

Projektabschlussvortrag Optimale Containment Probleme (Handouts)

Algorithmen zur Berechnung der Dicke und des Zylinderradius

Manuel Mayr Tobias Braunschöber

Technische Universität München

4. Juli 2006



Inhaltsverzeichnis

- 1 Hintergrund
 - Definitionen
 - Der Zylinderradius
 - Die Dicke
- 2 Keine SOCP
- 3 Semidefinite Programmierung
- 4 Zylinderradius in \mathbb{R}^∞
 - Theorem von Megiddo
 - Vorgehensweise (Binäre Suche)
- 5 Dicke
- 6 Codeoptimierung
 - Matrixmultiplikation vs Iterationen
 - A priori Reservierung von Speicher
 - C in Matlab einbetten
 - Prozessorcache
- 7 Testläufe



Der äußere j -Radius (R_j)

Zunächst ein paar grundlegende Definitionen

Definition (j -Zylinder)

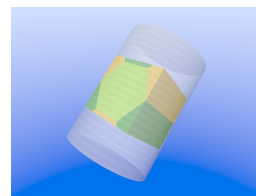
Sei $F \subset \mathbb{R}^d$ ein $(d - j)$ -dimensionaler Unterraum und $\rho \geq 0$, so heißt $F + \rho\mathbb{B} = F + \rho(\mathbb{B} \cap F)$ j -Zylinder.

Definition (Der äußere j -Radius (R_j))

Sei $K \subset \mathbb{R}^d$ und $1 \leq j \leq \dim(K)$, dann ist der äußere j -Radius das Minimum aller j -Zylinder, die K enthalten.



Der Zylinderradius

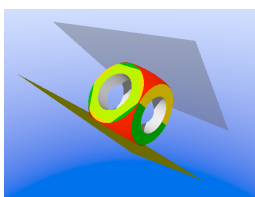


Definition (Zylinderradius (R_{d-1}))

Der äußere j -Radius, für $j = d - 1$ wird Zylinderradius genannt.



Die Dicke



Definition (Dicke (R_1))

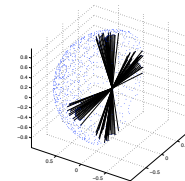
Der äußere j -Radius, für $j = 1$ wird Dicke genannt.



Alternativer Ansatz

Diplomarbeit von Simon Rittsteiger

`mincyl_mg`: Löse für jeden Richtungsvektor ein SOCP



Alternativer Ansatz

Diplomarbeit von Simon Rittsteiger

`mincyl_mg`: Löse für jeden Richtungsvektor ein SOCP

Problem

Nahezu die gesamte Zeit wird für das Lösen der SOCPs verbraucht.

Alternative

Anderes Verfahren ohne SOCPs zu lösen: `mincyl_mg_noSOCP`



Konzept von noSOCP

Algorithmus

- 1 Wähle ein $v \in \mathbb{R}^d$ aus Kugelapproximation.
- 2 Projiziere P entlang $v \in \mathbb{R}^d$ auf den orthogonalen $(d - 1)$ -dimensionalen Unterraum.
- 3 Berechne die kleinste umschließende Umkugel (Approximation).
- 4 Obere Schranke für den optimalen Radius.
- 5 Kugelapproximation abgearbeitet? Verfeinere lokal (adaptive).



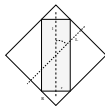
ϵ -Approximation

Güte

Die Güte der Approximation hängt neben dem Winkel der Kugelapproximation von den Ausmaßen von P ab.

Anfangsdaten

Für den Winkel θ für die Kugelapproximation gilt: $\theta = \epsilon \cdot \frac{2r}{\text{diam}(P)}$



Schranken für die Anfangsdaten

Schranken für die Anfangsdaten

- $\text{quotient} = \frac{2r}{\text{diam}(P)} \geq \frac{2r}{\text{diam}'(P)} \geq \frac{2r'}{\text{diam}'(P)}$
- $r' \leq r$: untere Schranke für den Zylinderradius
- $\text{diam}'(P) \geq \text{diam}(P)$: obere Schranke für den Durchmesser von P

Kugelapproximation

Berechne aus *quotient* die erste Kugelapproximation.
 $\theta = \epsilon \cdot \text{quotient}$



Untere Schranken mit SDP

Wozu SDP?

Durch die Relaxierung untere Schranke für Zylinderradius.

Lokale untere Schranken

- Durch Modifikation sehr gute lokale untere Schranken.
- Beschränkung des SDPs auf einzelne Kugelkappen.

Achtung! Leider nur für verankerten Zylinder

Ein Punkt auf der optimalen Zylinderachse muss bekannt sein.



Untere Schranken mit SDP

Wozu SDP?

Durch die Relaxierung untere Schranke für Zylinderradius.

Lokale untere Schranken

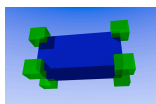
- Durch Modifikation sehr gute lokale untere Schranken.
- Beschränkung des SDPs auf einzelne Kugelkappen.

Die gute Nachricht

Praktisch relevant für die medizinische Berechnungen.



Megiddo



Definition (Boxen)

Sei $a, b \in \mathbb{R}^d$, so ist eine Box
 $B(a, b) = \{x | a \leq x \leq b\}$

Theorem (Megiddo)

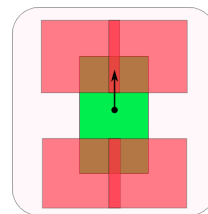
- Mit Megiddo wird überprüft, ob es eine Linie gibt, die alle Boxen $B^1, \dots, B^n \subset \mathbb{R}^n$ schneidet.
- Laufzeit linear bei fester Dimension.



Zylinderradius in ℓ_∞

Vorgehensweise (Binäre Suche)

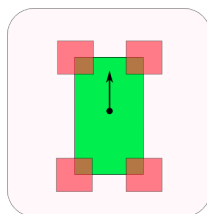
1. Gegeben sei ein Intervall $I = [us, os]$, mit us untere Schranke, os obere Schranke für den Radius.
2. Bestimme die Mitte m von I .
3. Lege um jeden Punkt $p \in P$ eine Box mit Radius m .
4. Prüfe mit Megiddo, ob es eine Linie gibt, die alle Boxen schneidet.
 - Wenn JA: $os \leftarrow m$
 - Wenn NEIN: $us \leftarrow m$
5. Gehe zu 2 ...



Zylinderradius in ℓ_∞

Vorgehensweise (Binäre Suche)

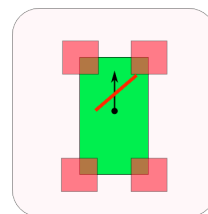
1. Gegeben sei ein Intervall $I = [us, os]$, mit us untere Schranke, os obere Schranke für den Radius.
2. Bestimme die Mitte m von I .
3. Lege um jeden Punkt $p \in P$ eine Box mit Radius m .
4. Prüfe mit Megiddo, ob es eine Linie gibt, die alle Boxen schneidet.
 - Wenn JA: $os \leftarrow m$
 - Wenn NEIN: $us \leftarrow m$
5. Gehe zu 2 ...



Zylinderradius in ℓ_∞

Vorgehensweise (Binäre Suche)

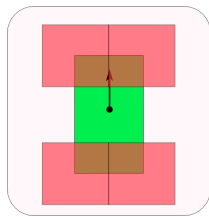
1. Gegeben sei ein Intervall $I = [us, os]$, mit us untere Schranke, os obere Schranke für den Radius.
2. Bestimme die Mitte m von I .
3. Lege um jeden Punkt $p \in P$ eine Box mit Radius m .
4. Prüfe mit Megiddo, ob es eine Linie gibt, die alle Boxen schneidet.
 - Wenn JA: $os \leftarrow m$
 - Wenn NEIN: $us \leftarrow m$
5. Gehe zu 2 ...



Zylinderradius in l_∞

Vorgehensweise (Binäre Suche)

- Gegeben sei ein Intervall $I = [us, os]$, mit us untere Schranke, os obere Schranke für den Radius.
- Bestimme die Mitte m von I .
- Lege um jeden Punkt $p \in P$ eine Box mit Radius m .
- Prüfe mit Megiddo, ob es eine Linie gibt, die alle Boxen schneidet.
 - Wenn JA: $os \leftarrow m$
 - Wenn NEIN: $us \leftarrow m$
- Gehe zu 2 ...



Dicke adaptiv

Dicke nun adaptiv

- Selbe Idee wie bei Zylinderberechnung.
- Löse für jeden Richtungsvektor ein LP.

$$\delta^* = \frac{\min(b_1 - b_2)}{\|x\|}$$

$$\langle x, p \rangle \leq b_1 \quad \forall p \in P$$

$$\langle x, p \rangle \geq b_2 \quad \forall p \in P$$

$$b_1 - b_2 \geq 0$$

$$x \in \mathbb{R}^d$$

$$b_1, b_2 \in \mathbb{R}$$



Matrixmultiplikationen

Regel (Matrixmultiplikationen vs Iterationen)

Matrixmultiplikationen sind Iterationen vorzuziehen.

Example (Maximaler Abstand)

Wir wollen den maximalen Abstand von einem Punkt $p \in P$ zu einem anderen in P berechnen (in Formeln: $\max_{p \in P} \|p - \hat{p}\|$).

- Subtrahiere von jedem Punkt in P den Punkt p

$$(X \leftarrow P - \begin{pmatrix} p \\ \vdots \\ p \end{pmatrix} = \begin{pmatrix} (1, \dots, 1) \\ \vdots \\ (1, \dots, 1) \end{pmatrix})$$

Zeilenvektor mit $|P|$ -Spalten

- Matrixmultiplikation ($\hat{X} \leftarrow X^T X$)
- Entnehme das maximale Element aus der Diagonale der entstandenen Matrix ($n \leftarrow \sqrt{\max\{\text{diag } \hat{X}\}}$)



A priori Reservierung von Speicher

Arrays

Wir betrachten nur den einfachsten Fall, nämlich **Arrays**.

Amortisierte worst-case-Analyse

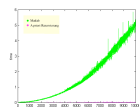
- Technik zum Analysieren von Algorithmen.
- Wir betrachten den worst-case einer Operation auf eine Datenstruktur in einer Folge von Operationen.
- Realistischeres Laufzeitverhalten.



Statische Datenstrukturen

Regel (A priori Reservierung)

Ist der Speicher einer Datenstruktur schon vorab bekannt und ändert sich nicht mehr, so ist der Speicher im voraus zu reservieren.



Vorteile

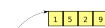
- Unter Umständen sehr große Laufzeiteinbußen wettmachen.
- Matlab operiert beim sukzessiven Hinzufügen von Elementen mit quadratischer Laufzeit.



Dynamische Datenstrukturen

Unterschied zu statischen Datenstrukturen

Bei dynamischen Datenstrukturen ist die Größe vorab nicht bekannt \Rightarrow sukzessives Hinzufügen von Elementen



Wir können das besser!

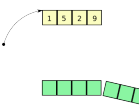
- Wenn Array voll, kreierte ein neues Array mit doppeltem Speicher.
- Kopiere alle Elemente vom alten Array in das neue Array
- Setze Zeiger auf neues Array.
- Amortisiert $O(n)$ für n Einfügeoperationen.



Dynamische Datenstrukturen

Unterschied zu statischen Datenstrukturen

Bei dynamischen Datenstrukturen ist die Größe vorab nicht bekannt \Rightarrow sukzessives Hinzufügen von Elementen



Wir können das besser!

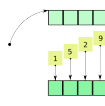
- Wenn Array voll, kreierte ein neues Array mit doppeltem Speicher.
- Kopiere alle Elemente vom alten Array in das neue Array
- Setze Zeiger auf neues Array.
- Amortisiert $O(n)$ für n Einfügeoperationen.



Dynamische Datenstrukturen

Unterschied zu statischen Datenstrukturen

Bei dynamischen Datenstrukturen ist die Größe vorab nicht bekannt \Rightarrow sukzessives Hinzufügen von Elementen



Wir können das besser!

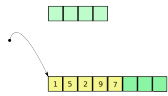
- Wenn Array voll, kreierte ein neues Array mit doppeltem Speicher.
- Kopiere alle Elemente vom alten Array in das neue Array
- Setze Zeiger auf neues Array.
- Amortisiert $O(n)$ für n Einfügeoperationen.



Dynamische Datenstrukturen

Unterschied zu statischen Datenstrukturen

Bei dynamischen Datenstrukturen ist die Größe vorab nicht bekannt
⇒ sukzessives Hinzufügen von Elementen



Wir können das besser!

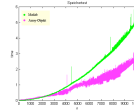
- 1 Wenn Array voll, kreierte ein neues Array mit doppeltem Speicher.
- 2 Kopiere alle Elemente vom alten Array in das neue Array
- 3 Setze Zeiger auf neues Array.
- 4 Amortisiert $O(n)$ für n Einfügeoperationen.



Dynamische Datenstrukturen

Unterschied zu statischen Datenstrukturen

Bei dynamischen Datenstrukturen ist die Größe vorab nicht bekannt
⇒ sukzessives Hinzufügen von Elementen



Wir können das besser!

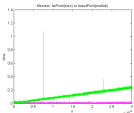
- 1 Wenn Array voll, kreierte ein neues Array mit doppeltem Speicher.
- 2 Kopiere alle Elemente vom alten Array in das neue Array
- 3 Setze Zeiger auf neues Array.
- 4 Amortisiert $O(n)$ für n Einfügeoperationen.



Die MEX-Schnittstelle

C in Matlab

Die MEX-Schnittstelle in Matlab bietet die Möglichkeit C-Code einzubetten.



Vorteile

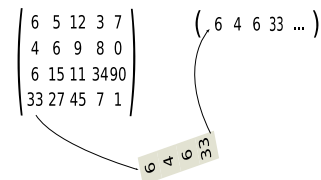
- 1 Effiziente Speicherverwaltung, allerdings fehleranfällig.
- 2 Schleifen sind sehr schnell.



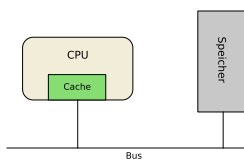
Vektorisieren einer Matrix

Matrizen in Matlab

Matrizen werden spaltenweise vektorisiert.



Vorteile des Prozessorcaches



Der Prozessorcache

- Prozessor lädt Blöcke in den Prozessorcache.
- ⇒ Speicher auf den als nächstes zugegriffen wird ist wahrscheinlich schon im Cache.

Fazit

- Sprünge bei Iterationen durch Arrays vermeiden.
- Bei Matrizen, wenn möglich durch Spalten iterieren.



Tests

Testinstanzen

- Reguläre Simplexe, gestreckte Einheitswürfel und Zylinder.
- in 2D, 3D, 4D
- Als Test wie sich die Algorithmen bei vielen Punkten verhalten wurde auch noch in 2D ein mit 100,000 Punkten aufgefüllter gestreckter Einheitswürfel getestet.



Ergebnisse – Zylinderradius – 2D

Gestreckter verankerter Einheitswürfel mit 4 Punkten

Mit SDP wird sofort eine Genauigkeit $\epsilon = 10^{-11}$ erreicht. Verbesserung der Laufzeit um Faktor 2 auf 1.16 Sekunden.

Gestreckter verankerter Einheitswürfel mit 100004 Punkten

$\epsilon = 0.1$

- SOCP: 463.80 Sekunden
- noSOCP: 71.04 Sekunden



Ergebnisse – Zylinderradius – 3D

Gestreckter verankerter Einheitswürfel mit 8 Punkten

Mit SDP wird sofort eine Genauigkeit $\epsilon = 10^{-5}$ erreicht.

- mit SDP

ϵ	Zeit in Sekunden
10^{-5}	12.69

- ohne SDP

ϵ	Zeit in Sekunden
0.1	14.55
0.01	65.81



Ergebnisse – Zylinderradius – 3D

Zylinder mit 22 Punkten

Mit SDP wird sofort eine Genauigkeit $\epsilon = 10^{-5}$ erreicht.

- mit SDP

ϵ	Zeit in Sekunden
10^{-5}	7.32

- ohne SDP

ϵ	Zeit in Sekunden
0.1	6.61
0.01	36.94
0.001	165.92

Ergebnisse – Dicke – 2D

Gestreckter Einheitswürfel mit 4 Punkten

- adaptiv

ϵ	Zeit in Sekunden
0.1	0.13
0.0001	0.34

- nicht-adaptiv

ϵ	Zeit in Sekunden
0.1	0.17
0.0001	3.84

Ergebnisse – Dicke – 3D

Gestreckter Einheitswürfel mit 8 Punkten

- adaptiv

ϵ	Zeit in Sekunden
0.1	1.61
0.01	6.27
0.001	30.03

- nicht-adaptiv

ϵ	Zeit in Sekunden
0.1	8.92
0.01	102.44

Ergebnisse – Dicke – 4D

Reguläres Simplex mit 5 Punkten

Zeit adaptiv für $\epsilon = 0.1$: 263.08 Sekunden