Technische Universität Braunschweig

Mathematische Methoden der Algorithmik – Vorlesung #06

Arne Schmidt

Letzte Woche

(Dualer) Simplex in Matrixform

Technischer Aufschrieb des Algorithmus.

Das lässt sich nun recht einfach implementieren.

Primal Simplex	Dual Simplex
Suppose $x^*_{\mathcal{B}} \geq 0$	Suppose $z^*_{\mathcal{N}} \geq 0$
while $(z^*_{\mathcal{N}} \not\geq 0)$ {	while $(x^*_{\mathcal{B}} \not\geq 0)$ {
pick $j \in \{j \in \mathcal{N} : z^*_j < 0\}$	pick $i \in \{i \in \mathcal{B} : x^*_i < 0\}$
$\Delta x_{\mathcal{B}} = B^{-1}Ne_j$	$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$
$t = \left(\max_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i} \right)^{-1}$	$s = \left(\max_{j \in \mathcal{N}} \frac{\Delta z_j}{z^*_j} \right)^{-1}$
pick $i \in \operatorname{argmax}_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i}$	pick $j \in \operatorname{argmax}_{j \in \mathcal{N}} \frac{\Delta z_j}{z^*_j}$
$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$	$\Delta x_{\mathcal{B}} = B^{-1}Ne_j$
$s = \frac{z^*_j}{\Delta z_j}$	$t = \frac{x^*_i}{\Delta x_i}$
$x^*_j \leftarrow t$	$x^*_j \leftarrow t$
$x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t\Delta x_{\mathcal{B}}$	$x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t\Delta x_{\mathcal{B}}$
$z^*_i \leftarrow s$	$z^*_i \leftarrow s$
$z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s\Delta z_{\mathcal{N}}$	$z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s\Delta z_{\mathcal{N}}$
$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$	$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$
}	}

Implementierung des Simplex-Algorithmus

Laufzeit

Das Dictionary hat m+1 Zeilen und n+1 Spalten.

In einer naiven Implementierung:

- Berechnung von j, t, i, s und die Berechnungen des Updates am Ende gehen in $O(n+m)$.
- Wie lange dauert das Berechnen von $\Delta x_{\mathcal{B}}$ bzw $\Delta z_{\mathcal{N}}$?
  - B ist quadratisch mit m Zeilen und Spalten.
  - B^{-1} ist quadratisch mit m Zeilen und Spalten.
    - Das müssen wir auch erst berechnen!
    - $\Rightarrow O(m^3)$ mit Gauss-Methode
  - N ist eine $m \times n$ Matrix.
  - $\Rightarrow O(m^3 + m^2 n)$ Berechnungen.
    Geht das besser?

Primal Simplex	Dual Simplex
Suppose $x^*_{\mathcal{B}} \geq 0$	Suppose $z^*_{\mathcal{N}} \geq 0$
while $(z^*_{\mathcal{N}} \ngeq 0)$ {	while $(x^*_{\mathcal{B}} \ngeq 0)$ {
pick $j \in \{j \in \mathcal{N} : z^*_j < 0\}$	pick $i \in \{i \in \mathcal{B} : x^*_i < 0\}$
$\Delta x_{\mathcal{B}} = B^{-1} N e_j$	$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$
$t = \left(\max_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i} \right)^{-1}$	$s = \left(\max_{j \in \mathcal{N}} \frac{\Delta z_j}{z^*_j} \right)^{-1}$
pick $i \in \operatorname{argmax}_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i}$	pick $j \in \operatorname{argmax}_{j \in \mathcal{N}} \frac{\Delta z_j}{z^*_j}$
$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$	$\Delta x_{\mathcal{B}} = B^{-1} N e_j$
$s = \frac{z^*_j}{\Delta z_j}$	$t = \frac{x^*_i}{\Delta x_i}$
$x^*_j \leftarrow t$	$x^*_j \leftarrow t$
$x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t \Delta x_{\mathcal{B}}$	$x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t \Delta x_{\mathcal{B}}$
$z^*_i \leftarrow s$	$z^*_i \leftarrow s$
$z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s \Delta z_{\mathcal{N}}$	$z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s \Delta z_{\mathcal{N}}$
$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$	$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$
}	}

Exkurs: Inverse von Matrizen

Eine $m \times m$-Matrix B hat genau dann eine inverse $m \times m$-Matrix B^{-1} (also $BB^{-1} = E$, wobei E eine $m \times m$ Einheitsmatrix ist), wenn $\mathrm{rg}(B) = m$.

Algorithmus	Laufzeit
Gauss-Jordan	$O(n^3)$
Strassen	$O(n^{2.807})$
Coppersmith-Winograd (CW)	$O(n^{2.376})$
Divide & Conquer	$O\left(n^{\omega+o(1)}\right), 2 \leq \omega < 2.37134$

Allerdings ist das explizite Berechnen der Inversen nicht immer notwendig.

Lösen von Subproblemen

$\Delta x_{\mathcal{B}} = B^{-1}Ne_j = B^{-1}a_j$ ist die Lösung zu $Bx = a_j$
$\Delta z_{\mathcal{N}} = -N^T v$, wobei v die Lösung zu $B^T v = e_j$ ist.

Lösen von Subproblemen

$\Delta x_{\mathcal{B}} = B^{-1}Ne_j = B^{-1}a_j$ ist die Lösung zu $Bx = a_j$
$\Delta z_{\mathcal{N}} = -N^T v$, wobei v die Lösung zu $B^T v = e_j$ ist.

Können wir die Matrix B so vorbearbeiten, dass...

... das Lösen des LGS schnell geht?

... die Datenstruktur bei einem Basiswechsel schnell aktualisiert werden kann?

Idee: LU-Zerlegung

Faktorisiere die $m \times m$-Matrix B in zwei $m \times m$-Matrizen L und U, sodass:

$$L = \underbrace{\begin{pmatrix} 1 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \ell_{m,1} & \cdots & \ell_{m,m-1} & 1 \end{pmatrix}}_{\text{Untere Dreiecksmatrix}}, U = \underbrace{\begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,m} \\ 0 & u_{2,2} & \cdots & u_{2,m} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{m,m} \end{pmatrix}}_{\text{Obere Dreiecksmatrix}}$$

Beispiel: Bestimme die LU-Zerlegung von

$$A = \begin{pmatrix} 2 & 4 & 1 \\ 4 & 9 & 3 \\ 1 & 3 & 2 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0.5 & 1 & 1 \end{pmatrix}, U = \begin{pmatrix} 2 & 4 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0.5 \end{pmatrix}$$

Wichtig: $B^T = (LU)^T = U^T L^T$

Berechnen von Gleichungssystemen mit LU

$\Delta x_{\mathcal{B}} = B^{-1}Ne_j = B^{-1}a_j$ ist die Lösung zu $Bx = LUx = a_j$

Sei $y \coloneqq Ux$, dann löse
- erst $Ly = a_j$,
- dann $Ux = y$

$\Delta z_{\mathcal{N}} = -N^T v$, wobei v die Lösung zu $B^T v = (LU)^T v = U^T L^T v = e_j$ ist.

Sei $y \coloneqq L^T v$, dann löse
- erst $U^T y = e_j$,
- dann $L^T v = y$

Beispiel I – LU-Zerlegung

$$B = \begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & & 1 & \\ -1 & & -1 & & -2 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1}}_{= L} \underbrace{\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}}_{= U}$$

Diese Matrizen könner innerhalb einer einzigen Matrix gespeichert werden während der Gauß-Elimination (siehe Vanderbei Kapitel 8).

$$L = \begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix}$$

Beispiel II – Lösung von LGS über LU-Zerlegung I

$$a_j = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Löse $LU\Delta x_{\mathcal{B}} = a_j$ über $Ly = a_j$ und $U\Delta x_{\mathcal{B}} = y$

$$\begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

Beispiel II – Lösung von LGS über LU-Zerlegung II

$$a_j = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Löse $LU\Delta x_{\mathcal{B}} = a_j$ über $Ly = a_j$ und $U\Delta x_{\mathcal{B}} = y$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

$$\begin{bmatrix} 2 & 4 & -2 & \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

$$\Delta x_{\mathcal{B}} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix}$$

Sparsity

$$\begin{pmatrix} 1 & 4 & 0 & 1 & 0 \\ 2 & 1 & 2 & 3 & 4 \\ 0 & 4 & 1 & 2 & 1 \\ 2 & 5 & 8 & 3 & 2 \\ 9 & 5 & 0 & 5 & 1 \end{pmatrix}$$

„Voll“ (dense)

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

„Dünn“ (sparse)

Generell: Je mehr 0en eine Matrix besitzt, desto **dünnbesetzter** (sparser) ist die Matrix.

Dünnbesetzte Matrizen lassen sich einfacher und schneller in eine LU-Zerlegung teilen, und benötigen deutlich weniger Speicher.

LU-Sparsity?

Können wir Spalten und Zeilen tauschen, um mehr Nullen in der LU-Zerlegung zu erhalten?

Dürfen wir das überhaupt?

Letztlich betrachten wir nur eine andere Reihenfolge von Variablen und Constraints!

LU-Zerlegung – Komplexität

Generell ist das Ziel, die L bzw U Matrix so dünnbesetzt wie möglich zu halten, d.h. die Zahl der Nicht-Nullen („fill-ins“) soll so klein wie möglich sein.

Problem: Das Minimieren der Fill-Ins ist NP-schwer.

Es existieren aber gute Heuristiken, bspw.:

1. Tausche die Zeile mit den meisten nicht-eliminierten Nullen mit der aktuellen Pivot-Zeile
2. Tausche die Spalte mit den meisten nicht-eliminierten Nullen mit der Pivot-Spalte.

Beispiel:

$$\begin{pmatrix} 0 & 4 & 1 & 0 & 0 \\ 2 & 1 & 0 & 3 & 2 \\ 0 & 1 & 0 & 2 & 1 \\ 2 & 2 & 2 & 3 & 2 \\ 1 & 2 & 0 & 2 & 1 \end{pmatrix}$$

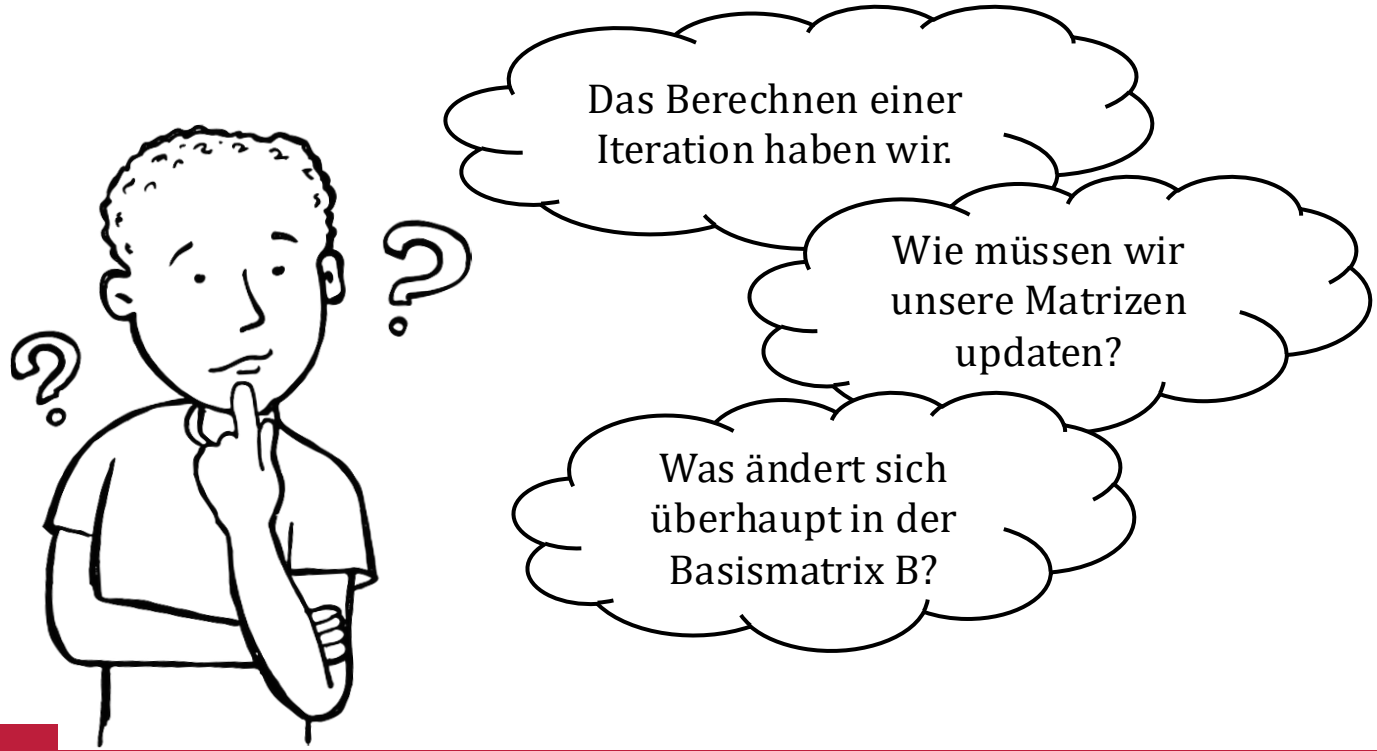
Beispiel – LU-Heuristik

$$\begin{pmatrix} 2 & 1 & 0 & 3 & 2 \\ 0 & 4 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \\ 2 & 2 & 2 & 0 & 2 \\ 1 & 2 & 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 2 \\ 0 & 1 & 0 & 2 & 0 \\ 2 & 2 & 2 & 0 & 2 \\ 0 & 2 & 1 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 \\ 0 & 3 & 2 & 1 & 2 \\ 2 & 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 1 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{3}{2} & 1 & 0 & 0 \\ 2 & -4 & 1 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{9}{26} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 2 & -\frac{1}{2} & 2 \\ 0 & 0 & 0 & \frac{13}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

LU-Zerlegung besitzt 11 Nullen.
Zerlegung über ursprüngliche Matrix hätte nur 8 Nullen.

Aktualisieren nach Pivotschritt



Basismatrix aktualisieren

Bei einem Pivotschritt (Basistausch) kann sich nur eine Zeile in B ändern, d.h.

$$B_{new} = B + (a_j - a_i)e_i^T = B\underbrace{\left(I + B^{-1}(a_j - a_i)e_i^T\right)}_{E :=}.$$

Wir kennen dabei $B^{-1}a_j = \Delta x_{\mathcal{B}}$ und $B^{-1}a_i = e_i$. Also ist:

$$E = \left(I + (\Delta x_{\mathcal{B}} - e_i)e_i^T\right)$$

Man kann nun zeigen, dass:

$$E^{-1} = I - \frac{(\Delta x_{\mathcal{B}} - e_i)e_i^T}{\Delta x_i}$$

Nächste Iteration

Für die nächste Iteration muss dann bspw. berechnet werden:

$$B_{new}\Delta\tilde{x}_{\mathcal{B}} = BE\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j \text{ und } B_{new}^T\tilde{v} = E^T B^T\tilde{v} = \tilde{e}_i$$

Berechne diese wieder in Schritten

- $Bu = \tilde{a}_j$
- $E\Delta\tilde{x}_{\mathcal{B}} = u \rightarrow \Delta\tilde{x}_{\mathcal{B}} = E^{-1}u = u - \frac{u_i}{\Delta x_i}(\Delta x_{\mathcal{B}} - e_i)$

Sowie

- $E^T y = \tilde{e}_i \rightarrow y = (E^{-1})^T\tilde{e}_i = \tilde{e}_i - e_i\frac{(\Delta x_{\mathcal{B}} - e_i)^T\tilde{e}_i}{\Delta x_i}$
- $B^T\tilde{v} = y$

Viele Pivotschritte

Wenn wir k Iterationen durchgeführt haben, können wir den nächsten Schritt über die erste Basis berechnen

Dann ist $B_{new} = BE_0E_1 \dots E_{k-1}$

Für B ist die LU-Zerlegung bekannt und die E-Matrizen sind sehr einfach.

Wenn wir B_{new} nicht explizit berechnen, dauert das ggf. irgendwann sehr lange.

Überlegungen für viele Iterationen

Also, nach k Iterationen müssen wir folgende Informationen speichern:

- Basismatrix B
- „Eta-Matrizen“ $E_0, \dots, E_{k-1}$ (dafür reicht es, $\Delta x_{\mathcal{B}}^{j}$ und einen Integer für die Spalte zu merken, die sogenannte eta-file)

Wann ist der Aufwand $BE_0 \dots E_{k-1}$ zu berechnen zu hoch?
Wann lohnt es sich B_{new} in LU zu zerlegen?

Eine sorgfältige Analyse zeigt, dass nach etwa $\sqrt{m}$ Iterationen eine neue Basismatrix explizit berechnet und zerlegt werden sollte.

In der Praxis ist die Anzahl dieser Iterationen sogar frei wählbar!

Partial Pricing

Viele Variablen, wenig Constraints

Viele Variablen erzeugen viel Arbeit:

Welches Pivot als nächstes?

Man muss die Länge des Pivotschritts berechnen.

Man muss reduzierte Kosten berechnen.

Müssen wir immer alle Variablen anschauen?

Interessant sind doch nur die „guten“ Variablen.

Partial Pricing

Idee: Betrachte zu jedem Zeitpunkt nur einen Teil der Variablen und aktualisiere das gesamte Dictionary nur ab und an.

Dazu:

- Suche unter den ersten $\frac{n}{3}$ Variablen die 40 Variablen, welche den größten negativen Koeffizienten besitzen.
- Betrachte nur diese Variablen für die Iterationen bis nur noch ein Teil dieser (bspw, weniger als die Hälfte) für einen Pivotschritt in Frage kommen.
- Aktualisiere dann das gesamte Dictionary.

Damit müssen wir in jeder Iteration nur mit wenig Variablen arbeiten. Nur ab und an trifft uns die Härte der Realität.

Steepest Edge

Wir gehen steil!

Bisher oft nach größtem Koeffizienten ausgewählt. Das liefert die größte Änderungsrate der Zielfunktion und betrachtet nur die Nichtbasisvariablen.

Schließe die Basisvariablen mit ein! Da die Zielfunktion in der Form

$$f(x) = c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}}$$

ist, ist die Ableitung von $f(x)$ in Richtung Δx für ein j:

$$\frac{\partial f}{\partial \Delta x} = \frac{c_{\mathcal{B}}^T \Delta x_{\mathcal{B}} + c_{\mathcal{N}}^T \Delta x_{\mathcal{N}}}{\|\Delta x\|} = \frac{-z_j^*}{\sqrt{\left\|B^{-1}Ne_j\right\|^2 + 1}}$$

Okay, der Nenner sieht nicht gut aus. Zu Beginn ist B aber quasi eine Einheitsmatrix. Dadurch können wir

$$\nu_k = \|B^{-1}Ne_k\|^2 \text{ für alle } k \in \mathcal{N}$$

einfach berechnen.

Aktualisiere ν_k

Wir können herleiten, wie sich die ν_k in der nächsten Iteration verändern.
Dazu sei

$$u \coloneqq (B^{-1})^T e_i$$
$$w \coloneqq (B^{-1})^T \Delta x_{\mathcal{B}}$$

u berechnen wir sowieso schon. w muss zusätzlich berechnet werden. Damit können wir folgendes berechnen:

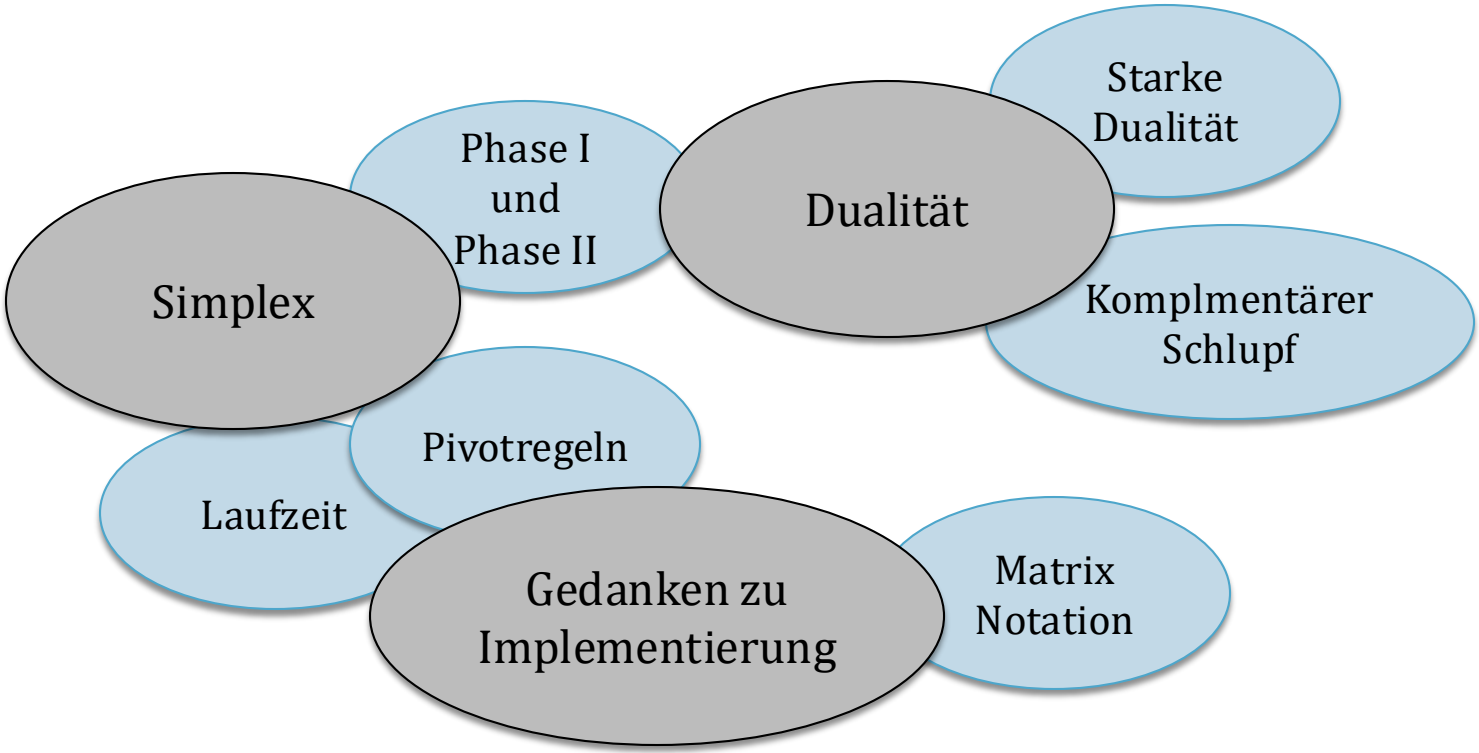
$$\tilde{\nu}_k = \nu_k - 2\frac{a_k^T u (w-u)^T a_k}{\Delta x_i} + \left(a_k^T u\right)^2 \frac{\|\Delta x_{\mathcal{B}} - e_i\|^2}{(\Delta x_{\mathcal{B}})^2}$$

Wähle damit $j \in \mathcal{N}$, sodass $\frac{-z_j^*}{\sqrt{\nu_j+1}}$ kleinstmöglich ist.

Die Steepest-Edge-Regel performt in der Praxis sehr gut mit den angegebenen Formeln.

Recap

Was bisher geschah...



... und was noch kommt

Bisher nur betrachtet: $Ax \leq b$.
Was ist, wenn wir LPs der Form $b_\ell \leq Ax \leq b_u$ haben?

Bisher nur reelle Lösungen angeschaut, aber viele Probleme fordern ganzzahlige Lösungen!

Wie unterschiedlich sind Probleme auf diskreten Strukturen (bspw. Graphen) oder in geometrischem Setting?