  CastleStatus();      // Determine castle status                                  0.5 cycles
  GenVictim();         // Generates next ToSquare.                                 2.0 cycles
  GenAggressor();    // Generates next FromSquare for a given ToSquare.  2.0 cycles
  TestCheck();        // Tests if the king is in check.                           2.0 cycles
  Board();             // Board respresenation and board update                    1.0 cycles
  SearchStack();     // "Local variables" for the search control.                 0.5 cycles.
  Evaluate();          // Evaluation.                                              3.0 cycles.
endmodule

module Evaluate();
  HandleSpecialCases; // E.g: Is there insufficient mate material?
  WeightedSum;          // Different weights for different game phases.
  GamePhase();          // Determines the game phase.
  PieceValues();        // Sums up the material value.
  FirstOrderValues();   // piece square tables
  PawnStructure();      // pawn structure and passed pawns
  KingCover();          // pawn-cover around the king
  Dynamic();            // Main part. This evaluation bases on attack and defense
                        // relations. E.g. attack on opponents king, mobility
endmodule
```

Hydra


BitCnt6
wCenterCntE4_1
wPD3
wPF3
wPAttE4
wNAttE4
wBAttE4
wRAttE4
wCenterCntE4
wC..CntE4
>
bC..CntE4
?
wCenterCtrlE4
BitCnt4
wCenterCtrl_1
wCenterValue
if MIDGAME
8*#controlled
fields
if LATEMIDGAME
8*#controlled
fields
if ENDGAME
4*#controlled
fields
BitCnt6
bCenterCntE4_1
bPD3
bPF3
bPAttE4
bNAttE4
bBAttE4
bRAttE4
bCenterCntE4
wC..CntE4
<
bC..CntE4
?
bCenterCtrlE4
BitCnt4
bCenterCtrl_1
bCenterValue
if MIDGAME
8*#controlled
fields
if LATEMIDGAME
8*#controlled
fields
if ENDGAME
4*#controlled
fields
the same for D4, D5, and E5
for D4
for D5
for E5
into large
adder tree


Eval
≤ α ,or depth
> 0 ?
Eval
FS_INIT
finished
no
yes
nullmove
applicable?
yes
no
FS_START
FS_VICTIM
new victim?
yes
no
futility cutoff ?
yes
no
FS_AGGR
aggressor
found ?
no
yes
do move
FS_DOWN
depth ≥2?
no
no
depth ≤0?
QS_VICTIM
new victim?
yes
no
QS_AGGR
aggressor
found ?
yes
no
victim
available?
yes
no
QS_RETURN
undo move
cutoff?
yes
QS_CUTOFF
no
depth ≤0?
no
FS_EVAL
eval ≤ α ?
yes
no
capture found ?
yes
no
side
to move in
check ?,
yes
no
futulity
cutoff at
horizon?
yes
do move

Hydra



1. Besetzte Felder senden ein Signal in GenVictim.
2. Freie Felder leiten Signale weiter
3. Alle Felder, die ein Signal bekommen, sind potentielle Zielfelder.
4. Der Comperator-tree wählt attraktivstes Zielfeld (z.B. Schlagzug).

1. Gewinnerfeld generiert Signal von 'super piece'.
2. Freie Felder leiten das Signal weiter.
3. Von eigenen Figuren besetzte Felder sind potentielle Von-Felder.
4. Der Comperator-tree wählt das attraktivste Von-Feld

Das Entwicklerteam hinter Hydra

TECHNISCHE UNIVERSITÄT DARMSTADT

Dr. Chrilly Donninger

GM Christopher Lutz

Dr. Ulf Lorenz

Universität Paderborn

Prof. Dr. B. Monien

Paderborn Center for Parallel Computing

Nasir Ali

PAL Computer Systems

PAL Gruppe Abu Dhabi

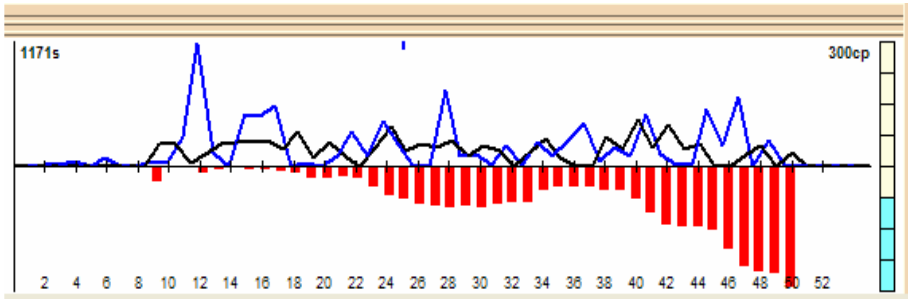
Scheich Tahnoon

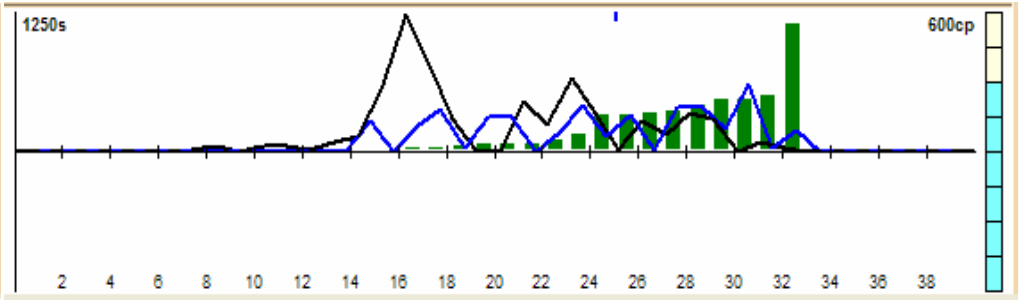
London: Hydra gegen Adams 5,5 : 0,5

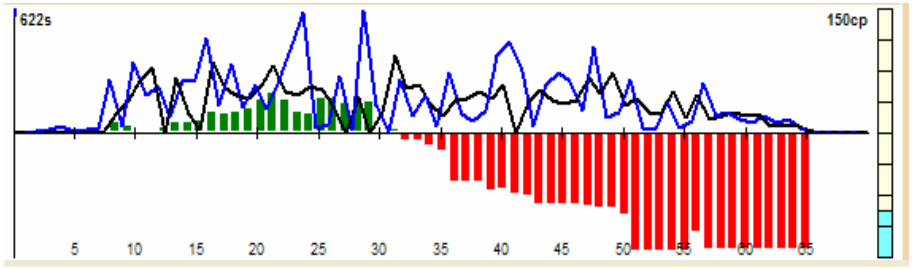
TECHNISCHE UNIVERSITÄT DARMSTADT

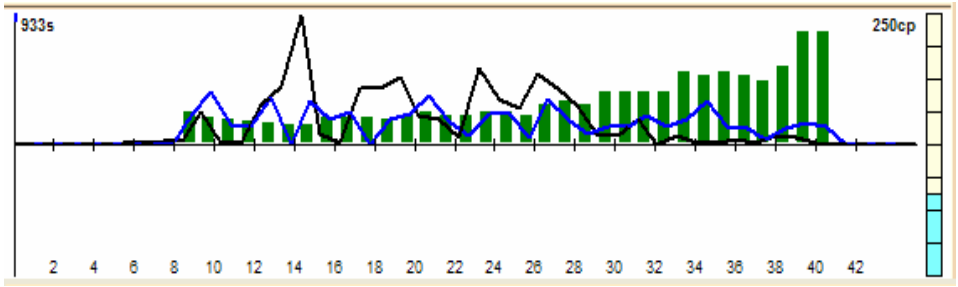
London 2005

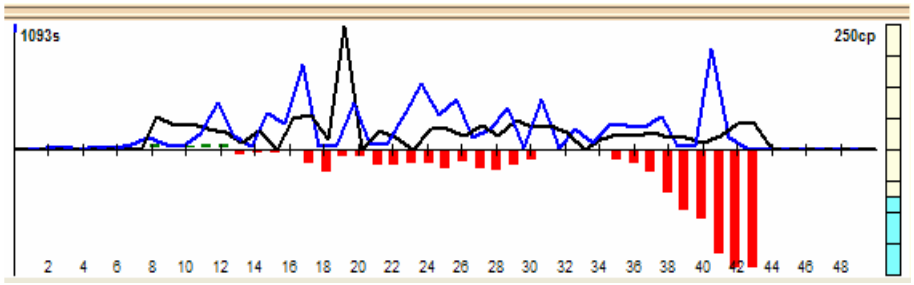
TECHNISCHE UNIVERSITÄT DARMSTADT

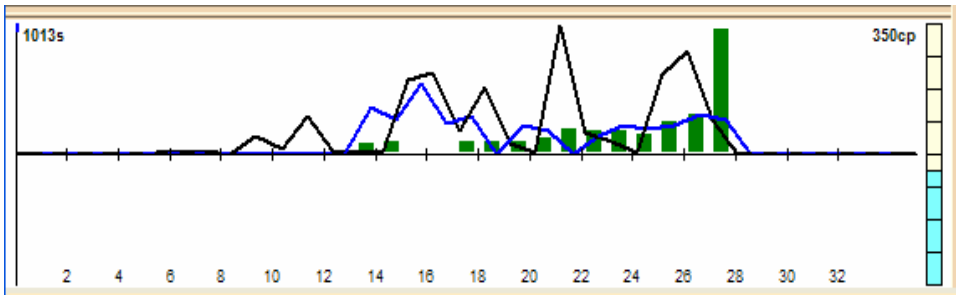












Typische Thesen aus den 80er Jahren (*) :

- Top-Großmeister spielen fast perfekt Schach
- Ein Top-Großmeister kann mit weiß immer Remis halten, wenn er nur kein Risiko eingeht und von vornherein auf Remis spielt.
- 3000 Elo ist eine natürliche Perfektionsgrenze
- Turmendspiele mit einem Wenigerbauern sind einfach Remis zu halten.

Alles Falsch!

Katastrophale Selbsteinschätzung.

(*) Diese Folie präsentiert auch aus Sicht Ihres Dozenten keine wissenschaftliche Erkenntnis. Sie dient allein der Anregung zu Diskussion!

Ein zarter Hauch von Intelligenz (*)

TECHNISCHE UNIVERSITÄT DARMSTADT

1. **Die Menschlichen Meister beanspruchen, daß sie, obwohl sie gegen ein Programm verlieren, das Spiel besser verstehen. Sie sprechen von einem 'tieferen Verständnis'.**

Frage: Tun sie das? Was könnte das sein: 'tiefer verstehen'? Gibt es überhaupt etwas zu verstehen, außer 'Jede Stellung ist entweder gewonnen, verloren oder remis'?

Antwort: Ja, es gibt da was ...

(*) Diese Folie präsentiert auch aus Sicht Ihres Dozenten keine wissenschaftliche Erkenntnis. Sie dient allein der Anregung zu Diskussion!

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computerschach

Grundlagen II

Spielbäume und Fehlerfilter

TECHNISCHE UNIVERSITÄT DARMSTADT

Untersuchung des Phänomens

Fragestellung: Was ist in diesen Spielbäumen z.B. des Schachspiels, das heuristische Spielbaumsuche so erfolgreich macht?

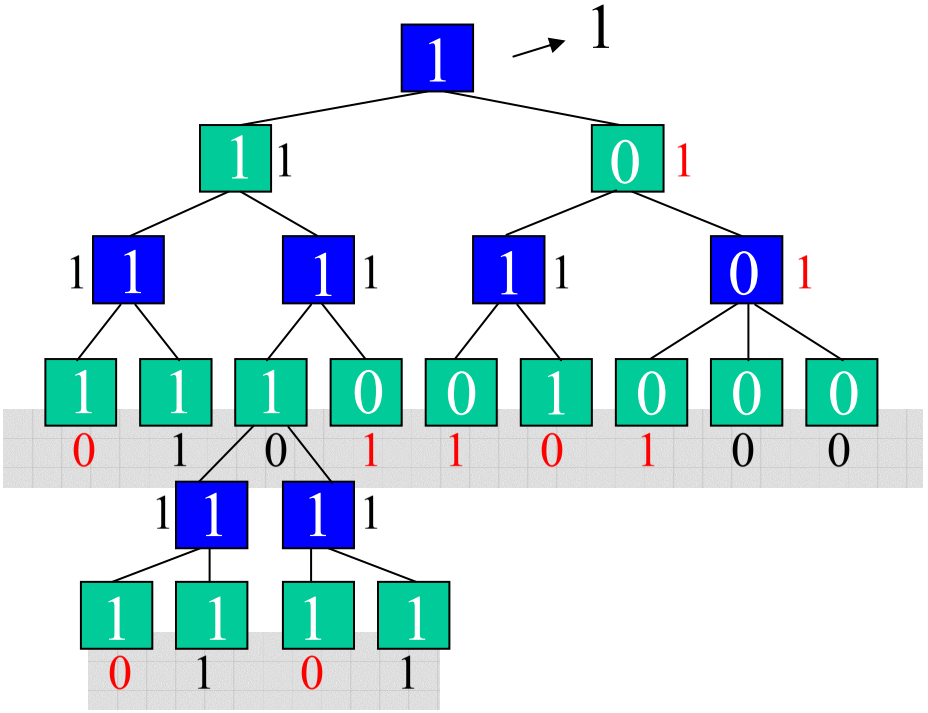
(Nau ’79; Pearl ’83; Schrüfer ’86; Althöfer ’88; Scheucher&Kaindl ’89, Lorenz&Monien STACS ’02, TCS ’05)

Anwendung: Starkes Spiel gegen schwächere Gegner
(Lorenz ESA ’04, ICGA Journal ’06)

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Gegeben: **Spielbaum G**, jeder Knoten hat einen **´echten´ Wert** 0|1; diese Werte gehorchen dem Minimax-Prinzip. So genannte **´heuristische´ Werte** werden den Blättern von G zugewiesen, und diese heuristischen Werte werden genutzt, um heuristische Minimax-Werte für innere Knoten zu bestimmen.



Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Verschiedene Fragestellungen sind möglich:

- Wie viele Fehler darf ich im günstigsten Fall machen?
  wenn Spielbaum G b/t-uniform ist: $b^t - b^{\lfloor t/2 \rfloor}$

- Sei n die Anzahl der Blätter von G. Wie wirkt es sich aus, wenn man (ungefähr) k Fehler bei Blattbewertungen macht?

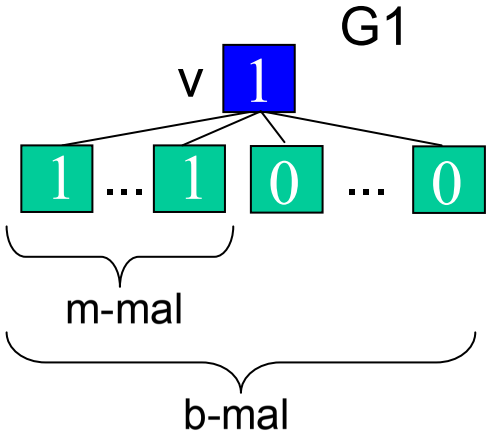
- Wie viele Fehler darf man an den Blättern im schlimmsten Fall machen?

TECHNISCHE UNIVERSITÄT DARMSTADT

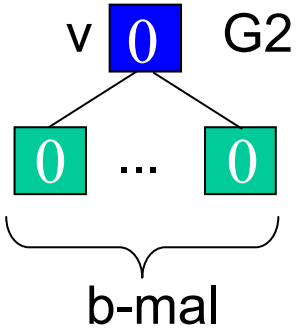
Modell I

- Fehler an Blättern werden mit Fehlerwahrscheinlichkeit $\overline{p}$ (= 1-p) gemacht.

  $v_1,...v_b$ seien die Nachfolger von v, $g_1(p),...,g_b(p)$ seien die Wahrscheinlichkeiten, dass die heuristischen Werte $h_1,...,h_b$ der Knoten $v_1,..,v_b$ gleich den echten Werten $w_1,..,w_b$ sind. Dann ist die Wahrscheinlichkeit dafür, dass der heuristische Minimaxxwert von v gleich dem echten Wert von v ist wie folgt:



$$Q_v(p) = 1 - (\prod_{i=1}^{m}(1-g_i(p)) \bullet \prod_{i=m+1}^{b} g_i(p))$$



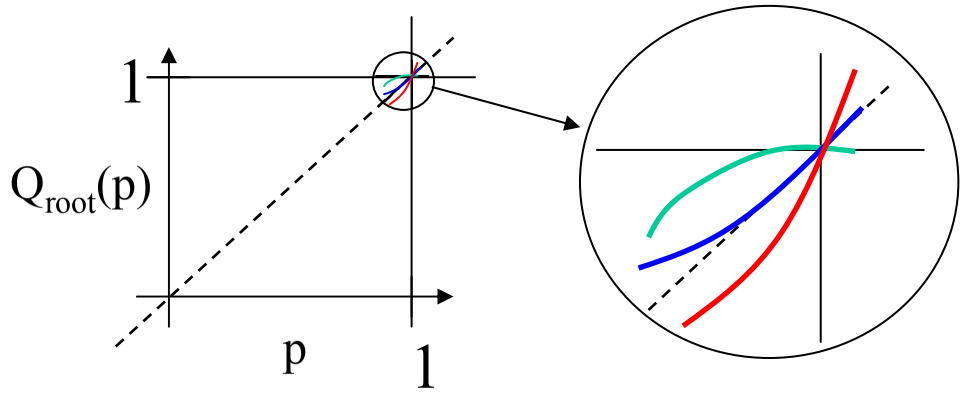
$$Q_v(p) = \prod_{i=1}^{b} g_i(p)$$

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Modell I

- Fehler an Blättern werden mit Fehlerwahrscheinlichkeit $\overline{p}$ (= 1-p) gemacht.
- Für jeden Knoten v des Spielbaums G, gibt es somit ein „Qualitätspolynom“ $Q_v(p)$, welches die Wahrscheinlichkeit dafür angibt, dass der echte und der heuristische Wert am Knoten v gleich sind.



- Intuitive Erklärung für besseres Spiel bei tieferer Suche:
  Super-Bäume haben kleinere Fehlerwahrscheinlichkeiten an ihren Wurzeln, zumindest wenn die Fehlerwahrscheinlichkeit am Blatt klein genug ist.
- Robustheit kann dann definiert werden mit Hilfe $Q_{root}^{(k)}(1)$, k=1,...

TECHNISCHE UNIVERSITÄT DARMSTADT

Modell I, Zusammenhang mit Average-Case Analyse

Sei G Spielbaum mit n Blättern und s der 0/1-Blattstring der echten Werte. s‘ sei der verfälschte String. p sei die Wahrscheinlichkeit dafür, einen heuristischen Blattwert korrekt zu erkennen. Die Anzahl korrekter heuristischer Blattbewertungen ist Binomialverteilt. Man kann dann sagen: „Man macht ungefähr n*(1-p) Fehler“

0 1 0 1 1 1 0 - s

0 0 1 0 0 0 1 - s'_1 } C_1 -> c_1

1 1 1 0 0 0 1 - s'_2 }

....

0 1 1 0 0 0 1 - s'_3 } C_2 -> c_2

....

C_i sind Cluster, die Strings mit genau c_i korrekten Bewertungen enthalten.

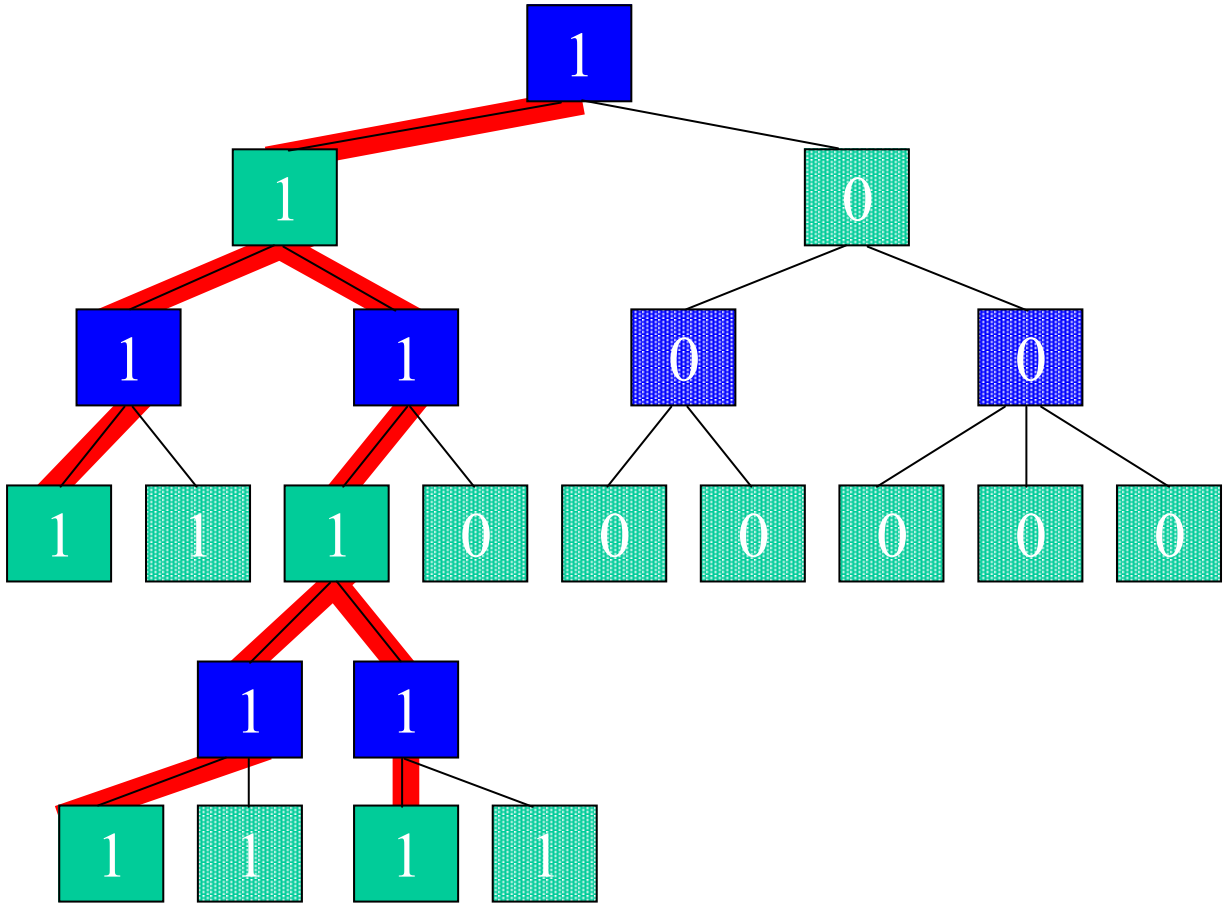
$Q_G(p) = \Sigma^n_{i=0}$ Prob(heur. Wurzelwert ist korrekt | es gibt genau i richtig klassifizierte Blätter) * Prob(genau i heur. Blattwerte sind korrekt)

$$= \sum_{i=0}^{n} c_i \mid C_i \mid \binom{n}{i} p^i (1-p)^{n-i}$$

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

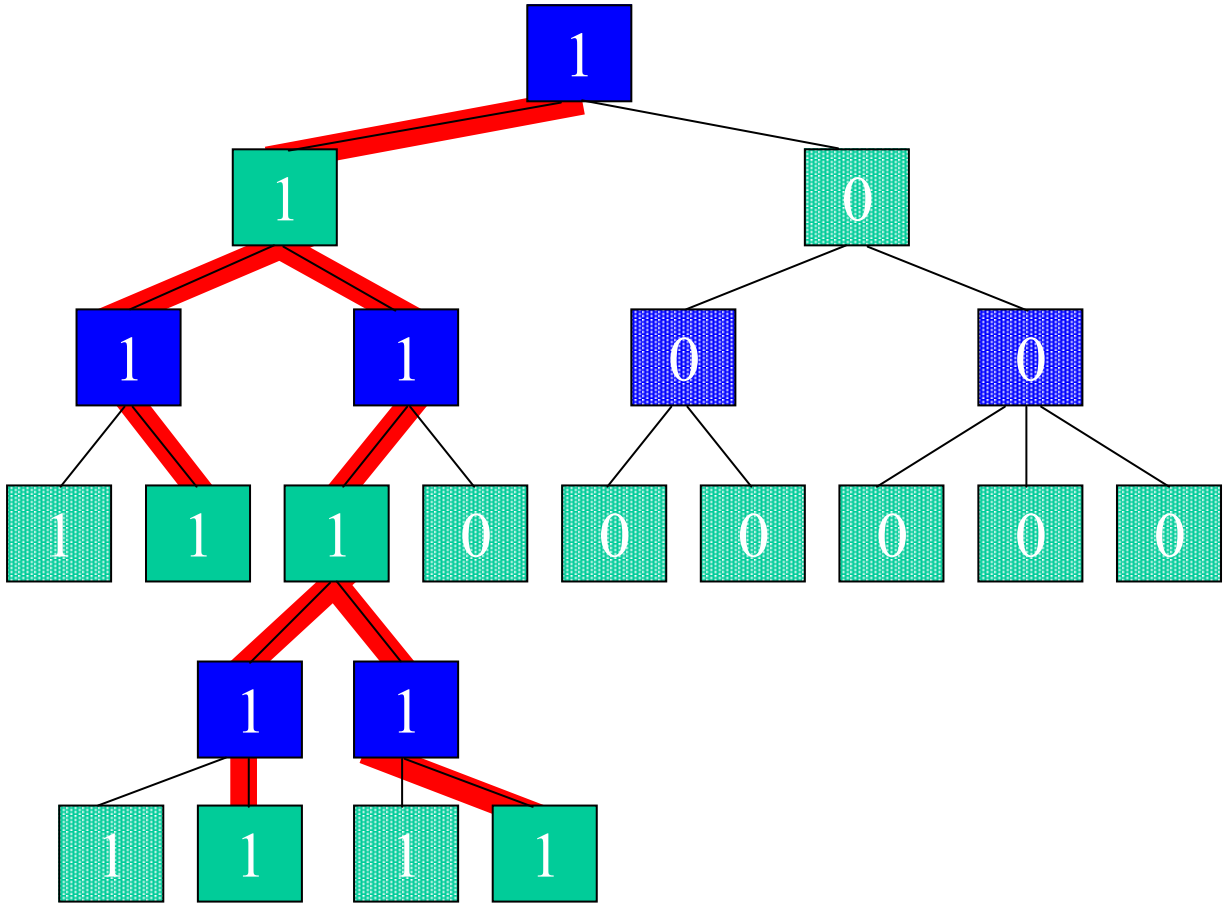
Modell II



Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Modell II



Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Ergebnisse

Lemma: $Q_{root}'(1) = 0$ oder $Q_{root}'(1) \geq 1$, wobei Q‘ erste Ableitung von Q

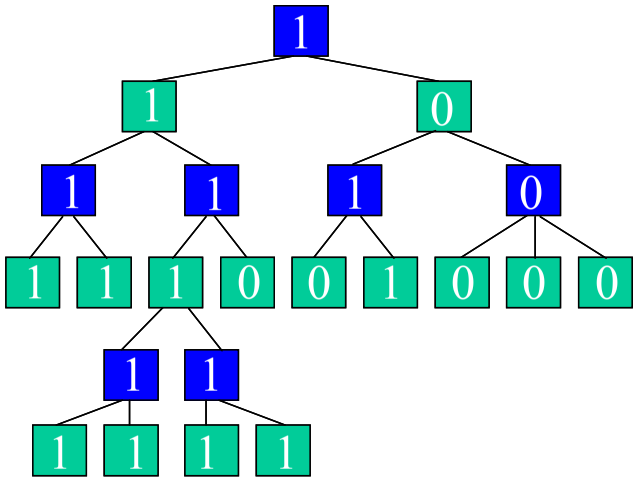
Lemma: Falls $Q_{root}'(1) \geq 1$, beschreibt $Q_{root}'(1)$ die Anzahl der Blätter, die den Wurzelwert durch einen Single-Flip ändern können.

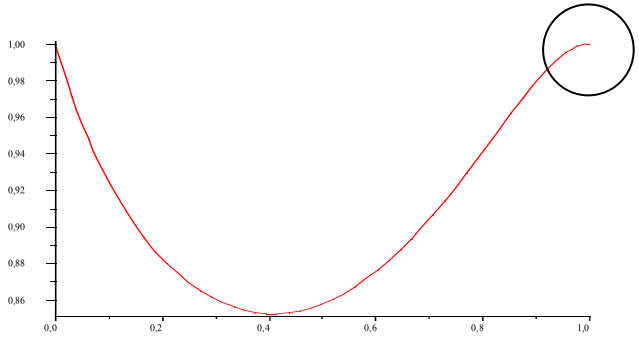
Theorem: $Q_{root}'(1) = 0$ gilt g.d.w. G mindestens 2 blatt-disjunkte Strategien enthält, die den Wurzelwert belegen.

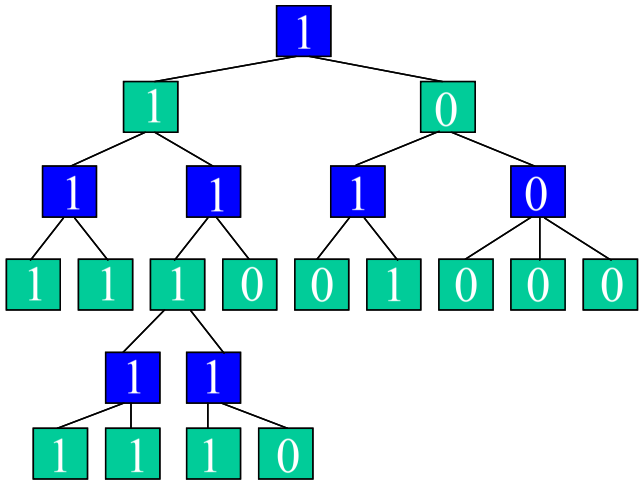
Fehleranalyse

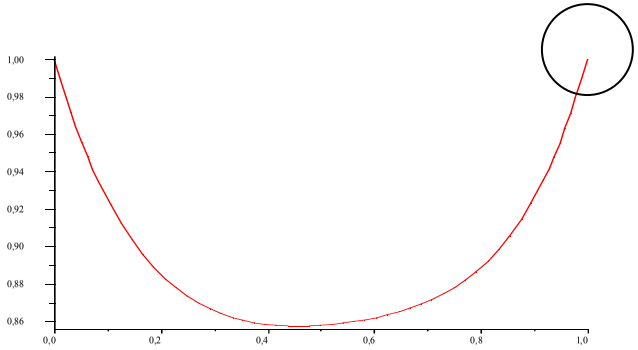
TECHNISCHE UNIVERSITÄT DARMSTADT

Ergebnisse









Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Ergebnisse

Die zwei Modelle mit ihren Robustheitsmaßen sind äquivalent zueinander.

Es gibt n+1 blattdisjunkte Strategien in G, die alle den Wurzelwert von G belegen.

(Modell II)

<=>

$$Q_{root}^{(n)}(1) = Q_{root}^{(n-1)}(1) = \ldots = Q_{root}^{(1)}(1) = 0$$

(Modell I)

Taylorreihenentwicklung $f(p) = f'(1)(p-1)+\ldots+(f^{(n)}(1)/n!)*(p-1)^n + R_{n+1}(p)$ führt uns zu

$$| Q_{root}(p) - Q_{root}(1) | = O((1-p)^{n+1})$$ **g.d.w. es n+1 viele blatt-disjunkte Startegien gibt.**

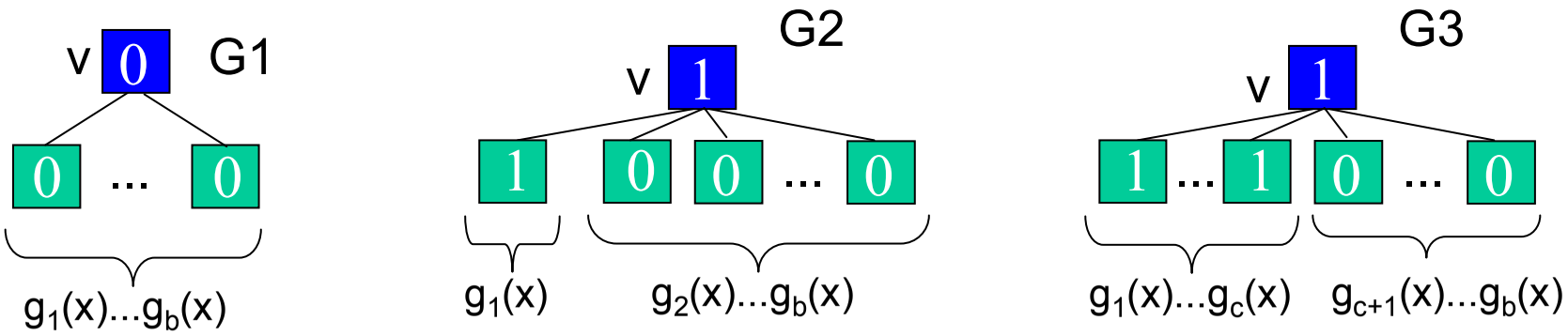
Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

ad Lemma: $Q_{root}'(1) = 0$ oder $Q_{root}'(1) \geq 1$, wobei Q' erste Ableitung von Q

ad Theorem: $Q_{root}'(1) = 0$ gilt g.d.w. G mindestens 2 blatt-disjunkte Strategien enthält, die den Wurzelwert belegen.

Betrachte die folgenden 3 Tiefe-1 Bäume:



Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

$$Q_{G1}(x) = g_1(x) \cdots g_b(x)$$

$$Q_{G2}(x) = 1 - (1 - g_1(x)) \cdot g_2(x) \cdots g_b(x)$$

$$Q_{G3}(x) = 1 - (1 - g_1(x)) \cdots (1 - g_c(x)) \cdot g_{c+1}(x) \cdots g_b(x)$$

$$Q'_{G1}(x) = \sum_{i=1}^{b} g_1(x) \cdots g'_i(x) \cdots g_b(x)$$

$$Q'_{G2}(x) = -(1 - g_1(x))' \cdot g_2(x) \cdots g_b(x) + \sum_{i=2}^{b} (1 - g_1(x)) \cdot g_2(x) \cdots g'_i(x) \cdots g_b(x)$$

$$Q'_{G3}(x) = -\sum_{i=1}^{c} (1 - g_1(x)) \cdots (1 - g_i(x))' \cdots (1 - g_c(x)) \cdot g_{c+1}(x) \cdots g_b(x)$$

$$- \sum_{i=c+1}^{b} (1 - g_1(x)) \cdots (1 - g_c(x)) \cdot g_{c+1}(x) \cdots g'_i(x) \cdots g_b(x)$$

$$Q'_{G1}(1) = g'_1(1) + \cdots + g'_b(1)$$

$$Q'_{G2}(1) = g'_1(1)$$

$$Q'_{G3}(1) = 0$$

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

ad Theorem: Es gibt n+1 blattdisjunkte Strategien in G, die alle den Wurzelwert von G belegen. <=> $Q_{root}^{(n)}(1) = Q_{root}^{(n-1)}(1) = \ldots = Q_{root}^{(1)}(1) = 0$

Allgemein läßt sich die n-te Ableitung eines Produkts von Polynomen darstellen als

$$\sum_{y_1+\ldots+y_b=n} a(y_1,\ldots,y_b) g_1^{(y_1)}(x) \cdots g_b^{(y_b)}(x)$$

mit geeigneten $a(y_1,\ldots,y_b) \in \mathbb{N}$.

Zu betrachten sind nun wieder die Ableitungen von Q_{G1}, Q_{G2}, Q_{G3}.

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahmen:

(i) Für alle i≤n gilt: Für alle Spielbäume G gibt es i blattdisjunkte Strategien in G, die alle den Wurzelwert von G belegen. <=> $Q_G^{(i-1)}(1) = \ldots = Q_G^{(1)}(1) = 0$

(ii) Für alle G‘ $\in$ {G1, G2, G3} soll gelten: Es gibt n blattdisjunkte Strategien in G‘, die alle den Wurzelwert von G' belegen **und** $Q_{G'}^{(n-1)}(1) = \ldots = Q_{G'}^{(1)}(1) = 0$

(iii) Für alle i $\in$ {1,..,n-1} gilt: das Vorzeichen von $Q_{G'}^{(i)}(1) = (-1)^{i-1}$

Bemerkung: Im folgenden machen wir Vorbetrachtungen für einen Induktionsbeweis über die Anzahl von blattdisjukten Strategien und über die Höhe der Bäume. (i) und (iii) werden die Induktionsvoraussetzung bilden, und (II) wird aus „$Q_G^{(n)}(1) = \ldots = Q_G^{(1)}(1) = 0$“ oder aus „es gibt n+1 blattdisjunkte Stragegien ...“ hergeleitet werden.

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

n-te Ableitung für G1:

$$Q_{G1}^{(n)}(x) = \sum_{y_1+\ldots+y_b=n} a(y_1,\ldots,y_b) g_1^{(y_1)}(x)\cdots g_b^{(y_b)}(x)$$

Alle Summanden, die weniger als Ableitungen kleiner als n enthalten sind Null bei x=1, wegen Voraussetzung (ii). Da $g_i(1) = 1$ für alle i, gilt:

$$Q_{G1}^{(n)}(1) = \sum_{i=1}^{b} g_i^{(n)}(1)$$

n-te Ableitung für G2:

$$Q_{G2}^{(n)}(x) = (-1)\cdot \sum_{y_1+\ldots+y_b=n} a(y_1,\ldots,y_b)(1-g_1(x))^{(y_1)} \cdot g_2^{(y_2)}(x)\cdots g_b^{(y_b)}(x)$$

Mit Hilfe von (ii) sieht man, dass bei x=1 nur ein Summand ungleich 0 wird:

$$Q_{G2}^{(n)}(1) = g_1^{(n)}(1)$$

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

n-te Ableitung für G3:

$$Q_{G3}^{(n)}(x) = (-1) \cdot \sum_{y_1+\ldots+y_b=n} a(y_1,\ldots,y_b)(1-g_1(x))^{(y_1)} \cdots (1-g_c(x))^{(y_c)} \cdot g_{c+1}^{(y_{c+1})}(x) \cdots g_b^{(y_b)}(x)$$

1. Fall n < c: Einer der ersten c Faktoren ist immer = 0 und es gibt, wegen der Definition von „Strategie“ n blattdisjunkte Strategien unterhalb der Wurzel. Also:
$$Q_{G3}^{(n)}(1) = 0$$

2. Fall n = c: Sei $S_{y1,\ldots,yb}$ ein beliebiger Summand von $Q^{(n)}_{G3}(x)$ bei x=1. Falls es ein l gibt mit l ≤ c und (y_l = 0 oder y_l > 1), folgt $S_{y1,\ldots,yb}$ = 0, weil einer der ersten c Wurzelnachfolger (sei das k) gilt: 1-$g_k(1)$ = 0. Falls es ein l>c gibt mit y_l > 0, folgt ebenfalls sofort $S_{y1,\ldots,yb}$ = 0. Sonst gilt
$$S_{y_1,\ldots,y_b}(1) = (-1)^c \cdot \prod_{i=1}^{c} g_i^{(1)}(1)$$

Vorzeichen: $(-1)(-1)^n \cdot k \cdot \prod_{i=1}^{n} g_i^{(1)}(1),\ \textit{für ein } k \in \mathbb{N}$

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

n-te Ableitung für G3:

3. Fall n > c: Sei $S_{y1,...,yb}$ ein beliebiger Summand von $Q^{(n)}_{G3}(x)$ bei x=1.
   a) Falls es ein l gibt mit l ≤ c und $y_l = 0$, folgt $S_{y1,...,yb} = 0$
   b) Falls es ein l gibt mit l > c und $y_l > 1$, gilt: $\sum^{c}_{i=1} y_i \leq n-1$. $S_{y1,...,yb}$ hat die Form $(1-g_1(x))^{y1} \cdots (1-g_c(x))^{yc} \cdot X$, X eine reele Zahl. Wegen Annahme (ii) gibt es unter n blattdisjunkte Strategien unter der Wurzel von G3. Wegen der Definition von Strategien gibt es die Summe der blattdisjunkten Strategien unter den ersten c Nachfolgern der Wurzel ebenfalls gleich n. Wir können also schließen, dass einer der ersten c Nachfolger mehr als y_i-viele blattdisjunkte Stretegien unter sich hat. Mit Voraussetzung (i) folgt, dass ein $(1-g_i(x))^{(y_i)}$ an der Stelle x=1 zu 0 wird, für ein $i \in \{1,...,c\}$
   c) $\sum^{c}_{i=1} y_i = n$ und $y_i > 1$ für sie ersten c Wurzelnachfolger

$$Q^{(n)}_{G3}(x) = - \sum_{y_1+\ldots+y_b=n} a(y_1,\ldots,y_c,0,\cdots 0)(1-g_1(x))^{(y_1)} \cdots (1-g_c(x))^{(y_c)}$$

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

n-te Ableitung für G3, 3. Fall:

$$Q_{G3}^{(n)}(x) = (-1)^{c+1} \sum_{y_1+\ldots+y_c=n} a(y_1,\ldots,y_c,0,\cdots 0) g_1^{(y_1)}(x)\cdots g_c^{(y_c)}(x)$$

Vorzeichen:
Wegen Voraussetzung (iii) ist das Vorzeichen von

$$sign(Q_{G3}^{(n)}(1)) = (-1)(-1)^c \cdot \prod_{i=1}^{c} sign(g_i^{(y_i)}(1)), \; mit \sum_{i=1}^{c} y_i = n$$

Sei $k_i = y_i - 1$, für i = 1,...,c. Somit ist $(-1)^{k_i}$ das Vorzeichen von $g_i^{(y_i)}(1)$. (vgl. (iii))
Da $\sum_{i=1}^{c} k_i = n-c$, folgt,

$$sign(Q_{G3}^{(n)}(1)) = (-1)(-1)^c \cdot \prod_{i=1}^{c} (-1)^{k_i} = (-1)^{n-1}$$

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Induktion:
Annahme:
(I) Für alle i≤n gilt: Für alle Spielbäume G gibt es i blattdisjunkte Strategien in G, die alle den Wurzelwert von G belegen. <=> $Q_G^{(i-1)}(1) = \ldots = Q_G^{(1)}(1) = 0$
(II) Für alle i $\in$ {1,..,n-1} gilt: das Vorzeichen von $Q_{G'}^{(i)}(1) = (-1)^{i-1}$

Induktionsschritt (n → n+1):
'<=': Es gibt n+1 blattdisjunkte Strategien unter der Wurzel von G. Insbesondere gibt es n blattdisjunkte Strategien und mit (I) und (II) wissen wir, dass die Voraussetzungen (i)-(iii) erfüllt sind. Mit hilfe einer inneren impliziten Induktion sehen wir, dass der Induktionsschritt bereits gemacht ist.
'=>': Sei Q_G ein Qualitätspolynom. Seien $Q_G^{(n)}(1) = \ldots = Q_G^{(1)}(1) = 0$. Offensichtlich gilt auch $Q_G^{(n-1)}(1) = \ldots = Q_G^{(1)}(1) = 0$. Von (I) wissen wir, dass es n blattdisjunkte Strategien in G gibt, und (i)-(iii) sind erfüllt. Mit Hilfe einer impliziten Induktion über die Tiefe von G ist der Induktionsschritt fertig. Man muss allerdings beachten, dass für alle i = 1,...,n gilt $\text{sign}(Q_G^{(i)}(1)) = (-1)^{i-1}$ ist.

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Worst-Case Betrachtung:
Satz:
Sei G ein Spielbaum mit Wert 0 oder 1 an der Wurzel. Sei oBdA die Wurzel ein MAX-Knoten. Dann sind folgende Aussagen äquivalent.

- es gibt c-viele blattdisjunkte Strategien unter der Wurzel von G, die beweisen, dass der wert der Wurzel 0 (bzw. 1) ist.
- man muss mindestens c-viele Blattwerte, bezogen auf die echten Werte, verändern, damit der heuristische Minimaxwert der Wurzel falsch wird.

‚=>' klar, denn mit der Veränderung eines Blattwertes kann man nur eine der blattdisjunkten Strategien „zerstören".
‚<=, Wir bauen per Induktion über die Baumtiefe t eine „Zerstörungsstrategie", die mit c Änderungen die c blattdisjunkten Strategien zerstört.
Start: Sei t = 1. Der Baum besteht nur aus einem Knoten, damit gibt es nur eine Strategie und mit Änderun eines Blattes wird der Wurzelwert verfälscht.

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Worst-Case Betrachtung:

,<=, ...

Annahme: Für alle Tiefe-(t-1)-Bäume gilt, wenn G genau c (c beliebig) blattdisjunkte Strategien enthält, die alle den wert 0 (bzw. 1) der Wurzel beweisen, läßt sich der Wurzelwert mit Hilfe von c Blattwertänderungen verfälschen.

Schritt t-1 → t: Betrachte die Wurzel eine Tiefe-t-Spielbaums.

- Ist der echte Wert 0, so gibt es für alle Nachfolger c blattdisjunkte Strategien, die den Wert 0 belegen. Mindestens 1 Nachfolger besitzt genau c solche Strategien. Auf den wenden wir die obige Annahme an und sind fertig.
- Ist der Wert der Wurzel 1, gibt es d viele Nachfolger, die ebenfalls den Wert 1 haben und die Summe der Anzahl blattdisjunkter Strategien unter diesen Wurzelnachfolgern ist gleich c. Mit Hilfe der Induktionsannahme zerstören wir alle diese Strategien.

Fehleranalyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Fazit:

- Der Wert einer Stellung hängt primär nicht vom 'Bild' der Stellung ab, sondern von der Dynamik, die sich in ihr entwickelt.
- Man kann also 'Chancen' abschätzen.
- Wie die Chancen sind hängt von der Struktur des Spielbaums ab, der unter der aktuell betrachteten Stellung drunterhängt. Wie sieht der Baum aus, wie sind Gewinn, Verlust, Remis darin verteilt (nicht nur in ihrer Anzahl, sondern in ihrere Zusammensetzung)?

Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

Folgende Themen sollen die nächsten Vorlesungsstunden füllen:

- Quantifizierte Linear Programme mit Existenz- und Allquantoren
- Dynamische Programmierung und Optimale Kontrolle
- Stochastische Programmierung
- Komplexität verschiedener stochastischer Problemstellungen
- Quantifizierte Lineare Programme mit Random-Quantoren und das Dynamische Graph-Reliability Problem
- PSPACE-vollständige Einpersonenspiele: ganz anders und doch wieder nicht

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Problemstellung QLP:

$$G: \exists x_1 \in [a_1, b_1] \, \forall y_1 \in [l_1, u_1] \, \exists x_2 \in [a_2, b_2] \ldots \exists x_n \in [a_n, b_n] \, \forall y_n \in [l_n, u_n]$$

$$A' \cdot \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{pmatrix} = \underbrace{A \cdot \begin{pmatrix} x \\ y \end{pmatrix}}_{\text{durch Umordnung}} \leq b, x \geq 0$$

„Entscheide, ob es ein x_1 gibt, so dass für alle y_1 es ein x_2 gibt u.s.w., so dass dass gegebene Ungleichungssystem eine gültige Lösung hat."

Dabei: $A \in \mathbb{Z}^{m \times 2n}$, $x,y \in \mathbb{Q}^n$, $b \in \mathbb{Z}^m$
(Vorsicht: nicht A, $a_1,..,a_n,b_1,...,b_n$, b) durcheinanderbringen. Bedeutung ist aus Zusammenhang entnehmbar.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Problemstellung QLP:

- Das Paar (A,b) wird als Nebenbedingungssystem bezeichnet
- o.B.d.A sollen die Quantifizierer sich strikt abwechseln. Andernfalls könnte man zusätzliche Dummyvariablen einführen, die in (A,b) nicht vorkommen.
- Die Zeichenkette

$$\exists x_1 \in [a_1, b_1] \, \forall y_1 \in [l_1, u_1] \, \forall x_2 \in [a_2, b_2] \ldots \exists x_n \in [l_n, u_n] \, \forall y_n \in [a_n, b_n]$$

  wird Quantifizierer-String Q(x,y) genannt. Die Länge von Q(x,y) wird mit |Q(A,b)| bezeichnet.

- Bedingungen an die ∃–Variablen können in die Matrix A kodiert werden, Bedingungen an die ∀–Variablen nicht.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Problemstellung QLP, Beispiel:

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2 : \begin{pmatrix} 1 & -1 & 1 & 0 \\ -1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix} \leq \begin{pmatrix} -2 \\ 5 \\ 15 \end{pmatrix}, y_1 \in [3,5], y_2 \in [7,10]$$

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2 : \begin{pmatrix} x_1 - x_2 + y_1 \leq -2 \\ -x_1 + x_2 - y_1 \leq 5 \\ x_2 + y_2 \leq 15 \end{pmatrix}$$

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

QLP als Spiel:

- Wir können ein QLP als Spiel zwischen einem Spieler X, der die Existenz-variablen setzt und einem Spieler Y, der die All-Variablen setzt, auffassen.

- Die Züge werde alternierend ausgeführt.

- Ein *Spiel* (x,y) ist eine Abfolge von Zügen
  $x = [x_1, x_2, \ldots, x_n]$
  $y = [y_1, y_2, \ldots, y_n]$

- Eine *Strategie* für X bezeichnen wir mit $\underline{x}$ und eine Strategie für Y mit $\underline{y}$.
  $\underline{x} = [x_1, \quad x_2(x_1,y_1), \ldots \quad , x_n(x_1,y_1,x_2 \ldots y_{n-1})]$
  $\underline{y} = [y_1(x_1), y_2(x_1,y_1,x_2), \ldots, y_n(x_1,y_1,x_2 \ldots y_{n-1},x_n)]$
  Es gibt für jedes QLP entweder eine Gewinnstrategie für X oder eine für Y. (XOR)

Diese Art der Strategie wird meist, z.B. im Gebiet der Kontrolltheorie als *Politik* bezeichnet. Strategien sind streng genommen Politiken, die Bäume sind.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Lösungsalgorithmus, QLP-Decide(A,b)

1: $A_{n+1} = A$; $b_{n+1} = b$;
2: for (i = n down to 2) do
3: (A_i,b_i) = Elim-Univ-Variable($A_{i+1},b_{i+1},y_i,l_i,u_i$);
4: (A_i,b_i) = Elim-Exist-Variable(A_i,b_i,x_i);
5: if (not Check-consistency()) then System is infeasable
6: Prune-Constraints()
7: (A_1,b_1) = Elim-Univ-Variable(A_2,b_2,y_1,l_1,u_1);
8: // The System is now reduced to a one-variable-system, that
9: // can easily be checked on consistency. Let the system have
10: // the form $x_1 \leq v$ and $x_1 \geq w$
11: if ($w \leq x_1 \leq v$ and $w \leq v$) then return System is feasible
12: else return System is infeasible

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Elim-Univ-Variable(A,b,y^n,l^n,u^n)

// beachte: Variablenindizierung hier oben statt unten

1: // Elimination der innersten Variable y_n (All-Variable).
2: for all constraints from i = 1 to m do
3: // assume the worst case for the ∃-Player
4: if ($a_{i,2n}l^n \le a_{i,2n}u^n$) $y_i^n = u^n$
5: else $y_i^n = l^n$
6: build the new A and b by incorporating all constants to into b

Beispiel: $y_1 \in [-1/3,1]$

(A,b) =

$3x_1 + 4y_1 \le 5 \quad \rightarrow y_1 = 1$

$4x_1 - 3y_1 \le 1 \quad \rightarrow y_1 = -1/3$

Damit sieht das neue System wie folgt aus:

$3x_1 + 4 \le 5$ → $3x_1 \le 1$

$4x_1 + 1 \le 1$ → $4x_1 \le 0$

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Korrektheit

Seien

$$L: \exists x_1 \in [a_1,b_1]\, \forall y_1 \in [l_1,u_1]\, \exists x_2 \in [a_2,b_2] \ldots \exists x_n \in [a_n,b_n]\, \forall y_n \in [l_n,u_n] \quad A \cdot \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{pmatrix} \leq b, x \geq 0$$

und

$$R: \exists x_1 \in [a_1,b_1]\, \forall y_1 \in [l_1,u_1]\, \exists x_2 \in [a_2,b_2] \ldots \exists x_n \in [a_n,b_n] \quad A' \cdot \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ y_{n-1} \\ x_n \end{pmatrix} \leq b', x \geq 0$$

zwei Systeme, wobei R aus L durch
Elim-Univ-Variable(A,b,y_n,l_n,u_n) hervorgegangen sei.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Intuition

Sei $\underline{x}$ eine Gewinnstrategie für X, d.h., X kann in jedem Schritt x_i so wählen, dass egal wie Y spielt, X gewinnt und also insbesondere alle Nebenbedingungen eingehalten werden. Für alle Spiele, die der Strategie $\underline{x}$ folgen, muss also X seinen Zug x_n machen, bevor Y seinen letzten Zug y_n macht.
Wenn X dem Y eine Ungleichung zu zerstören ermöglicht, gewinnt Y.
Also muss X für jede einzelne Ungleichung sicherstellen, dass Y auch im schlimmsten Fall nicht gewinnt.

Korrektheit

L => R ist klar. X gibt Y keine Möglichkeit zu gewinnen.
R => L ist ebenso klar. Wenn es in R eine Gewinnstrategie für X gibt, ist diese durch unsere Konstruktion entstanden. Verfolgt man die Umformungen zurück, bekommt Y zwar mehr Freiheiten (eine zusätzliche) Aktion, die rechte Seite b wird aber so stark erweitert, dass die zusätzlichen Aktionen gerade nicht ermöglichen, eine Ungleichung in L zu zerstören.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Elim-Exist-Variable(A,b,x_i)

// entspricht Fourier-Motzkin-Elemination

1: Form the set $L_{\leq}$ of every constraint that can be written in the form $x_i \leq 0$. If $x_i \leq m_j$ is a constraint in (A,b), m_j is added to $L_{\leq}$.

2: Form the set $L_{\geq}$ of every constraint that can be written in the form $x_i \geq 0$. If $x_i \geq n_k$ is a constraint in (A,b), n_k is added to $L_{\leq}$.

3: Form the set $L_{=}$ of every constraint that does not contain x_i

4: $\mathcal{J} = \emptyset$

5: for each constraint $m_j \in L_{\leq}$ do

6: for each constraint $n_k \in L_{\geq}$ do

7: Create new constraint l_{kj} from $n_k \leq m_j$; $\mathcal{J} = \mathcal{J} \cup l_{kj}$

8: Form a new system (A‘,b‘) from the constarints in $\mathcal{J} = \mathcal{J} \cup L_{=}$

9: return (A‘,b‘)

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Korrektheit

Seien

$$L: \exists x_1 \ \forall y_1 \in [l_1, u_1] \, \exists x_2 \ldots \forall y_{n-1} \in [l_{n-1}, u_{n-1}] \, \exists x_n \qquad A \cdot \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{pmatrix} \leq b, x \geq 0$$

und

$$R: \exists x_1 \ \forall y_1 \in [l_1, u_1] \, \exists x_2 \ldots \exists x_{n-1} \ \forall \ y_{n-1} \in [l_{n-1}, u_{n-1}] \qquad A' \cdot \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ y_{n-1} \\ x_n \end{pmatrix} \leq b', x \geq 0$$

zwei Systeme, wobei R aus L durch
Elim-Univ-Variable(A,b,x_n) hervorgegangen sei.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Korrektheit

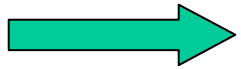
Seien

$\underline{x}_L = [x_1, x_2(x_1,y_1),...,x_n(x_1,y_1,x_2 ...y_{n-1})]$ eine Strategie für L und
$\underline{x}_R = [x_1, x_2(x_1,y_1),..., x_{n-1}(x_1,y_1,x_2 ...y_{n-2})]$ eine Strategie für R.
$\underline{x}_R$ entsteht aus $\underline{x}_L$ durch weglassen der letzten Komponente.

Annahme: $\underline{x}_L$ ist eine Gewinnstrategie für L, aber $\underline{x}_R$ ist keine Gewinnstrategie für R.

Dann gibt es eine Gewinnstrategie $\underline{y}_R(\underline{x}_R)$, $\underline{y}_R$ (möglicherweise) abhängig von $\underline{x}_R$. Betrachte nun das Spiel (x_R,y_R), welches entsteht, indem X und Y die Züge ihrer Strategien $\underline{y}_R(\underline{x}_R)$und $\underline{x}_R$ anwenden.

Da $\underline{y}_R(\underline{x}_R)$ eine Gewinnstrategie ist, gibt es eine Nebenbedingung in $A'\cdot(x_R,y_R) \leq b'$, die verletzt wird. Sei dies in der i-te Zeile und sei $p'_i := A'_{i*} \cdot (x_R,y_R) > b'_i$. Schreibe p'_i auch als $A'^{x}_{i*} \cdot x_R + A'^{y}_{i*} \cdot y_R \leq b'$, wobei A'^{x}_{i*} und A'^{y}_{i} entsprechende Teil matrizen von A' sind.

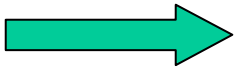


TECHNISCHE UNIVERSITÄT DARMSTADT

Korrektheit

1. Fall: p'_i gibt es auch als p_i in L. Dann kommt Variable x_n offenbar in p_i nicht vor. D.h., die gleiche Partie wird von X auch in L verloren.
2. Fall: p'_i wurde in L zusammengesetzt aus $l_i : m_i \leq x_n$ und $l_j : x_n \leq n_j$, um mit *Elim-Exist-Variable* $m_i \leq n_j$ zu erhalten. Wir schreiben
   l_i als $(A^x_{i,\{1,...,2n-2\}} \cdot x_R - A^y_{i,\{1,...,2n-1\}} \cdot y_R - b_i) / A_{i,n} \leq x_n$ und
   l_j als $x_n \leq (b_j - A^x_{i,\{1,...,2n-2\}} \cdot x_R - A^y_{i,\{1,...,2n-1\}} \cdot y_R) / A_{i,n}$.
   Da $\underline{y}_R(\underline{x}_R)$ eine Gewinnstrategie in R ist, wissen wir aber, dass
   $(A^x_{i,\{1,...,2n-2\}} \cdot x_R - A^y_{i,\{1,...,2n-1\}} \cdot y_R - b_i) / A_{i,n} > (b_j - A^x_{i,\{1,...,2n-2\}} \cdot x_R - A^y_{i,\{1,...,2n-1\}} \cdot y_R) / A_{i,n}$
   D.h., es kann kein x_n geben, welches zwischen den beiden Termen liegt.
   $y_R(\underline{x}_R)$ wäre also auch für L eine Gewinnstrategie für Spieler Y.

Es gilt also L => R. Wir müssen noch die Rückrichtung zeigen.



QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Korrektheit

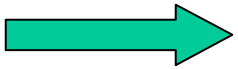
Seien $S_1 = \{m_1 \leq x_n, m_2 \leq x_n, ..., m_p \leq x_n\}$ und $S_2 = \{x_n \leq n_1, x_n \leq n_2, ..., x_n \leq n_q\}$ die Nebenbedingungen in L, die als $x_n \geq ()$ bzw. als $x_n \leq ()$ geschrieben werden können. $\underline{x}_R$ sei eine Gewinnstrategie für Spieler X in R.

Betrachte nun x_n mit: $\max^{p}_{i=1} m_i \leq x_n \leq \min^{q}_{j=1} n_j$. Die Behauptung ist, dass $\underline{x}_L = [\underline{x}_R, x_n(x_L, x_R)]$ eine Gewinnstrategie für L ist.

Annahme: Das Gegenteil ist der Fall, $\underline{x}_L$ ist also keine Gewinnstrategie für L.

Dann gibt es eine Gewinnstrategie $\underline{y}_L(\underline{x}_L)$ für Y. Seien y_L und x_L wieder die Züge des entstehenden Spiels. $\underline{x}_R$, $\underline{y}_R$, x_R und y_R analog.

Da $\underline{y}_L(\underline{x}_L)$ eine Gewinnstrategie von Y in L ist, gibt es eine Nebenbedingung p_i, die im System $A \cdot (x_L, y_L) \leq b$ verletzt wird.



QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Korrektheit

1. Fall: p_i taucht in gleicher Form is R auf. Offenbar war x_n dann nicht Teil von p_i. Somit gewinnt Y mit $\underline{y}_R(x_R)$ auch in R.
2. p_i sei von der Form $l_i : m_i \leq x_n \in S_1$. Betrachte nun eine beliebiges Element $l_j : n_j \leq x_n \in S_2$. Dann ist $m_i \leq n_j$ Teil von R.

Wir schreiben l_i als $(A^x_{i,\{1,\ldots,2n-2\}} \cdot x_R - A^y_{i,\{1,\ldots,2n-1\}} \cdot y_R - b_i) / A_{i,n} \leq x_n$ und l_j als $x_n \leq (b_j - A^x_{i,\{1,\ldots,2n-2\}} \cdot x_R - A^y_{i,\{1,\ldots,2n-1\}} \cdot y_R) / A_{i,n}$. Weil $\underline{x}_R$ eine Strategie für R ist, wissen wir, dass

$$(A^x_{i,\{1,\ldots,2n-2\}} \cdot x_R - A^y_{i,\{1,\ldots,2n-1\}} \cdot y_R - b_i) / A_{i,n} \leq (b_j - A^x_{i,\{1,\ldots,2n-2\}} \cdot x_R - A^y_{i,\{1,\ldots,2n-1\}} \cdot y_R) / A_{i,n}.$$

Da aber $y_L(x_L)$ eine Gewinnstrategie für Y in L ist, gilt auch:

$(A^x_{i,\{1,\ldots,2n-2\}} \cdot x_R - A^y_{i,\{1,\ldots,2n-1\}} \cdot y_R - b_i) / A_{i,n} > x_n$, für alle $x_n \in \mathbb{Q}$.

D.h., X konnte kein $x_n \in \mathbb{Q}$ finden, so dass

$(A^x_{i,\{1,\ldots,2n-2\}} \cdot x_R - A^y_{i,\{1,\ldots,2n-1\}} \cdot y_R - b_i) / A_{i,n} \leq x_n$ und

$x_n \leq (b_j - A^x_{i,\{1,\ldots,2n-2\}} \cdot x_R - A^y_{i,\{1,\ldots,2n-1\}} \cdot y_R) / A_{i,n}$.

$\underline{y}_L(\underline{x}_L)$ kann also keine Gewinnstrategie für L sein, da anderenfalls $\underline{y}_R(\underline{x}_R)$ eine Gewinnstrategie von Y in R wäre. (analog für p_i von der Form $l_j : x_n \leq n_j \in S_2$.)

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz:

Wir haben gezeigt:

1. Wenn die innerste Variable des Quantorblocks eine All-Variable ist, lässt sie sich in dem Sinn eleminieren, dass ein neues System entsteht, welches eine Variable weniger besitzt, und genau dann eine Gewinnstrategie für Spieler X enthält, wenn auch das alte System eine Gewinnstrategie für Spieler X enthält.
2. Wenn die innerste Varaible eine Existenzvariable ist, lässt sich die Fourier-Motzkin Elemination zur Elemination der innersten Variable anwenden.
3. Diese beiden Schritte lassen sich mehrfach anwenden.

Es folgt: Der Algorithmus QLP-Decide(A,b) entscheidet Problem G korrekt.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz:

Bezeichne $y_i \in \{l_i, u_i\}$, dass y_i aus der Menge der Zahlen $[l_i, u_i] \cap \mathbb{Z}$ stammen soll. Dann gilt, sofern alle Existenzvariablen kontinuierlich sind:

$$L : \exists x_1 \ \forall y_1 \in [l_1, u_1] \, \exists x_2 \ldots \exists x_n \forall y_n \in [l_n, u_n] \quad A \cdot \begin{pmatrix} x \\ y \end{pmatrix} \leq b, x \geq 0$$

$$R : \exists x_1 \ \forall y_1 \in \{l_1, u_1\} \, \exists x_2 \ldots \exists x_n \forall y_n \in \{l_n, u_n\} \quad A \cdot \begin{pmatrix} x \\ y \end{pmatrix} \leq b, x \geq 0$$

L $\Leftrightarrow$ R

Korrektheit: Für die Elemination des innersten Quantors, wenn All-Quantor, sind nur die beiden Grenzen l_i und u_i benutzt worden. Nach einer Fourier-Motzkin-Elemination der nächsten, dann existenzquantifizierten, Variable, ist wiederum eine All-Quantifizierte Variable die Innerste.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Komplexität:

QIPs der Form

$$G : \exists x_1 \in \{a_1, b_1\}\ \forall y_1 \in \{l_1, u_1\} \ldots \exists x_n \in \{a_n, b_n\} \forall y_n \in \{l_n, u_n\} \quad A \cdot \begin{pmatrix} x \\ y \end{pmatrix} \leq b, x \geq 0$$

sind PSPACE volständig.

a) PSPACE-schwer:
   Ausgehend von einem QSAT-Problem bilde für jede Klausel
   $k=(l_{k1} \vee \ldots \vee l_{kr})$ bilde eine Nebenbedingung der Form $L_{k1} + \ldots + L_{kr} \geq 1$, wobei hier
   - $L_{ki} = x_j$, falls $l_{ki} = x_j$ nicht-negiert und Existenzquantifiziert,
   - $L_{ki} = 1-x_j$, falls $l_{ki} = x_j$ negiert und Existenzquantifiziert,
   - $L_{ki} = y_j$, falls $l_{ki} = y_j$ nicht-negiert und Allquantifiziert,
   - $L_{ki} = 1-y_j$, falls $l_{ki} = y_j$ negiert und Allquantifiziert.

   Offenbar ist die SAT-Formel genau dann erfüllt, wenn das Nebenbedingungssystem nicht verletzt wird. Dies setzt sich über die Quantoren fort.
b) in PSPACE: Nutze den Alphabeta-Algorithmus

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Komplexität von QLPs ist unbekannt. Jedoch:

Ein E-QLP ist ein QLP, bei demerst die existenzquantifizierten Variablen kommen:

$$G : \exists x_1 \ldots \exists x_n \forall y_1 \in \{l_1, u_1\} \ldots \forall y_n \in \{l_n, u_n\} \quad A \cdot \begin{pmatrix} x \\ y \end{pmatrix} \leq b, x \geq 0$$

E-QLPs sind in P.

Beweis: Wir eleminieren zuerst alle Allvariablen. Übrig bleibt ein LP.

Damit ist auch klar: E-QIPs sind NP-vollständig.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Komplexität von QLPs ist unbekannt. Jedoch:

Ein F-QLP ist ein QLP, bei dem erst die allquantifizierten Variablen kommen:

$$G: \forall y_1 \in \{l_1, u_1\} \ldots \forall y_n \in \{l_n, u_n\} \exists x_1 \ldots \exists x_n \quad A \cdot \begin{pmatrix} x \\ y \end{pmatrix} \leq b, x \geq 0$$

F-QLPs sind coNP-vollständig.

Beweis: Ein Problem P ist in coNP, wenn sein negiertes Problem in NP ist.
Wir bilden deshalb

$$\overline{G}: \exists y_1 \in \{l_1, u_1\} \ldots \exists y_n \in \{l_n, u_n\} : not \left(\exists x_1 \ldots \exists x_n \quad A \cdot \begin{pmatrix} x \\ y \end{pmatrix} \leq b, x \geq 0 \right)$$

Algorithmus: Rate $y_1,\ldots,y_n$ und prüfe, of das Polyeder

$$\exists x_1 \ldots \exists x_n \quad A \cdot x \leq b, x \geq 0$$

leer ist.

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

Komplexität von QLPs ist unbekannt. Jedoch:

F-QLPs sind coNP-schwer.

Beweis: Nehme eine SAT-Formel und negiere sie:

$$\exists x_1 \ldots \exists x_n \;\; (l_{1,1} \vee \ldots \vee l_{1,k1}) \wedge (l_{2,1} \vee \ldots \vee l_{2,k2}) \wedge \ldots \wedge (l_{m,1} \vee \ldots \vee l_{m,km})$$

$$\Leftrightarrow$$

$$not\left(\forall y_1 \ldots \forall y_n \;\; (\bar{l}_{1,1} \wedge \ldots \wedge \bar{l}_{1,k1}) \vee (\bar{l}_{2,1} \wedge \ldots \wedge \bar{l}_{2,k2}) \vee \ldots \vee (\bar{l}_{m,1} \wedge \ldots \wedge \bar{l}_{m,km})\right)$$

Hieraus bilden wir folgendes QIP G:

$$\forall y_1 \ldots \forall y_n \; \exists x_1 \ldots \exists x_m :$$

$$\bar{l}_{1,j} \geq x_1, \forall j \in \{1,\ldots,k_1\}; \; wobei \;\; l_{1,j} = y_l \; oder \; l_{1,j} = 1 - y_l, \; mit \;\; passendem \;\; l$$

$$\bar{l}_{2,j} \geq x_2, \forall j \in \{1,\ldots,k_2\}; \; wobei \;\; l_{2,j} = y_l \; oder \; l_{2,j} = 1 - y_l, \; mit \;\; passendem \;\; l$$

$$\ldots$$

$$\bar{l}_{m,j} \geq x_m, \forall j \in \{1,\ldots,k_m\}; \; wobei \;\; l_{m,j} = y_l \; oder \; l_{m,j} = 1 - y_l, \; mit \;\; passendem \;\; l$$

$$x_1 + \ldots + x_m \geq 1$$

QLPs und QIPs

TECHNISCHE UNIVERSITÄT DARMSTADT

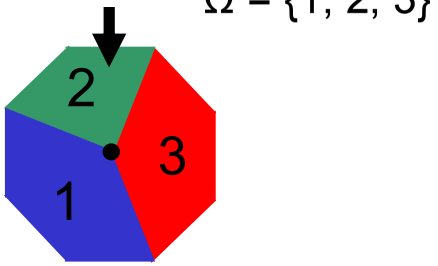
Komplexität von QLPs ist unbekannt. Jedoch:

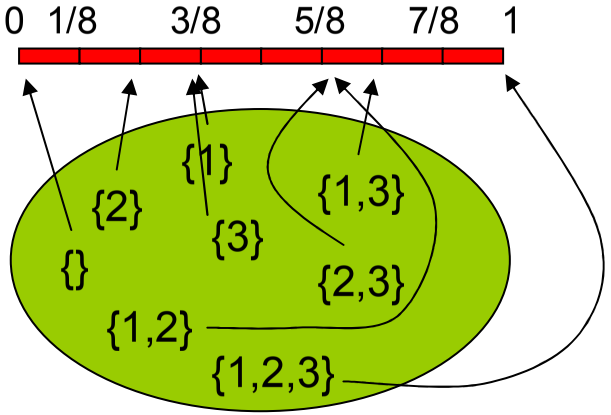
F-QLPs sind coNP-schwer.
Beweis (Forts.):
Sei das QLP G‘ die LP-Relaxierung von QIP.
Behauptung: G hat eine Lösung, genau dann, wenn G‘ eine Lösung besitzt.
In beiden Fällen sollen die Allvariablen diskret sein, was in QIPs gegeben ist, und was nach vorigen Recherchen nicht die Lösbarkeit der QLP-Systeme ändert.
G => G‘: klar. Wenn es gegen diskrete y-Variablen eine ganzzahlige Gewinnstrategie für X gibt, gibt es auch eine kontinuierlche Lösung.
G‘ => G:
Es gibt eine nicht-ganzzahlige Strategie $\underline{x}$ für G‘.
$\Rightarrow$ mindestens eine x-Variable x_i wird in der Strategie $\underline{x}$ auf einen Wert > 0 gesetzt.
$\Rightarrow$ auf der „linken Seite“ der Ungleichungen, die x_i beschränken, sind alle Terme > 0.
$\Rightarrow$ alle diese Terme sind = 1, denn sie sind von der Form $y \geq x_i$ oder $1-y \geq x_i$.
$\Rightarrow$ man kann x_i auf 1 setzen.

TECHNISCHE UNIVERSITÄT DARMSTADT

Voraussetzungen

Ein **Wahrscheinlichkeitsraum** ist ein Tripel (Ω,Σ,*P*). Ω bezeichnet die Menge aller Elementarereignissee, Σ eine Sigma-Algebra und *P* ein Wahrscheinlichkeitsmaß auf Σ. Aus P ergeben sich die Wahrscheinlichkeitsverteilungen für die Ereignisse aus Σ.





Beispiel: Ein Glücksrad mit Ergebnismenge Ω, Ereignisraum Σ (hier die Potenzmenge von Ω) und Wahrscheinlichkeitsmaß *P*.

Stochastische Problemstellungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Im folgenden sind alle Wahrscheinlichkeitsverteilungen diskret und endlich, also sehr einfach. Kombinierte Gesamtverteilungen sind zum Teil nur implizit gegeben.

Es sollen folgende Problemstellungen betrachtet werden:

- Stochastic Satisfibility (SSAT)
- Dynamic Graph Reliability (DGR)
- Optimal Control mit Dynamic Programming
- Multi-Stage Stochastic Programming

SSAT [Papadimitriou 1985]

TECHNISCHE UNIVERSITÄT DARMSTADT

Problem SSAT: Gegeben sei eine boolsche Formel C in CNF, mit den Variablen $x_1,\ldots,x_n$ (n gerade) und eine rationale Zahl $b \in [0,1]$.

Ist die Wahrscheinlichkeit für

$\exists\, x_1\, \Re\, x_2\, \exists\, x_3 \ldots \Re\, x_n : C(x_1, x_2, \ldots, x_n) = \text{TRUE}$

größer oder gleich 1/2?

$\Re$ ist ein stochastischer Quantor, der so quantifizierte Variablen mit Wahrscheinlichkeit ½ TRUE bzw. FALSE setzt.

Satz: SSAT ist PSPACE complete

SSAT

TECHNISCHE UNIVERSITÄT DARMSTADT

Beweis: Wir nehmen eine QSAT-Instanz, interpretieren deren Allquantifizierte Variablen als Random-Variablen, die mit Wahrscheoinlichkeit ½ auf true oder false gesetzt werden und fügen eine Variable x_0 und eine Klausel (x_0) hinzu:

Ist die Wahrscheinlichkeit für

$\Re\ x_0 \exists\ x_1 \Re\ x_2 \exists\ x_3 \ldots \Re\ x_n : (x_0) \wedge C(x_1, x_2, \ldots, x_n) = \text{TRUE}$

größer oder gleich 1/2?

Mit Wahrscheinlichkeit ½ ist die CNF wegen x_0 false. Wenn also die SSAT-Antwort auf unsere Konstruktion “ja” ist, kann das nur daran liegen, dass es eine Gewinnstrategie für den Existenzspieler im ursprünglichen QSAT Problem gibt.

Dynamic Graph Reliability (DGR)

TECHNISCHE UNIVERSITÄT DARMSTADT

Problem DGR: Gegeben sei ein gerichteter Graph ohne Kreise, also ein DAG (Directed Acyclic Graph), G = (V,E). Zwei seiner Knoten s (source) und t (sink) seien speziell ausgezeichnet.

Frage: Wie ist die Startegie, die die Wahrscheinlichkeit das Ziel t zu erreichen maximiert, wenn

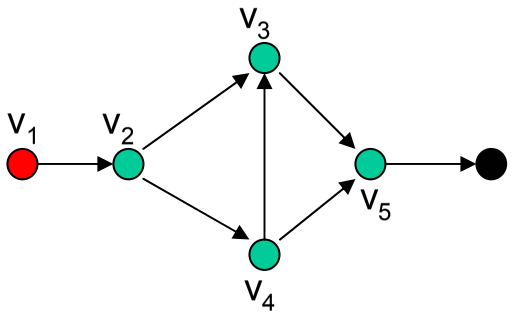
- man bei Knoten s startet

- p(e,v) die Wahrscheinlichkeit angibt, dass Kante e ausfällt, wenn wir Knoten v betreten, und der Graph somit während unserer Wanderung abhängig von unseren Entscheidungen und vom Zufall zerfällt?

Dynamic Graph Reliability (DGR)

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel A:

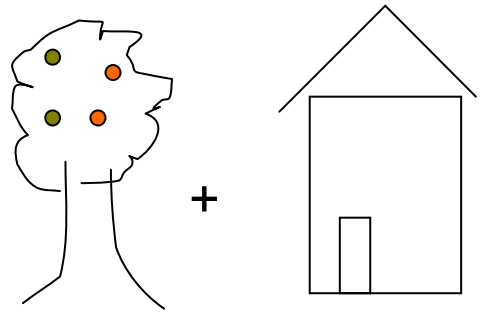


$p((v_3, v_5), v_3) = 0.5$ → 0.5
$p((v_4, v_5), v_3) = 0.9$ → 0.1
$p((v_3, v_5), v_4) = 0.5$ → 0.5
$p((v_4, v_5), v_4) = 0.6$ → 0.4

W., dass man ans Ziel kommt, wenn man über v_4 geht:

0.4 + 0.5 * 0.5 = 0.65

Beispiel B:



Frage: Baum fällen? Dann kann er nicht mehr umfallen. Bringt aber später auch keine Ernte.

-> Entscheidung beeinflusst Wahrscheinlichkeiten, deren Auswirkungen erst spät spürbar werden.

Dynamic Graph Reliability (DGR)

TECHNISCHE UNIVERSITÄT DARMSTADT

Hilfsproblem SSAT': Gegeben ist eine boolsche 3-SAT Formel mit alternierendem $\exists$ - $\Re$' - Quantor-Präfix, sowie eine rationale Zahl $b \in [0,1]$.

In dieser Verion:

- Die Wahrscheinlichkeit, dass eine der Belegungen true oder false für eine Randomvariable *unverfügbar* wird ist ½.
- Eine Existenzstrategie wählt eine Belegung für die Existenzvariable x_1, und dann wird bestimmt, welche Belegungen für x_2 verfügbar sind. Mit Wahrscheinlichkeit ¼ ist keine verfügbar, mit Wahrscheinlichkeit ¼ sind beide verfügbar, mit Wahrscheinlichkeit ¼ ist nur true verfügbar und mit Wahrscheinlichkeit ¼ nur false.
- Dann wählt die Strategie eine der verfügbaren Belegungen für x_2 etc. Wenn an einer $\Re$'-quantifizierten Variable keine Belegung verfügbar ist, hat der Existenzspieler sein Spiel verloren.

Dynamic Graph Reliability (DGR)

TECHNISCHE UNIVERSITÄT DARMSTADT

Hilfsproblem SSAT':

- **Die Frage:**

  Ist die Wahrscheinlichkeit für

  $$\exists\, x_1\ \Re'\ x_2\ \exists\, x_3 \ldots \Re'\ x_n : C(x_1, x_2, \ldots, x_n) = \text{TRUE}$$

  größer oder gleich b?

  ist PSPACE-vollständig.

Härte: Wir nehmen eine QSAT Formel und interpretieren die n vielen Allquantoren als $\Re'$-Quantoren; b setzen wir auf $(¾)^n$. Mit Wahrscheinlichkeit $(1- ¾)^n$ geht ein Spiel für den Existenzspieler dadurch verloren, dass die SAT-Formel C "nicht erreicht" wird. Falls es nun eine Gewinnstrategie für den Existenzspieler gibt, die immer gewinnt, ist die Wahrscheinlichkeit, das C erfüllt wird gerade gleich $(¾)^n$. Sonst ist sie kleiner.

Dynamic Graph Reliability (DGR)

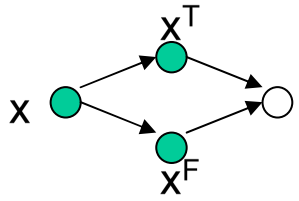
TECHNISCHE UNIVERSITÄT DARMSTADT

DGR ist PSPACE-schwer

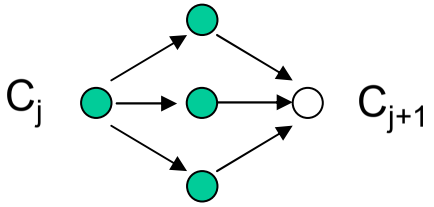
Beweis:

Sei (X,C,b) eine SSAT' Instanz mit $X=\{x_1,..,x_n\}$ der Variablenmenge, den Klauseln $C=\{C_1,..,C_m\}$ und der Schranke b.

- Für jede Variable x habe G 3 Knoten: x, x^T, x^F und die Kanten (x, x^T), (x, x^F)



- von x^T und x^F führen Kanten zum nächsten Variablenknoten.
- Für jede Klausel C_j haben wir 4 Knoten C_j, C_j^1 , C_j^2 , C_j^3 . C_j^1 , C_j^2 , C_j^3 sind jeweils mit C_j und C_{j+1} verbunden.



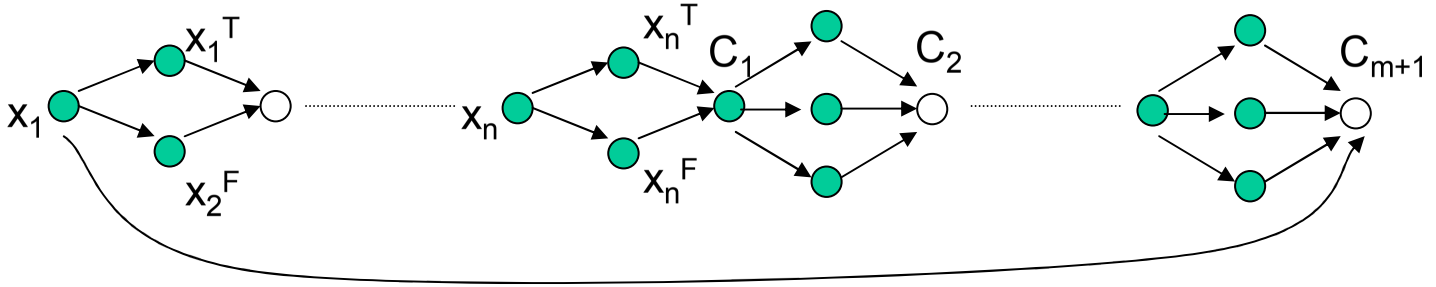
Dynamic Graph Reliability (DGR)

TECHNISCHE UNIVERSITÄT DARMSTADT

DGR ist PSPACE-schwer

Beweis Forts:

- Zusätzlich gibt es Kanten von x_n^T und x_n^F nach C_1 und eine Kante von x_1 nach C_{m+1}.



Definiere nun die Wahrscheinlichkeiten p(e,v):

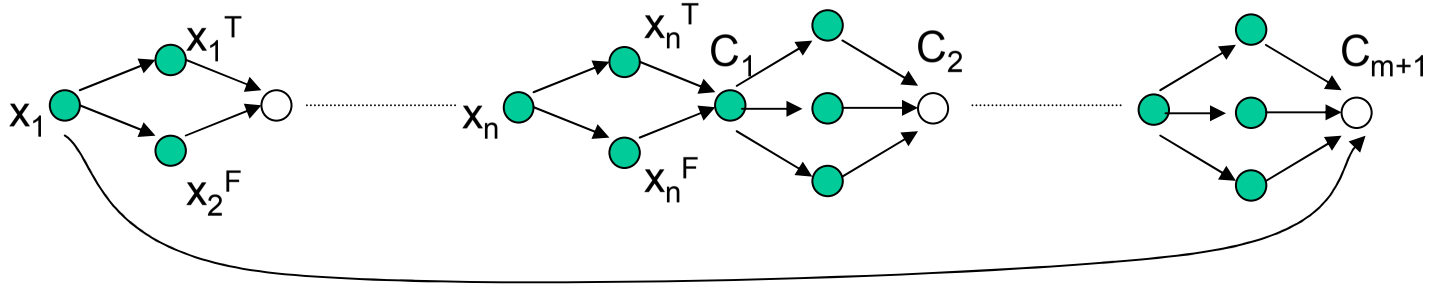
- für jede stochastische Variable setzen wir $p((x,x^T),x) = p((x,x^F),x) = 0.5$.
- für alle Variablen x setzen wir: Wenn $e=(C_j,C_j^{i\in\{1,2,3\}})$ eine Kante ist, die zum Literal x im Klauselpart gehört: $p(e,x^F) = 1$. Wenn $e=(C_j,C_j^{i\in\{1,2,3\}})$ eine Kante ist, die zum Literal *not* x im Klauselpart gehört: $p(e,x^T) = 1$.
- $p((x_1,C_{m+1}),x_1) := 0.5 + b - 2^{-3n}$
- p(e,v) = 0 für alle anderen Paare (e,v).

Dynamic Graph Reliability (DGR)

TECHNISCHE UNIVERSITÄT DARMSTADT

DGR ist PSPACE-schwer

Beweis Forts:



Beste Strategie:
Wenn (x_1,C_{m+1}) existiert gehen wir dort entlang. Wir erreichen also das Ziel mit Wahrscheinlichkeit 1- $p((x_1,C_{m+1}),x_1)$ plus der Wahrscheinlichkeit, dass die Formel C erfüllt wird. Die Gesamtwahrscheinlichkeit, das Ziel zu erreichen ist über 0.5, g.d.w. die Wahrscheinlichkeit, dass die Formel erfüllbar wird, größer oder gleich b ist.

Optimale Kontrolle

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorab: Einordungen weiterer Begriffe

diskrete Zustände	vs.	kontinuierliche Zustände
endliche Zustandsmenge	vs.	unendliche Zustandsmenge
diskrete Zeit	vs.	kontinuierliche Zeit
endlicher Horizont	vs.	unendlicher Horizont
open-loop	vs.	closed-loop Optimierung

Optimale Kontrolle: Einführung

nach Dimitri P. Bertsekas

TECHNISCHE UNIVERSITÄT DARMSTADT

Basissystem

- zeitdiskretes dynamisches System
- Kostenfunktion ist additiv über die Zeit
- das System besitzt Zustandsvariablen, die sich im Lauf der Zeit verändern, unter dem Einfluss eigener Entscheidungen und des Zufalls

- das System hat die Form

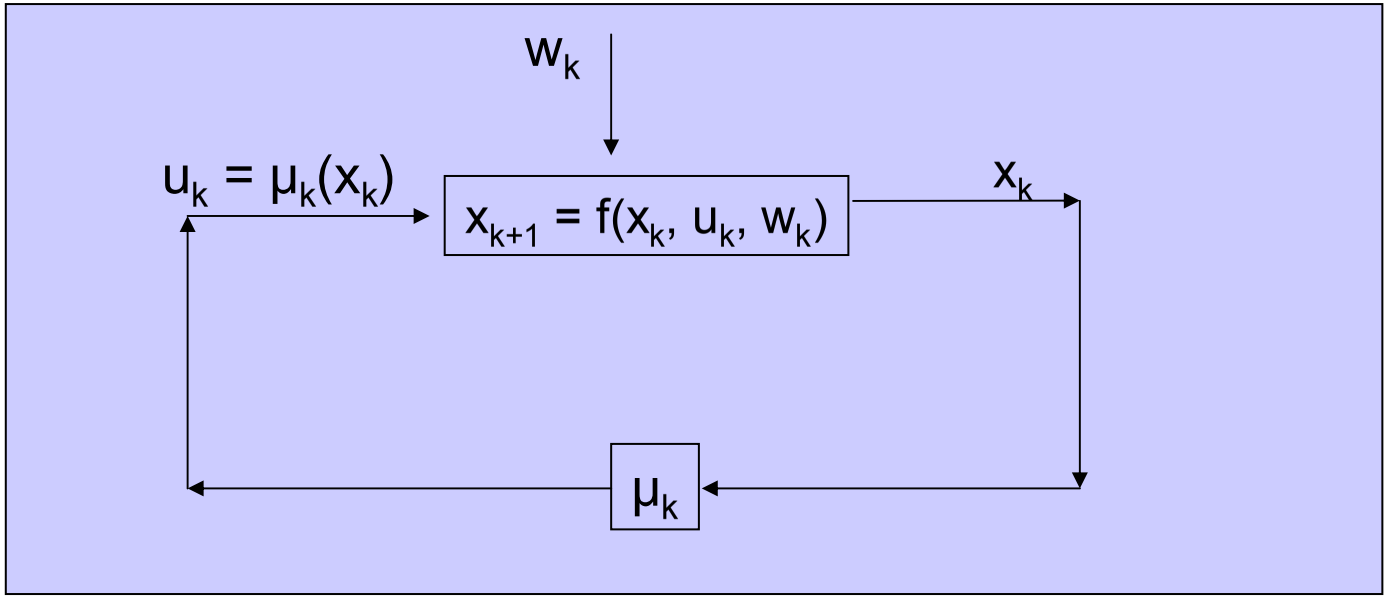
$$x_{k+1} = f_k(x_k, u_k, w_k) \quad \text{mit} \quad k = 0,\ldots,N-1, \text{ wobei}$$

  - k indiziert die diskrete Zeit
  - x_k ist der Zustand des Systems (mit Zustandsmenge S_k) und summiert vergangene Informationen, die für die Zukunft relevant sind
  - $u_k \in U_k(x_k) \subseteq C_k$ sind so genannte Kontrollvariablen. Die Menge $U_k(x_k)$ der möglichen Aktionen („Züge“) hängt vom aktuellen Zustand ab.
  - $w_k \in D_k$ ist ein Zufallsparameter, eine „Störung“. w_k gehorcht einer Wahrscheinlichkeitsverteilung $P_k(\cdot \mid x_k, u_k)$, die von x_k und u_k abhängen kann, jedoch nicht von früheren Realisierungen $w_{k-1}, \ldots, w_0$.
  - N ist der Horizont, die Anzahl der Zeitschritte, die wir untersuchen

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Basissystem



Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Basissystem I

- das System hat die Form

  ...

  - f_k ist eine Funktion, die das System und den Mechanismus, mit dem ein Zustand in einen nächsten überführt wird, beschreibt.
  - die dazugehörige Kostenfunktion ist additiv, d.h. die Kosten $g_k(x_k,u_k,w_k)$ werden akkumuliert. Zusätzlich gibt es Abschlußkosten $g_N(x_N)$ zum Zeitpunkt N. Die Gesamtkosten werden beschrieben als

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)$$

  - Eine **Politik** ist eine Funktionenfolge

    $\pi = \{\mu_0,\ldots, \mu_{N-1}\}$

    wobei μ_k Zustände x_k auf Kontrollvariablen $u_k = \mu_k(x_k)$ so abbildet, dass für alle $x_k \in S_k$ gilt: $\mu_k(x_k) \in U_k(x_k)$.

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Basissystem

- Wegen des Zufallseinflusses w_k sind die Kosten im Allgemeinen eine Zufallsvariable und können deshalb nicht sinnvoll optimiert werden. Wir betrachten deshalb das Problem der erwarteten Kosten für eine bei x_0 startende Politik π:

$$J_\pi(x_0) = E\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}$$

  wobei die Erwartung sich auf die (möglicherweise implizite) Gesamtverteilung der Zufallsvariablen bezieht.
- Eine optimale Politik π* ist eine, die die optimalen Kosten $J^*(x_0)$ minimiert:

$$J^*(x_0) = J_{\pi^*}(x_0) = \min_{\pi \in \Pi} \quad J_\pi(x_0)$$

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel I, Optimierung einer Schach-Turnier-Strategie

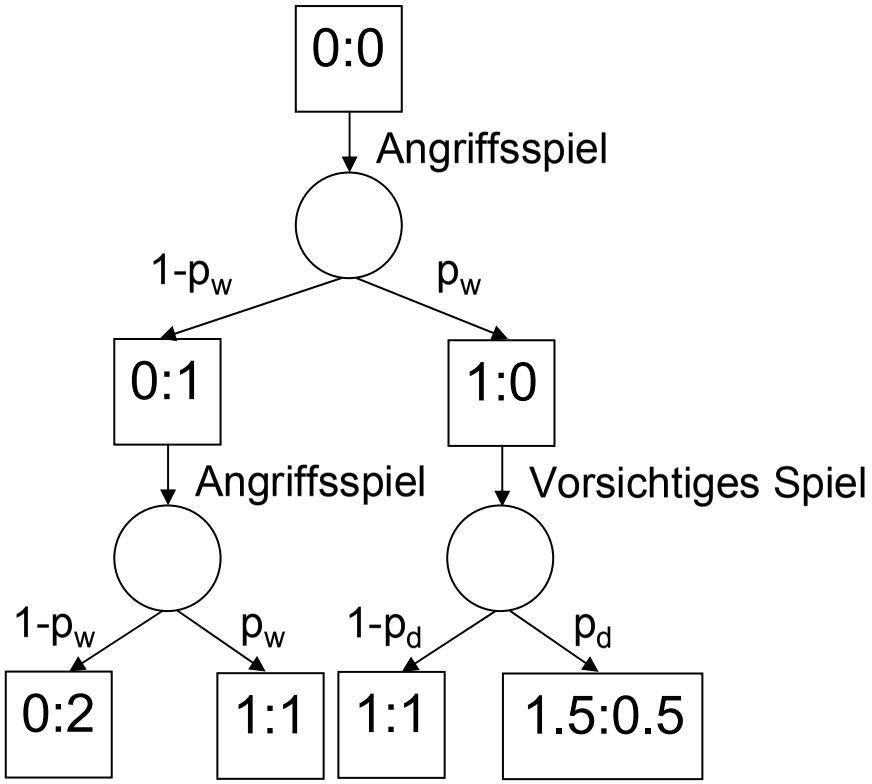
- Ein Spieler muss ein Match mit 2 Partien spielen. Jede Partie hat einen der 3 Ausgänge „win“, „loss“, and „draw“.
- Beim Spielstand von 1:1 wird so lange gespielt, bis einer eine Partie gewinnt (sudden-death)
- Unser Spieler habe 2 Spielmodi:
  - „Vorsichtiges Spiel“. Hier bekommt er ein Remis (draw) mit Wahrscheinlichkeit $p_d > 0$ und verliert mit Wahrscheinlichkeit $1-p_d$. Kein Gewinn.
  - „Angriffsspiel“. Gewinnwahrscheinlichkeit p_w und Verlustwahrscheinlichkeit $1-p_w$. Nie Remis.
- Sobald der Sudden-death beginnt, sollte der Spieler Angriffsspiel zeigen.

Frage: Wie sollte er sich in den ersten beiden Runden verhalten?
Frage: Wie groß sind seine Gewinnchancen, wenn z.B. $p_w = 0.45$ und $p_d = 0.9$?

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel I, Optimierung einer Schach-Turnier-Strategie



Nach 2 Spielen:

- W. für Matchwin ist $p_w \cdot p_d$
- W. für Matchverlust ist $(1-p_w)^2$
- W. für Gleichstand ist $p_w(1-p_d)+(1-p_w)p_w$. Danach ist die Gewinnwahrscheinlichkeit p_w.
- Somit: Gewinnw. dieser Strategie:
  $p_w p_d + p_w(p_w(1-p_d)+(1-p_w)p_w)$

- Für $p_w = 0.45$ und $p_d = 0.9$ ergibt sich eine Gesamtgewinnwahrscheinlichkeit von ca. 0.53.

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel I, Optimierung einer Schach-Turnier-Strategie

´Betrachte nun alle open-loop-Politiken (sind nur 4):

Bezeichne W die Wahrscheinlichkeit, das Match zu gewinnen.

1. Spiele vorsichtig in beiden Spielen. → $W = p_d^2 p_w$.
2. Zeige beide Male Angriffsspiel → $W = p_w^2 + 2\, p_w^2(1-p_w) = p_w^2(3 - 2p_w^2)$
3. Spiele erst auf Angriff, dann vorsichtig → $W = p_w\, p_d + p_w^2(1-p_d)$
4. Spiele erst vorsichtig, dann auf Angriff → $W = p_w\, p_d + p_w^2(1-p_d)$

mit ein bisschen „Herumrechnen“ ergibt sich:

$$W = p_w^2 + p_w(1-p_w)\, \max(2p_w\, p_d)$$

Für $p_w = 0.45$ und $p_d = 0.9$ ergibt sich eine Gesamtgewinnwahrscheinlichkeit von ca. 0.425.

Die Differenz 0.53 – 0.425 nennt man den “Value of Information”.

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Dynamic Programming (DP)

Grundlage für DP ist das **Optimalitätsprinzip**

Sei $\pi^* = \{\mu^*_0, \ldots, \mu^*_{N-1}\}$ eine optimale Strategie. Nehmen wir an, aufgrund der ersten i Schritte wird Zustand x_i erreicht. Betrachten wir nun das Teilproblem von Zeitpunkt i bis N. Die restlichen Kosten bis zum Zeitpunkt N sind:

$$E\left\{ g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}$$

Dann ist die abgeschnittene Politik $\{\mu^*_i, \ldots, \mu^*_{N-1}\}$ optimal für das Restproblem.

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Der DP-Algorithmus (Dynamic Programming Algorithmus)

Für jeden Startzustand x_0 sind die optimalen Kosten $J^*(x_0)$ gleich $J_0(x_0)$, welches durch den letzten Schritt des folgenden Algorithmus berechnet wird, der sich rückwärts von Periode N-1 zu Periode 0 in der Zeit bewegt:

$$J_N(x_N) = g_N(x_N),$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\},$$

$$k = 0,1,\ldots,N-1$$

Die Erwartung bezieht sich hier auf Wahrscheinlichkeitsverteilung von w_k, die von x_k und u_k abhängt. Wenn $u^*_k = \mu^*_k$ die rechte Seite von (*) für alle x_k und k minimiert, ist die Politik $\pi^* = \{\mu^*_0,\ldots, \mu^*_{N-1}\}$ eine optimale Strategie.

TECHNISCHE UNIVERSITÄT DARMSTADT

Der DP-Algorithmus (Dynamic Programming Algorithmus)

Unter der Annahme, dass alle involvierten Wahrscheinlichkeitsverteilungen endlich und diskret sind, ergibt sich folgender einfach Korrektheitsbeweis über Induktion.

Bezeichne dafür für jede gültige Politik π und jedes k

$$\pi^k = \{\mu_k, \ldots, \mu_{N-1}\}$$

die Restpolitik von π der letzten Perioden.
Für k = 0,..,N-1 seien

$$J^*_k(x_k) = \min_{\pi_k} \underset{w_k,\ldots,w_{N-1}}{E} \left\{ g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i), w_i) \right\}$$

die optimalen Kosten für das (N-k)-stufige Restproblem, welches in x_k zum Zeitpunkt k startet und zum Zeitpunkt N endet.

TECHNISCHE UNIVERSITÄT DARMSTADT

Der DP-Algorithmus (Dynamic Programming Algorithmus)

$$J^*_N(x_N) = g_N(x_N),$$

$$J^*_k(x_k) = \min_{(\mu_k, \pi^{k+1})} \underset{w_k, \ldots, w_{N-1}}{E} \left\{ g_k(x_k, \mu_k(x_k), w_k) + g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i), w_i) \right\}$$

$$= \min_{\mu_k} \underset{w_k}{E} \left\{ g_k(x_k, \mu_k(x_k), w_k) + \min_{\pi^{k+1}} \left[\underset{w_{k+1}, \ldots, w_{N-1}}{E} \left\{ g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i), w_i) \right\} \right] \right\}$$

$$= \min_{\mu_k} \underset{w_k}{E} \left\{ g_k(x_k, \mu_k(x_k), w_k) + J^*_{k+1}(g_k(x_k, \mu_k(x_k), w_k)) \right\}$$

$$= \min_{u_k \in U_k(x_k)} \underset{w_k}{E} \left\{ g_k(x_k, u_k, w_k) + J_{k+1}(g_k(x_k, u_k, w_k)) \right\}$$

$$= J_k(x_k)$$

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

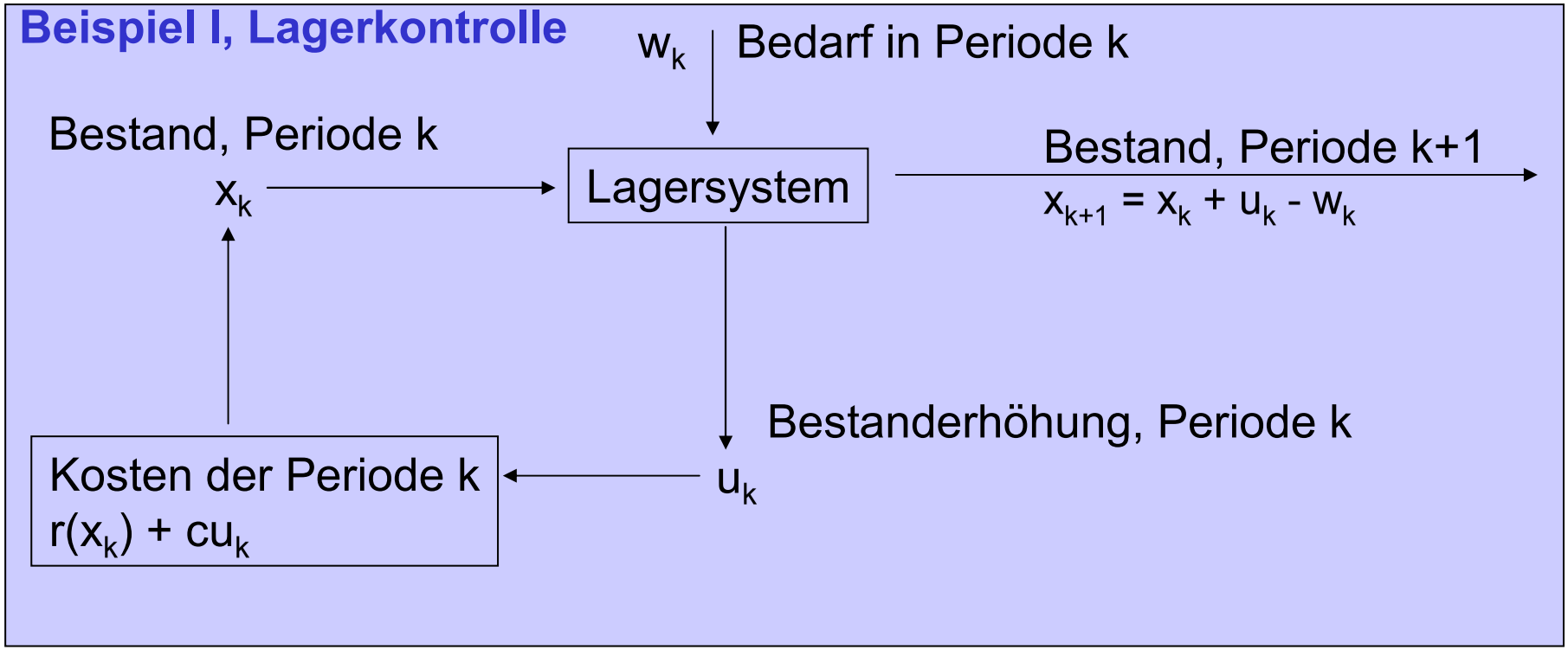
Beispiel I, Lagerkontrolle

- Im Verlauf von N Zeitschritten wird zu jedem Zeitschritt eine bestimmte Anzahl eines bestimmten Gutes von Außen geordert. Wir müssen den Lagerbestand möglichst klein halten, gleichzeitig aber verhindern, dass Anforderungen nicht erfüllt werden können.
  - x_k ist der Lagerbestand zu Beginn von Periode k
  - u_k ist die Menge des Gutes, mit dem wir das Lager nach Periode k auffüllen.
  - w_k Bedarf während der k-ten Periode mit gegebener Zufallsverteilung, $w_0,...,w_{N-1}$ seien unabhängige Zufallsvariablen.
  - Bedarf, den wir nicht decken können wird als negativer Lagerbestand fortgeführt, und wird so bald wie möglich bedient.
  - - Kosten $r(x_k)$ repräsentieren Strafen für positiven und negativen Bestand.
    - $R(x_N)$ sind Endkosten für Lagerbestand am Ende
    - cu_k sind Bestellkosten, wobei c die Kosten pro Einheit des Gutes sind.
  - Der Lagerbestand entwickelt sich also wie folgt:

$$x_{k+1} = x_k + u_k - w_k$$

Einführung, Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT



Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel I, Lagerkontrolle

Rest-Teilproblem der Länge 1:

Zu Beginn der Periode N-1 sei der Lagerbestand x_{N-1}. Klar: Egal was vorher war, sollte man mittels $u_{N-1} \geq 0$ die Orderkosten plus die erwarteten Lager/Strafkosten minimieren:

$$\min_{u_{N-1}} \quad cu_{N-1} + \underset{w_{N-1}}{E}\{R(x_{N-1} + u_{N-1} - w_{N-1})\}$$

Die optimalen Kosten für die letzte Periode sind

$$J_{N-1}(x_{N-1}) = r(x_{N-1}) + \min_{u_{N-1} \geq 0}\left[cu_{N-1} + \underset{w_{N-1}}{E}\{R(x_{N-1} + u_{N-1} - w_{N-1})\} \right]$$

Optimale Kontrolle: Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel I, Lagerkontrolle

Rest-Teilproblem der Länge 2:

Zu Beginn der Periode N-2 sei der Lagerbestand x_{N-2}. Klar: Egal was vorher war, sollte man mittels $u_{N-2} \geq 0$

(erwarteten Kosten der Periode N-2) +
(erwarteten Kosten der Periode N-1, bei optimaler Politik)
→ $r(x_{N-2}) + cu_{N-2} + E\{J_{N-1}(x_{N-1})\}$

minimieren.

$$J_{N-2}(x_{N-2}) = r(x_{N-2}) + \min_{u_{N-2} \geq 0} \left[cu_{N-2} + \underset{w_{N-2}}{E} \{J_{N-1}(x_{N-2} + u_{N-2} - w_{N-2})\} \right]$$

Rest-Teilproblem der Länge k:
minimiere mittel u_{N-k} :

(erwarteten Kosten der Periode N-k) +
(erwarteten Kosten der Periode N-k+1, bei optimaler Politik)

$$J_{N-k}(x_{N-k}) = r(x_{N-k}) + \min_{u_{N-k} \geq 0} \left[cu_{N-k} + \underset{w_k}{E} \{J_{N-k+1}(x_{N-k} + u_{N-k} - w_{N-k})\} \right] \longrightarrow \text{DP}$$

TECHNISCHE UNIVERSITÄT DARMSTADT

Lineares System mit quadratischen Kosten

- Betrachtet wird folgender Spezialfall:

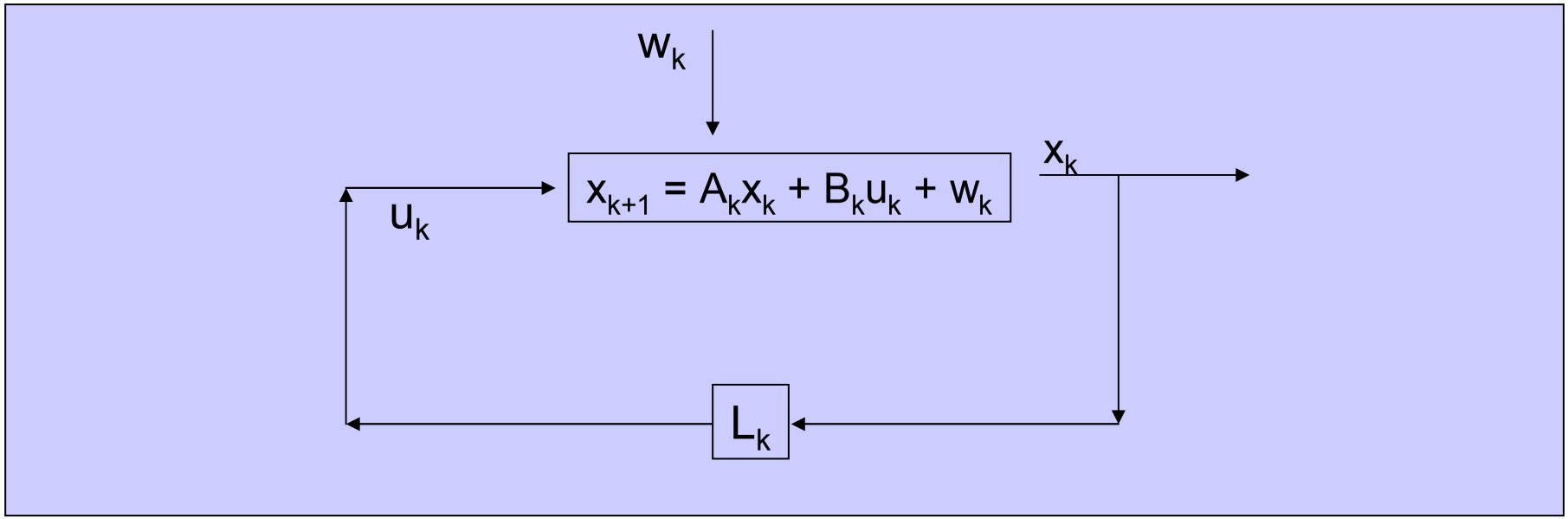
  $$x_{k+1} = A_k x_k + B_k u_k + w_k,\ k = 0,\ldots,N-1$$

  mit den quadratischen Kosten

$$\underset{\substack{w_k \\ k=0,\ldots,N-1}}{E} \left\{ x_N^T Q_N x_N + \sum_{k=0}^{N-1} (x_k^T Q_k x_k + u_k^T R_k u_k) \right\}$$

- x_k und u_k sind Vektoren der Dimensionen n und m
- A_k, B_k, Q_k, R_k sind Matrizen mit passenden Dimensionen
- Q_k positiv semidefinit symmetrisch
- R_k positiv definit symmetrisch w_k seien unabhängig zufällig
- w_k habe Durchschnittswert 0

Einführung, Beispiele



Linearer Feedback-Controller für linear-quadratisches System

TECHNISCHE UNIVERSITÄT DARMSTADT

Lineares System mit quadratischen Kosten

→ Populäre Formulierung von Steuerungsproblemen.

1. die Kostenfunktion $J_k(x_k)$ ist quadratisch und die optimale Einstellungen der Kontrollvariablen u_k bekommt man durch Anwendung einer linearen Funktion auf den Zustand x_k.
2. Problem, geeignete Steuerung zu finden ist damit in P (quadratische Zeit) aber:
   Papadimitriou: R nonpositive-definite => Problem wird PSPACE-hard

   Herleitung zu 1.: Anwenden des DP-Algorithmus liefert:

$$J_N(x_N) = x_N^T Q_N x_N,$$
$$J_k(x_k) = \min_{u_k} \underset{w_k}{E}\left\{x_k^T Q_k x_k + u_k^T R_k u_k + J_{k+1}(A_k x_k + B_k u_k + w_k)\right\}$$

TECHNISCHE UNIVERSITÄT DARMSTADT

Lineares System mit quadratischen Kosten

→ Einsetzen von hinten ergibt:

$$J_N(x_N) = x_N^T Q_N x_N,$$
$$J_k(x_k) = \min_{u_k} E_{w_k}\left\{x_k^T Q_k x_k + u_k^T R_k u_k + J_{k+1}(A_k x_k + B_k u_k + w_k)\right\}$$

→ insbesondere für k = N-1:

$$J_{N-1}(x_{N-1}) = \min_{u_{N-1}} E_{w_{N-1}}\left\{\begin{array}{l} x_{N-1}^T Q_{N-1} x_{N-1} + u_{N-1}^T R_{N-1} u_{N-1} + \\ \left(A_{N-1}x_{N-1} + B_{N-1}u_{N-1} + w_{N-1}\right)^T Q_N \left(A_{N-1}x_{N-1} + B_{N-1}u_{N-1} + w_{N-1}\right) \end{array}\right\}$$

→ Durch Ausmultiplizieren des unteren Terms und Elimination des Terms $E\{w^T_{N-1} Q_N (A_{N-1}x_{N-1} + B_{N-1}u_{N-1})\}$, da $E\{w_{N-1}\}=0$, ergibt sich:

$$J_{N-1}(x_{N-1}) = x_{N-1}^T Q_{N-1} x_{N-1} + \min_{u_{N-1}}\left[\begin{array}{l} u_{N-1}^T R_{N-1} u_{N-1} + \\ u_{N-1}^T B_{N-1}^T Q_N B_{N-1} u_{N-1} + 2x_{N-1}^T A_{N-1}^T Q_N B_{N-1} u_{N-1} \end{array}\right]$$

$$+ x_{N-1}^T A_{N-1}^T Q_N A_{N-1} x_{N-1} A_{N-1} x_{N-1} + E\{w_{N-1}^T Q_N w_{N-1}\}$$

TECHNISCHE UNIVERSITÄT DARMSTADT

Lineares System mit quadratischen Kosten

→ Durch Bilden der totalen Ableitung nach u_{N-1}, und durch setzen der Ableitung zu Null ergibt sich:

$$(R_{N-1} + B_{N-1}^T Q_N B_{N-1}) u_{N-1} = - B_{N-1}^T Q_N A_{N-1} x_{N-1}$$

→ Die Matrix, die von links an u_{N-1} anmultipliziert wird, ist positive definit (und somit) invertierbar), weil R_{N-1} positiv definit ist und $B^T_{N-1} Q_N B_{N-1}$ positiv semidefinit ist. Minimierung des Kontrollvektors ergibt:

$$u^*_{N-1} = -(R_{N-1} + B_{N-1}^T Q_N B_{N-1})^{-1} B_{N-1}^T Q_N A_{N-1} x_{N-1}$$

→ Wenn wir diesen Ausdruck in J_{N-1} einsetzen, erhalten wir

$$J_{N-1}(x_{N-1}) = x_{N-1}^T K_{N-1} x_{N-1} + E\{w_{N-1}^T Q_N w_{N-1}\}, \; wobei$$

$$K_{N-1} = A_{N-1}^T (Q_N - Q_N B_{N-1} (B_{N-1}^T Q_N B_{N-1} + R_{N-1})^{-1} B_{N-1}^T Q_N) A_{N-1} + Q_{N-1}$$

TECHNISCHE UNIVERSITÄT DARMSTADT

Lineares System mit quadratischen Kosten

→ K_{N-1} ist positiv semidefinit und symmetrisch
- K_{N-1} wird aus symmetrischen Matrizen zusammengesetzt und
- $x^T K_{N-1} x = \min[x^T Q x + u^T R u + (A_{N-1}x+B_{N-1}u)^T Q_N (A_{N-1}x+B_{N-1}u)]$
  Q_{N-1}, R_{N-1} und Q_N sind pos. semidef. also ist die rechte Seite nicht negativ.
- also ist auch J_{N-1} positiv semidefinit

→ durch mehrfaches induktives Einsetzen erhält man

$$\mu^*_k(x_k) = L_k x_k$$
$$\text{mit } L_k = -(B^T{}_k K_{k+1} B_k + R_k)^{-1} B^T{}_k K_{k+1} A_k$$

→ und mit $K_N = Q_N$ und $K_k = A^T{}_k (K_{k+1} - K_{k+1} B_k (B^T{}_k K_{k+1} B_k + R_k)^{-1} B^T{}_k K_{k+1}) A_k + Q_k$

Diskrete Riccati Gleichung

→ und

$$J_0(x_0) = x_0^T K_0 x_0 + \sum_{k=0}^{N-1} E\{w_k^T K_{k+1} w_k\}$$

2-stufige Programme , Beispiel

Farmer – Problem

- 500 ha Land
- mindestens 200t Weizen und 240t Mais werden fürs Vieh benötigt.
- Verkaufspreis von Weizen/Mais = 170/150 Euro/t, für nicht benötigtes Getreide
- Einkaufspreis von Weizen/Mais = 238/210 Euro/t
- Verkaufspreis von Zuckerrüben: 36 Euro/t unter 6000t; 10 Euro ab mehr als 6000t (wegen Agrarregeln in der EU)
- Planzkosten Weizen/Mais/Zuckerrüben = 150/230/260 Euro/ha
- Ertrag in t/ha für Weizen/Mais/Zuckerrüben = 2.5/3/20

x_1 = Land für Weizen; x_2 = Land für Mais; x_3 = Land für Zuckerrüben;
w_1 = verkaufter Weizen in Tonnen; w_2 = verkaufter Mais in Tonnen; w_3 = verkaufte Zuckerüben zu gutem Preis; w_4 = verkaufte Zuckerüben zu schlechtem Preis;
y_1 = Tonnen gekaufter Weizen; y_2 = Tonnen gekaufter Mais

TECHNISCHE UNIVERSITÄT DARMSTADT

2-stufige Programme , Beispiel

$$\min 150x_1 + 230x_2 + 260x_3 + 238y_1 - 170w_1 + 210y_2 - 150w_2 - 36w_3 - 10w_4$$

s.t.

$$x_1 + x_2 + x_3 \leq 500$$
$$2.5x_1 + y_1 - w_1 \geq 200$$
$$3x_2 + y_2 - w_2 \geq 240$$
$$w_3 + w_4 \leq 20x_3$$
$$w_3 \leq 6000$$
$$x_1, x_2, x_3, y_1, y_2, w_1, w_2, w_3, w_4 \geq 0$$

2-stufige Programme , Beispiel

Pflanzenart	Weizen	Mais	Zuckerrüben
Anzahl ha	120	80	300
Ertrag in Tonnen	300	240	6000
Verkauf in Tonnen	100	-	6000
Einkauf in Tonnen	-	-	-

Gewinn bei optimaler Lösung: 118.600 Euro

Unser Farmer ist verunsichert: Ertrag hängt doch sehr vom Wetter ab. Annahme, der Ertrag je ha erhöht / erniedrigt sich um 20%:

Pflanzenart	Weizen	Mais	Zuckerr
Anzahl ha	183.33	66.67	250
Ertrag in T.	550	240	6000
Verkauf in T.	350	-	6000
Einkauf in T.	-	-	-

max. Gewinn bei +20%: 167.667 Euro

Pflanzenart	Weizen	Mais	Zuckerr.
Anzahl ha	100	25	375
Ertrag in T.	200	60	6000
Verkauf in T.	-	-	6000
Einkauf in T.	-	180	-

max. Gewinn bei -20%: 59.950 Euro

TECHNISCHE UNIVERSITÄT DARMSTADT

2-stufige Programme , Beispiel

Unser Farmer würde gerne flexibel reagieren können. Entscheidungen für das Land (x_1,x_2,x_3) müssen sofort gefällt werden, aber die anderen Entscheidungen hängen von den Erträgen je ha ab.

→ Bilde 3 Szenarios mit index 1,2,3. Jedes Szenario habe Eintrittswahrscheinlichkeit 1/3.

$$\min\ 150x_1 + 230x_2 + 260x_3$$
$$- 1/3*(170w_{11}-238y_{11}+150w_{21}-210y_{21}+36w_{31}+10w_{41}) \quad \text{Szenario 1}$$
$$- 1/3*(170w_{12}-238y_{12}+150w_{22}-210y_{22}+36w_{32}+10w_{42}) \quad \text{Szenario 2}$$
$$- 1/3*(170w_{13}-238y_{13}+150w_{23}-210y_{23}+36w_{33}+10w_{43}) \quad \text{Szenario 3}$$
$$\text{s.t. } x_1 + x_2 + x_3 \le 500,\ 3x_1 + y_{11} - w_{11} \ge 200,\ 3.6x_2 + y_{21} - w_{21} \ge 240,$$
$$w_{31} + w_{41} \le 24x_3,\ w_{31} \le 6000,\ 2.5x_1 + y_{12} - w_{12} \ge 200$$
$$3x_2 + y_{22} - w_{22} \ge 240,\ w_{32} + w_{42} \le 20x_3,\ w_{32} \le 6000,$$
$$2x_1 + y_{13} - w_{13} \ge 200,\ 2.4x_2 + y_{23} - w_{23} \ge 240,\ w_{33} + w_{43} \le 16x_3,\ w_{33} \le 6000,$$
$$x, y, w \ge 0$$

Stochastic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

2-stufige Programme, Beispiel

	Pflanzenart	Weizen	Mais	Zuckerrüben
1. Stufe	Anzahl ha	170	80	250
s=1 (+20%)	Ertrag in Tonnen	510	288	6000
	Verkauf in Tonnen	310	48	6000
	Einkauf in Tonnen	-	-	-
s=2	Ertrag in Tonnen	425	240	5000
	Verkauf in Tonnen	225	-	5000
	Einkauf in Tonnen	-	-	-
s=3 (-20%)	Ertrag in Tonnen	340	192	4000
	Verkauf in Tonnen	140	-	4000
	Einkauf in Tonnen	-	48	-

Erwarteter Gewinn bei optimaler Lösung: 108.390 Euro

Durchschnittlicher Gewinn der optimalen Einzellösungen: 115.406 Euro
Differenz = 7016 Euro ist „erwarteter Wert von perfekter Information“

Stochastic Programming

TECHNISCHE UNIVERSITÄT DARMSTADT

mehrstufige Programme, formal

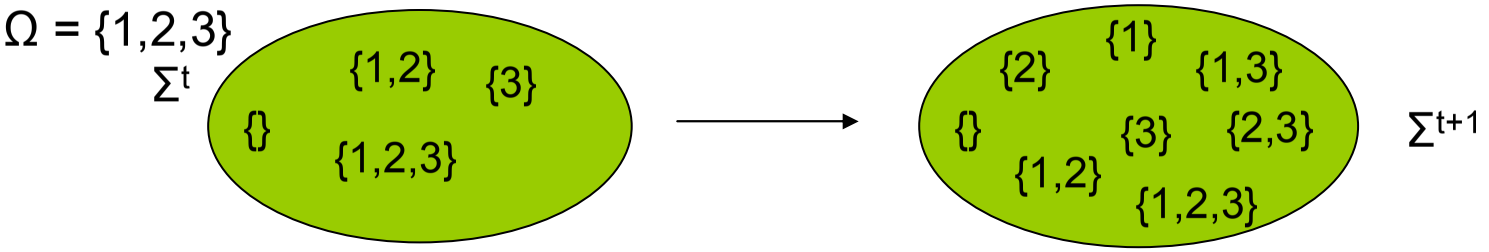
$$\min z = c^1 x^1 + E_{\xi^2}\left[\min c^2(\omega)x^2(\omega^2) + \ldots + E_{\xi^H}\left[\min c^H(\omega)x^H(\omega^H)\right]..\right]$$

$$\begin{aligned} s.t. \quad & W^1 x^1 = h^1, \\ & T^1(\omega)x^1 + W^2 x^2(\omega^2) = h^2(\omega), \\ & \vdots \\ & T^{H-1}(\omega)x^{H-1}(\omega^{H-1}) + W^H x^H(\omega^H) = h^H(\omega), \\ & x^1 \geq 0; x^t(\omega^t) \geq 0, t = 2,\ldots,H \end{aligned}$$

TECHNISCHE UNIVERSITÄT DARMSTADT

mehrstufige Programme, formal

c_1 Vektor in $\mathbb{Q}^{n1}$, h_1 Vektor in $\mathbb{Q}^{m1}$, $\xi^t(\omega) = (c^t(\omega),h^t(\omega),T_1^{t-1}(\omega),...,T_{mt}^{t-1}(\omega))$
ist ein Zufallsvektor, definiert auf (Ω,Σ^t,P), für alle $t = 2,...,H$. Dabei ist $\Sigma^t \subseteq \Sigma^{t+1}$.
W^t ist eine feste Matrix und spiegelt den festen Recourse wider.



Entscheidungen x hängen von der Historie bis zum Zeitpunkt t ab,
die Historie bezeichnen wir mit ω^t

mehrstufige Programme, deterministisches Äquivalent, Version 1

als dynamisches Programm:

$$Q^H(x^{H-1}, \xi^H(\omega)) = \min c^H(\omega) x^H(\omega) \qquad \text{letzte Stufe}$$

$$s.t. \quad W^H x^H(\omega) = h^H(\omega) - T^{H-1}(\omega) x^{H-1},$$

$$x^H(\omega) \geq 0 \quad und$$

$$Q^t(x^{t-1}, \xi^t(\omega)) = \min c^t(\omega) x^x(\omega) + E_\xi\{Q(x^t, \xi)\} \qquad \text{Stufen 2,...H-1}$$

$$s.t. \quad W^t x^t(\omega) = h^t(\omega) - T^{t-1}(\omega) x^{t-1},$$

$$x^t(\omega) \geq 0.$$

Der Wert, den wir suchen ist: 1. Stufe

$$\min c^1 x^1(\omega) + E_\xi\{Q(x^1, \xi)\}$$

$$s.t. \quad W^1 x^1 = h^1, \quad x^1 \geq 0.$$

Mehrstufige Programme, deterministisches Äquivalent, Version 2

Als (gemischt ganzzahliges) lineares Programm:

Annahmen:

- klare **Stufung** von Entscheidungsvariablen und Zufallsereignissen

- Zufallsereignisse **unabhängig** von unseren Entscheidungen
  (Das Wetter wird im Normalfall morgen sonnig oder regnerisch sein,
  egal, welche Entscheidung wir in unserem Optimierungsproblem treffen.)

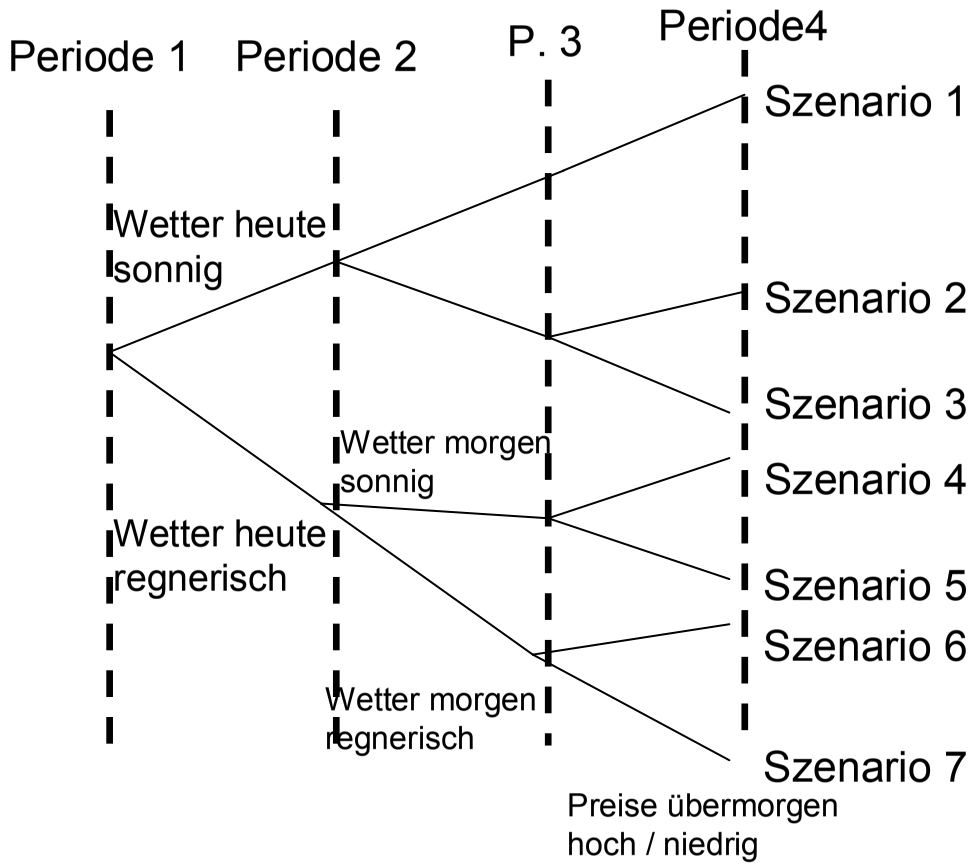
  → essentielle Annahmen für stochastische Programme
  → erlaubt die Aufspaltung von Zufallsprozess und Entscheidungsprozeß
  → führt zu **Szenariobäumen**

- **endliche Anzahl von möglichen Realisierungen** für zukünftigen Ausgängen

TECHNISCHE UNIVERSITÄT DARMSTADT

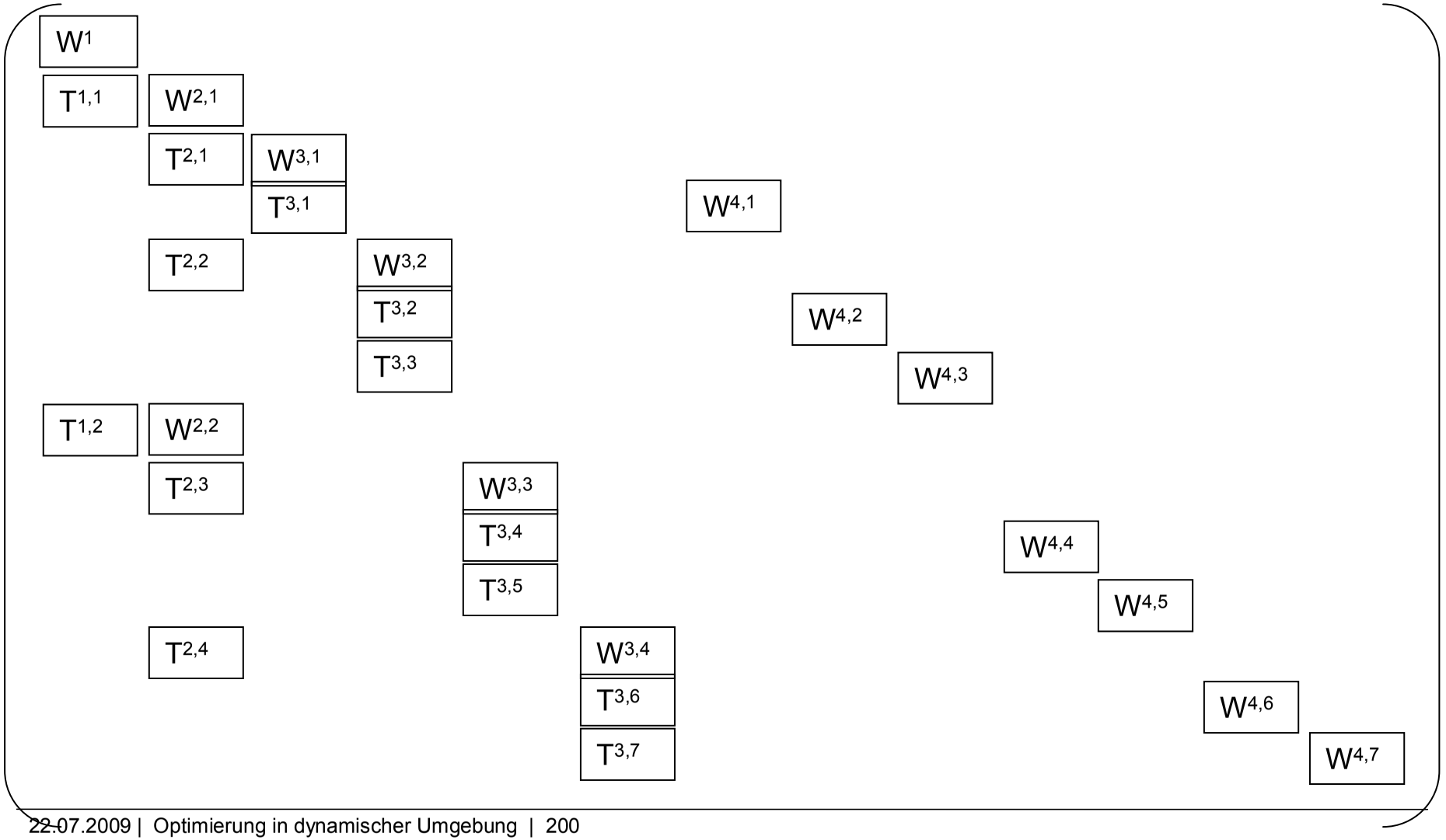
Mehrstufige Programme, deterministisches Äquivalent, Version 2

Szenariobaum:



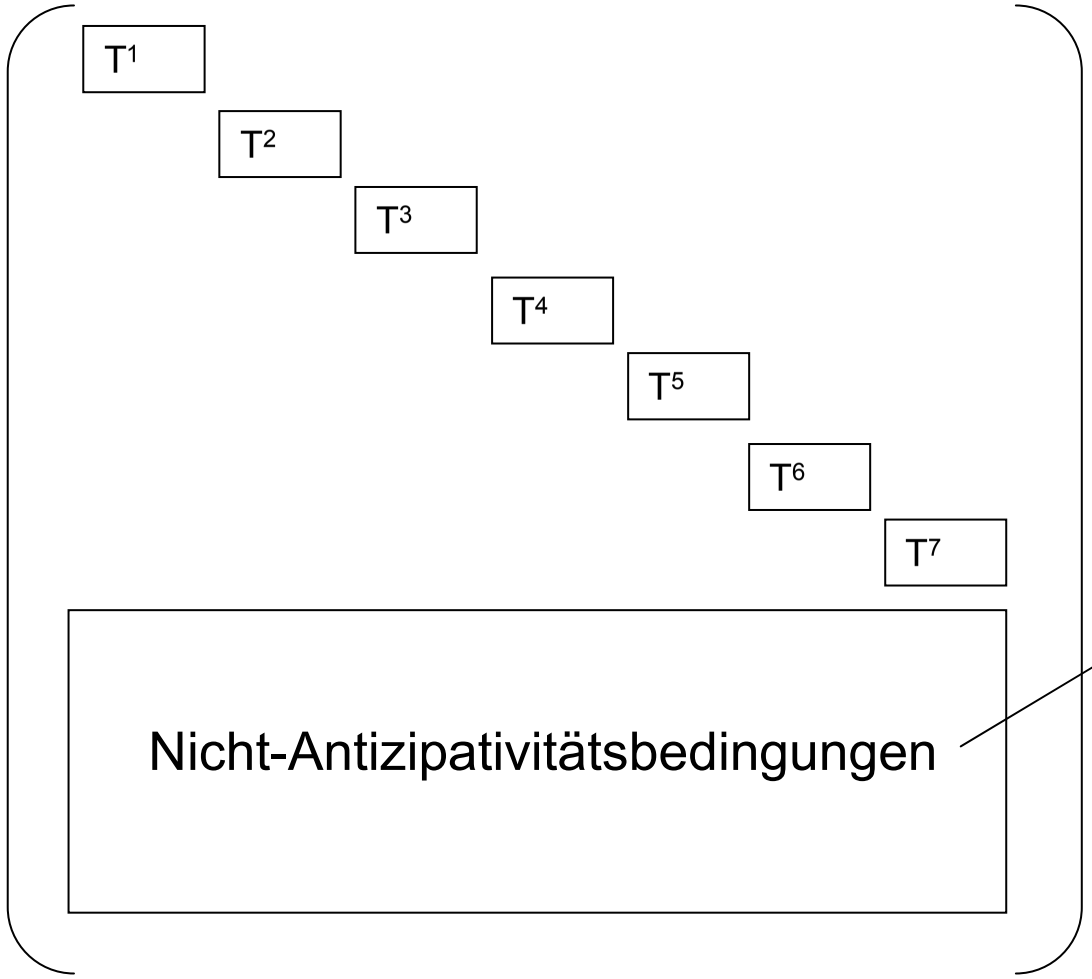
Stochastic Programming

Matrix



Stochastic Programming

Matrix, andere Darstellung



Vervielfachung von Variablen führt zu entkoppelten Szenarien

Einige Variablen müssen in allen (manchen) anderen Szenarien gleich sein.

Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

Eine NCL-“Maschine“ ist definiert über einen Graphen:
gegeben:

- ungerichteter Graph mit
  - nicht-negativen ganzzahligen ***Gewichten*** an den Kanten und
  - ganzzahligen ***minimum*** *in-flow constraints* an Knoten

Eine ***Konfiguration*** der Maschine ist eine Orientierung (Richtung) der Kanten, so dass die Summe der eingehenden Kantengewichte an jedem Knoten mindestens so groß ist, wie der minimum in-flow constraint für den jeweiligen Knoten.

Ein Zug von einer Konfiguration zu einer Anderen ist die Umkehrung einer Kante, so dass die Bedingungen an Konfigurationen eingehalten werden.

Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

Fragestellungen:

- Seien zwei Konfigurationen gegeben. Gibt es eine Folge von Zügen von A nach B?
- Gegeben seien 2 Kanten E_A und E_B und Orientierungen für diese. Gibt es Konfigurationen A und B, so dass E_A die gewünschte Richtung in A und E_B in B haben, und es eine Folge von Zügen von A nach B gibt?

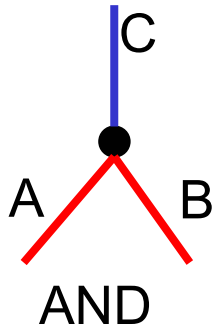
Nondeterministic Constraint Logic (NCL)

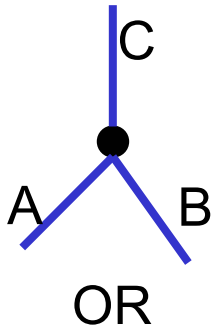
TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

AND/OR Constraint Graphen (spezielle NCL-Graphen)

- Knoten mit minimum in-flow constraint 2 und Kanten-Eingangsgewichte 1, 1 und 2 verhalten sich in folgendem Sinne wie ein AND:
  Die Kante mit Gewicht 2 kann nur nach außen zeigen, wenn beide Kanten mit Gewicht 1 nach innen zeigen.
- Knoten mit minimum in-flow constraint 2 und Kanten-Eingangsgewichte 2, 2 und 2 verhalten sich in folgendem Sinne wie ein OR:
  Eine der Kanten kann nur nach außen zeigen, wenn eine der beiden anderen Kanten nach innen zeigt.





Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

Unterschiede NCL und klassische Logik

NCL	klassisch
* nicht-deterministisch	* deterministisch
* nicht a priori festgelegt, was Inputs, und was Outputs sind	* Inputs und Outputs statisch
* kein NOT	* Inverter essentiell

Nicht-Determinismus:
Wenn z.B. zwei Kanten in ein AND hineingehen, ist es der 3. Kante **erlaubt**, nach aussen gedreht zu werden. Es ist **nicht zwingend**. Ob es möglich ist, eine bestimmte Kante nach aussen zu drehen, ist nicht lokal ersichtlich.

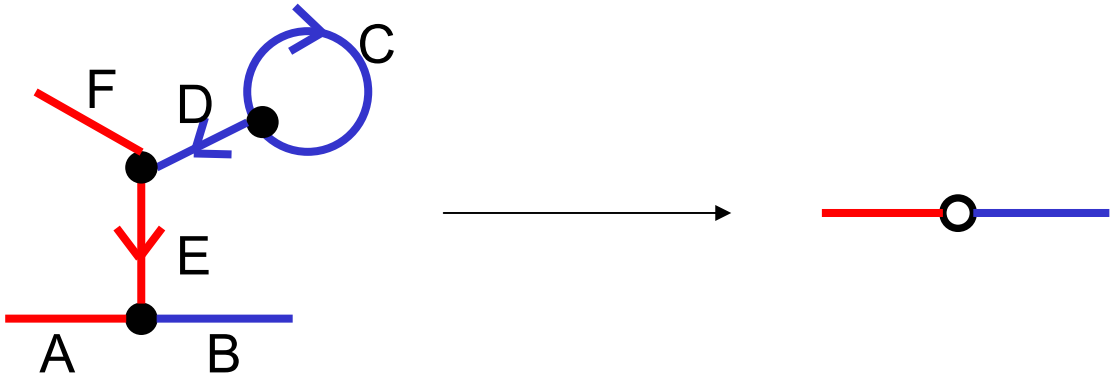
Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

Blau-Rot-Adapter

Betrachtet man AND/OR-Graphen als Schaltkreise, möchten wir verschiedene Outputs mit Inputs verbinden. Bei verschiedenartigen Kanten (rot/blau) ist das nicht direkt möglich.



Es gibt eine nicht- verbundene Kante bei F. Etwas Überlegung besagt, solche Kanten treten in Paaren auf: Rote Kanten treten nur in OR-Elementen auf, d.h., F-E-A führt zu einer weiteren freien Kante X. Wir verbinden F mit X.

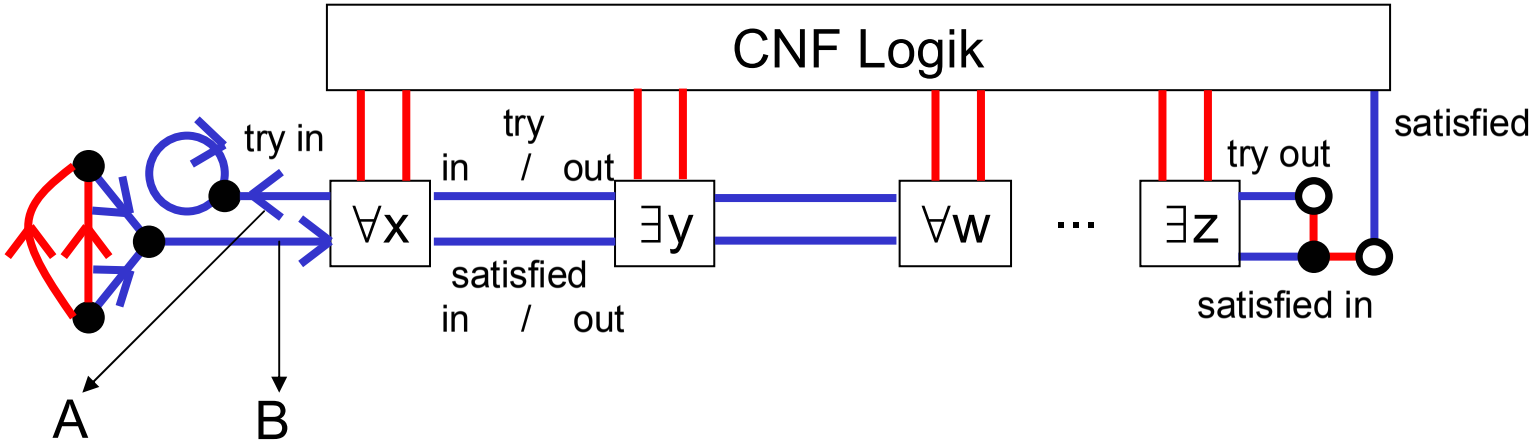
Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

PSPACE-completeness (zunächst: PSPACE-hard)

Überblick:



Kann man A umdrehen, so dass sich auch B umdrehen lässt?

Eine Möglichkeit, den Wahrheitsgehalt einer QBF zu finde geht so:
Gehe von außen nach innen durch die Quantifizierer. Bei Allvariaben: setze diese erst auf false, dann auf true und prüfe jeweils rekursiv, ob der Rest erfüllbar ist. Falls ja, return true, sonst false. Bei Existenzvariablen: return true, g.d.w. eine der Zuweisungen (true/false) erfolgreich ist.

Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

PSPACE-completeness (zunächst: PSPACE-hard)

Überblick:

Quantifizierer-Schaltungen:
Wenn ein Quantifizierer **aktiviert** wird
- sind alle Variablen links (im Sinne des Bildes, vorige Folie) von ihm fixiert
- und die Variablen des Quantifizierers und rechts davon sind frei

Ein Quantifizierer kann sich dann und nur dann als **satisfied** melden, wenn die Formel von diesem Quantifizierer bis ganz nach rechts unter der Berücksichtigung der Quantoren erfüllbar ist.

Ein Quantifizierer wird aktiviert, indem seine *try-in*-Kante so gedreht wird, dass sie in den Quantifizierer hineinzeigt. Seine *try-out*-Kante kann nur vom Quantifizierer weg zeigen, wenn die *try-in*-Kante hineinzeigt, und seine Variablen fixiert sind.

Die Variablenzuweisung wird durch zwei herausgehende Kanten (x und $\overline{x}$) repräsentiert, von denen nur eine aus dem Quantifizierer hauszeigen kann.

Nondeterministic Constraint Logic (NCL)

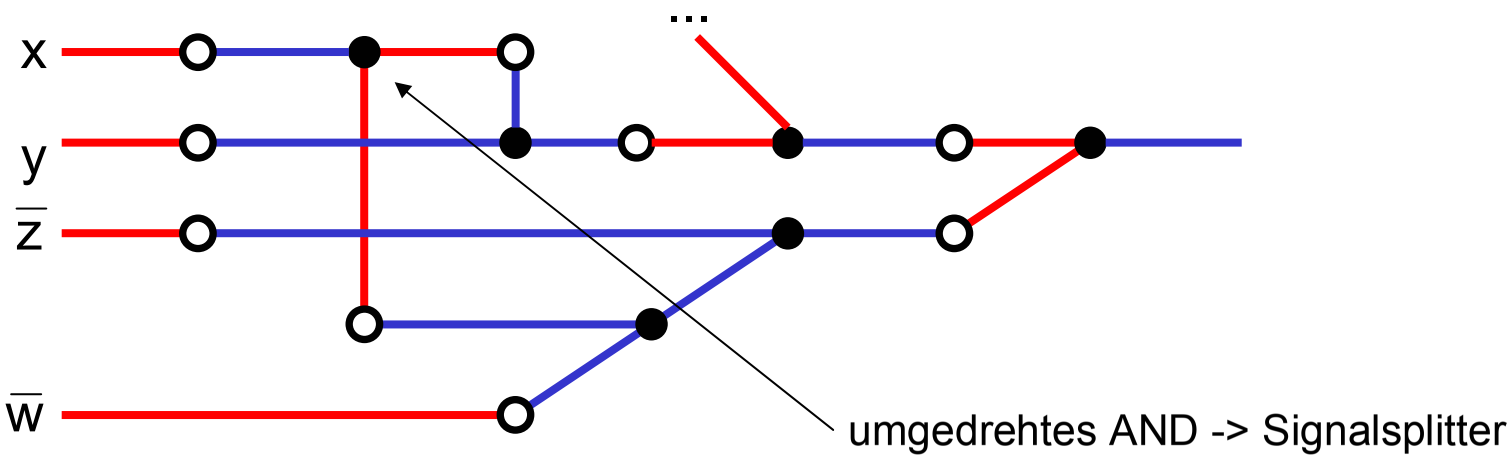
TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

PSPACE-completeness (zunächst: PSPACE-hard)

Details:

Kodierung der CNF-Formel
Die CNF Formel wird entsprechend herkömmlicher Logik verdrahtet.
Bsp:



$(x \vee y) \wedge \ldots \wedge (\overline{z} \vee x \vee \overline{w})$

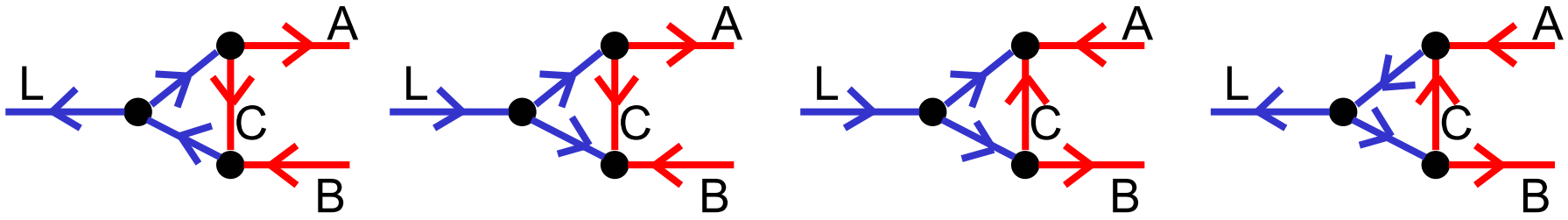
Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

PSPACE-completeness

Variablen fixieren („Bits fangen“)
Latches speichern Bits fest:



fixiert, A aktiv | gelöst, A aktiv | gelöst, B aktiv | fixiert, B aktiv

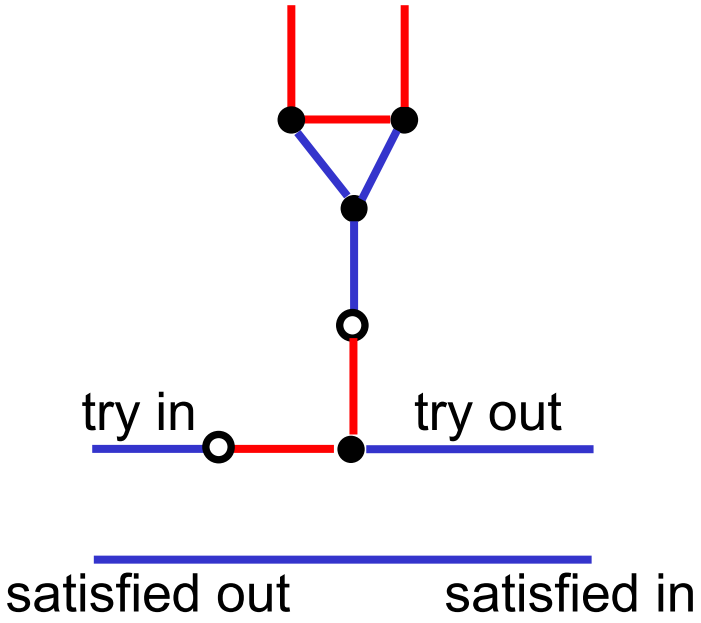
L n.links => eine der beiden anderen OR-Kanten nach links =>
B links (A egal) und **C nach unten**
L n.rechts => die Orientierung der beiden anderen OR-Kanten nach rechts ist möglich => **Umdrehen von C ist möglich**. Wurde C gedreht, kann man das Latch wieder festfrieren und L nach links zeigen lassen.

Nondeterministic Constraint Logic (NCL)

Graph-Formulierung

PSPACE-completeness

Existenzquantor



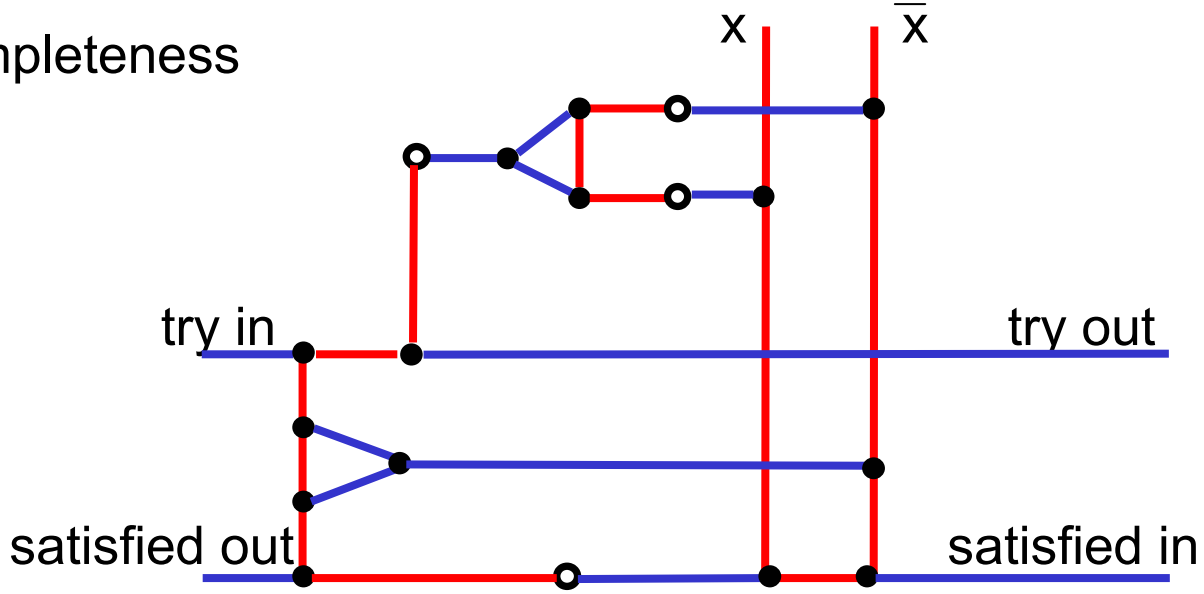
Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

PSPACE-completeness

Allquantor



Beh: Ein Allquantor Schaltkreis kann seine satisfied-out Kante nur nach rechts (gem. Bild) richten, wenn zuerst $\bar{x}$ nach oben zeigt und die satisfied-in Kante nach links zeigt, und später x noch oben und satisfied-in nochmal nach links zeigt.

Bew. s. Hearn and Demain: PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation

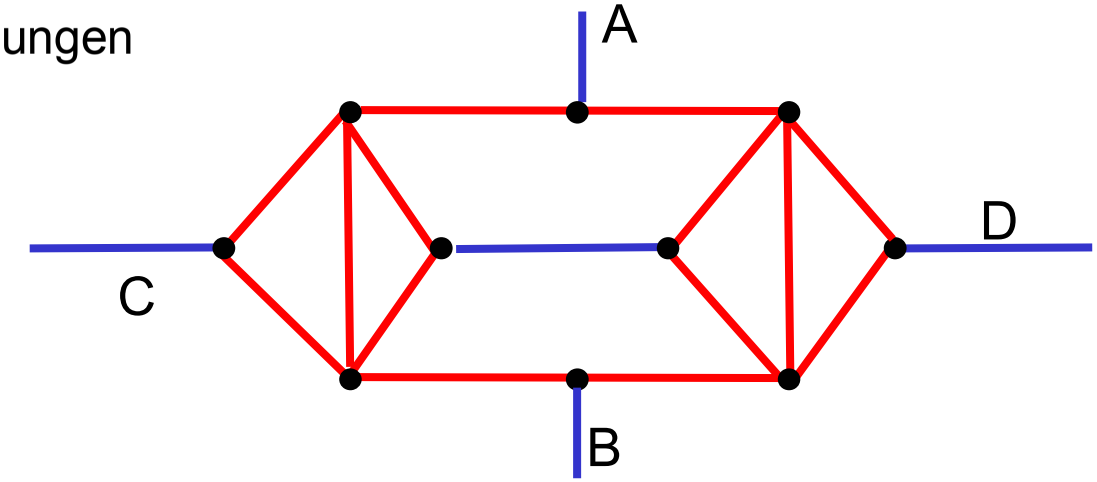
Alles zusammen: NCL ist PSPACE-schwer

Nondeterministic Constraint Logic (NCL)

Graph-Formulierung

PSPACE-completeness

Kreuzungen



Die bisherigen Graphen waren nicht planar. Mit Hilfe von Kreuzungen kann man die Graphen in planare Graphen umwandeln. In der Konstruktion oben kann von A und B nur einer nach aussen zeigen. Ebenso von C und D.

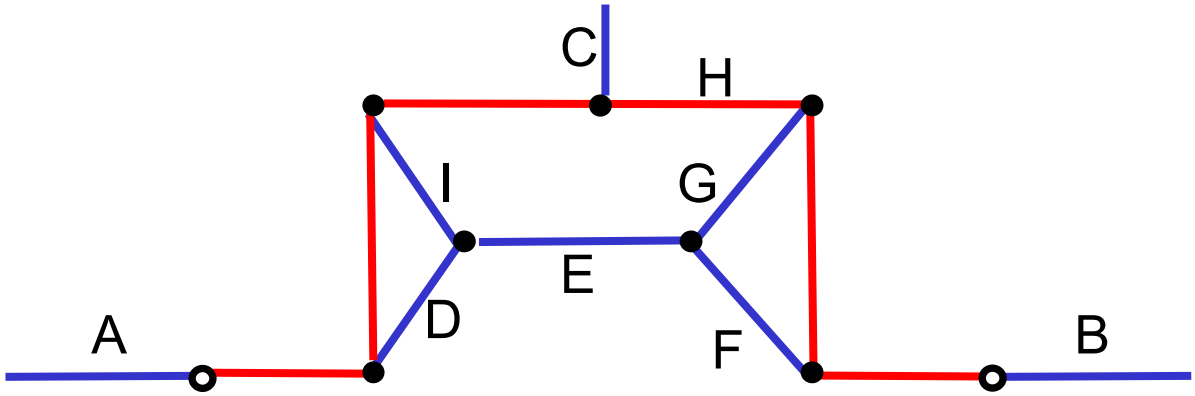
(Begründung s. Hearn and Demain: PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation)

Nondeterministic Constraint Logic (NCL)

Graph-Formulierung

PSPACE-completeness

Protected OR
Ein OR ist protected, wenn wegen globaler Bedingungen es nicht vorkommen kann, dass 2 der 3 Kanten in das OR hineinzeigen. Dann ist es nicht schlimm, wenn das OR für den Fall, dass doch 2 Kanten hineinzeigen, nicht richtig funktioniert und Fehler macht.
Man kann jedoch aus Protected ORs ein „echtes“ OR bauen:



A, B, und C verhalten sich wie ein OR; alle ORs innerhalb der Schaltung sind protected ORs

Nondeterministic Constraint Logic (NCL)

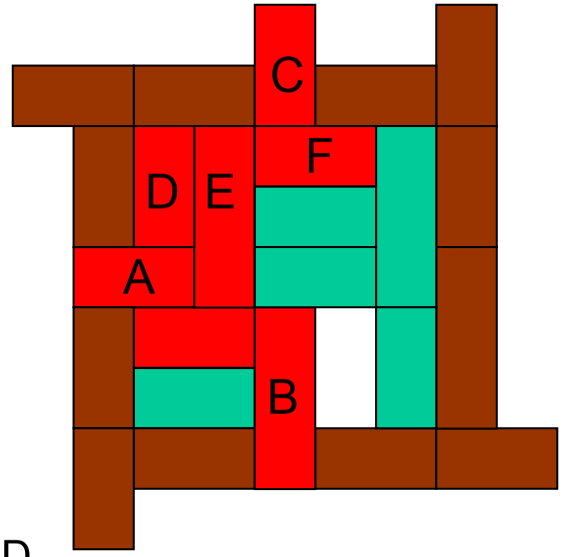
TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

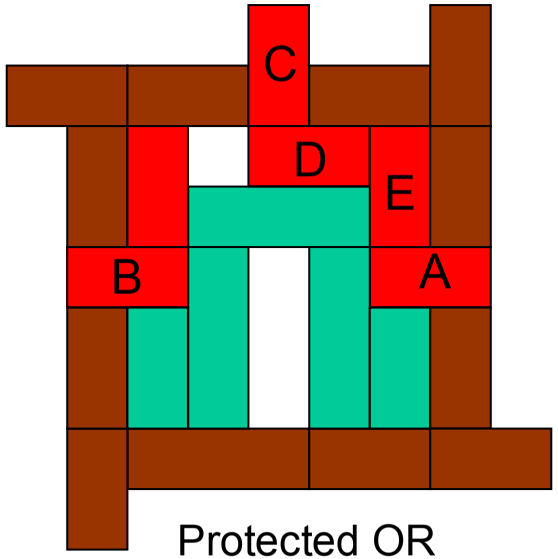
PSPACE-completeness

Rush Hour ist PSPACE schwer; AND und OR

Wie beschränken uns auf cars of 1 x 2 and 1 x 3 blocks.



AND
C darf nur runter, wenn B runter und A nach links geht



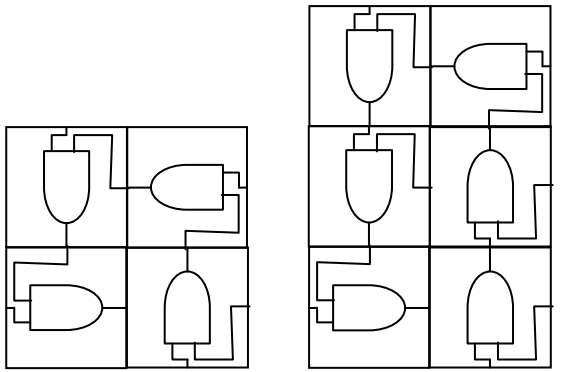
Protected OR
C darf runter, wenn B nach links oder A nach rechts geht

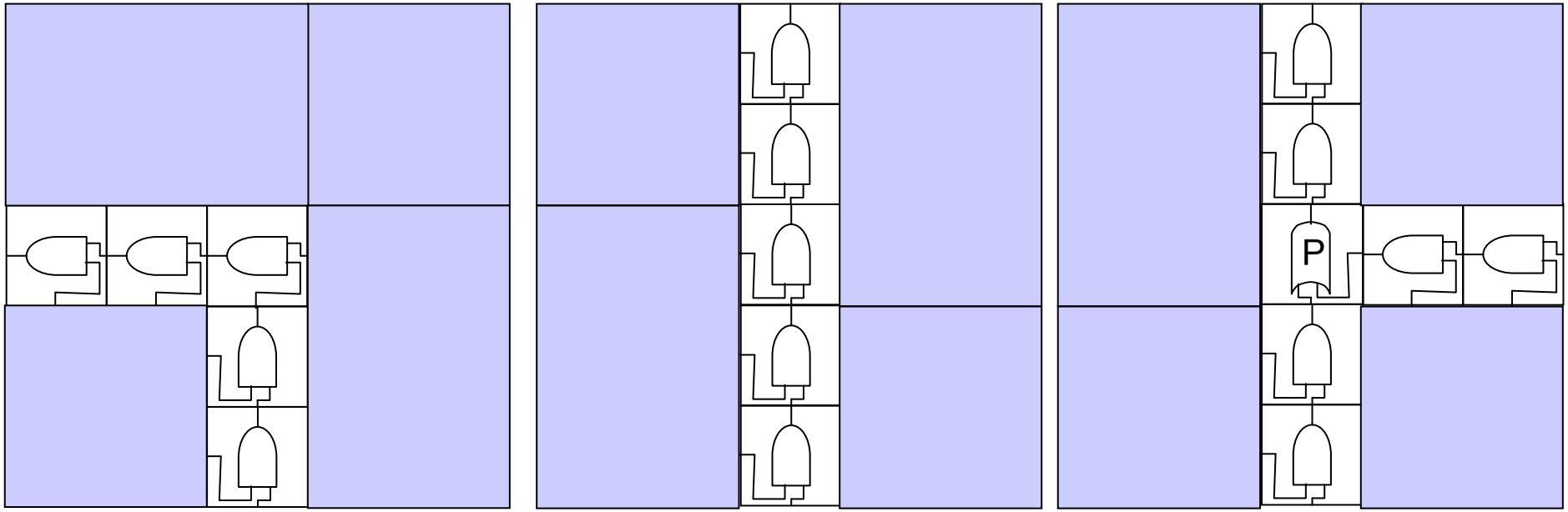
Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Graph-Formulierung

PSPACE-completeness: Rush Hour ist
PSPACE schwer; Graphen





Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz von Savitch

NPSPACE = DPSPACE

Theorem: Für jede Funktion $s(n) \geq n$ gilt: $NPSPACE_1(s(n)) \subseteq DPSPACE_3(s^2(n))$
In Worten: Jede Berechnung einer t-Zeit / 1 Band Turingmaschine kann von einer 3-Band DTM in Platz $s^2(n)$ durchgeführt werden.
(Platzbedarf wird betrachtet ohne read-only-Eingabe)

Beweis:

- Betrachte NTM für $L \in NSPACE_1(s(n))$, die mit einer eindeutigen Konfiguration C_{akz} akzeptiert. D.h., es gibt nur einen akzeptierenden Endzustand und am Ende der Berechnung wird das Band gelöscht.
- Betrachte das Prädikat Erreicht-Konf(C,C‘,S,T): Dieses Prädikat ist wahr, wenn die S-Platz-NTM M ausgehend von C die Konfiguration C‘ innerhalb von T Schritten erreicht.
- **Lemma (noch zu zeigen): Erreicht-Konf(C,C‘,S,T) kann von einer 2-Band-DTM mit S·log(T) Platz entschieden werden.**
- Nun ist $T \leq 2^{O(s(n))}$ und damit ist $\log(T) = O(s(n)) \leq c \cdot s(n)$ für eine Konstante $c > 0$.
    - Sei C_{start} die Startkonfiguration
    - Das Prädikat Erreicht-Konf(C_{start},C_{akz},s(n),$2^{O(s(n))}$) entscheidet L.
- Dann kann eine 3-Band-DTM L in Platz $c \cdot s(n) \cdot s(n) = c \cdot s^2(n) = O(s^2(n))$ die Sprache L entscheiden.

Nondeterministic Constraint Logic (NCL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz von Savitch

NPSPACE = DPSPACE

Lemma: Das Prädikat **Erreicht-Konf(C,C‘,S,T) kann von einer 2-Band- DTM mit S·log(T) Platz entschieden werden.**

Beweis: Betrachte folgende DTM M‘ auf Eingabe (C,C‘,S,T)

- falls T = 0 dann
  - akzeptiere falls C=C‘ und akzeptiere nicht falls C ungleich C‘.
- falls T = 1 dann
  - akzeptiere fals C‘ eine erlaubte Nachfolgekonfiguration von C ist, oder falls C=C‘. Sonst akzeptiere nicht.
- falls T > 1 dann
  - für alle Konfigurationen Z der Länge S
    - Berechne rekursiv r_1 = Erreicht-Konf(C,Z,S, $\lfloor T/2 \rfloor$)
    - Berechne rekursiv r_2 = Erreicht-Konf(Z,C‘,S, $\lceil T/2 \rceil$)
    - falls r_1 und r_2 gilt, halte und akzeptiere
  - akzeptiere nicht.

Platz: s(n)+1 in jeder Rekrsionstiefe bei Anzahl der Rekursionstiefen log(T).