

Algorithmen für Chip-Entwurfswerkzeuge

Mathematische Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorlesung
WS 2018/2019

Florian Stock

Eingebette Systeme und Anwendungen
Technische Universität Darmstadt



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)
- ▶ Exakt lösbar nur für kleine Problemgrößen



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)
- ▶ Exakt lösbar nur für kleine Problemgrößen
- ▶ Falls suboptimale Lösungen akzeptabel:



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)
- ▶ Exakt lösbar nur für kleine Problemgrößen
- ▶ Falls suboptimale Lösungen akzeptabel:
 - ▶ Näherungsverfahren, Approximationen



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)
- ▶ Exakt lösbar nur für kleine Problemgrößen
- ▶ Falls suboptimale Lösungen akzeptabel:
 - ▶ Näherungsverfahren, Approximationen
 - ▶ Garantieren eine bestimmte Lösungsqualität



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)
- ▶ Exakt lösbar nur für kleine Problemgrößen
- ▶ Falls suboptimale Lösungen akzeptabel:
 - ▶ Näherungsverfahren, Approximationen
 - ▶ Garantieren eine bestimmte Lösungsqualität
 - ▶ Nicht allgemein formulierbar



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)
- ▶ Exakt lösbar nur für kleine Problemgrößen
- ▶ Falls suboptimale Lösungen akzeptabel:
 - ▶ Näherungsverfahren, Approximationen
 - ▶ Garantieren eine bestimmte Lösungsqualität
 - ▶ Nicht allgemein formulierbar
 - ▶ Heuristiken



- ▶ Alle Probleme im Bereich VLSI CAD sind Optimierungsprobleme, viele davon sind
 - ▶ NP-vollständig
 - ▶ NP-hart (mindestens so aufwendig wie NP-vollständig)
- ▶ Exakt lösbar nur für kleine Problemgrößen
- ▶ Falls suboptimale Lösungen akzeptabel:
 - ▶ Näherungsverfahren, Approximationen
 - ▶ Garantieren eine bestimmte Lösungsqualität
 - ▶ Nicht allgemein formulierbar
 - ▶ Heuristiken
 - ▶ Schwankende Lösungsqualität



- ▶ Gesamtverbindungslänge



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele
- ▶ Verdrahtungsdichte / Verdrahtbarkeit



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele
- ▶ Verdrahtungsdichte / Verdrahtbarkeit
- ▶ Signalverzögerung



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele
- ▶ Verdrahtungsdichte / Verdrahtbarkeit
- ▶ Signalverzögerung
- ▶ Energieverbrauch



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele
- ▶ Verdrahtungsdichte / Verdrahtbarkeit
- ▶ Signalverzögerung
- ▶ Energieverbrauch
- ▶ Thermisches Verhalten



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele
- ▶ Verdrahtungsdichte / Verdrahtbarkeit
- ▶ Signalverzögerung
- ▶ Energieverbrauch
- ▶ Thermisches Verhalten



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele
- ▶ Verdrahtungsdichte / Verdrahtbarkeit
- ▶ Signalverzögerung
- ▶ Energieverbrauch
- ▶ Thermisches Verhalten

⇒ Zielfunktion (objective function) (Funktion die max./min. werden soll)
entsprechend wählen



- ▶ Gesamtverbindungslänge
 - ▶ Optimiert indirekt auch andere Ziele
- ▶ Verdrahtungsdichte / Verdrahtbarkeit
- ▶ Signalverzögerung
- ▶ Energieverbrauch
- ▶ Thermisches Verhalten

⇒ Zielfunktion (objective function) (Funktion die max./min. werden soll)
entsprechend wählen

Am gebräuchlichsten:

(Gewichtete) Gesamtverbindungslänge



► Wozu?



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen



▶ Wozu?

- ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
- ▶ Analysiere fertige Layouts
- ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang
- ▶ Worauf?



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang
- ▶ Worauf?
 - ▶ Schaltungselement



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang
- ▶ Worauf?
 - ▶ Schaltungselement
 - ▶ Gatter, Werttabellen (LUT), Register, I/O-Blöcke, ...



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang
- ▶ Worauf?
 - ▶ Schaltungselement
 - ▶ Gatter, Werttabellen (LUT), Register, I/O-Blöcke, ...
 - ▶ Bleiben konstant, exakte Verzögerungen bekannt



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang
- ▶ Worauf?
 - ▶ Schaltungselement
 - ▶ Gatter, Werttabellen (LUT), Register, I/O-Blöcke, . . .
 - ▶ Bleiben konstant, exakte Verzögerungen bekannt
 - ▶ Netze



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang
- ▶ Worauf?
 - ▶ Schaltungselement
 - ▶ Gatter, Werttabellen (LUT), Register, I/O-Blöcke, ...
 - ▶ Bleiben konstant, exakte Verzögerungen bekannt
 - ▶ Netze
 - ▶ Nur nach Layouterzeugung bekannt



- ▶ Wozu?
 - ▶ Zielfunktion für Optimierung? **Nein!**
Häufige Auswertung zu teuer
 - ▶ Analysiere fertige Layouts
 - ▶ Analysiere einzelne Verbindungen *während* Layouterzeugung
 - ▶ Erkenne kritische Verbindungen
 - ▶ Behandle diese mit Vorrang
- ▶ Worauf?
 - ▶ Schaltungselement
 - ▶ Gatter, Werttabellen (LUT), Register, I/O-Blöcke, ...
 - ▶ Bleiben konstant, exakte Verzögerungen bekannt
 - ▶ Netze
 - ▶ Nur nach Layouterzeugung bekannt
 - ▶ Vorher schätzen



- ▶ Ankunftszeit (arrival) an Knoten v
 $T_a(v) = \max_{(u,v) \in E} (T_a(u) + w(u, v))$



- ▶ Ankunftszeit (arrival) an Knoten v
$$T_a(v) = \max_{(u,v) \in E} (T_a(u) + w(u, v))$$
- ▶ Idee: BFS oder DAG LP



- ▶ Ankunftszeit (arrival) an Knoten v

$$T_a(v) = \max_{(u,v) \in E} (T_a(u) + w(u, v))$$

- ▶ Idee: BFS oder DAG LP

- ▶ Beginne mit $T_a(v) = 0$ bei folgenden Knoten:
Externer Eingänge, Registerausgänge



- ▶ Ankunftszeit (arrival) an Knoten v

$$T_a(v) = \max_{(u,v) \in E} (T_a(u) + w(u, v))$$

- ▶ Idee: BFS oder DAG LP

- ▶ Beginne mit $T_a(v) = 0$ bei folgenden Knoten:
Externer Eingänge, Registerausgänge
- ▶ Bearbeite Knoten bei denen alle Vorgänger bearbeitet sind



- ▶ Ankunftszeit (arrival) an Knoten v

$$T_a(v) = \max_{(u,v) \in E} (T_a(u) + w(u, v))$$

- ▶ Idee: BFS oder DAG LP

- ▶ Beginne mit $T_a(v) = 0$ bei folgenden Knoten:
Externer Eingänge, Registerausgänge
- ▶ Bearbeite Knoten bei denen alle Vorgänger bearbeitet sind
- ▶ Späteste Gesamtankunftszeit $D_{max} =$ Taktperiode
(an Externen Ausgängen, Registereingängen)



- ▶ Ankunftszeit (arrival) an Knoten v

$$T_a(v) = \max_{(u,v) \in E} (T_a(u) + w(u, v))$$

- ▶ Idee: BFS oder DAG LP

- ▶ Beginne mit $T_a(v) = 0$ bei folgenden Knoten:
Externer Eingänge, Registerausgänge
- ▶ Bearbeite Knoten bei denen alle Vorgänger bearbeitet sind
- ▶ Späteste Gesamtankunftszeit $D_{max} =$ Taktperiode
(an Externen Ausgängen, Registereingängen)

In Beispielschaltung

$$D_{max} = 13.6 \text{ ns}$$



- ▶ Wie unwichtig sind unkritische Netze?



- ▶ Wie unwichtig sind unkritische Netze?
 - ▶ Hier auf Zeitintervalle anwenden



- ▶ Wie unwichtig sind unkritische Netze?
 - ▶ Hier auf Zeitintervalle anwenden

Beantwortet die Frage

Wieviel langsamer kann ein Netz werden ohne das die gesamte Schaltung leidet?



- ▶ Wie unwichtig sind unkritische Netze?
 - ▶ Hier auf Zeitintervalle anwenden

Beantwortet die Frage

Wieviel langsamer kann ein Netz werden ohne das die gesamte Schaltung leidet?

- ▶ Berechnung



- ▶ Wie unwichtig sind unkritische Netze?
 - ▶ Hier auf Zeitintervalle anwenden

Beantwortet die Frage

Wieviel langsamer kann ein Netz werden ohne das die gesamte Schaltung leidet?

- ▶ Berechnung
 - ▶ Mittels spätestmöglicher Ankunftszeit



- ▶ Wie unwichtig sind unkritische Netze?
 - ▶ Hier auf Zeitintervalle anwenden

Beantwortet die Frage

Wieviel langsamer kann ein Netz werden ohne das die gesamte Schaltung leidet?

- ▶ Berechnung
 - ▶ Mittels spätestmöglicher Ankunftszeit
 - ▶ Required Time $T_r(v)$ an Knoten v



- ▶ Wie unwichtig sind unkritische Netze?
 - ▶ Hier auf Zeitintervalle anwenden

Beantwortet die Frage

Wieviel langsamer kann ein Netz werden ohne das die gesamte Schaltung leidet?

- ▶ Berechnung
 - ▶ Mittels spätestmöglicher Ankunftszeit
 - ▶ Required Time $T_r(v)$ an Knoten v
 - ▶ Spätestmöglicher Ankunftszeitpunkt von Signalen
 - ⇒ **Sonst Verlangsamung der gesamten Schaltung**

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge
- ▶ Nun BFS/LP Rückwärts

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge
- ▶ Nun BFS/LP Rückwärts
- ▶ Rückwärts \Rightarrow Graph mit umgedrehten Kanten

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge
- ▶ Nun BFS/LP Rückwärts
- ▶ Rückwärts \Rightarrow Graph mit umgedrehten Kanten
- ▶ Bearbeite Knoten

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge
- ▶ Nun BFS/LP Rückwärts
- ▶ Rückwärts \Rightarrow Graph mit umgedrehten Kanten
- ▶ Bearbeite Knoten
 - ▶ Nur mit komplett bearbeiteten Vorgängern

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge
- ▶ Nun BFS/LP Rückwärts
- ▶ Rückwärts \Rightarrow Graph mit umgedrehten Kanten
- ▶ Bearbeite Knoten
 - ▶ Nur mit komplett bearbeiteten Vorgängern
 - ▶ $T_r(u) = \min_{(u,v) \in E} (T_r(v) - w(u, v))$

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge
- ▶ Nun BFS/LP Rückwärts
- ▶ Rückwärts \Rightarrow Graph mit umgedrehten Kanten
- ▶ Bearbeite Knoten
 - ▶ Nur mit komplett bearbeiteten Vorgängern
 - ▶ $T_r(u) = \min_{(u,v) \in E} (T_r(v) - w(u, v))$
- ▶ *Slack (Schlupf)* einer Verbindung von u nach v
 $slack(u, v) = T_r(v) - T_a(u) - w(u, v)$

Berechnung Spätestmögl. Ankunftszeit und Slack

- ▶ Beginne mit $T_r(u) = D_{max}$ bei folgenden Knoten:
Externen Ausgänge, Registereingänge
- ▶ Nun BFS/LP Rückwärts
- ▶ Rückwärts \Rightarrow Graph mit umgedrehten Kanten
- ▶ Bearbeite Knoten
 - ▶ Nur mit komplett bearbeiteten Vorgängern
 - ▶ $T_r(u) = \min_{(u,v) \in E} (T_r(v) - w(u, v))$
- ▶ *Slack (Schlupf)* einer Verbindung von u nach v
 $slack(u, v) = T_r(v) - T_a(u) - w(u, v)$
- ▶ Auf kritischem Pfad: $slack = 0$

Beispiel



Node: 4 INPAD_SOURCE Block #2 (s27_in_3_)
T_arr: 0 T_req: -3.88578e-16 Tdel: 5e-10

Node: 5 INPAD_OPIN Block #2 (s27_in_3_)
Pin: 0
T_arr: 5e-10 T_req: 5e-10 Tdel: 5e-09
Net to next node: #2 (s27_in_3_). Pins on net: 5.

Node: 12 CLB_IPIN Block #6 (s27_out)
Pin: 0
T_arr: 5.5e-09 T_req: 5.5e-09 Tdel: 0

Node: 17 SUBBLK_IPIN Block #6 (s27_out)
Pin: 0 Subblock #0
T_arr: 5.5e-09 T_req: 5.5e-09 Tdel: 9e-10

Node: 21 SUBBLK_OPIN Block #6 (s27_out)
Pin: 4 Subblock #0
T_arr: 6.4e-09 T_req: 6.4e-09 Tdel: 0

Node: 16 CLB_OPIN Block #6 (s27_out)
Pin: 4
T_arr: 6.4e-09 T_req: 6.4e-09 Tdel: 1e-09
Net to next node: #5 (s27_out). Pins on net: 2.

Node: 10 OUTPAD_IPIN Block #5 (out:s27_out)
Pin: 0
T_arr: 7.4e-09 T_req: 7.4e-09 Tdel: 3e-10

Node: 11 OUTPAD_SINK Block #5 (out:s27_out)
T_arr: 7.7e-09 T_req: 7.7e-09

Tnodes on crit. path: 8 Non-global nets on crit. path: 2.
Global nets on crit. path: 0.
Total logic delay: 1.7e-09 (s) Total net delay: 6e-09 (s)

Gesamtverbindungslänge

Exakte Kosten



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Häufigstes Optimierungsziel in der Praxis

Gesamtverbindungslänge

Exakte Kosten



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Häufigstes Optimierungsziel in der Praxis
- ▶ Optimal: Exaktes (und perfektes) Routing während der Platzierung

Gesamtverbindungslänge

Exakte Kosten



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Häufigstes Optimierungsziel in der Praxis
- ▶ Optimal: Exaktes (und perfektes) Routing während der Platzierung
- ▶ Exaktes Routing zu aufwendig

Gesamtverbindungslänge

Exakte Kosten



- ▶ Häufigstes Optimierungsziel in der Praxis
 - ▶ Optimal: Exaktes (und perfektes) Routing während der Platzierung
 - ▶ Exaktes Routing zu aufwendig
- ⇒ Verbindungslänge für jedes Netz schätzen und aufsummieren

Gesamtverbindungslänge

Exakte Kosten



- ▶ Häufigstes Optimierungsziel in der Praxis
 - ▶ Optimal: Exaktes (und perfektes) Routing während der Platzierung
 - ▶ Exaktes Routing zu aufwendig
- ⇒ Verbindungslänge für jedes Netz schätzen und aufsummieren
- ▶ Annahme optimales Routing möglich (d.h. keine Umwege wegen Hindernissen)

Gesamtverbindungslänge

Exakte Kosten

- ▶ Häufigstes Optimierungsziel in der Praxis
 - ▶ Optimal: Exaktes (und perfektes) Routing während der Platzierung
 - ▶ Exaktes Routing zu aufwendig
- ⇒ Verbindungslänge für jedes Netz schätzen und aufsummieren
- ▶ Annahme optimales Routing möglich (d.h. keine Umwege wegen Hindernissen)
 - ▶ Für bis 3 Pins trivial: HPWL Abschätzung exakt

Gesamtverbindungslänge

Exakte Kosten



- ▶ Häufigstes Optimierungsziel in der Praxis
 - ▶ Optimal: Exaktes (und perfektes) Routing während der Platzierung
 - ▶ Exaktes Routing zu aufwendig
- ⇒ Verbindungslänge für jedes Netz schätzen und aufsummieren
- ▶ Annahme optimales Routing möglich (d.h. keine Umwege wegen Hindernissen)
 - ▶ Für bis 3 Pins trivial: HPWL Abschätzung exakt
 - ▶ Für (viel) mehr: HPWL Abschätzung zunehmend ungenau

Gesamtverbindungslänge

Exakte Kosten



- ▶ Häufigstes Optimierungsziel in der Praxis
 - ▶ Optimal: Exaktes (und perfektes) Routing während der Platzierung
 - ▶ Exaktes Routing zu aufwendig
- ⇒ Verbindungslänge für jedes Netz schätzen und aufsummieren
- ▶ Annahme optimales Routing möglich (d.h. keine Umwege wegen Hindernissen)
 - ▶ Für bis 3 Pins trivial: HPWL Abschätzung exakt
 - ▶ Für (viel) mehr: HPWL Abschätzung zunehmend ungenau
 - ▶ Andere Möglichkeiten RSMT, RMST, korrigierte HPWL, FLUTE, quadratische Verbindungslänge, LSE, WA, Star+, ...



- ▶ Abstand $d(x, y)$ zweier Punkte (Pins) im Raum (in der Ebene)

- ▶ Abstand $d(x, y)$ zweier Punkte (Pins) im Raum (in der Ebene)
- ▶ Definit: $x = y \Leftrightarrow d(x, y) = 0$



- ▶ Abstand $d(x, y)$ zweier Punkte (Pins) im Raum (in der Ebene)
- ▶ Definit: $x = y \Leftrightarrow d(x, y) = 0$
- ▶ Symmetrisch: $d(x, y) = d(y, x)$



- ▶ Abstand $d(x, y)$ zweier Punkte (Pins) im Raum (in der Ebene)
- ▶ Definit: $x = y \Leftrightarrow d(x, y) = 0$
- ▶ Symmetrisch: $d(x, y) = d(y, x)$
- ▶ Erfüllt Dreiecksungleichung: $d(x, y) \leq d(x, z) + d(z, y)$



- ▶ Abstand $d(x, y)$ zweier Punkte (Pins) im Raum (in der Ebene)
- ▶ Definit: $x = y \Leftrightarrow d(x, y) = 0$
- ▶ Symmetrisch: $d(x, y) = d(y, x)$
- ▶ Erfüllt Dreiecksungleichung: $d(x, y) \leq d(x, z) + d(z, y)$
- ▶ Üblicherweise benutzt man von Norm induzierte Metrik

$$d(x, y) := \|x - y\|$$



- ▶ Abstand $d(x, y)$ zweier Punkte (Pins) im Raum (in der Ebene)
- ▶ Definit: $x = y \Leftrightarrow d(x, y) = 0$
- ▶ Symmetrisch: $d(x, y) = d(y, x)$
- ▶ Erfüllt Dreiecksungleichung: $d(x, y) \leq d(x, z) + d(z, y)$
- ▶ Üblicherweise benutzt man von Norm induzierte Metrik

$$d(x, y) := ||x - y||$$

- ▶ **Wichtige Frage:** Wie handhabt man Netze mit mehr als 2 Pins?

1-Norm

Manhattan-/Taxi-Metrik



- ▶ p -Norm induzierte Metrik

$$d(x, y) = \sqrt[p]{\sum_i |x_i - y_i|^p}$$

1-Norm

Manhattan-/Taxi-Metrik



- ▶ p -Norm induzierte Metrik

$$d(x, y) = \sqrt[p]{\sum_i |x_i - y_i|^p}$$

- ▶ 1-Norm gibt Abstand wenn man sich rechtwinklig bewegt

1-Norm

Manhattan-/Taxi-Metrik



- ▶ p -Norm induzierte Metrik

$$d(x, y) = \sqrt[p]{\sum_i |x_i - y_i|^p}$$

- ▶ 1-Norm gibt Abstand wenn man sich rechtwinklig bewegt
- ▶ Gibt exakte Länge in einem Raster

1-Norm

Manhattan-/Taxi-Metrik



- ▶ p -Norm induzierte Metrik

$$d(x, y) = \sqrt[p]{\sum_i |x_i - y_i|^p}$$

- ▶ 1-Norm gibt Abstand wenn man sich rechtwinklig bewegt
- ▶ Gibt exakte Länge in einem Raster
- ▶ Sehr schnell zu berechnen

1-Norm

Manhattan-/Taxi-Metrik



- ▶ p -Norm induzierte Metrik

$$d(x, y) = \sqrt[p]{\sum_i |x_i - y_i|^p}$$

- ▶ 1-Norm gibt Abstand wenn man sich rechtwinklig bewegt
 - ▶ Gibt exakte Länge in einem Raster
 - ▶ Sehr schnell zu berechnen
- ⇒ Sehr weit verbreitet

HPWL

Half-Perimeter-Wirelength



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Verallgemeinerung der 1-Norm für mehr als 2 Pins

HPWL

Half-Perimeter-Wirelength



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Verallgemeinerung der 1-Norm für mehr als 2 Pins
- ▶ Halber Umfang des umschliessenden Rechtecks aller Pins

HPWL

Half-Perimeter-Wirelength



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Verallgemeinerung der 1-Norm für mehr als 2 Pins
- ▶ Halber Umfang des umschliessenden Rechtecks aller Pins
- ▶ Exakt für bis zu 3 Pins

HPWL

Half-Perimeter-Wirelength



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Verallgemeinerung der 1-Norm für mehr als 2 Pins
- ▶ Halber Umfang des umschliessenden Rechtecks aller Pins
- ▶ Exakt für bis zu 3 Pins
- ▶ Untere Grenze für > 3 Pins

HPWL

Half-Perimeter-Wirelength



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Verallgemeinerung der 1-Norm für mehr als 2 Pins
- ▶ Halber Umfang des umschliessenden Rechtecks aller Pins
- ▶ Exakt für bis zu 3 Pins
- ▶ Untere Grenze für > 3 Pins

HPWL

Half-Perimeter-Wirelength



- ▶ Verallgemeinerung der 1-Norm für mehr als 2 Pins
- ▶ Halber Umfang des umschliessenden Rechtecks aller Pins
- ▶ Exakt für bis zu 3 Pins
- ▶ Untere Grenze für > 3 Pins

Vorteile

- ▶ Sehr schnell zu berechnen:

$$HPWL = (\max x_i - \min x_i) + (\max y_i - \min y_i)$$

HPWL

Half-Perimeter-Wirelength



- ▶ Verallgemeinerung der 1-Norm für mehr als 2 Pins
- ▶ Halber Umfang des umschließenden Rechtecks aller Pins
- ▶ Exakt für bis zu 3 Pins
- ▶ Untere Grenze für > 3 Pins

Vorteile

- ▶ Sehr schnell zu berechnen:

$$HPWL = (\max x_i - \min x_i) + (\max y_i - \min y_i)$$

Nachteile

- ▶ Für große Anzahl Pins wird der schnell Fehler groß
⇒ Korrekturfaktoren für große Netze (Abhängig von # Pins)
- ▶ Nicht differenzierbar

HPWL

Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT



HPWL

Beispiele



HPWL

Beispiele



Exakt



HPWL

Beispiele



Exakt

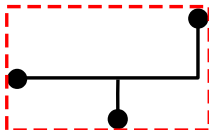


HPWL

Beispiele



Exakt



HPWL

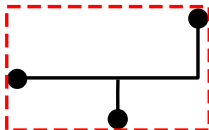
Beispiele



Exakt



Exakt

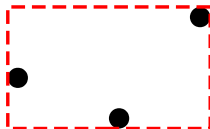


HPWL

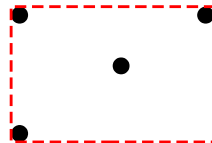
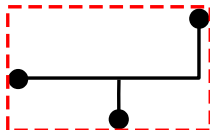
Beispiele



Exakt



Exakt



HPWL

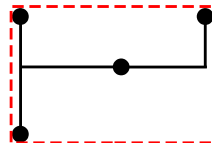
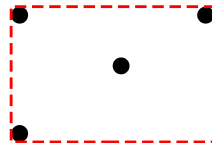
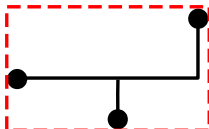
Beispiele



Exakt



Exakt



HPWL

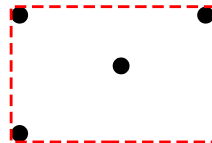
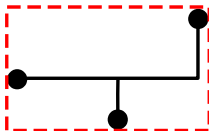
Beispiele



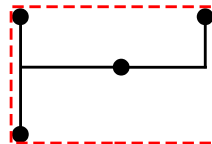
Exakt



Exakt



Untere Grenze



RSMT

Rectilinear Steiner Minimal Tree



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Rechtwinkliger Minimaler Steiner Baum, d.h. Steinerbaum mit rechtwinkliger Metrik

RSMT

Rectilinear Steiner Minimal Tree



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Rechtwinkliger Minimaler Steiner Baum, d.h. Steinerbaum mit rechtwinkliger Metrik
- ▶ Steinerbaumproblem: Finde kleinsten geometrisch aufspannenden Baum eines Graph (Verzweigungen nicht zwangsweise an Knoten)

RSMT

Rectilinear Steiner Minimal Tree



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Rechtwinkliger Minimaler Steiner Baum, d.h. Steinerbaum mit rechtwinkliger Metrik
- ▶ Steinerbaumproblem: Finde kleinsten geometrisch aufspannenden Baum eines Graph (Verzweigungen nicht zwangsweise an Knoten)

RSMT

Rectilinear Steiner Minimal Tree

- ▶ Rechtwinkliger Minimaler Steiner Baum, d.h. Steinerbaum mit rechtwinkliger Metrik
- ▶ Steinerbaumproblem: Finde kleinsten geometrisch aufspannenden Baum eines Graph (Verzweigungen nicht zwangsweise an Knoten)

Vorteile

- ▶ Exakt

RSMT

Rectilinear Steiner Minimal Tree

- ▶ Rechtwinkliger Minimaler Steiner Baum, d.h. Steinerbaum mit rechtwinkliger Metrik
- ▶ Steinerbaumproblem: Finde kleinsten geometrisch aufspannenden Baum eines Graph (Verzweigungen nicht zwangsweise an Knoten)

Vorteile

- ▶ Exakt

Nachteile

- ▶ NP-Vollständig \Rightarrow Impraktikabel
- ▶ Normalerweise sogar Heuristiken lange Laufzeit
- ▶ **NEU:** FLUTE
Tabellenbasierte RSMT-Heuristik
- ▶ Nicht differenzierbar

RMST

Rectilinear Minimum Spanning Tree



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Rechtwinkliger minimal aufspannender Baum

RMST

Rectilinear Minimum Spanning Tree



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Rechtwinkliger minimal aufspannender Baum

RMST

Rectilinear Minimum Spanning Tree



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Rechtwinkliger minimal aufspannender Baum

Vorteile

- ▶ Viel schneller als RSMT:
 $\mathcal{O}(n \log n)$
- ▶ Gütegarantie: Ergebnis
max. 50% schlechter

RMST

Rectilinear Minimum Spanning Tree

- ▶ Rechtwinkliger minimal aufspannender Baum

Vorteile

- ▶ Viel schneller als RSMT:
 $\mathcal{O}(n \log n)$
- ▶ Gütegarantie: Ergebnis
max. 50% schlechter

Nachteile

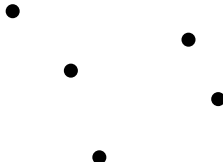
- ▶ Schlechter als RMST
- ▶ Vielfaches langsamer als HPWL
- ▶ Nicht differenzierbar

Vergleich HPWL – RMST – RSMT

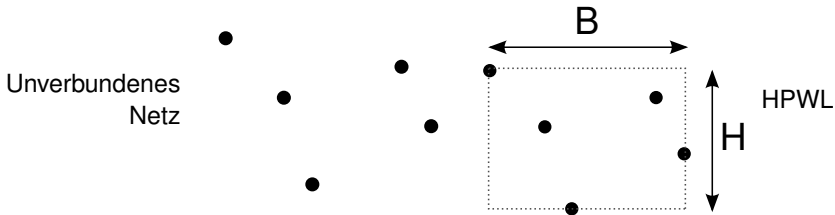


TECHNISCHE
UNIVERSITÄT
DARMSTADT

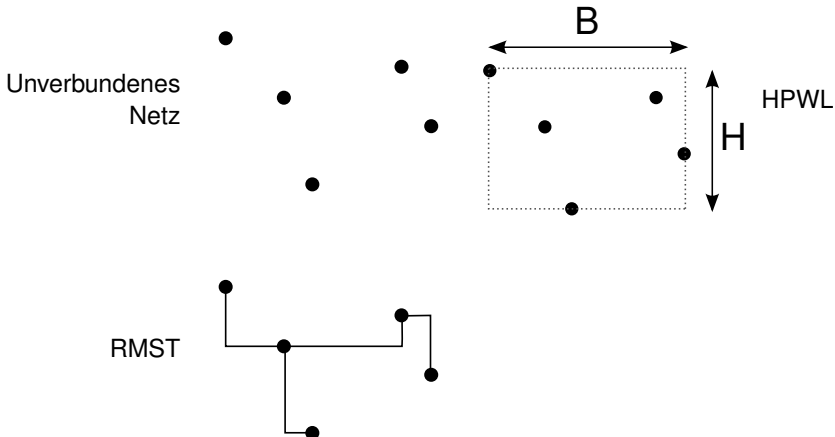
Unverbundenes
Netz



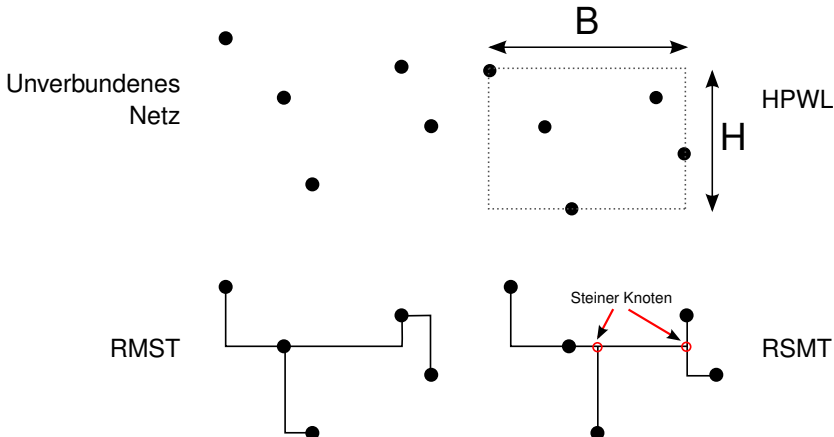
Vergleich HPWL – RMST – RSMT



Vergleich HPWL – RMST – RSMT



Vergleich HPWL – RMST – RSMT



Nichtlineare Metriken

Quadratische Verdrahtungslänge

- ▶ 2-Norm basiert, allerdings ohne Wurzel und mit Faktor

$$QWL = \frac{1}{2}((x_i - x_j)^2 + (y_i - y_j)^2)$$

Nichtlineare Metriken

Quadratische Verdrahtungslänge



- ▶ 2-Norm basiert, allerdings ohne Wurzel und mit Faktor

$$QWL = \frac{1}{2}((x_i - x_j)^2 + (y_i - y_j)^2)$$

- ▶ Da oft mit Ableitung gearbeitet wird: Faktor von $\frac{1}{2}$

Nichtlineare Metriken

Quadratische Verdrahtungslänge



- ▶ 2-Norm basiert, allerdings ohne Wurzel und mit Faktor

$$QWL = \frac{1}{2}((x_i - x_j)^2 + (y_i - y_j)^2)$$

- ▶ Da oft mit Ableitung gearbeitet wird: Faktor von $\frac{1}{2}$

Nichtlineare Metriken

Quadratische Verdrahtungslänge

- ▶ 2-Norm basiert, allerdings ohne Wurzel und mit Faktor

$$QWL = \frac{1}{2}((x_i - x_j)^2 + (y_i - y_j)^2)$$

- ▶ Da oft mit Ableitung gearbeitet wird: Faktor von $\frac{1}{2}$

Vorteile

- ▶ Differenzierbar
- ▶ Konvex

Nichtlineare Metriken

Quadratische Verdrahtungslänge

- ▶ 2-Norm basiert, allerdings ohne Wurzel und mit Faktor

$$QWL = \frac{1}{2}((x_i - x_j)^2 + (y_i - y_j)^2)$$

- ▶ Da oft mit Ableitung gearbeitet wird: Faktor von $\frac{1}{2}$

Vorteile

- ▶ Differenzierbar
- ▶ Konvex

Nachteile

- ▶ Weicht von linearer ($p = 1$) Norm ab \Rightarrow Funktion linearisieren (z.B. Star+)

Nichtlineare Metriken

Log-Summe-Exp-Verdrahtungslänge (LSE)



- ▶ Logarithmus-Summe-Exponentialfunktion:

$$LSE(z_i) = \alpha \left(\log \left(\sum_i e^{\frac{z_i}{\alpha}} \right) \right)$$

Nichtlineare Metriken

Log-Summe-Exp-Verdrahtungslänge (LSE)



- ▶ Logarithmus-Summe-Exponentialfunktion:

$$LSE(z_i) = \alpha(\log(\sum_i e^{\frac{z_i}{\alpha}}))$$

- ▶ Approximation der max-Funktion

Nichtlineare Metriken

Log-Summe-Exp-Verdrahtungslänge (LSE)



- ▶ Logarithmus-Summe-Exponentialfunktion:

$$LSE(z_i) = \alpha \left(\log \left(\sum_i e^{\frac{z_i}{\alpha}} \right) \right)$$

- ▶ Approximation der max-Funktion
- ▶ α Genauigkeit der Approximation

Nichtlineare Metriken

Log-Summe-Exp-Verdrahtungslänge (LSE)



- ▶ Logarithmus-Summe-Exponentialfunktion:

$$LSE(z_i) = \alpha(\log(\sum_i e^{\frac{z_i}{\alpha}}))$$

- ▶ Approximation der max-Funktion
- ▶ α Genauigkeit der Approximation
- ▶ Für $\alpha \rightarrow 0$: LSE konvergiert gegen max

Nichtlineare Metriken

Log-Summe-Exp-Verdrahtungslänge (LSE)

- ▶ Logarithmus-Summe-Exponentialfunktion:

$$LSE(z_i) = \alpha \left(\log \left(\sum_i e^{\frac{z_i}{\alpha}} \right) \right)$$

- ▶ Approximation der max-Funktion
- ▶ α Genauigkeit der Approximation
- ▶ Für $\alpha \rightarrow 0$: LSE konvergiert gegen max
- ▶ Approximiert somit HPWL

$$\alpha \times \left(\log \left(\sum_i e^{\frac{x_i}{\alpha}} \right) + \log \left(\sum_i e^{\frac{-x_i}{\alpha}} \right) + \log \left(\sum_i e^{\frac{y_i}{\alpha}} \right) + \log \left(\sum_i e^{\frac{-y_i}{\alpha}} \right) \right)$$

Nichtlineare Metriken

Log-Summe-Exp-Verdrahtungslänge (LSE)

- ▶ Logarithmus-Summe-Exponentialfunktion:

$$LSE(z_i) = \alpha \left(\log \left(\sum_i e^{\frac{z_i}{\alpha}} \right) \right)$$

- ▶ Approximation der max-Funktion
- ▶ α Genauigkeit der Approximation
- ▶ Für $\alpha \rightarrow 0$: LSE konvergiert gegen max
- ▶ Approximiert somit HPWL

$$\alpha \times \left(\log \left(\sum_i e^{\frac{x_i}{\alpha}} \right) + \log \left(\sum_i e^{\frac{-x_i}{\alpha}} \right) + \log \left(\sum_i e^{\frac{y_i}{\alpha}} \right) + \log \left(\sum_i e^{\frac{-y_i}{\alpha}} \right) \right)$$

- ▶ Differenzierbar und konvex

(Nicht)lineare Metriken

Netze mit mehr als 2 Pins



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ *Auflösen* des Multi-Pin-Netzes in mehrere 2-Pin-Netze

(Nicht)lineare Metriken

Netze mit mehr als 2 Pins



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ *Auflösen* des Multi-Pin-Netzes in mehrere 2-Pin-Netze
Cliquen Modell Netz wird durch vollständigen Teilgraphen (Clique) ersetzt

(Nicht)lineare Metriken

Netze mit mehr als 2 Pins

- ▶ *Auflösen* des Multi-Pin-Netzes in mehrere 2-Pin-Netze
 - Cliquen Modell** Netz wird durch vollständigen Teilgraphen (Clique) ersetzt
 - Stern Modell** Netz wird durch zusätzlichen Knoten und 2-Pin-Netze zu diesen Stern-Knoten ersetzt

(Nicht)lineare Metriken

Netze mit mehr als 2 Pins

- ▶ *Auflösen* des Multi-Pin-Netzes in mehrere 2-Pin-Netze
 - Cliquen Modell** Netz wird durch vollständigen Teilgraphen (Clique) ersetzt
 - Stern Modell** Netz wird durch zusätzlichen Knoten und 2-Pin-Netze zu diesen Stern-Knoten ersetzt
 - Hybrid Modell** Bis 4 Pins Clique Modell, > 4 Stern Modell



- ▶ Basierend auf Stern-Netzmodell:
Multipin-Netz wird ersetzt durch fiktiven Knoten und alle Pins verbinden zu diesem



- ▶ Basierend auf Stern-Netzmodell:
Multipin-Netz wird ersetzt durch fiktiven Knoten und alle Pins verbinden zu diesem
- ▶ x - und y -Koordinate werden getrennt berechnet (und später addiert)



- ▶ Basierend auf Stern-Netzmodell:
Multipin-Netz wird ersetzt durch fiktiven Knoten und alle Pins verbinden zu diesem
- ▶ x - und y -Koordinate werden getrennt berechnet (und später addiert)
- ▶ Fast lineare Funktion

- ▶ Basierend auf Stern-Netzmodell:
Multipin-Netz wird ersetzt durch fiktiven Knoten und alle Pins verbinden zu diesem
- ▶ x - und y -Koordinate werden getrennt berechnet (und später addiert)
- ▶ Fast lineare Funktion
- ▶ **Schnell**: Nach Veränderungen ein Update in $\mathcal{O}(1)$ möglich

- ▶ Basierend auf Stern-Netzmodell:
Multipin-Netz wird ersetzt durch fiktiven Knoten und alle Pins verbinden zu diesem
- ▶ x - und y -Koordinate werden getrennt berechnet (und später addiert)
- ▶ Fast lineare Funktion
- ▶ **Schnell**: Nach Veränderungen ein Update in $\mathcal{O}(1)$ möglich
- ▶ Schwerpunkt x_g für Netz l mit k_l Blocks

$$x_g := \frac{1}{k_l} \sum_{i \in \text{Netz}_l} x_i$$

$$\|\text{Netz}_l\|_x := \gamma \sqrt{\sum_{i \in \text{Netz}_l} (x_i - x_g)^2 + \phi}$$

ϕ Konstante > 0 , so dass es immer diff'bar

γ Faktor zur Kompensation von Unterschätzung

- ▶ Basierend auf Stern-Netzmodell:
Multipin-Netz wird ersetzt durch fiktiven Knoten und alle Pins verbinden zu diesem
- ▶ x - und y -Koordinate werden getrennt berechnet (und später addiert)
- ▶ Fast lineare Funktion
- ▶ **Schnell**: Nach Veränderungen ein Update in $\mathcal{O}(1)$ möglich
- ▶ Schwerpunkt x_g für Netz l mit k_l Blocks
$$x_g := \frac{1}{k_l} \sum_{i \in \text{Netz}_l} x_i$$
$$\|\text{Netz}_l\|_x := \gamma \sqrt{\sum_{i \in \text{Netz}_l} (x_i - x_g)^2 + \phi}$$
 ϕ Konstante > 0 , so dass es immer diff'bar
 γ Faktor zur Kompensation von Unterschätzung
- ▶ y -Koordinate analog

- ▶ Basierend auf Stern-Netzmodell:
Multipin-Netz wird ersetzt durch fiktiven Knoten und alle Pins verbinden zu diesem
- ▶ x - und y -Koordinate werden getrennt berechnet (und später addiert)
- ▶ Fast lineare Funktion
- ▶ **Schnell:** Nach Veränderungen ein Update in $\mathcal{O}(1)$ möglich
- ▶ Schwerpunkt x_g für Netz l mit k_l Blocks
$$x_g := \frac{1}{k_l} \sum_{i \in \text{Netz}_l} x_i$$
$$\|\text{Netz}_l\|_x := \gamma \sqrt{\sum_{i \in \text{Netz}_l} (x_i - x_g)^2 + \phi}$$
 ϕ Konstante > 0 , so dass es immer diff'bar
 γ Faktor zur Kompensation von Unterschätzung
- ▶ y -Koordinate analog
- ▶ Unterschied zu normalem Stern-Modell: $\sqrt{\dots + \phi}$

- **Schnell:** Nach Veränderung von einzelnen Positionen ist ein Update in $\mathcal{O}(1)$ möglich

$$\|\text{Netz}_l\|_x = \gamma \sqrt{\sum_{i \in \text{Netz}_l} (x_i - x_g)^2 + \phi} = \gamma \sqrt{\sum_{i \in \text{Netz}_l} x_i^2 - k_l x_g^2 + \phi}$$

- **Schnell:** Nach Veränderung von einzelnen Positionen ist ein Update in $\mathcal{O}(1)$ möglich

$$\|\text{Netz}_l\|_x = \gamma \sqrt{\sum_{i \in \text{Netz}_l} (x_i - x_g)^2 + \phi} = \gamma \sqrt{\sum_{i \in \text{Netz}_l} x_i^2 - k_l x_g^2 + \phi}$$

- **Schnell:** Nach Veränderung von einzelnen Positionen ist ein Update in $\mathcal{O}(1)$ möglich

$$\|\text{Netz}_l\|_x = \gamma \sqrt{\sum_{i \in \text{Netz}_l} (x_i - x_g)^2 + \phi} = \gamma \sqrt{\sum_{i \in \text{Netz}_l} x_i^2 - k_l x_g^2 + \phi}$$

$$U_l := \sum_{i \in \text{Netz}_l} x_i^2, V_l := \sum_{i \in \text{Netz}_l} x_i = k_l x_g$$

- **Schnell:** Nach Veränderung von einzelnen Positionen ist ein Update in $\mathcal{O}(1)$ möglich

$$\|\text{Netz}_I\|_x = \gamma \sqrt{\sum_{i \in \text{Netz}_I} (x_i - x_g)^2 + \phi} = \gamma \sqrt{\sum_{i \in \text{Netz}_I} x_i^2 - k_I x_g^2 + \phi}$$

$$U_I := \sum_{i \in \text{Netz}_I} x_i^2, V_I := \sum_{i \in \text{Netz}_I} x_i = k_I x_g$$

$$\|\text{Netz}_I\|_x = \gamma \sqrt{U_I - \frac{V_I^2}{k_I} + \phi}$$

- ▶ **Schnell:** Nach Veränderung von einzelnen Positionen ist ein Update in $\mathcal{O}(1)$ möglich

$$\| \text{Netz}_l \|_x = \gamma \sqrt{\sum_{i \in \text{Netz}_l} (x_i - x_g)^2 + \phi} = \gamma \sqrt{\sum_{i \in \text{Netz}_l} x_i^2 - k_l x_g^2 + \phi}$$

$$U_l := \sum_{i \in \text{Netz}_l} x_i^2, V_l := \sum_{i \in \text{Netz}_l} x_i = k_l x_g$$

$$\| \text{Netz}_l \|_x = \gamma \sqrt{U_l - \frac{V_l^2}{k_l} + \phi}$$

Block $b \in \text{Netz}_l$ wird bewegt, dann Update in $\mathcal{O}(1)$

$$U_l^{\text{new}} = U_l - x_b^2 + (x_b^{\text{new}})^2$$

$$V_l^{\text{new}} = V_l - x_b + x_b^{\text{new}}$$

Verbesserung der Kostenfunktion

Gewichtete Netzkosten



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Bisher: Jedes Netz gleich behandelt

Verbesserung der Kostenfunktion

Gewichtete Netzkosten



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Bisher: Jedes Netz gleich behandelt
- ▶ Gesamtkosten = \sum Netzkosten

Verbesserung der Kostenfunktion

Gewichtete Netzkosten

- ▶ Bisher: Jedes Netz gleich behandelt
- ▶ Gesamtkosten = \sum Netzkosten
- ▶ Verbesserung: Gewichtung einzelner Teilnetze bzgl. bestimmter (anderer) Optimierungsziele

Verbesserung der Kostenfunktion

Gewichtete Netzkosten

- ▶ Bisher: Jedes Netz gleich behandelt
- ▶ Gesamtkosten = \sum Netzkosten
- ▶ Verbesserung: Gewichtung einzelner Teilnetze bzgl. bestimmter (anderer) Optimierungsziele
- ▶ Nun: Gesamtkosten = $\sum c_i(\dots) \cdot$ Netzkosten

Verbesserung der Kostenfunktion

Gewichtete Netzkosten

- ▶ Bisher: Jedes Netz gleich behandelt
- ▶ Gesamtkosten = \sum Netzkosten
- ▶ Verbesserung: Gewichtung einzelner Teilnetze bzgl. bestimmter (anderer) Optimierungsziele
- ▶ Nun: Gesamtkosten = $\sum c_i(\dots) \cdot$ Netzkosten
- ▶ Gewicht für jedes Netz muß separat bestimmt/berechnet werden



- ▶ Im folgenden



- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)



- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:



- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme



- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme
 - ▶ Bestimmung einer nicht optimalen (aber hoffentlich guten) Lösung



- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme
 - ▶ Bestimmung einer nicht optimalen (aber hoffentlich guten) Lösung
 - ▶ Bei nicht exponentieller Laufzeit



- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme
 - ▶ Bestimmung einer nicht optimalen (aber hoffentlich guten) Lösung
 - ▶ Bei nicht exponentieller Laufzeit
 - ▶ Einige **exakte** mathematische Optimierungsalgorithmen:



- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme
 - ▶ Bestimmung einer nicht optimalen (aber hoffentlich guten) Lösung
 - ▶ Bei nicht exponentieller Laufzeit
 - ▶ Einige **exakte** mathematische Optimierungsalgorithmen:
 - ▶ Oft vereinfachtes Modell \Rightarrow Nicht exponentielle Laufzeit

- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme
 - ▶ Bestimmung einer nicht optimalen (aber hoffentlich guten) Lösung
 - ▶ Bei nicht exponentieller Laufzeit
 - ▶ Einige **exakte** mathematische Optimierungsalgorithmen:
 - ▶ Oft vereinfachtes Modell \Rightarrow Nicht exponentielle Laufzeit
 - ▶ Nur für sehr kleine Problemgrößen (bei NP-Vollständigen Algorithmen)

- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme
 - ▶ Bestimmung einer nicht optimalen (aber hoffentlich guten) Lösung
 - ▶ Bei nicht exponentieller Laufzeit
 - ▶ Einige **exakte** mathematische Optimierungsalgorithmen:
 - ▶ Oft vereinfachtes Modell \Rightarrow Nicht exponentielle Laufzeit
 - ▶ Nur für sehr kleine Problemgrößen (bei NP-Vollständigen Algorithmen)
- ▶ Später:

- ▶ Im folgenden
 - ▶ Beispiel: Abstraktes vereinfachtes Platzierungsproblem (UPP)
 - ▶ Einige **heuristische** mathematische Optimierungsalgorithmen:
 - ▶ Zur Lösung der NP-Vollständigen Probleme
 - ▶ Bestimmung einer nicht optimalen (aber hoffentlich guten) Lösung
 - ▶ Bei nicht exponentieller Laufzeit
 - ▶ Einige **exakte** mathematische Optimierungsalgorithmen:
 - ▶ Oft vereinfachtes Modell \Rightarrow Nicht exponentielle Laufzeit
 - ▶ Nur für sehr kleine Problemgrößen (bei NP-Vollständigen Algorithmen)
- ▶ Später:
 - ▶ Konkrete Varianten/Implementierungen einzelner Verfahren

Beispielanwendung

Unit-Size Placement Problem (UPP)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Eingabe

- ▶ 1×1 Zellen
- ▶ Netzliste

Beispielanwendung

Unit-Size Placement Problem (UPP)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Eingabe

- ▶ 1×1 Zellen
- ▶ Netzliste

Platziere Zellen

- ▶ Auf 1×1 Raster
- ▶ Überlappungsfrei

Beispielanwendung

Unit-Size Placement Problem (UPP)

Eingabe

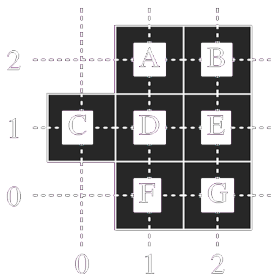
- ▶ 1×1 Zellen
- ▶ Netzliste

Platzierte Zellen

- ▶ Auf 1×1 Raster
- ▶ Überlappungsfrei

Minimiere Fläche

- ▶ Platz für Verdrahtung



n_1 : A, B, F, G

n_2 : B, E

n_3 : D, E

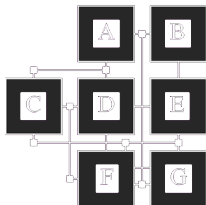
n_4 : A, C, D

n_5 : C, D, F

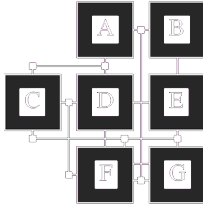
n_6 : C, E, F, G

n_7 : D, F

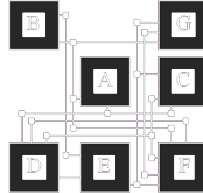
n_8 : F, G



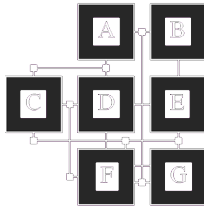
Gute Platzierung mit Verdrahtung



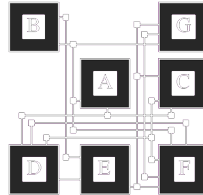
Gute Platzierung mit Verdrahtung



Schlechte Platzierung
Mehr Verdrahtungsspuren



Gute Platzierung mit Verdrahtung



Schlechte Platzierung
Mehr Verdrahtungsspuren

Problem: Bestimmung der Qualität

- ▶ Komplette Verdrahtung dauert zu lange
- ▶ Abschätzen

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problemspezifisch

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung
 - ▶ Alle Unbekannten haben (immer) gültige Werte

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung
 - ▶ Alle Unbekannten haben (immer) gültige Werte
 - ▶ Algorithmus könnte beliebig beendet werden

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung
 - ▶ Alle Unbekannten haben (immer) gültige Werte
 - ▶ Algorithmus könnte beliebig beendet werden
 - ⇒ Hat ggf. nur eine schlechtere Lösung

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung
 - ▶ Alle Unbekannten haben (immer) gültige Werte
 - ▶ Algorithmus könnte beliebig beendet werden
 - ⇒ Hat ggf. nur eine schlechtere Lösung
 - ▶ Unvollständige Lösung

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung
 - ▶ Alle Unbekannten haben (immer) gültige Werte
 - ▶ Algorithmus könnte beliebig beendet werden
 - ⇒ Hat ggf. nur eine schlechtere Lösung
 - ▶ Unvollständige Lösung
 - ▶ Einige/Alle Unbekannte sind noch unbestimmt oder ungültig

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*

- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung
 - ▶ Alle Unbekannten haben (immer) gültige Werte
 - ▶ Algorithmus könnte beliebig beendet werden
 - ⇒ Hat ggf. nur eine schlechtere Lösung
 - ▶ Unvollständige Lösung
 - ▶ Einige/Alle Unbekannte sind noch unbestimmt oder ungültig
 - ▶ Algorithmus muß weiter rechnen

Heuristische Optimierungsalgorithmen

Darstellung einer *Lösung*



- ▶ Problemspezifisch
- ▶ Algorithmusspezifisch
- ▶ Grundsätzlich unterscheidbar
 - ▶ Vollständige Lösung
 - ▶ Alle Unbekannten haben (immer) gültige Werte
 - ▶ Algorithmus könnte beliebig beendet werden
 - ⇒ Hat ggf. nur eine schlechtere Lösung
 - ▶ Unvollständige Lösung
 - ▶ Einige/Alle Unbekannte sind noch unbestimmt oder ungültig
 - ▶ Algorithmus muß weiter rechnen
 - ⇒ Bei vorzeitiger Beendigung gar keine Lösung



- ▶ Probleminstanz $I = (F, c)$



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F
 - ▶ Kostenfunktion $c : F \mapsto \mathbf{R}$



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F
 - ▶ Kostenfunktion $c : F \mapsto \mathbf{R}$
- ▶ Lösung $\vec{f} \in F : \vec{f} = [f_1, \dots, f_n]^T$



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F
 - ▶ Kostenfunktion $c : F \mapsto \mathbf{R}$
- ▶ Lösung $\vec{f} \in F : \vec{f} = [f_1, \dots, f_n]^T$
 - ▶ Explizite Einschränkungen: Wertebereiche der f_i



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F
 - ▶ Kostenfunktion $c : F \mapsto \mathbf{R}$
- ▶ Lösung $\vec{f} \in F : \vec{f} = [f_1, \dots, f_n]^T$
 - ▶ Explizite Einschränkungen: Wertebereiche der f_i
 - ▶ Implizite Einschränkung: Abhängigkeiten



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F
 - ▶ Kostenfunktion $c : F \mapsto \mathbf{R}$
- ▶ Lösung $\vec{f} \in F : \vec{f} = [f_1, \dots, f_n]^T$
 - ▶ Explizite Einschränkungen: Wertebereiche der f_i
 - ▶ Implizite Einschränkung: Abhängigkeiten
- ▶ Teillösung \vec{f}



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F
 - ▶ Kostenfunktion $c : F \mapsto \mathbf{R}$
- ▶ Lösung $\vec{f} \in F : \vec{f} = [f_1, \dots, f_n]^T$
 - ▶ Explizite Einschränkungen: Wertebereiche der f_i
 - ▶ Implizite Einschränkung: Abhängigkeiten
- ▶ Teillösung \vec{f}
 - ▶ Einige f_i undefiniert



- ▶ Probleminstanz $I = (F, c)$
 - ▶ Lösungsraum F
 - ▶ Kostenfunktion $c : F \mapsto \mathbf{R}$
- ▶ Lösung $\vec{f} \in F : \vec{f} = [f_1, \dots, f_n]^T$
 - ▶ Explizite Einschränkungen: Wertebereiche der f_i
 - ▶ Implizite Einschränkung: Abhängigkeiten
- ▶ Teillösung \vec{f}
 - ▶ Einige f_i undefiniert
 - ▶ Spannt Unterraum von F auf

Nachbarsuche Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Starte mit einer vollständigen Lösung

Nachbarsuche Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Starte mit einer vollständigen Lösung
- ▶ Bestimme *Nachbarn* der Lösung

Nachbarsuche

Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Starte mit einer vollständigen Lösung
- ▶ Bestimme *Nachbarn* der Lösung
 - ▶ Andere Lösungen *nahe* an existierender



- ▶ Starte mit einer vollständigen Lösung
- ▶ Bestimme *Nachbarn* der Lösung
 - ▶ Andere Lösungen *nahe* an existierender
 - ▶ Definition von *Nähe* ist problemspezifisch



- ▶ Starte mit einer vollständigen Lösung
- ▶ Bestimme *Nachbarn* der Lösung
 - ▶ Andere Lösungen *nahe* an existierender
 - ▶ Definition von *Nähe* ist problemspezifisch
- ▶ Wähle *besseren* Nachbarn aus



- ▶ Starte mit einer vollständigen Lösung
- ▶ Bestimme *Nachbarn* der Lösung
 - ▶ Andere Lösungen *nahe* an existierender
 - ▶ Definition von *Nähe* ist problemspezifisch
- ▶ Wähle *besseren* Nachbarn aus
- ▶ Wiederhole



- ▶ Problem $I = (F, c)$

Nachbarsuche

Formal



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem $I = (F, c)$
- ▶ Lösung $\vec{f} \in F$



- ▶ Problem $I = (F, c)$
- ▶ Lösung $\vec{f} \in F$
- ▶ Nachbarschaft $N : F \mapsto 2^F$
(2^F Potenzmenge = Menge aller Teilmengen von F)



- ▶ Problem $I = (F, c)$
- ▶ Lösung $\vec{f} \in F$
- ▶ Nachbarschaft $N : F \mapsto 2^F$
(2^F Potenzmenge = Menge aller Teilmengen von F)
- ▶ Nachbar $\vec{g} \in N(\vec{f})$



- ▶ Problem $I = (F, c)$
- ▶ Lösung $\vec{f} \in F$
- ▶ Nachbarschaft $N : F \mapsto 2^F$
(2^F Potenzmenge = Menge aller Teilmengen von F)
- ▶ Nachbar $\vec{g} \in N(\vec{f})$

- ▶ Problem $I = (F, c)$
- ▶ Lösung $\vec{f} \in F$
- ▶ Nachbarschaft $N : F \mapsto 2^F$
(2^F Potenzmenge = Menge aller Teilmengen von F)
- ▶ Nachbar $\vec{g} \in N(\vec{f})$

Beispiel UPP

- ▶ \vec{g} ist Nachbar von $\vec{f} : \Leftrightarrow \vec{g}$ ergibt sich durch Vertauschung von 2 Zellen von \vec{f} .

- ▶ Problem $I = (F, c)$
- ▶ Lösung $\vec{f} \in F$
- ▶ Nachbarschaft $N : F \mapsto 2^F$
(2^F Potenzmenge = Menge aller Teilmengen von F)
- ▶ Nachbar $\vec{g} \in N(\vec{f})$

Beispiel UPP

- ▶ \vec{g} ist Nachbar von $\vec{f} : \Leftrightarrow \vec{g}$ ergibt sich durch Vertauschung von 2 Zellen von \vec{f} .
- ▶ n Zellen, $n - 1$ Tauschpartner $\Rightarrow |N(\vec{f})| = \frac{n(n-1)}{2}$



- ▶ Problem $I = (F, c)$
- ▶ Lösung $\vec{f} \in F$
- ▶ Nachbarschaft $N : F \mapsto 2^F$
(2^F Potenzmenge = Menge aller Teilmengen von F)
- ▶ Nachbar $\vec{g} \in N(\vec{f})$

Beispiel UPP

- ▶ \vec{g} ist Nachbar von $\vec{f} : \Leftrightarrow \vec{g}$ ergibt sich durch Vertauschung von 2 Zellen von \vec{f} .
- ▶ n Zellen, $n - 1$ Tauschpartner $\Rightarrow |N(\vec{f})| = \frac{n(n-1)}{2}$
- ▶ Komplexere *Züge* möglich
z.B. tausche 3 Zellen, tausche Regionen, ...



- ▶ Welches $\vec{g} \in N(\vec{f})$ wählen?
- ▶ Ziel: Kostenreduzierung bezüglich c
- ▶ Also wähle \vec{g} mit $c(\vec{g}) < c(\vec{f})$
- ▶ Ende mit \vec{f} falls $\forall \vec{g} \in N(\vec{f}) : c(\vec{g}) \geq c(\vec{f})$



Local_search() : **begin**

Feasible_solution f ;

Set<Feasible_solution> G ;

f := Initial_solution() ;

repeat

$G := \{g \mid g \in N(f) \wedge c(g) < c(f)\}$;

if $G \neq \emptyset$ **then**

 f := G.pickany()

until $G = \emptyset$;

report(f) ;

► Initialisierung:



Local_search() : **begin**

Feasible_solution f ;

Set<Feasible_solution> G ;

f := Initial_solution() ;

repeat

 G := {g | g ∈ N(f) ∧ c(g) < c(f)} ;

if G ≠ ∅ **then**

 f := G.pickany()

until G = ∅;

report(f) ;

- ▶ Initialisierung: Initial_solution? Zufällig, triviale Lösung
- ▶ Strategien: pickany?



Local_search() : **begin**

Feasible_solution f ;

Set<Feasible_solution> G ;

f := Initial_solution() ;

repeat

$G := \{g \mid g \in N(f) \wedge c(g) < c(f)\}$;

if $G \neq \emptyset$ **then**

 f := G.pickany()

until $G = \emptyset$;

report(f) ;

- ▶ Initialisierung: Initial_solution? Zufällig, triviale Lösung
- ▶ Strategien: pickany? Erste Verbesserung, Steilster Abstieg

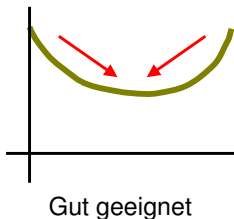


- ▶ *Form der Kostenfunktion:*

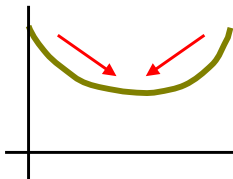


- ▶ *Form der Kostenfunktion:*

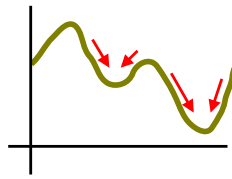
- ▶ *Form* der Kostenfunktion:



► *Form* der Kostenfunktion:

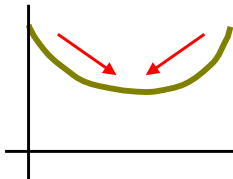


Gut geeignet



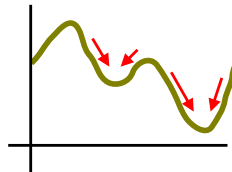
Schlecht geeignet

- ▶ *Form* der Kostenfunktion:



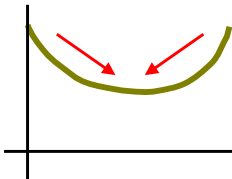
Gut geeignet

- ▶ Steckenbleiben in lokalen Minima



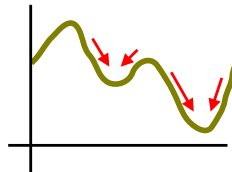
Schlecht geeignet

- ▶ *Form* der Kostenfunktion:



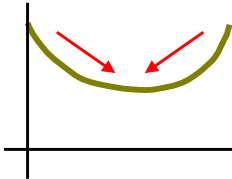
Gut geeignet

- ▶ Steckenbleiben in lokalen Minima
- ▶ Mögliche Lösungen:



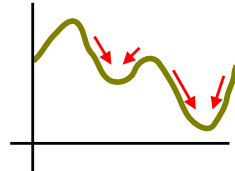
Schlecht geeignet

- ▶ *Form* der Kostenfunktion:



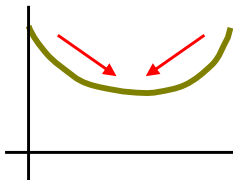
Gut geeignet

- ▶ Steckenbleiben in lokalen Minima
- ▶ Mögliche Lösungen:
 - ▶ Mehrere Läufe mit anderen Startlösungen

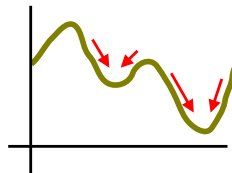


Schlecht geeignet

- ▶ *Form* der Kostenfunktion:



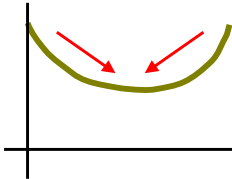
Gut geeignet



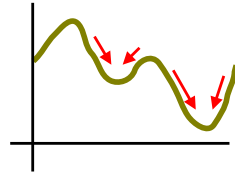
Schlecht geeignet

- ▶ Steckenbleiben in lokalen Minima
- ▶ Mögliche Lösungen:
 - ▶ Mehrere Läufe mit anderen Startlösungen
 - ▶ Größere Nachbarschaft

- ▶ *Form* der Kostenfunktion:



Gut geeignet



Schlecht geeignet

- ▶ Steckenbleiben in lokalen Minima
- ▶ Mögliche Lösungen:
 - ▶ Mehrere Läufe mit anderen Startlösungen
 - ▶ Größere Nachbarschaft
 - ▶ Adaptiere Größe der Nachbarschaft

Simulated Annealing

Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Akzeptiere verschlechternde Züge
Aber bessere Strategie als reiner Zufall

Simulated Annealing

Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Akzeptiere verschlechternde Züge
Aber bessere Strategie als reiner Zufall
- ▶ Simulated Annealing

Simulated Annealing

Idee



- ▶ Akzeptiere verschlechternde Züge
Aber bessere Strategie als reiner Zufall
- ▶ Simulated Annealing
 - ▶ Simuliertes Erstarren von Metallen

Simulated Annealing

Idee



- ▶ Akzeptiere verschlechternde Züge
Aber bessere Strategie als reiner Zufall
- ▶ Simulated Annealing
 - ▶ Simuliertes Erstarren von Metallen
 - ▶ Inspiriert von physikalischen Erstarrungsprozessen

Simulated Annealing

Idee



- ▶ Akzeptiere verschlechternde Züge
Aber bessere Strategie als reiner Zufall
- ▶ Simulated Annealing
 - ▶ Simuliertes Erstarren von Metallen
 - ▶ Inspiriert von physikalischen Erstarrungsprozessen
 - ▶ Schnelles Erstarren (*Schockfrosten*)
Hohe innere Spannung = hohe innere Energie

Simulated Annealing

Idee



- ▶ Akzeptiere verschlechternde Züge
Aber bessere Strategie als reiner Zufall
- ▶ Simulated Annealing
 - ▶ Simuliertes Erstarren von Metallen
 - ▶ Inspiriert von physikalischen Erstarrungsprozessen
 - ▶ Schnelles Erstarren (*Schockfrosten*)
Hohe innere Spannung = hohe innere Energie
 - ▶ Langsames Abkühlen Niedrige innere Spannung = niedrige innere Energie

Simulated Annealing

Physikalischer Hintergrund



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hohe Anfangstemperatur (flüssiges Material)

Simulated Annealing

Physikalischer Hintergrund



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hohe Anfangstemperatur (flüssiges Material)
- ▶ Moleküle können sich frei anordnen

Simulated Annealing

Physikalischer Hintergrund



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hohe Anfangstemperatur (flüssiges Material)
- ▶ Moleküle können sich frei anordnen
- ▶ Langsames Abkühlen

Simulated Annealing

Physikalischer Hintergrund



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hohe Anfangstemperatur (flüssiges Material)
- ▶ Moleküle können sich frei anordnen
- ▶ Langsames Abkühlen
- ▶ Bewegungsfreiheit wird schrittweise weiter eingeschränkt

Simulated Annealing

Physikalischer Hintergrund



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hohe Anfangstemperatur (flüssiges Material)
- ▶ Moleküle können sich frei anordnen
- ▶ Langsames Abkühlen
- ▶ Bewegungsfreiheit wird schrittweise weiter eingeschränkt
- ▶ Moleküle ordnen sich in Konfiguration niedrigster Energie an

Simulated Annealing

Physikalischer Hintergrund



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hohe Anfangstemperatur (flüssiges Material)
- ▶ Moleküle können sich frei anordnen
- ▶ Langsames Abkühlen
- ▶ Bewegungsfreiheit wird schrittweise weiter eingeschränkt
- ▶ Moleküle ordnen sich in Konfiguration niedrigster Energie an
- ▶ Am besten bei sehr langsamer Abkühlung

Simulated Annealing für die Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Energie entspricht Kostenfunktion

Simulated Annealing für die Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Energie entspricht Kostenfunktion
- ▶ Bewegung der Moleküle entspricht Zügen

Simulated Annealing für die Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Energie entspricht Kostenfunktion
- ▶ Bewegung der Moleküle entspricht Zügen
- ▶ Temperatur entspricht Kontrollparameter T

Simulated Annealing für die Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Energie entspricht Kostenfunktion
- ▶ Bewegung der Moleküle entspricht Zügen
- ▶ Temperatur entspricht Kontrollparameter T
 - ▶ Wie frei dürfen sich Moleküle bewegen?
= Welche Züge sind noch akzeptabel?

Simulated Annealing für die Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Energie entspricht Kostenfunktion
- ▶ Bewegung der Moleküle entspricht Zügen
- ▶ Temperatur entspricht Kontrollparameter T
 - ▶ Wie frei dürfen sich Moleküle bewegen?
= Welche Züge sind noch akzeptabel?
 - ▶ Niedrigere Energie/Kosten: Immer akzeptiert
 $c(\vec{g}) \leq c(\vec{f})$



- ▶ Energie entspricht Kostenfunktion
- ▶ Bewegung der Moleküle entspricht Zügen
- ▶ Temperatur entspricht Kontrollparameter T
 - ▶ Wie frei dürfen sich Moleküle bewegen?
= Welche Züge sind noch akzeptabel?
 - ▶ Niedrigere Energie/Kosten: Immer akzeptiert
 $c(\vec{g}) \leq c(\vec{f})$
 - ▶ Höhere Energie/Kosten:
Akzeptiert mit Wahrscheinlichkeit $e^{\frac{-\Delta c}{T}}$ mit $\Delta c = c(\vec{g}) - c(\vec{f})$

Simulated Annealing

Temperatur



TECHNISCHE
UNIVERSITÄT
DARMSTADT

$$e^{\frac{-\Delta c}{T}} = \frac{1}{e^{\frac{\Delta c}{T}}}$$

- ▶ Hohe Temperaturen
Akzeptiere fast alle schlechten Züge

Simulated Annealing

Temperatur

$$e^{\frac{-\Delta c}{T}} = \frac{1}{e^{\frac{\Delta c}{T}}}$$

- ▶ Hohe Temperaturen
Akzeptiere fast alle schlechten Züge
- ▶ Niedrige Temperaturen
Akzeptiere fast keine schlechten Züge mehr



$$e^{\frac{-\Delta c}{T}} = \frac{1}{e^{\frac{\Delta c}{T}}}$$

- ▶ Hohe Temperaturen
Akzeptiere fast alle schlechten Züge
- ▶ Niedrige Temperaturen
Akzeptiere fast keine schlechten Züge mehr
- ▶ Physik: Boltzmann-Verteilung
Statistische Mechanik

Simulated Annealing Algorithmus



Simulated_annealing(): **begin**

```
Feasible_solution f, g, bsf ;  
float T ;  
T := Initial_temperature() ;  
f := Initial_solution() ;  
bsf := f ;  
repeat  
    repeat  
        g := N(f).pickany() ;  
        if Accept(f, g) then f := g ;  
        until Thermal_equilibrium(T);  
        T := New_temperature(T) ;  
until Stop();  
report(bsf) ;
```

int Accept(Feasible_solution f, g):

begin

```
float DeltaC ;  
DeltaC := c(g) - c(f) ;  
if DeltaC ≤ 0 then  
    if c(g) < c(bsf) then bsf := g ;  
    return (1);  
else  
    return (exp(-DeltaC/T) >  
        random(1))
```

Simulated Annealing

Zusätzliche Funktionen



`Initial_temperature()`

Bestimmt ausreichend hohe Starttemperatur

`Initial_solution()`

- ▶ Bestimmt Startlösung
- ▶ Zufällige, aber gültige Lösung OK!

`Thermal_equilibrium()`

Gleichgewicht auf einer Temperaturstufe

`New_temperature()`

Bestimmt nächsten Temperaturschritt

`Stop()`

Abbruchkriterium

BSF: *Best so far*

- ▶ Beste bisherige Lösung
- ▶ Letzte Lösung ist nicht immer die beste!



- ▶ Bei geeigneter Cooling Schedule



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung
 - ▶ Praktisch aber nicht relevant (zu langsam)



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung
 - ▶ Praktisch aber nicht relevant (zu langsam)
- ▶ Viele Variationsmöglichkeiten



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung
 - ▶ Praktisch aber nicht relevant (zu langsam)
- ▶ Viele Variationsmöglichkeiten
 - ▶ stop() abhängig von accept()



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung
 - ▶ Praktisch aber nicht relevant (zu langsam)
- ▶ Viele Variationsmöglichkeiten
 - ▶ stop() abhängig von accept()
 - ▶ Adaptive Cooling Schedules



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung
 - ▶ Praktisch aber nicht relevant (zu langsam)
- ▶ Viele Variationsmöglichkeiten
 - ▶ stop() abhängig von accept()
 - ▶ Adaptive Cooling Schedules
- ▶ Bibliotheken: ASA, EBSA



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung
 - ▶ Praktisch aber nicht relevant (zu langsam)
- ▶ Viele Variationsmöglichkeiten
 - ▶ stop() abhängig von accept()
 - ▶ Adaptive Cooling Schedules
- ▶ Bibliotheken: ASA, EBSA
- ▶ SA ist allgemein verwendbar



- ▶ Bei geeigneter Cooling Schedule
 - ▶ SA findet immer die optimale Lösung
 - ▶ Praktisch aber nicht relevant (zu langsam)
- ▶ Viele Variationsmöglichkeiten
 - ▶ stop() abhängig von accept()
 - ▶ Adaptive Cooling Schedules
- ▶ Bibliotheken: ASA, EBSA
- ▶ SA ist allgemein verwendbar
- ▶ Aber: Spezialisierte Lösungen sind besser



- ▶ Simulated Annealing



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden **immer** akzeptiert



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden immer akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden immer akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$
 - ▶ Auch wenn $c(\vec{g}) > c(\vec{f})!$



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden immer akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$
 - ▶ Auch wenn $c(\vec{g}) > c(\vec{f})!$
 - ▶ Problem: Zyklen



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden immer akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$
 - ▶ Auch wenn $c(\vec{g}) > c(\vec{f})!$
 - ▶ Problem: Zyklen
 - ▶ Ständige Wiederholung der letzten Züge



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden **immer** akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$
 - ▶ Auch wenn $c(\vec{g}) > c(\vec{f})!$
 - ▶ Problem: Zyklen
 - ▶ Ständige Wiederholung der letzten Züge
- ▶ Lösung: Verbiete letzten k Lösungen



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden **immer** akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$
 - ▶ Auch wenn $c(\vec{g}) > c(\vec{f})!$
 - ▶ Problem: Zyklen
 - ▶ Ständige Wiederholung der letzten Züge
- ▶ Lösung: Verbiete letzten k Lösungen
 - ▶ Lösungen sind als *tabu* markiert



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden **immer** akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$
 - ▶ Auch wenn $c(\vec{g}) > c(\vec{f})!$
 - ▶ Problem: Zyklen
 - ▶ Ständige Wiederholung der letzten Züge
- ▶ Lösung: Verbiete letzten k Lösungen
 - ▶ Lösungen sind als *tabu* markiert
 - ▶ Vermeidet Zyklen bis zu der Länge k



- ▶ Simulated Annealing
 - ▶ Verschlechternde Züge zu Beginn akzeptiert
- ▶ Tabu-Suche (TS)
 - ▶ Verschlechternde Züge werden **immer** akzeptiert
 - ▶ Gehe immer zu $\vec{g} \in N(\vec{f})$ mit
$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$
 - ▶ Auch wenn $c(\vec{g}) > c(\vec{f})!$
 - ▶ Problem: Zyklen
 - ▶ Ständige Wiederholung der letzten Züge
- ▶ Lösung: Verbiete letzten k Lösungen
 - ▶ Lösungen sind als *tabu* markiert
 - ▶ Vermeidet Zyklen bis zu der Länge k
 - ▶ Realisierung: FIFO mit Lösungen der Länge k



Tabu_search(): **begin**

```
Feasible_solution f, g, bsf ;
Set<Feasible_solution> G ;
FIFO<Feasible_solution,k> Q ;
f := Initial_solution(); bsf := f; Q :=  $\emptyset$  ;
repeat
  G := {s | s  $\in$  N(f)  $\wedge$  s  $\notin$  Q} ;
  if G  $\neq \emptyset$  then
    g := G.findmin(c) ;
    Q.shiftin(g) ;
    f := g ;
    if c(f) < c(bsf) then bsf := f;
until (G  $\neq \emptyset$ ) or Stop();
report(bsf);
```



- ▶ stop()
Keine Verbesserung in den letzten k Zügen



- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel



- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel

- ▶ 10.000 Zellen Lösung beschreibt 10.000 Koordinatenpaare



- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel

- ▶ 10.000 Zellen Lösung beschreibt 10.000 Koordinatenpaare
- ▶ Sehr große Tabu-Liste



- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel

- ▶ 10.000 Zellen Lösung beschreibt 10.000 Koordinatenpaare
- ▶ Sehr große Tabu-Liste
- ▶ Abhilfe: Setze nur einzelne Züge Tabu



- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel

- ▶ 10.000 Zellen Lösung beschreibt 10.000 Koordinatenpaare
- ▶ Sehr große Tabu-Liste
- ▶ Abhilfe: Setze nur einzelne Züge Tabu
- ▶ Aber: Einschränkung des Lösungsraumes



- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel

- ▶ 10.000 Zellen Lösung beschreibt 10.000 Koordinatenpaare
 - ▶ Sehr große Tabu-Liste
 - ▶ Abhilfe: Setze nur einzelne Züge Tabu
 - ▶ Aber: Einschränkung des Lösungsraumes
-
- ▶ Viele Variationsmöglichkeiten

- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel

- ▶ 10.000 Zellen Lösung beschreibt 10.000 Koordinatenpaare
 - ▶ Sehr große Tabu-Liste
 - ▶ Abhilfe: Setze nur einzelne Züge Tabu
 - ▶ Aber: Einschränkung des Lösungsraumes
- ▶ Viele Variationsmöglichkeiten
 - ▶ Kein theoretischer Hintergrund Erreichen des Optimums?



- ▶ stop()
Keine Verbesserung in den letzten k Zügen

UPP-Beispiel

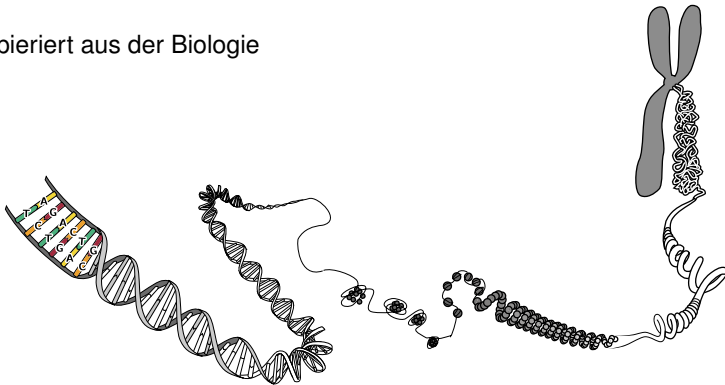
- ▶ 10.000 Zellen Lösung beschreibt 10.000 Koordinatenpaare
- ▶ Sehr große Tabu-Liste
- ▶ Abhilfe: Setze nur einzelne Züge Tabu
- ▶ Aber: Einschränkung des Lösungsraumes

- ▶ Viele Variationsmöglichkeiten
 - ▶ Kein theoretischer Hintergrund Erreichen des Optimums?
 - ▶ Wie stop() oder k wählen?

Genetische Algorithmen

Idee

- ▶ Inspiriert aus der Biologie

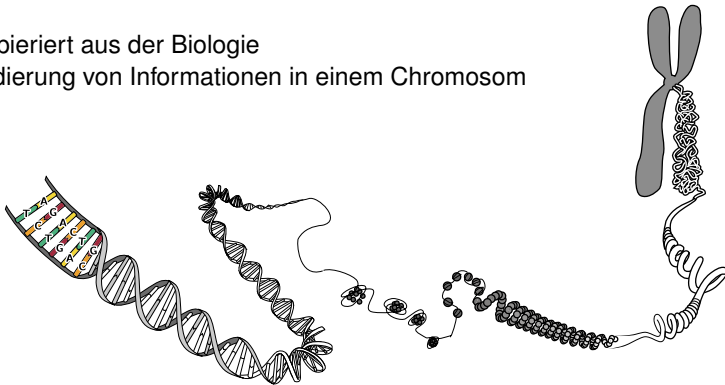


Basiert auf einem Bild der NIH

Genetische Algorithmen

Idee

- ▶ Inspiriert aus der Biologie
- ▶ Kodierung von Informationen in einem Chromosom

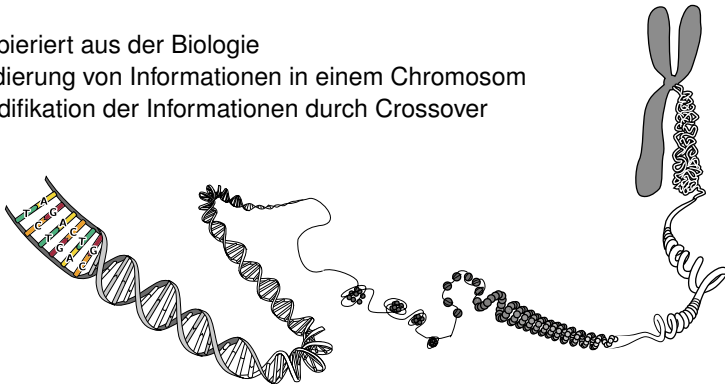


Basiert auf einem Bild der NIH

Genetische Algorithmen

Idee

- ▶ Inspiriert aus der Biologie
- ▶ Kodierung von Informationen in einem Chromosom
- ▶ Modifikation der Informationen durch Crossover



Basiert auf einem Bild der NIH

Genetische Algorithmen

Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung

Genetische Algorithmen

Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung
- ▶ Nun: Gleichzeitig auf *mehreren* Lösungen

Genetische Algorithmen

Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung
- ▶ Nun: Gleichzeitig auf *mehreren* Lösungen
 - ▶ Menge P von Lösungen: Population

Genetische Algorithmen

Idee



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung
- ▶ Nun: Gleichzeitig auf *mehreren* Lösungen
 - ▶ Menge P von Lösungen: Population
 - ▶ Iterationszähler k : Generation

Genetische Algorithmen

Idee

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung
- ▶ Nun: Gleichzeitig auf *mehreren* Lösungen
 - ▶ Menge P von Lösungen: Population
 - ▶ Iterationszähler k : Generation
 - ▶ Ersetze $P^{(k)}$ durch $P^{(k+1)}$ während Optimierung

Genetische Algorithmen

Idee

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung
- ▶ Nun: Gleichzeitig auf *mehreren* Lösungen
 - ▶ Menge P von Lösungen: Population
 - ▶ Iterationszähler k : Generation
 - ▶ Ersetze $P^{(k)}$ durch $P^{(k+1)}$ während Optimierung
- ▶ Bestimmung von $\vec{f}^{(k+1)} \in P^{(k+1)}$ mittels

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung
- ▶ Nun: Gleichzeitig auf *mehreren* Lösungen
 - ▶ Menge P von Lösungen: Population
 - ▶ Iterationszähler k : Generation
 - ▶ Ersetze $P^{(k)}$ durch $P^{(k+1)}$ während Optimierung
- ▶ Bestimmung von $\vec{f}^{(k+1)} \in P^{(k+1)}$ mittels
 - ▶ Crossover von zwei Eltern $\vec{f}^{(k)}, \vec{g}^{(k)} \in P^{(k)}$
Vererbung von Eigenschaften von $\vec{f}^{(k)}$ und $\vec{g}^{(k)}$

- ▶ Bisher: Algorithmen arbeiten auf *einer* vollständigen Lösung
- ▶ Nun: Gleichzeitig auf *mehreren* Lösungen
 - ▶ Menge P von Lösungen: Population
 - ▶ Iterationszähler k : Generation
 - ▶ Ersetze $P^{(k)}$ durch $P^{(k+1)}$ während Optimierung
- ▶ Bestimmung von $\vec{f}^{(k+1)} \in P^{(k+1)}$ mittels
 - ▶ Crossover von zwei Eltern $\vec{f}^{(k)}, \vec{g}^{(k)} \in P^{(k)}$
Vererbung von Eigenschaften von $\vec{f}^{(k)}$ und $\vec{g}^{(k)}$
 - ▶ Ggf. Mutation von $\vec{f}^{(k+1)}$

Genetische Algorithmen

Darstellung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

Genetische Algorithmen

Darstellung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

Genetische Algorithmen

Darstellung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster



- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate



- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate
 - ▶ 8 Bit pro Koordinatenpaar



- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate
 - ▶ 8 Bit pro Koordinatenpaar
 - ▶ 100×8 Bit = 800 Bit lange Bitfolge als Chromosom



- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate
 - ▶ 8 Bit pro Koordinatenpaar
 - ▶ $100 \times 8 \text{ Bit} = 800 \text{ Bit}$ lange Bitfolge als Chromosom
 - ▶ L Länge des Chromosoms in Bit



- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate
 - ▶ 8 Bit pro Koordinatenpaar
 - ▶ 100×8 Bit = 800 Bit lange Bitfolge als Chromosom
 - ▶ L Länge des Chromosoms in Bit
- ▶ Wichtige Unterscheidung zwischen



- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate
 - ▶ 8 Bit pro Koordinatenpaar
 - ▶ 100×8 Bit = 800 Bit lange Bitfolge als Chromosom
 - ▶ L Länge des Chromosoms in Bit
- ▶ Wichtige Unterscheidung zwischen
Lösung Biologie: Phänotyp



- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate
 - ▶ 8 Bit pro Koordinatenpaar
 - ▶ 100×8 Bit = 800 Bit lange Bitfolge als Chromosom
 - ▶ L Länge des Chromosoms in Bit
- ▶ Wichtige Unterscheidung zwischen
Lösung Biologie: Phänotyp
Kodierung der Lösung Biologie: Genotyp

- ▶ Kodierung für Chromosom
Bitfolge für Lösungsvektor

UPP Kodierung

- ▶ 100 Zellen, 10×10 Raster
 - ▶ 4 Bit pro Koordinate
 - ▶ 8 Bit pro Koordinatenpaar
 - ▶ 100×8 Bit = 800 Bit lange Bitfolge als Chromosom
 - ▶ L Länge des Chromosoms in Bit
- ▶ Wichtige Unterscheidung zwischen
Lösung Biologie: Phänotyp
Kodierung der Lösung Biologie: Genotyp
- ▶ Hier im Beispiel äquivalent benutzt

Genetische Algorithmen

Vererbung mit dem Crossover-Operator



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kombiniere die Bitfolgen der Eltern

Genetische Algorithmen

Vererbung mit dem Crossover-Operator



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kombiniere die Bitfolgen der Eltern
- ▶ Verschiedene Realisierungen möglich

Genetische Algorithmen

Vererbung mit dem Crossover-Operator



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kombiniere die Bitfolgen der Eltern
- ▶ Verschiedene Realisierungen möglich

Beispiel

Genetische Algorithmen

Vererbung mit dem Crossover-Operator



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kombiniere die Bitfolgen der Eltern
- ▶ Verschiedene Realisierungen möglich

Beispiel

1. Wähle zufällig Crossover-Position $1 \leq r \leq L$

Genetische Algorithmen

Vererbung mit dem Crossover-Operator



- ▶ Kombiniere die Bitfolgen der Eltern
- ▶ Verschiedene Realisierungen möglich

Beispiel

1. Wähle zufällig Crossover-Position $1 \leq r \leq L$
2. Kopiere Bits $1 \dots (r - 1)$ aus $\vec{f}^{(k)}$ nach $\vec{f}^{(k+1)}$

Genetische Algorithmen

Vererbung mit dem Crossover-Operator



- ▶ Kombiniere die Bitfolgen der Eltern
- ▶ Verschiedene Realisierungen möglich

Beispiel

1. Wähle zufällig Crossover-Position $1 \leq r \leq L$
2. Kopiere Bits $1 \dots (r - 1)$ aus $\vec{f}^{(k)}$ nach $\vec{f}^{(k+1)}$
3. Kopiere Bits $r \dots L$ aus $\vec{g}^{(k)}$ nach $\vec{f}^{(k+1)}$

Genetische Algorithmen

Vererbung mit dem Crossover-Operator

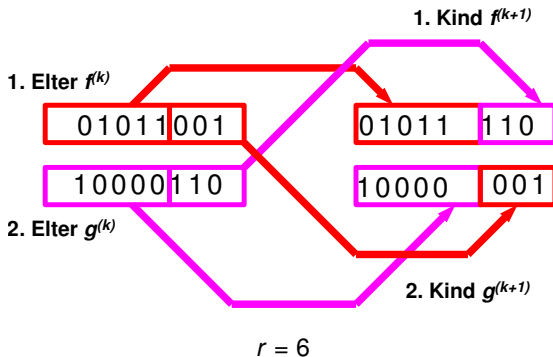


- ▶ Kombiniere die Bitfolgen der Eltern
- ▶ Verschiedene Realisierungen möglich

Beispiel

1. Wähle zufällig Crossover-Position $1 \leq r \leq L$
2. Kopiere Bits $1 \dots (r - 1)$ aus $\vec{f}^{(k)}$ nach $\vec{f}^{(k+1)}$
3. Kopiere Bits $r \dots L$ aus $\vec{g}^{(k)}$ nach $\vec{f}^{(k+1)}$
4. Ggf. erzeuge 2. Kind $\vec{g}^{(k+1)}$ mit vertauschten Rollen

10 × 10 Raster, platziere einzelne Zelle



Genetische Algorithmen

Problem



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Crossover erzeugt ungültige Lösungen

Genetische Algorithmen

Problem



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Crossover erzeugt ungültige Lösungen
- ⇒ Abhilfe: Mehr Struktur als einfache Bitfolge

Genetische Algorithmen

Problem



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Crossover erzeugt ungültige Lösungen
- ⇒ Abhilfe: Mehr Struktur als einfache Bitfolge

Bei UPP-Beispiel

- ▶ Crossover erzeugt ungültige Lösungen
- ⇒ Abhilfe: Mehr Struktur als einfache Bitfolge

Bei UPP-Beispiel

- ▶ Folgen von 4-bit Koordinaten



- ▶ Crossover erzeugt ungültige Lösungen
- ⇒ Abhilfe: Mehr Struktur als einfache Bitfolge

Bei UPP-Beispiel

- ▶ Folgen von 4-bit Koordinaten
- ▶ Nun zwar intern konsistente Koordinaten

- ▶ Crossover erzeugt ungültige Lösungen
- ⇒ Abhilfe: Mehr Struktur als einfache Bitfolge

Bei UPP-Beispiel

- ▶ Folgen von 4-bit Koordinaten
- ▶ Nun zwar intern konsistente Koordinaten
- ▶ Reicht im allgemeinen Fall aber nicht aus!



- ▶ Crossover erzeugt ungültige Lösungen
- ⇒ Abhilfe: Mehr Struktur als einfache Bitfolge

Bei UPP-Beispiel

- ▶ Folgen von 4-bit Koordinaten
- ▶ Nun zwar intern konsistente Koordinaten
- ▶ Reicht im allgemeinen Fall aber nicht aus!
- ⇒ Problemspezifisches Crossover



- ▶ Bisher noch keine Optimierung
Nur neue Lösungen erzeugt

Genetische Algorithmen

Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Bisher noch keine Optimierung
Nur neue Lösungen erzeugt
- ▶ Bevorzuge gute Lösungen vor schlechten



- ▶ Bisher noch keine Optimierung
Nur neue Lösungen erzeugt
- ▶ Bevorzuge gute Lösungen vor schlechten
 - ▶ Wähle *gute* Eltern aus: Niedrige Kosten



- ▶ Bisher noch keine Optimierung
Nur neue Lösungen erzeugt
- ▶ Bevorzuge gute Lösungen vor schlechten
 - ▶ Wähle *gute* Eltern aus: Niedrige Kosten
 - ▶ Kombiniere gute Eigenschaften in Nachwuchs



- ▶ Bisher noch keine Optimierung
Nur neue Lösungen erzeugt
- ▶ Bevorzuge gute Lösungen vor schlechten
 - ▶ Wähle *gute* Eltern aus: Niedrige Kosten
 - ▶ Kombiniere gute Eigenschaften in Nachwuchs
 - ▶ Aber: Auch Gegenteil möglich (Crossoverposition r zufällig)



- ▶ Bisher noch keine Optimierung
Nur neue Lösungen erzeugt
- ▶ Bevorzuge gute Lösungen vor schlechten
 - ▶ Wähle *gute* Eltern aus: Niedrige Kosten
 - ▶ Kombiniere gute Eigenschaften in Nachwuchs
 - ▶ Aber: Auch Gegenteil möglich (Crossoverposition r zufällig)
 - ▶ Vererbung schlechter Eigenschaften

- ▶ Bisher noch keine Optimierung
Nur neue Lösungen erzeugt
- ▶ Bevorzuge gute Lösungen vor schlechten
 - ▶ Wähle *gute* Eltern aus: Niedrige Kosten
 - ▶ Kombiniere gute Eigenschaften in Nachwuchs
 - ▶ Aber: Auch Gegenteil möglich (Crossoverposition r zufällig)
 - ▶ Vererbung schlechter Eigenschaften
 - ▶ Idee: Schlechte Nachkommen sterben in nächster Generation



Genetic_Algorithm(int popsize) : **begin**

Set<chromosome> pop, newpop ;

Chromosome parent1, parent2, child ;

pop := \emptyset ;

for $i:=1 \dots popsize$ **do** pop := pop \cup {Chromosom einer zufälligen Lösung} ;

repeat

 newpop := \emptyset ;

for $i:=1 \dots popsize$ **do**

 parent1 := pop.select() ; parent2 := pop.select() ;

 child := crossover(parent1, parent2) ;

 newpop := newpop \cup { child } ;

 pop := newpop;

until Stop();

report(pop.findmin(c));



▶ Stop()



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen



- ▶ `Stop()`
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftkommende Fehler beim Kopieren/durch Fremdeinwirkungen



- ▶ `Stop()`
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftkommende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftkommende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima
- ▶ Sehr viele Variationsmöglichkeiten



- ▶ `Stop()`
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftretende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima
- ▶ Sehr viele Variationsmöglichkeiten
 - ▶ Komplexere Crossover (mehrere r, \dots)



- ▶ `Stop()`
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftretende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima
- ▶ Sehr viele Variationsmöglichkeiten
 - ▶ Komplexere Crossover (mehrere r, \dots)
 - ▶ Zusätzliche Funktionen: Mutation, Inversion



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftretende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima
- ▶ Sehr viele Variationsmöglichkeiten
 - ▶ Komplexere Crossover (mehrere r, \dots)
 - ▶ Zusätzliche Funktionen: Mutation, Inversion
 - ▶ Mehrere Generationen gleichzeitig



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftretende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima
- ▶ Sehr viele Variationsmöglichkeiten
 - ▶ Komplexere Crossover (mehrere r, \dots)
 - ▶ Zusätzliche Funktionen: Mutation, Inversion
 - ▶ Mehrere Generationen gleichzeitig
 - ▶ Elite-Selektion



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftretende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima
- ▶ Sehr viele Variationsmöglichkeiten
 - ▶ Komplexere Crossover (mehrere r, \dots)
 - ▶ Zusätzliche Funktionen: Mutation, Inversion
 - ▶ Mehrere Generationen gleichzeitig
 - ▶ Elite-Selektion
 - ▶ Meta-Genetische Algorithmen



- ▶ Stop()
 - ▶ Keine Verbesserung in den letzten m Iterationen
 - ▶ m problemspezifischer Parameter
- ▶ Mutation
 - ▶ Modelliert in Natur auftretende Fehler beim Kopieren/durch Fremdeinwirkungen
 - ▶ Vermeidet Steckenbleiben in lokalen Minima
- ▶ Sehr viele Variationsmöglichkeiten
 - ▶ Komplexere Crossover (mehrere r, \dots)
 - ▶ Zusätzliche Funktionen: Mutation, Inversion
 - ▶ Mehrere Generationen gleichzeitig
 - ▶ Elite-Selektion
 - ▶ Meta-Genetische Algorithmen

...

Stop(): 10.000 Generationen ohne Verbesserung der BSF-Lösung

Initiale Population: Anzahl: Hunderte

25% Zufällig, 75% sequentiell angeordnet

.select(): Zufällig mit Gewichtung auf *fitness* ($< \infty$ nie)

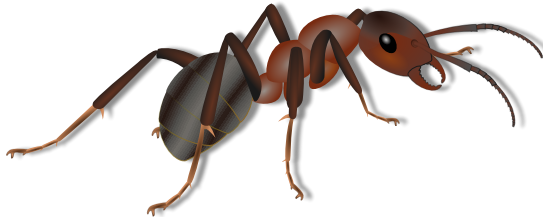
Resultat: Ähnlich gut wie TimberWolf

(Tendenziell bessere Ergebnisse auf Kosten der Laufzeit)

Ameisenalgorithmus

Idee

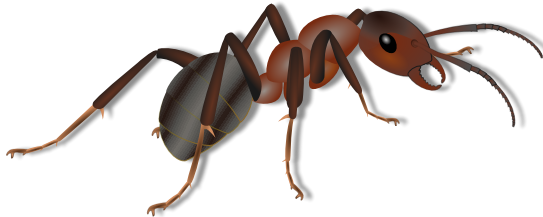
- ▶ Inspiriert von Ameisenpfaden



Ameisenalgorithmus

Idee

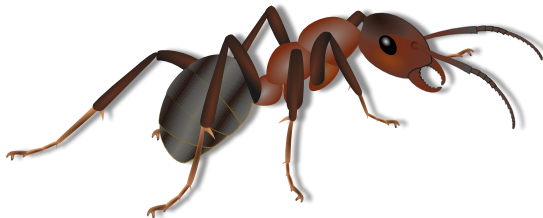
- ▶ Inspiriert von Ameisenpfaden
- ▶ Ameisen hinterlassen beim marschieren Pheromone



Ameisenalgorithmus

Idee

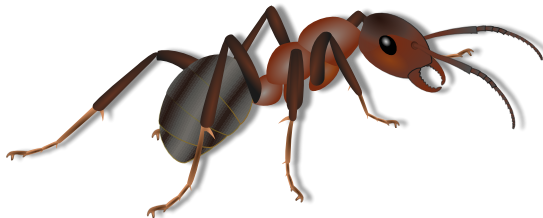
- ▶ Inspiriert von Ameisenpfaden
- ▶ Ameisen hinterlassen beim marschieren Pheromone
- ▶ Kurze Wege akkumulieren stärkere Pheromonkonzentration



Ameisenalgorithmus

Idee

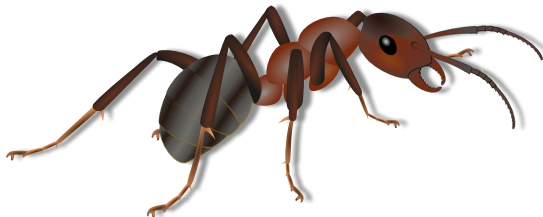
- ▶ Inspiriert von Ameisenpfaden
 - ▶ Ameisen hinterlassen beim marschieren Pheromone
 - ▶ Kurze Wege akkumulieren stärkere Pheromonkonzentration
- ⇒ werden entsprechend attraktiver für andere Ameisen



Ameisenalgorithmus

Idee

- ▶ Inspiriert von Ameisenpfaden
- ▶ Ameisen hinterlassen beim marschieren Pheromone
- ▶ Kurze Wege akkumulieren stärkere Pheromonkonzentration
- ⇒ werden entsprechend attraktiver für andere Ameisen
- ▶ Agentenbasierter Ansatz: Finden von besten Wegen im Graph



Ameisenalgorithmus

Anwendung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

Ameisenalgorithmus

Anwendung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

Ameisenalgorithmus

Anwendung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

Ameisenalgorithmus

Anwendung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

$d(l_1, l_2)$ Distanz zwischen zwei Orten ($d : L \times L \mapsto R$)

- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

$d(l_1, l_2)$ Distanz zwischen zwei Orten ($d : L \times L \mapsto R$)

$w(f_1, f_2)$ Fluß zwischen zwei Einrichtungen ($w : F \times F \mapsto R$)

Ameisenalgorithmus

Anwendung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

$d(l_1, l_2)$ Distanz zwischen zwei Orten ($d : L \times L \mapsto R$)

$w(f_1, f_2)$ Fluß zwischen zwei Einrichtungen ($w : F \times F \mapsto R$)

Ameisenalgorithmus

Anwendung



- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

$d(l_1, l_2)$ Distanz zwischen zwei Orten ($d : L \times L \mapsto R$)

$w(f_1, f_2)$ Fluß zwischen zwei Einrichtungen ($w : F \times F \mapsto R$)

Gesucht: Zuordnung $a : F \mapsto L$ bei der $\sum_{f_1, f_2 \in F} w(f_1, f_2) \cdot d(a(f_1), a(f_2))$ minimal wird

Ameisenalgorithmus

Anwendung



- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

$d(l_1, l_2)$ Distanz zwischen zwei Orten ($d : L \times L \mapsto R$)

$w(f_1, f_2)$ Fluß zwischen zwei Einrichtungen ($w : F \times F \mapsto R$)

Gesucht: Zuordnung $a : F \mapsto L$ bei der $\sum_{f_1, f_2 \in F} w(f_1, f_2) \cdot d(a(f_1), a(f_2))$ minimal wird

- ▶ Lösung wird iterativ in jedem *Agenten* (Ameise) aufgebaut

Ameisenalgorithmus

Anwendung



- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

$d(l_1, l_2)$ Distanz zwischen zwei Orten ($d : L \times L \mapsto R$)

$w(f_1, f_2)$ Fluß zwischen zwei Einrichtungen ($w : F \times F \mapsto R$)

Gesucht: Zuordnung $a : F \mapsto L$ bei der $\sum_{f_1, f_2 \in F} w(f_1, f_2) \cdot d(a(f_1), a(f_2))$ minimal wird

- ▶ Lösung wird iterativ in jedem *Agenten* (Ameise) aufgebaut
- ▶ Jede Ameise läuft zufälligen Weg
Wahrscheinlichkeit bei Weg mit vielen Pheromonen höher

Ameisenalgorithmus

Anwendung



- ▶ Oft benutzt bei *TSP* (Travelling-Salesman-Problem) oder QAP

QAP (Quadratic-Assignment-Problem)

Gegeben:

$f \in F$ Einrichtungen

$l \in L$ Orte

$d(l_1, l_2)$ Distanz zwischen zwei Orten ($d : L \times L \mapsto R$)

$w(f_1, f_2)$ Fluß zwischen zwei Einrichtungen ($w : F \times F \mapsto R$)

Gesucht: Zuordnung $a : F \mapsto L$ bei der $\sum_{f_1, f_2 \in F} w(f_1, f_2) \cdot d(a(f_1), a(f_2))$ minimal wird

- ▶ Lösung wird iterativ in jedem *Agenten* (Ameise) aufgebaut
- ▶ Jede Ameise läuft zufälligen Weg
Wahrscheinlichkeit bei Weg mit vielen Pheromonen höher
- ▶ Nach Konstruktion des Wege, entsprechend der Kosten Pheromongehalt der Wege aller Ameisen aktualisieren

Ameisenalgorithmus

UPP-Beispiel



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Zellen \mapsto Einrichtungen

Ameisenalgorithmus

UPP-Beispiel



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Zellen \mapsto Einrichtungen
- ▶ Orte \mapsto Positionen im Raster

Ameisenalgorithmus

UPP-Beispiel



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Zellen \mapsto Einrichtungen
- ▶ Orte \mapsto Positionen im Raster
- ▶ Fluß \mapsto Anzahl der adjazenten Zellen

Ameisenalgorithmus

UPP-Beispiel



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Zellen \mapsto Einrichtungen
- ▶ Orte \mapsto Positionen im Raster
- ▶ Fluß \mapsto Anzahl der adjazenten Zellen
- ▶ Distanz \mapsto Distanz

Ameisenalgorithmus

Algorithmus



Ant_colony_optimization:

begin

 Init_heuristics() ;

repeat

forall *Ants* *i* **do**

 └ Ant *i*.generate_solution()

 Update_pheromons() ;

until *Stop()*;

Ant.generate_solution():

begin

for *NumberCells* **do**

 Select_cell() ;

 └ Assign_cell_to_location()

 Improve_solution() ;

if *solution* < *bsf* **then**

 └ *bsf* := *solution*

Stop(): Abbruchkriterium: Anzahl Iterationen abhängig von Problemgröße, keine Änderung der BSF, ...

Improve_solution(): Lokale Suche zum verbessern des Ergebnisses

Select_cell(): Zufällig, feste Reihenfolge

(z.B. sortiert nach Summe der inzidenten Flüße)

Ameisenalgorithmus

Initialisierung



► Heuristische Information

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}$$

f_i *Flußpotenzial* der Zelle (Anzahl verbundener Zellen)

d_j *Distanzpotenzial* einer Position

(Summe der Distanzen zu allen anderen Positionen)

Ameisenalgorithmus

Initialisierung



- ▶ Heuristische Information

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}$$

f_i *Flußpotenzial* der Zelle (Anzahl verbundener Zellen)

d_j *Distanzpotenzial* einer Position

(Summe der Distanzen zu allen anderen Positionen)

- ▶ Setzen sonstiger Parameter:

Ameisenalgorithmus

Initialisierung



- ▶ Heuristische Information

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}$$

f_i *Flußpotential* der Zelle (Anzahl verbundener Zellen)

d_j *Distanzpotential* einer Position

(Summe der Distanzen zu allen anderen Positionen)

- ▶ Setzen sonstiger Parameter:

k Anzahl Ameisen

Ameisenalgorithmus

Initialisierung



- ▶ Heuristische Information

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}$$

f_i *Flußpotential* der Zelle (Anzahl verbundener Zellen)

d_j *Distanzpotential* einer Position

(Summe der Distanzen zu allen anderen Positionen)

- ▶ Setzen sonstiger Parameter:

k Anzahl Ameisen

Q Wert der Pheromonenerhöhung (abhängig von Zielfunktion)

Ameisenalgorithmus

Initialisierung

- ▶ Heuristische Information

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}$$

f_i *Flußpotential* der Zelle (Anzahl verbundener Zellen)

d_j *Distanzpotential* einer Position

(Summe der Distanzen zu allen anderen Positionen)

- ▶ Setzen sonstiger Parameter:

k Anzahl Ameisen

Q Wert der Pheromonenerhöhung (abhängig von Zielfunktion)

$\alpha, \beta > 0$ Gewichtung der Pheromone (α) und heuristischen Information (β)

Ameisenalgorithmus

Initialisierung

- ▶ Heuristische Information

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}$$

f_i *Flußpotential* der Zelle (Anzahl verbundener Zellen)

d_j *Distanzpotential* einer Position

(Summe der Distanzen zu allen anderen Positionen)

- ▶ Setzen sonstiger Parameter:

k Anzahl Ameisen

Q Wert der Pheromonenerhöhung (abhängig von Zielfunktion)

$\alpha, \beta > 0$ Gewichtung der Pheromone (α) und heuristischen Information (β)

ρ Persistenz der Pheromone

Ameisenalgorithmus

Initialisierung

- ▶ Heuristische Information

$$\eta_{ij} = \frac{1}{f_i \cdot d_j}$$

f_i *Flußpotential* der Zelle (Anzahl verbundener Zellen)

d_j *Distanzpotential* einer Position

(Summe der Distanzen zu allen anderen Positionen)

- ▶ Setzen sonstiger Parameter:

k Anzahl Ameisen

Q Wert der Pheromonerhöhung (abhängig von Zielfunktion)

$\alpha, \beta > 0$ Gewichtung der Pheromone (α) und heuristischen Information (β)

ρ Persistenz der Pheromone

τ_{ij} Pheromone, Initialisierung mit Wert > 0

Ameisenalgorithmus

Zuweisung Zelle \mapsto Position

Wahrscheinlichkeit das Zelle i Position j zugewiesen wird
(in Iteration t der Ameise k):

$$p_{ij}^k = \frac{\tau_{ij}(t)^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}(t)^\alpha \cdot \eta_{il}^\beta}$$

- ▶ N_i^k ist die Nachbarschaft von Knoten i

Ameisenalgorithmus

Zuweisung Zelle \mapsto Position

Wahrscheinlichkeit das Zelle i Position j zugewiesen wird
(in Iteration t der Ameise k):

$$p_{ij}^k = \frac{\tau_{ij}(t)^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}(t)^\alpha \cdot \eta_{il}^\beta}$$

- ▶ N_i^k ist die Nachbarschaft von Knoten i
- ▶ $\sum_{l \in N_i^k} p_{il}(t) = 1$

Ameisenalgorithmus

Zuweisung Zelle \mapsto Position

Wahrscheinlichkeit das Zelle i Position j zugewiesen wird
(in Iteration t der Ameise k):

$$p_{ij}^k = \frac{\tau_{ij}(t)^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}(t)^\alpha \cdot \eta_{il}^\beta}$$

- ▶ N_i^k ist die Nachbarschaft von Knoten i
- ▶ $\sum_{l \in N_i^k} p_{il}(t) = 1$
- ▶ τ_{ij} Pheromonwerte, η_{ij} heuristische Information



$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^m \Delta\tau_{ij}^k$$

Mit

$$\Delta\tau_{ij}^k = \begin{cases} Q/\text{cost}(\text{solution}) & \text{Zelle } i \text{ ist Position } j \text{ zugeordnet} \\ 0 & \text{sonst} \end{cases}$$

Q Menge der Pheromone einer Ameise

$0 < \rho < 1$ Persistenz der Pheromone ($(1 - \rho)$ Evaporation)
Modelliert verdampfen von Pheromonen (gegen lokale Minima)

Ameisenalgorithmus

Zusammenfassung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel

Ameisenalgorithmus

Zusammenfassung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung

Ameisenalgorithmus

Zusammenfassung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung
- ▶ Qualität und Laufzeit vergleichbar mit Simulated Annealing

Ameisenalgorithmus

Zusammenfassung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung
- ▶ Qualität und Laufzeit vergleichbar mit Simulated Annealing
- ▶ Ähnlich zu SA: Theoretisch optimal

Ameisenalgorithmus

Zusammenfassung



- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung
- ▶ Qualität und Laufzeit vergleichbar mit Simulated Annealing
- ▶ Ähnlich zu SA: Theoretisch optimal
- ▶ Kann dynamisch auf Graphänderungen reagieren

Ameisenalgorithmus

Zusammenfassung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung
- ▶ Qualität und Laufzeit vergleichbar mit Simulated Annealing
- ▶ Ähnlich zu SA: Theoretisch optimal
- ▶ Kann dynamisch auf Graphänderungen reagieren
- ▶ Viele Varianten: AS, ANTS, MAX-MIN, FANT, HAS-QAP, ... Hauptunterschied sind andere ...

Ameisenalgorithmus

Zusammenfassung

- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung
- ▶ Qualität und Laufzeit vergleichbar mit Simulated Annealing
- ▶ Ähnlich zu SA: Theoretisch optimal
- ▶ Kann dynamisch auf Graphänderungen reagieren
- ▶ Viele Varianten: AS, ANTS, MAX-MIN, FANT, HAS-QAP, ...
Hauptunterschied sind andere ...
 - ▶ heuristische Informationen

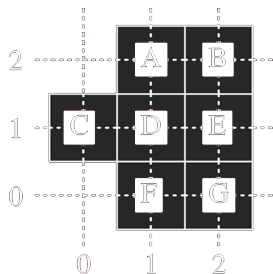
- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung
- ▶ Qualität und Laufzeit vergleichbar mit Simulated Annealing
- ▶ Ähnlich zu SA: Theoretisch optimal
- ▶ Kann dynamisch auf Graphänderungen reagieren
- ▶ Viele Varianten: AS, ANTS, MAX-MIN, FANT, HAS-QAP, ...
Hauptunterschied sind andere ...
 - ▶ heuristische Informationen
 - ▶ Pheromon Updates



- ▶ Schlechte Konvergenz in der Nähe des Optimums, da kaum ein Unterschied im Pheromonlevel
- ▶ Schlechte Skalierung
- ▶ Qualität und Laufzeit vergleichbar mit Simulated Annealing
- ▶ Ähnlich zu SA: Theoretisch optimal
- ▶ Kann dynamisch auf Graphänderungen reagieren
- ▶ Viele Varianten: AS, ANTS, MAX-MIN, FANT, HAS-QAP, ...
Hauptunterschied sind andere ...
 - ▶ heuristische Informationen
 - ▶ Pheromon Updates
 - ▶ Wahrscheinlichkeitsfunktion

Ameisenalgorithmus

Beispiel UPP



n_1 : A, B, F, G

n_2 : B, E

n_3 : D, E

n_4 : A, C, D

n_5 : C, D, F

n_6 : C, E, F, G

n_7 : D, F

n_8 : F, G

$$\vec{f} = \begin{pmatrix} 5 \\ 4 \\ 7 \\ 6 \\ 5 \\ 10 \\ 7 \end{pmatrix}, \vec{d} = \begin{pmatrix} 18 \\ 15 \\ 18 \\ 15 \\ 12 \\ 15 \\ 18 \\ 15 \\ 18 \end{pmatrix}$$

Zellen sortiert nach Flußpotential: F, C, G, D, A, E, B

Ameisenalgorithmus

Beispiel UPP – Init

$$\eta = \begin{pmatrix} 0.0111 & 0.0139 & 0.0079 & 0.0093 & 0.0111 & 0.0056 & 0.0079 \\ 0.0133 & 0.0167 & 0.0095 & 0.0111 & 0.0133 & 0.0067 & 0.0095 \\ 0.0111 & 0.0139 & 0.0079 & 0.0093 & 0.0111 & 0.0056 & 0.0079 \\ 0.0133 & 0.0167 & 0.0095 & 0.0111 & 0.0133 & 0.0067 & 0.0095 \\ 0.0167 & 0.0208 & 0.0119 & 0.0139 & 0.0167 & 0.0083 & 0.0119 \\ 0.0133 & 0.0167 & 0.0095 & 0.0111 & 0.0133 & 0.0067 & 0.0095 \\ 0.0111 & 0.0139 & 0.0079 & 0.0093 & 0.0111 & 0.0056 & 0.0079 \\ 0.0133 & 0.0167 & 0.0095 & 0.0111 & 0.0133 & 0.0067 & 0.0095 \\ 0.0111 & 0.0139 & 0.0079 & 0.0093 & 0.0111 & 0.0056 & 0.0079 \end{pmatrix}$$

$$Q = 2, \alpha = \beta = 1, \rho = 0.8$$

Ameisenalgorithmus

Beispiel UPP – Init

$$\tau = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$Q = 2, \alpha = \beta = 1, \rho = 0.8$$

Ameisenalgorithmus

Beispiel UPP – Auswahl

$$p_{ij} = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{l \in N_i^k} \tau_{il} \cdot \eta_{il}}$$

Platziere Zelle F

η_{Fi}	0.0056	0.0067	0.0056	0.0067	0.0083	0.0067	0.0056	0.0067	0.0056
τ_{Fi}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
$\tau_{Fi} \cdot \eta_{Fi}$	0.00056	0.00067	0.00056	0.00067	0.00083	0.00067	0.00056	0.00067	0.00056

$$\Sigma = 0.00575$$

p_{Fi}	0.0973	0.1165	0.0973	0.1165	0.1443	0.1165	0.0973	0.1165	0.0973
----------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Ameisenalgorithmus

Beispiel UPP – Auswahl

$$p_{ij} = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{l \in N_i^k} \tau_{il} \cdot \eta_{il}}$$

Platziere Zelle C

η_{Ci}	0.0079	0.0095	0.0079	0.0095	0.0079	0.0095	0.0079	0.0095	0.0079
τ_{Ci}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
$\tau_{Ci} \cdot \eta_{Ci}$	0.00079	0.00095	0.00079	0.00095	0.00079	0.00095	0.00079	0.00095	0.00079

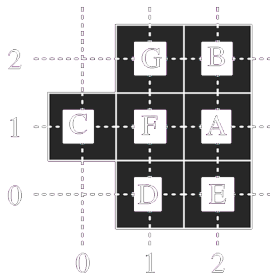
$$\Sigma = 0.00696$$

p_{Ci}	0.1135	0.1365	0.1135	0.1365	0.1135	0.1365	0.1135	0.1365	0.1135
----------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Ameisenalgorithmus

Beispiel UPP

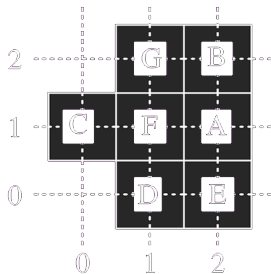
- ▶ Analog G, D, A, E, B



Ameisenalgorithmus

Beispiel UPP

- ▶ Analog G, D, A, E, B

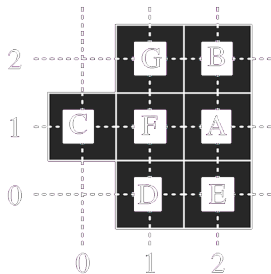


- ▶ Kosten (HPWL): 24

Ameisenalgorithmus

Beispiel UPP

- ▶ Analog G, D, A, E, B



- ▶ Kosten (HPWL): 24
- ▶ Nächster Schritt: Pheromone aktualisieren

Ameisenalgorithmus

Beispiel UPP – Update

$$\tau = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

Ameisenalgorithmus

Beispiel UPP – Update

$$\tau = \begin{pmatrix} 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \end{pmatrix}$$

$\rho = 0.8$ Evaporation

Ameisenalgorithmus

Beispiel UPP – Update



$$\tau = \begin{pmatrix} 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.163 \\ 0.08 & 0.163 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.163 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.163 & 0.08 \\ 0.163 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.163 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.163 & 0.08 & 0.08 \end{pmatrix}$$

$\frac{2}{24}$ auf genommenen Pfad hinzufügen



- ▶ Erschöpfende Suche



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking
- ▶ Eliminierung *schlechter* Ansätze



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking
- ▶ Eliminierung *schlechter* Ansätze
 - ▶ Abschätzung aus Teillösung



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking
- ▶ Eliminierung *schlechter* Ansätze
 - ▶ Abschätzung aus Teillösung
 - ▶ Beispiel: Branch-and-Bound



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking
- ▶ Eliminierung *schlechter* Ansätze
 - ▶ Abschätzung aus Teillösung
 - ▶ Beispiel: Branch-and-Bound
- ▶ Wiederverwendung alter Ergebnisse



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking
- ▶ Eliminierung *schlechter* Ansätze
 - ▶ Abschätzung aus Teillösung
 - ▶ Beispiel: Branch-and-Bound
- ▶ Wiederverwendung alter Ergebnisse
 - ▶ Beispiel: Dynamic Programming



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking
- ▶ Eliminierung *schlechter* Ansätze
 - ▶ Abschätzung aus Teillösung
 - ▶ Beispiel: Branch-and-Bound
- ▶ Wiederverwendung alter Ergebnisse
 - ▶ Beispiel: Dynamic Programming
- ▶ Mathematisches Modell



- ▶ Erschöpfende Suche
 - ▶ Durchlaufen des gesamten Lösungsraums
 - ▶ Beispiel: Backtracking
- ▶ Eliminierung *schlechter* Ansätze
 - ▶ Abschätzung aus Teillösung
 - ▶ Beispiel: Branch-and-Bound
- ▶ Wiederverwendung alter Ergebnisse
 - ▶ Beispiel: Dynamic Programming
- ▶ Mathematisches Modell
 - ▶ Beispiel: Linear Programming

- ▶ Den gesamten Lösungsraum enumerieren

```
float bsfCost ;  
solution_element current[n], bsf[n] ;  
main(): begin  
    bsfCost :=  $\infty$  ;  
    backtrack(0) ;  
    report(bsf) ;
```

```
backtrack(int k): begin  
    float newCost ;  
    solution_element sol_el ;  
    if  $k = n$  then  
        new_cost := cost(current) ;  
        if  $newCost < bsfCost$  then  
            bsfCost := newCost ;  
            bsf := copy(bsf) ;  
    else  
        foreach ( $sol\_el \in$   
             $allowed(current, k)$ ) do  
            current[k] := sol_el ;  
            backtrack(k+1) ;
```


- ▶ Den gesamten Lösungsraum enumerieren
- ▶ Jede Lösung wird dabei schrittweise aufgebaut (f_1, f_2, \dots)

```
float bsfCost ;  
solution_element current[n], bsf[n] ;  
main(): begin  
    bsfCost :=  $\infty$  ;  
    backtrack(0) ;  
    report(bsf) ;
```

```
backtrack(int k): begin  
    float newCost ;  
    solution_element sol_el ;  
    if  $k = n$  then  
        new_cost := cost(current) ;  
        if  $newCost < bsfCost$  then  
            bsfCost := newCost ;  
            bsf := copy(bsf) ;  
    else  
        foreach ( $sol\_el \in$   
             $allowed(current, k)$ ) do  
            current[k] := sol_el ;  
            backtrack(k+1) ;
```

- ▶ Den gesamten Lösungsraum enumerieren
- ▶ Jede Lösung wird dabei schrittweise aufgebaut (f_1, f_2, \dots)
- ▶ Zu jeder Lösung die Kosten bestimmen und ggf. BSF aktualisieren

```
float bsfCost ;  
solution_element current[n], bsf[n] ;  
main(): begin  
    bsfCost :=  $\infty$  ;  
    backtrack(0) ;  
    report(bsf) ;
```

```
backtrack(int k): begin  
    float newCost ;  
    solution_element sol_el ;  
    if  $k = n$  then  
        new_cost := cost(current) ;  
        if  $newCost < bsfCost$  then  
            bsfCost := newCost ;  
            bsf := copy(bsf) ;  
    else  
        foreach ( $sol\_el \in$   
             $allowed(current, k)$ ) do  
            current[k] := sol_el ;  
            backtrack(k+1) ;
```

- ▶ Den gesamten Lösungsraum enumerieren
- ▶ Jede Lösung wird dabei schrittweise aufgebaut (f_1, f_2, \dots)
- ▶ Zu jeder Lösung die Kosten bestimmen und ggf. BSF aktualisieren
- ▶ Üblicherweise: Rekursiv

```
float bsfCost ;  
solution_element current[n], bsf[n] ;  
main(): begin  
    bsfCost :=  $\infty$  ;  
    backtrack(0) ;  
    report(bsf) ;
```

```
backtrack(int k): begin  
    float newCost ;  
    solution_element sol_el ;  
    if  $k = n$  then  
        new_cost := cost(current) ;  
        if  $newCost < bsfCost$  then  
            bsfCost := newCost ;  
            bsf := copy(bsf) ;  
    else  
        foreach ( $sol\_el \in$   
             $allowed(current, k)$ ) do  
            current[k] := sol_el ;  
            backtrack(k+1) ;
```

Backtracking

UPP Beispiel



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:

$$\vec{f} = [f_1, \dots, f_n]^T$$

Zelle i wird auf Position f_i platziert

Backtracking

UPP Beispiel



- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$

Backtracking

UPP Beispiel



- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:

$$\vec{f} = [f_1, \dots, f_n]^T$$

Zelle i wird auf Position f_i platziert

- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$

Backtracking

UPP Beispiel



- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$
- ▶ Konkretes Beispiel: $n = 3, m = 4$, HPWL-Kosten

Backtracking

UPP Beispiel

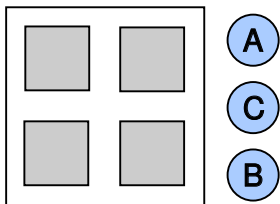


- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$
- ▶ Konkretes Beispiel: $n = 3, m = 4$, HPWL-Kosten

Backtracking

UPP Beispiel

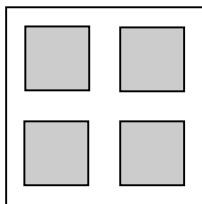
- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$
- ▶ Konkretes Beispiel: $n = 3, m = 4$, HPWL-Kosten



Backtracking

UPP Beispiel

- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$
- ▶ Konkretes Beispiel: $n = 3, m = 4$, HPWL-Kosten

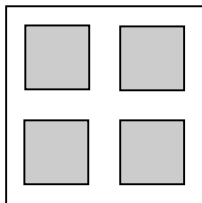


Netzliste:

Backtracking

UPP Beispiel

- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$
- ▶ Konkretes Beispiel: $n = 3, m = 4$, HPWL-Kosten



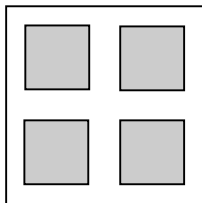
Netzliste:

n_1 : A, B, C

Backtracking

UPP Beispiel

- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$
- ▶ Konkretes Beispiel: $n = 3, m = 4$, HPWL-Kosten



A

C

B

Netzliste:

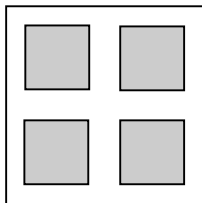
n_1 : A, B, C

n_2 : A, B

Backtracking

UPP Beispiel

- ▶ Lösung: Bei n Zellen auf m möglichen Positionen:
 $\vec{f} = [f_1, \dots, f_n]^T$
Zelle i wird auf Position f_i platziert
- ▶ Explizite Einschränkungen: $f_i \in 1, \dots, m$
- ▶ Implizite Einschränkungen: $i \neq j \Rightarrow f_i \neq f_j$
- ▶ Konkretes Beispiel: $n = 3, m = 4$, HPWL-Kosten



A

C

B

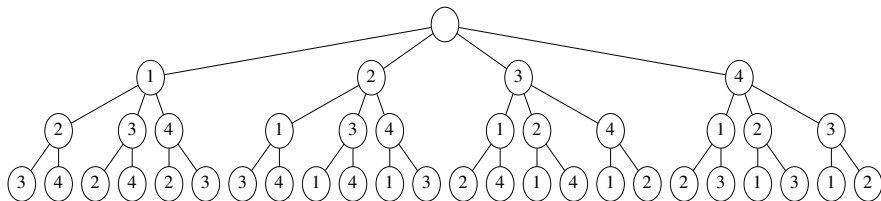
Netzliste:




n_1 : A, B, C




n_2 : A, B




n_3 : B, C

Backtracking UPP Suchbaum



Lsg.	\vec{f}	Kosten
	123	5
	124	5
	132	5

	134	4
	142	5
	143	5

	213	4
	214	5
	124	5
⋮	⋮	⋮



- ▶ Branch = Verzweigen des Suchbaums



- ▶ Branch = Verzweigen des Suchbaums
- ▶ Bound = Beschneiden der Äste wenn möglich



- ▶ Branch = Verzweigen des Suchbaums
- ▶ Bound = Beschneiden der Äste wenn möglich
- ▶ Teillösung $\underline{f}^{(k)} = [f_1, \dots, f_k, \perp, \dots, \perp]$



- ▶ Branch = Verzweigen des Suchbaums
- ▶ Bound = Beschneiden der Äste wenn möglich
- ▶ Teillösung $\underline{f}^{(k)} = [f_1, \dots, f_k, \perp, \dots, \perp]$
- ▶ $D(\underline{f}^{(k)})$: Menge aus $\underline{f}^{(k)}$ herleitbarer Lösungen



- ▶ Branch = Verzweigen des Suchbaums
- ▶ Bound = Beschneiden der Äste wenn möglich
- ▶ Teillösung $\vec{f}^{(k)} = [f_1, \dots, f_k, \perp, \dots, \perp]$
- ▶ $D(\vec{f}^{(k)})$: Menge aus $\vec{f}^{(k)}$ herleitbarer Lösungen
- ▶ Minimale mögliche Kosten cost einer Teillösung $\vec{f}^{(k)}$ abschätzen
Verwerfe $\vec{f}^{(k)}$ falls $\text{cost}(\vec{f}^{(k)}) > \text{cost}(BSF)$
⇒ Suchbaum wird gestutzt

Branch-and-Bound

Abschätzfunktion cost



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Effekt der Abschätzung

Branch-and-Bound

Abschätzfunktion cost



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten

Branch-and-Bound

Abschätzfunktion cost



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)

Branch-and-Bound

Abschätzfunktion cost



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte

Branch-and-Bound

Abschätzfunktion cost



- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte
 - ▶ Reale Kosten niedriger als geschätzte Kosten

Branch-and-Bound

Abschätzfunktion cost



- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte
 - ▶ Reale Kosten niedriger als geschätzte Kosten
 - ⇒ Zu pessimistisch (“Nein, das bringt nichts mehr”)

Branch-and-Bound

Abschätzfunktion cost



- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte
 - ▶ Reale Kosten niedriger als geschätzte Kosten
 - ⇒ Zu pessimistisch (“Nein, das bringt nichts mehr”)
 - ⇒ Optimum wird möglicherweise weggestutzt

Branch-and-Bound

Abschätzfunktion cost



- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte
 - ▶ Reale Kosten niedriger als geschätzte Kosten
 - ⇒ Zu pessimistisch (“Nein, das bringt nichts mehr”)
 - ⇒ Optimum wird möglicherweise weggestutzt
 - ⇒ **Keine exakte Lösung mehr!**

Branch-and-Bound

Abschätzfunktion cost

- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte
 - ▶ Reale Kosten niedriger als geschätzte Kosten
 - ⇒ Zu pessimistisch (“Nein, das bringt nichts mehr”)
 - ⇒ Optimum wird möglicherweise weggestutzt
 - ⇒ Keine exakte Lösung mehr!
 - ▶ cost sollte möglichst genau sein

Branch-and-Bound

Abschätzfunktion cost



- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte
 - ▶ Reale Kosten niedriger als geschätzte Kosten
 - ⇒ Zu pessimistisch (“Nein, das bringt nichts mehr”)
 - ⇒ Optimum wird möglicherweise weggestutzt
 - ⇒ Keine exakte Lösung mehr!
 - ▶ cost sollte möglichst genau sein

Branch-and-Bound

Abschätzfunktion cost



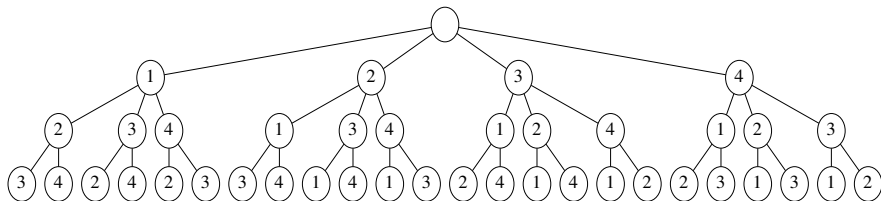
- ▶ Effekt der Abschätzung
 - ▶ Reale Kosten höher als geschätzte Kosten
 - ⇒ Zu optimistisch (“Ja, es lohnt sich weiterzumachen”)
 - ⇒ Überflüssige Schritte
 - ▶ Reale Kosten niedriger als geschätzte Kosten
 - ⇒ Zu pessimistisch (“Nein, das bringt nichts mehr”)
 - ⇒ Optimum wird möglicherweise weggestutzt
 - ⇒ Keine exakte Lösung mehr!
 - ▶ cost sollte möglichst genau sein




UPP Beispiel: Kostenabschätzung




- ▶ Gute Abschätzung: Minimal mögliche Kosten eines Netzes,




Branch-and-Bound

UPP Beispiel



Lsg.	\vec{f}	Kosten
	123	5
	124	5
	132	5

	134	4
	142	5
	143	5

	213	4
	214	5
	124	5
⋮	⋮	⋮



- ▶ Wiederverwenden von Lösungen
- ▶ Prinzip der Optimalität:
 - ▶ Annahme: Lösung eines komplex Problems kann optimal aus dem optimalen Lösungen von Teilproblemen zusammengesetzt werden (ebenfalls Idee bei Teile-Und-Herrsche)
 - ▶ Gilt aber nicht für alle Probleme!
- ▶ p Parameter für Problemkomplexität
 - $p = k$: Gesamtproblem
 - $p < k$: Teilproblem
 - $p = 0$ oder $= 1$: Kleinstes Problem
- ▶ Für viele Probleme anwendbar. Beispielsweise:
 - ▶ Klassisches Beispiel: Fibonacci-Zahlen
 - ▶ Dijkstra SP
 - ▶ Aber auch für NP-vollständige Probleme

Dynamisches Programmieren

Beispiel Fibonacci-Zahlen

- ▶ Fibonacci-Zahlen: 0, 1, 1, 2, 3, 5, ...
- ▶ $F_n = F_{n-1} + F_{n-2}$ mit $F_0 = 0, F_1 = 1$

Klassische Implementierung:

```
fib(int n): begin
  if (n=0) then
    | return 0
  if (n=1) then
    | return 1
  else
    | return (fib(n-1)+fib(n-2))
```

- ▶ $\frac{F_{n+1}}{F_n} \rightarrow 1.6$ (genau $\frac{1+\sqrt{5}}{2}$) $\Rightarrow F_n > 1.6^n$
- ▶ Da immer 1 aufsummiert werden $\Rightarrow \mathcal{O}(1.6^n)$

Dynamisches Programmieren

fib(int n): **begin**

```
Fib[0] := 0 ;
Fib[1] := 1 ;
for (i=2; i ≤ n; i++) do
  | Fib[i] := Fib[i-1] + Fib[i-2]
return Fib[n] ;
```

- ▶ Teillösungen bei $p = i$
- ▶ Gesamtlösung bei $p = n$



- ▶ Nahezu alle Probleme Optimierungsprobleme
- ▶ Entsprechende mathematische Modelle wählen und bekannte mathematische Verfahren anwenden

Mathematische Optimierung

Lineare Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Verfahren aus dem Operations Research

Mathematische Optimierung

Lineare Programmierung

- ▶ Verfahren aus dem Operations Research
- ▶ Beispiel:

	Produkt 1	Produkt 2	Liefermenge
Rohstoff A	42	14	100
Rohstoff B	23	53	200
Preis	550	250	

Ziel: Optimierte auf maximalen Umsatz

Mathematische Optimierung

Lineare Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Modell: x_1, x_2 Anzahl Produkte 1 und 2

Mathematische Optimierung

Lineare Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Modell: x_1, x_2 Anzahl Produkte 1 und 2
- ▶ Kanonische Form (üblicherweise max und $x_i \geq 0$ implizit)

$$\text{max: } 550x_1 + 250x_2$$

$$\text{subject to: } 42x_1 + 14x_2 \leq 100$$

$$23x_1 + 53x_2 \leq 200$$

$$\text{and } x_i \geq 0$$

Mathematische Optimierung

Lineare Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Modell: x_1, x_2 Anzahl Produkte 1 und 2
- ▶ Kanonische Form (üblicherweise max und $x_i \geq 0$ implizit)

$$\text{max: } 550x_1 + 250x_2$$

$$\text{subject to: } 42x_1 + 14x_2 \leq 100$$

$$23x_1 + 53x_2 \leq 200$$

$$\text{and } x_i \geq 0$$

- ▶ Lineares Programm

Mathematische Optimierung

Lineare Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Modell: x_1, x_2 Anzahl Produkte 1 und 2
- ▶ Kanonische Form (üblicherweise max und $x_i \geq 0$ implizit)

$$\begin{aligned} \max: & 550x_1 + 250x_2 \\ \text{subject to:} & 42x_1 + 14x_2 \leq 100 \\ & 23x_1 + 53x_2 \leq 200 \end{aligned}$$

$$\text{and } x_i \geq 0$$

- ▶ Lineares Programm
- ▶ Matrixformulierung:

$$\begin{aligned} \max: & \vec{c} \cdot \vec{x} \\ \text{s. t. :} & A\vec{x} \leq \vec{b}, x_i \geq 0 \end{aligned}$$



- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11



- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken
 - ▶ Worst-Case-Komplexität: Exponentiell

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken
 - ▶ Worst-Case-Komplexität: Exponentiell
 - ▶ Praktisch immer Polynomial

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

Mathematische Optimierung

Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken
 - ▶ Worst-Case-Komplexität: Exponentiell
 - ▶ Praktisch immer Polynomial
- ▶ Problem ist in **P**

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

Mathematische Optimierung

Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken
 - ▶ Worst-Case-Komplexität: Exponentiell
 - ▶ Praktisch immer Polynomial
- ▶ Problem ist in **P**
 - ▶ Ellipsoid-Verfahren (1979)

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken
 - ▶ Worst-Case-Komplexität: Exponentiell
 - ▶ Praktisch immer Polynomial
- ▶ Problem ist in **P**
 - ▶ Ellipsoid-Verfahren (1979)
 - ▶ Innere Punkte Methoden (1984)
wandern nicht die Ecken ab, sondern gehen durch den Simplex

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

Mathematische Optimierung

Lineare Optimierung

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken
 - ▶ Worst-Case-Komplexität: Exponentiell
 - ▶ Praktisch immer Polynomial
- ▶ Problem ist in **P**
 - ▶ Ellipsoid-Verfahren (1979)
 - ▶ Innere Punkte Methoden (1984)
wandern nicht die Ecken ab, sondern gehen durch den Simplex
- ▶ Lösung durch *LP Solver*

Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

Mathematische Optimierung

Lineare Optimierung

- ▶ Lösung mittels Simplex-Verfahren (Danzig, 1947)
 - ▶ Darstellung der Kostenfunktion und Restriktionen als Hyperebene im Raum
 - ▶ Restriktionen bilden ein Simplex (Polyeder), Lösung eine der Ecken
 - ▶ Worst-Case-Komplexität: Exponentiell
 - ▶ Praktisch immer Polynomial
- ▶ Problem ist in **P**
 - ▶ Ellipsoid-Verfahren (1979)
 - ▶ Innere Punkte Methoden (1984)
wandern nicht die Ecken ab, sondern gehen durch den Simplex
- ▶ Lösung durch *LP Solver*
 - ▶ Ip_solve, GLPK, CLP, CPLEX, ...

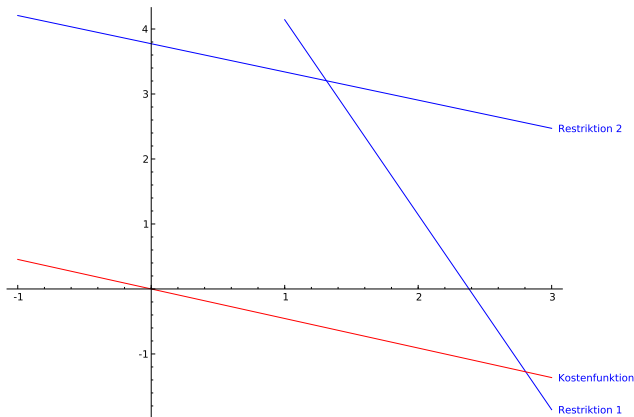
Beispiel

Optimum: $x_1 = 1.31303$, $x_2 = 3.20378$

Umsatz: 1523.11

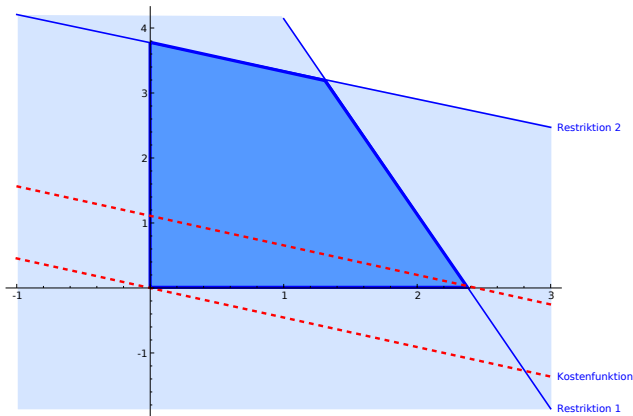
Mathematische Optimierung

Lineare Programmierung



Mathematische Optimierung

Lineare Programmierung



Mathematische Optimierung

Integer Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem: Oft nur ganzzahlige Variablen erlaubt!
Es lassen sich nicht 1.31303 Produkte herstellen/verkaufen

Beispiel

$$x_1 = 2, x_2 = 1$$

Umsatz: 1350

Mathematische Optimierung

Integer Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem: Oft nur ganzzahlige Variablen erlaubt!
Es lassen sich nicht 1.31303 Produkte herstellen/verkaufen
- ⇒ Integer Lineare Programmierung (ILP)

Beispiel

$$x_1 = 2, x_2 = 1$$

Umsatz: 1350

Mathematische Optimierung

Integer Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem: Oft nur ganzzahlige Variablen erlaubt!
Es lassen sich nicht 1.31303 Produkte herstellen/verkaufen
- ⇒ Integer Lineare Programmierung (ILP)
- ▶ Lösungsmethoden komplizierter
Lösungsverfahren NP-vollständig

Beispiel

$$x_1 = 2, x_2 = 1$$

Umsatz: 1350

Mathematische Optimierung

Integer Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem: Oft nur ganzzahlige Variablen erlaubt!
Es lassen sich nicht 1.31303 Produkte herstellen/verkaufen
- ⇒ Integer Lineare Programmierung (ILP)
 - ▶ Lösungsmethoden komplizierter
Lösungsverfahren NP-vollständig
 - ▶ Einfachste Heuristik: Rundung

Beispiel

$$x_1 = 2, x_2 = 1$$

Umsatz: 1350

Mathematische Optimierung

Integer Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem: Oft nur ganzzahlige Variablen erlaubt!
Es lassen sich nicht 1.31303 Produkte herstellen/verkaufen
- ⇒ Integer Lineare Programmierung (ILP)
 - ▶ Lösungsmethoden komplizierter
Lösungsverfahren NP-vollständig
 - ▶ Einfachste Heuristik: Rundung
 - ▶ Nicht sinnvoll

Beispiel

$$x_1 = 2, x_2 = 1$$

Umsatz: 1350

Mathematische Optimierung

Integer Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem: Oft nur ganzzahlige Variablen erlaubt!
Es lassen sich nicht 1.31303 Produkte herstellen/verkaufen
- ⇒ Integer Lineare Programmierung (ILP)
 - ▶ Lösungsmethoden komplizierter
Lösungsverfahren NP-vollständig
 - ▶ Einfachste Heuristik: Rundung
 - ▶ Nicht sinnvoll
 - ▶ Sub-Optimal

Beispiel

$$x_1 = 2, x_2 = 1$$

Umsatz: 1350

Mathematische Optimierung

Integer Lineare Optimierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Problem: Oft nur ganzzahlige Variablen erlaubt!
Es lassen sich nicht 1.31303 Produkte herstellen/verkaufen
- ⇒ Integer Lineare Programmierung (ILP)
 - ▶ Lösungsmethoden komplizierter
Lösungsverfahren NP-vollständig
 - ▶ Einfachste Heuristik: Rundung
 - ▶ Nicht sinnvoll
 - ▶ Sub-Optimal
 - ▶ Unzulässige Lösung

Beispiel

$$x_1 = 2, x_2 = 1$$

Umsatz: 1350



- ▶ Lösungsverfahren:



- ▶ Lösungsverfahren:
 - ▶ LP kombiniert mit branch-and-bound (auf- und abrunden)



- ▶ Lösungsverfahren:
 - ▶ LP kombiniert mit branch-and-bound (auf- und abrunden)
 - ▶ SAT (Erfüllbarkeitsproblem) (nur bei 0 – 1-ILP)



▶ Lösungverfahren:

- ▶ LP kombiniert mit branch-and-bound (auf- und abrunden)
- ▶ SAT (Erfüllbarkeitsproblem) (nur bei 0 – 1-ILP)
- ▶ Schnittebenenverfahren (Cutting-Planes)
Wiederholt normales LP lösen und Restriktionen für Ganzzahligkeit hinzufügen



▶ Lösungsverfahren:

- ▶ LP kombiniert mit branch-and-bound (auf- und abrunden)
- ▶ SAT (Erfüllbarkeitsproblem) (nur bei 0 – 1-ILP)
- ▶ Schnittebenenverfahren (Cutting-Planes)
Wiederholt normales LP lösen und Restriktionen für Ganzzahligkeit hinzufügen
- ▶ Manchmal durch Matrixstruktur implizit garantiert, dass Lösung ganzzahlig ist:
[Total unimodulare Matrizen](#)

⇒ Standard Löser benutzen

▶ Lösungsverfahren:

- ▶ LP kombiniert mit branch-and-bound (auf- und abrunden)
- ▶ SAT (Erfüllbarkeitsproblem) (nur bei 0 – 1-ILP)
- ▶ Schnittebenenverfahren (Cutting-Planes)
Wiederholt normales LP lösen und Restriktionen für Ganzzahligkeit hinzufügen
- ▶ Manchmal durch Matrixstruktur implizit garantiert, dass Lösung ganzzahlig ist:
[Total unimodulare Matrizen](#)

Beispiele

⇒ Standard Löser benutzen



▶ Lösungsverfahren:

- ▶ LP kombiniert mit branch-and-bound (auf- und abrunden)
- ▶ SAT (Erfüllbarkeitsproblem) (nur bei 0 – 1-ILP)
- ▶ Schnittebenenverfahren (Cutting-Planes)
Wiederholt normales LP lösen und Restriktionen für Ganzzahligkeit hinzufügen
- ▶ Manchmal durch Matrixstruktur implizit garantiert, dass Lösung ganzzahlig ist:
Total unimodulare Matrizen

Beispiele

- ▶ Inzidenzmatrix eines bipartiten Graphen

⇒ Standard Löser benutzen



▶ Lösungsverfahren:

- ▶ LP kombiniert mit branch-and-bound (auf- und abrunden)
- ▶ SAT (Erfüllbarkeitsproblem) (nur bei 0 – 1-ILP)
- ▶ Schnittebenenverfahren (Cutting-Planes)
Wiederholt normales LP lösen und Restriktionen für Ganzzahligkeit hinzufügen
- ▶ Manchmal durch Matrixstruktur implizit garantiert, dass Lösung ganzzahlig ist:
Total unimodulare Matrizen

Beispiele

- ▶ Inzidenzmatrix eines bipartiten Graphen
- ▶ Restriktionsmatrix von Max-Flow-Problemen

⇒ Standard Löser benutzen

Mathematische Optimierung

LP Varianten



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Häufige Varianten:

- ▶ Variablen nur 0 oder 1:
0 – 1-ILP

Mathematische Optimierung

LP Varianten



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Häufige Varianten:

- ▶ Variablen nur 0 oder 1:
0 – 1-ILP
- ▶ Gemischt LP und ILP:
Mixed Linear Programm (MLP)

Häufige Varianten:

- ▶ Variablen nur 0 oder 1:
0 – 1-ILP
- ▶ Gemischt LP und ILP:
Mixed Linear Programm (MLP)
- ▶ Quadratische Zielfunktion (weiterhin Restriktionen linear):
Quadratisches Programmieren (QP)
Vorteil bei Quadratisch-Konvexen-Funktionen: Lokale Extrema = Globale

ILP

UPP Beispiel



TECHNISCHE
UNIVERSITÄT
DARMSTADT

► Modell:

ILP

UPP Beispiel

► Modell:

$x_{ij} \in \mathbf{B}$: Zelle $i = 1, \dots, N$ wird auf Position $j = 1, \dots, M$ platziert.

ILP

UPP Beispiel

► Modell:

$x_{ij} \in \mathbf{B}$: Zelle $i = 1, \dots, N$ wird auf Position $j = 1, \dots, M$ platziert.

$n_k \in \mathbf{R}$: Kosten für ein Netz $k = 1, \dots, K$.

ILP

UPP Beispiel

► Modell:

$x_{ij} \in \mathbf{B}$: Zelle $i = 1, \dots, N$ wird auf Position $j = 1, \dots, M$ platziert.

$n_k \in \mathbf{R}$: Kosten für ein Netz $k = 1, \dots, K$.

Zielfunktion: $\min: \sum_k n_k$

ILP

UPP Beispiel

► Modell:

$x_{ij} \in \mathbf{B}$: Zelle $i = 1, \dots, N$ wird auf Position $j = 1, \dots, M$ platziert.

$n_k \in \mathbf{R}$: Kosten für ein Netz $k = 1, \dots, K$.

Zielfunktion: $\min: \sum_k n_k$

ILP

UPP Beispiel

► Modell:

$x_{ij} \in \mathbf{B}$: Zelle $i = 1, \dots, N$ wird auf Position $j = 1, \dots, M$ platziert.

$n_k \in \mathbf{R}$: Kosten für ein Netz $k = 1, \dots, K$.

Zielfunktion: $\min: \sum_k n_k$

Restriktionen: ► Alle Platzieren: $\forall i \sum_j x_{ij} = 1$ (2N Restriktionen)

► Modell:

$x_{ij} \in \mathbf{B}$: Zelle $i = 1, \dots, N$ wird auf Position $j = 1, \dots, M$ platziert.

$n_k \in \mathbf{R}$: Kosten für ein Netz $k = 1, \dots, K$.

Zielfunktion: $\min: \sum_k n_k$

Restriktionen:

- Alle Platzieren: $\forall i \sum_j x_{ij} = 1$ (2N Restriktionen)
- Überlappungsfreiheit: $\forall j : \sum_i x_{ij} \leq 1$ (M Restriktionen)
- Kostenfunktion: $n_k = c_{kV} + c_{kH}$

► Modell:

$x_{ij} \in \mathbf{B}$: Zelle $i = 1, \dots, N$ wird auf Position $j = 1, \dots, M$ platziert.

$n_k \in \mathbf{R}$: Kosten für ein Netz $k = 1, \dots, K$.

Zielfunktion: $\min: \sum_k n_k$

Restriktionen: ► Alle Platzieren: $\forall i \sum_j x_{ij} = 1$ (2N Restriktionen)

► Überlappungsfreiheit: $\forall j : \sum_i x_{ij} \leq 1$ (M Restriktionen)

► Kostenfunktion: $n_k = c_{kV} + c_{kH}$

► Netz k:

$$c_{kV} = c_{kMaxV} - c_{kMinV}$$

$$c_{kMaxV} = \max_{i \in \text{Netz } k} \{x_{ij} \cdot \text{'zu } j \text{ gehöriger } x\text{-Wert'}\}$$

$$c_{kMinV} = \min_{i \in \text{Netz } k} \{x_{ij} \cdot \text{'zu } j \text{ gehöriger } x\text{-Wert'}\}$$

c_{kH} analog

(2K(2 + 2 · $\sum_k |\text{Netz } k|$) Restriktionen)

Mathematische Optimierung

Konjugiertes Gradientenverfahren (CG)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Eigentlich benutzt um große LGS zu lösen

Mathematische Optimierung

Konjugiertes Gradientenverfahren (CG)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Eigentlich benutzt um große LGS zu lösen
- ▶ Vorteilhaft bei dünnbesetzten (sparse) Matrizen

Mathematische Optimierung

Konjugiertes Gradientenverfahren (CG)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Eigentlich benutzt um große LGS zu lösen
- ▶ Vorteilhaft bei dünnbesetzten (sparse) Matrizen
- ▶ Minimieren einer Funktion $f(\vec{x}) = \frac{1}{2} \cdot \vec{x}^T A \vec{x} - b \cdot \vec{x} + c$,
A symmetrisch, positiv definit

Mathematische Optimierung

Konjugiertes Gradientenverfahren (CG)



- ▶ Eigentlich benutzt um große LGS zu lösen
- ▶ Vorteilhaft bei dünnbesetzten (sparse) Matrizen
- ▶ Minimieren einer Funktion $f(\vec{x}) = \frac{1}{2} \cdot \vec{x}^T A \vec{x} - b \cdot \vec{x} + c$,
 A symmetrisch, positiv definit
- ▶ Gradient: Vektor der partiellen Ableitungen $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \dots \right)$
in unserem Fall $f'(\vec{x}) = A\vec{x} - b$.

Mathematische Optimierung

Konjugiertes Gradientenverfahren (CG)



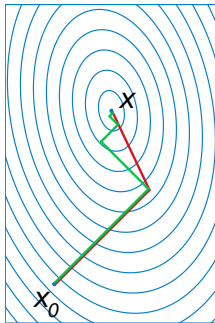
TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Eigentlich benutzt um große LGS zu lösen
- ▶ Vorteilhaft bei dünnbesetzten (sparse) Matrizen
- ▶ Minimieren einer Funktion $f(\vec{x}) = \frac{1}{2} \cdot \vec{x}^T A \vec{x} - b \cdot \vec{x} + c$,
A symmetrisch, positiv definit
- ▶ Gradient: Vektor der partiellen Ableitungen $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \dots \right)$
in unserem Fall $f'(\vec{x}) = A\vec{x} - b$.
- ▶ $f'(\vec{x}) = 0 \Leftrightarrow$ lösen des LGS $Ax = b$

Mathematische Optimierung

Konjugiertes Gradientenverfahren

- ▶ Krylow-Unterraum-Verfahren
 - ▶ Steilster Anstieg in Punkt x_k ist Gradient
 $f'(\vec{x}_k) = A\vec{x}_k - b$
 - ▶ Residuum $r_k := -(A\vec{x}_k - b)$
 - ▶ Minimiere Funktion in Richtung d_k
 d_i sind bzgl. A orthogonal zueinander
(konjugiert)
- ⇒ α_k in entsprechende Richtung gehen
- ⇒ Neues x_k
- ▶ Neues Residuum r_{k+1} aus Altem und der Richtung d_k (mit zugehöriger Schrittweite β_k bestimmen)



[Quelle: Wikimedia]

Mathematische Optimierung

Konjugiertes Gradientenverfahren

cg(A,b) begin

Wähle $x_0 \in \mathbf{R}^n$;

$r_0 := b - Ax_0$; $d_0 := r_0$;

$k := 0$;

repeat

$$\alpha_k := \frac{r_k^T r_k}{d_k^T A d_k} ;$$

$$x_{k+1} := x_k + \alpha_k \cdot d_k ;$$

$$r_{k+1} := r_k + \alpha_k \cdot A d_k ;$$

$$\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} ;$$

$$d_{k+1} := r_{k+1} + \beta_k \cdot d_k ;$$

$k++$;

until Residuum $\|r_k\| < \epsilon$;

return x_k ;

- ▶ x_0 beliebig
schnellere Konvergenz, falls x_0 nahe an Lösung
- ▶ Optimierung: In der Schleife $A \cdot d_k$ nur einmal berechnen
- ▶ Auch andere Abbruchbedingungen möglich:
 - ▶ $\Delta x_k < \epsilon$
 - ▶ Max. Anzahl Iterationen
 - ▶ ...

Mathematische Optimierung

Konjugiertes Gradientenverfahren (nichtlinear)

cg(A,b) **begin**

Wähle $x_0 \in \mathbf{R}^n$;

$r_0 := -\nabla f(x_0)$; $d_0 := r_0$;

$k := 0$;

repeat

$$\alpha_k := \frac{r_k^T r_k}{d_k^T \nabla^2(f(x_k)) d_k} ;$$

$$x_{k+1} := x_k + \alpha \cdot d_k ;$$

$$r_{k+1} := -\nabla f(x_{k+1}) ;$$

$$\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} ;$$

$$d_{k+1} := r_{k+1} + \beta_{k+1} \cdot d_k ;$$

$k++$;

until Residuum $\|r_k\| < \epsilon$;

return x_k ;

- ▶ Hesse-Matrix zur α -Bestimmung
Üblicherweise $\mathcal{O}(n^2)$,
problemabhängig ggf. nur $\mathcal{O}(n)$
- ▶ Andere Möglichkeit:
 α mittels Linesearch
- ▶ Andere Abbruchbedingungen

Mathematische Optimierung

Konjugiertes Gradientenverfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)

Mathematische Optimierung

Konjugiertes Gradientenverfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:

Mathematische Optimierung

Konjugiertes Gradientenverfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:

Theoretisch: Nach n Schritten

Mathematische Optimierung

Konjugiertes Gradientenverfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:

Theoretisch: Nach n Schritten

Praktisch I: Numerische Ungenauigkeiten machen mehr notwendig

Mathematische Optimierung

Konjugiertes Gradientenverfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:

Theoretisch: Nach n Schritten

Praktisch I: Numerische Ungenauigkeiten machen mehr notwendig

Praktisch II: Schneller, abhängig von Anzahl verschiedener Eigenwerte

Mathematische Optimierung

Konjugiertes Gradientenverfahren



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:
 - Theoretisch:** Nach n Schritten
 - Praktisch I:** Numerische Ungenauigkeiten machen mehr notwendig
 - Praktisch II:** Schneller, abhängig von Anzahl verschiedener Eigenwerte
- ▶ Wenn nötig Matrix konditionieren
⇒ Preconditioner

Mathematische Optimierung

Konjugiertes Gradientenverfahren

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:
 - Theoretisch:** Nach n Schritten
 - Praktisch I:** Numerische Ungenauigkeiten machen mehr notwendig
 - Praktisch II:** Schneller, abhängig von Anzahl verschiedener Eigenwerte
- ▶ Wenn nötig Matrix konditionieren
 - ⇒ Preconditioner
- ▶ Verschiedene Varianten, andere Bestimmung von β
 - ⇒ ggf. besseres Konvergenzverhalten

Mathematische Optimierung

Konjugiertes Gradientenverfahren

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:
 - Theoretisch:** Nach n Schritten
 - Praktisch I:** Numerische Ungenauigkeiten machen mehr notwendig
 - Praktisch II:** Schneller, abhängig von Anzahl verschiedener Eigenwerte
- ▶ Wenn nötig Matrix konditionieren
 - ⇒ Preconditioner
- ▶ Verschiedene Varianten, andere Bestimmung von β
 - ⇒ ggf. besseres Konvergenzverhalten
- ▶ Optimierung von Funktionen mit Nebenbedingungen:

Mathematische Optimierung

Konjugiertes Gradientenverfahren

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:
 - Theoretisch:** Nach n Schritten
 - Praktisch I:** Numerische Ungenauigkeiten machen mehr notwendig
 - Praktisch II:** Schneller, abhängig von Anzahl verschiedener Eigenwerte
- ▶ Wenn nötig Matrix konditionieren
 - ⇒ Preconditioner
- ▶ Verschiedene Varianten, andere Bestimmung von β
 - ⇒ ggf. besseres Konvergenzverhalten
- ▶ Optimierung von Funktionen mit Nebenbedingungen:
 - ▶ Lagrange-Multiplikatoren

Mathematische Optimierung

Konjugiertes Gradientenverfahren

- ▶ Konvergenz abhängig von Matrix (Konditionszahl)
- ▶ Konvergenz:
 - Theoretisch:** Nach n Schritten
 - Praktisch I:** Numerische Ungenauigkeiten machen mehr notwendig
 - Praktisch II:** Schneller, abhängig von Anzahl verschiedener Eigenwerte
- ▶ Wenn nötig Matrix konditionieren
 - ⇒ Preconditioner
- ▶ Verschiedene Varianten, andere Bestimmung von β
 - ⇒ ggf. besseres Konvergenzverhalten
- ▶ Optimierung von Funktionen mit Nebenbedingungen:
 - ▶ Lagrange-Multiplikatoren
 - ▶ Nebenbedingungen werden mittels Kostenfunktion berücksichtigt

- ▶ Metriken
 - ▶ 1-/2-Norm, HPWL, RSMT, RMST, LSE, Star+
- ▶ Timing-Analyse
- ▶ UPP
- ▶ Allgemeine Heuristiken
 - ▶ Nachbarsuche, Simulated Annealing, Tabu-Suche, Genetische Algorithmen, Ameisenalgorithmus
- ▶ Exakte Lösungsverfahren
 - ▶ Backtracking, Branch-and-Bound, Dynamic Programming, Linear Programming