

Interne Aufteilung

Kai

- Eingabedatei einlesen (2 min)
- Probleminst. vorbereiten (3 min)

Dominik

- Vor- und Nachteile (2 min)
- simulierte Abkühlung (4 min)
- grafische Ausgabe (1 min)
- Demo (Restzeit)
- Einzelverschiebung (1 min)

Stefan

- Ausgabe der Lösung (2 min)
- Bayern & Brandenburg (2 min)
- Aufgabenstellung(optional)+Einleitung +Gliederung (2 min)

Matthias

- Eröffnungsalgorithmen (2 min)
- Metaheuristiken (1 min)
- Tabu (4 min)
- Güte+Gültigkeit (3 min)

Jeder bitte ein Backup(Krankheit)

Backup für Matthias Backup für Stefan Backup für Dominik Backup für Kai

Wer sind wir?



Stefan

- HS Regensburg
- 1. Semester medizinische Informatik



Dominik

- Hasso-Plattner-Institut Potsdam
- 1. Semester IT-Systems Engineering



Matthias

- Hasso-Plattner-Institut Potsdam
- 1. Semester IT-Systems Engineering



Kai

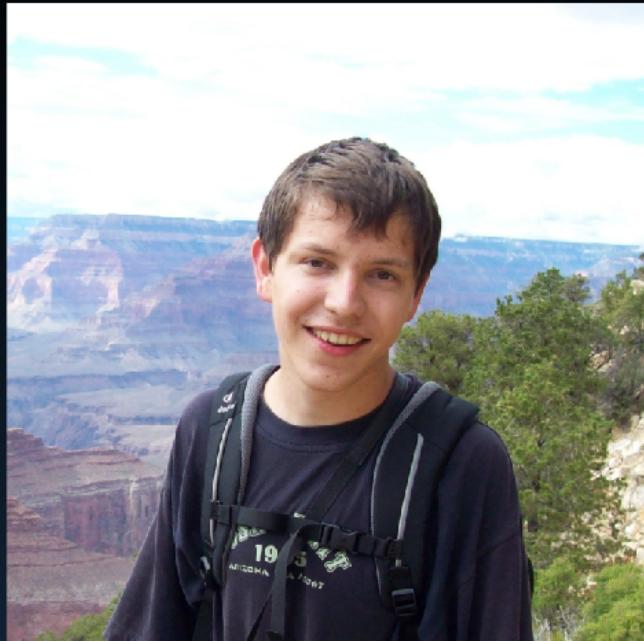
- Hasso-Plattner-Institut Potsdam
- 1. Semester IT-Systems Engineering



Stefan

- HS Regensburg
- 1. Semester medizinische Informatik

- Hass
- 1. Se



Dominik

- Hasso-Plattner-Institut Potsdam
- 1. Semester IT-Systems Engineering

- Ha
- 1.



Matthias

am
neering

- Hasso-Plattner-Institut Potsdam
- 1. Semester IT-Systems Engineering

- Hass
- 1. Se



Kai

S

tsdam
engineering

- Hasso-Plattner-Institut Potsdam
- 1. Semester IT-Systems Engineering



Kai
• Hochbegabter mit hoher
Motivation



Matthias
• Hochbegabter mit hoher
Motivation



Lukas
• Hochbegabter mit hoher
Motivation



Probleminstanz vorbereiten



Gültigkeit und Güte von Lösungen

Eröffnungsalgorithmen



Optimierungsalgorithmen



Grafische Ausgabe



Ausgabedatei erzeugen

Textausgabeformat:

- Eine Zeile probl. der statuarien Automaten
- Nach einer Zeile Koordinaten und Gewichte w. der positionierten Parkpositionen
- Letzte Zeile Ges. Hignight der Automaten
- Gewichte auf Ganzzahlen gerundet

Beispiele:
Germiston Hatfield Soweto

Typischer Programmablauf

Vorteile

GUI	simpel, funktional
Algorithmen	viele verschiedene, schnell (v.a. dank Gewichtskarte, kein Polygon Clipping)
Leistungsfähigkeit	große Probleminstanzen mit vielen Automaten möglich
Sonstiges	negative Koordinaten erlaubt

Nachteile

manchmal etwas langsam
Rundungsfehler durch Pixelkarte
hoher Speicherbedarf

B&B - Bayern&Brandenburg

1 Mitglied an der HS Regensburg
3 Mitglieder am HPI in Potsdam.

Wie geht das?

Einheitliche Software



LaTeX → TeX Live 2010

Collaborational Computing
Kommunikation



Prezi

Persönliche Treffen
Gute Konzeption

Eingabeparser

Eingabe: Textdateiformat siehe Aufgabenstellung
Ausgabe: Datenstruktur Vektorkarte

Exception Handling, Gültigkeit der Eingabedatei wird überprüft

Gewichtung der Attribute

Eingabe: Vektorkarte
Benutzereingabe: Gewichtung der Attribute
Ausgabe: Gewichtete Vektorkarte



Datenstruktur (Klassendiagramm)



Eingabedatei einlesen

Prob

Eingabeparser

Eingabe: Textdateiformat siehe Aufgabenstellung

Ausgabe: Datenstruktur Vektorkarte

Exception Handling, Gültigkeit der Eingabedatei
wird überprüft

Datenstruktur (Klassendiagramm)



Gewichtung der Attribute

Eingabe: Vektorkarte

Benutzereingabe: Gewichtung der Attribute

Ausgabe: Gewichtete Vektorkarte

$$gewichtNachFkt(s) = \sum_{a \in \text{Attribute}} \text{Gewichtung}(a) \cdot \text{Wert}(a, s)$$

Gewichtung : Attribut $\rightarrow \mathbb{N}$

additional parameters	
Attribute name	Attribute weigh...
Name	1
Gender: Male	15
Gender: Female	24
Age: 0-4	0
Age: 5-9	3
Age: 10-14	56
Age: 15-19	1
Age: 20-24	1
Age: 25-29	1

Approximation der Karte



Eingabe: Gewichtete Vektorkarte
Benutzereingabe: Approximationsfaktor

Berechnung der neuen Koordinaten
 $\text{Neu} = \text{Stadtteil} \cdot (\text{Faktor} + \epsilon) + \text{Umwelt} \cdot (\text{Faktor} - \epsilon)$
Berechnung der Polygonfläche
Gaußsche Trapezformel

Nachteil: Rundungsfehler (Integer)
Vorteil: weniger Speicherplatzbedarf/
Rechenaufwand

Ausgabe: Approx. und gew. Vektorkarte

Erstellen der Pixelkarte

Eingabe: Approx. und gew. Vektorkarte



- quadr. Speicherbedarf
- negative Koordinaten
- nach außen:
Abstraktion von
exakten Koordinaten

0	0	0	0	0	0	0
0	45	45	0	0	0	0
0	45	45	45	0	0	0
0	45	45	45	45	0	0
0	45	45	45	45	45	0
8	8	8	8	8	0	0
8	8	8	8	8	0	0
0	0	0	0	0	0	0

Berechnung: Java-API (performant)

Ausgabe: Pixelkarte (Pixel-Matrix)

Generieren der Gewichtskarte

Eingabe: Pixelkarte

Trivialer Ansatz



Für jeden Wert Pixel im Umkreis
aufsummieren

$$\sum_{P \in \text{Pixelkarte}, P \in \text{Umkreis von } M} \frac{\text{wkt}(P)}{\text{Fläche(StdZeile}(P))}$$

Komplexität: $O(4 \cdot \text{rechtsAutomaten}^2 \cdot P)$

P: Anzahl der Pixel

Delta-Kreis-Methode



Nur Veränderungen zu den
benachbarten Pixeln berechnen

Deutlich reduzierte Laufzeit, da nicht jeder Kreis
aufaddiert werden muss

$$O(P \cdot \text{rechtsAutomaten}^2 \cdot P + P \cdot \frac{3 \cdot \text{rechtsAutomaten}^2 \cdot P}{2})$$

Probleminstanz vorbereiten

Approximation der Karte

Koordinate
 $_x$: Integer
 $_y$: Integer



Approximation



Eingabe: Gewichtete Vektorkarte
Benutzereingabe: Approximationsfaktor

Berechnung der neuen Koordinaten

$$\forall s \in \text{Stadtteil} : \forall k \in s : (x_{neu}, y_{neu}) = \left(\frac{x}{\text{Faktor}}, \frac{y}{\text{Faktor}} \right)$$

Berechnung der Polygonfläche
Gaußsche Trapezformel

Nachteil: Rundungsfehler (Integer)
Vorteil: weniger Speicherplatzbedarf/
Rechenaufwand

Ausgabe: Approx. und gew. Vektorkarte

Erstellen der Pixelkarte

Eingabe: Approx. und gew. Vektorkarte

Karte

```
_karte : Integer[][]  
_maxX : Integer  
_minX : Integer  
_maxY : Integer  
_minY : Integer
```



- quadr. Speicherbedarf
- negative Koordinaten
- nach außen:
Abstraktion von
exakten Koordinaten

Welt

...

0	0	0	0	0	0
0	45	45	0	0	0
0	45	45	45	0	0
0	45	45	45	45	0
0	45	45	45	45	45
8	8	8	8	8	0
8	8	8	8	8	0
0	0	0	0	0	0

Berechnung: Java-API (performant)

Ausgabe: Pixelkarte (Pixel-Matrix)

Generieren der Gewichtskarte

Eingabe: Pixelkarte

Trivialer Ansatz



Für jeden Wert Pixel im Umkreis
aufsummieren

$$\sum_{P \in \text{Pixel}: dist(P, M) \leq \text{_radiusAutomaten}} \frac{_karte(P)}{\text{Fläche}(\text{Stadtteil}(P))}$$

Komplexität: $\mathcal{O}(4 \cdot \text{_radiusAutomaten}^2 \cdot P)$

P: Anzahl der Pixel

Delta-Kreis-Methode



Nur Veränderungen zu den
benachbarten Pixeln berechnen

Deutlich reduzierte Laufzeit, da nicht jeder Kreis
aufaddiert werden muss

$$\mathcal{O}(4 \cdot \text{_radiusAutomaten}^2 + P \cdot 2 \cdot \frac{2 \cdot \text{_radiusAutomaten} \cdot \pi}{2})$$

Trivialer Ansatz

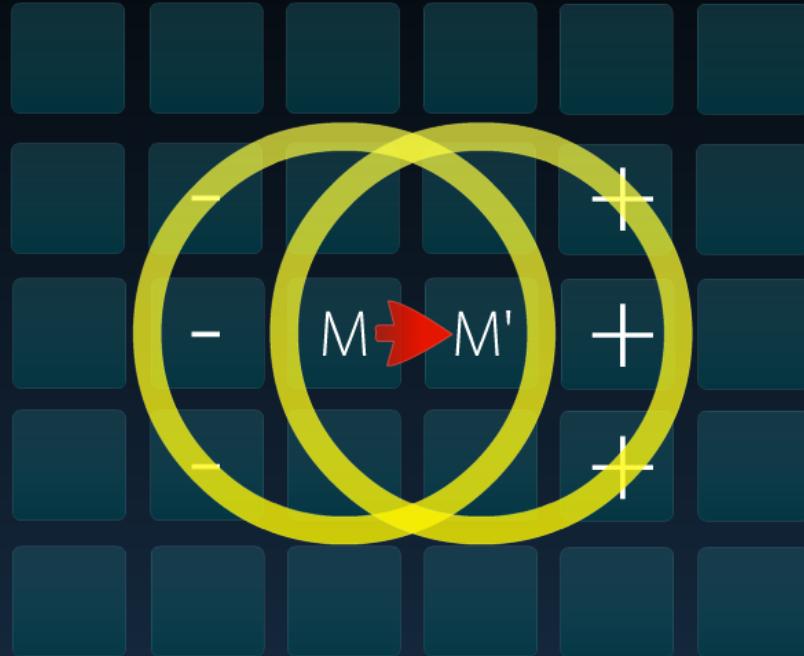


Für jeden Wert Pixel im Umkreis
aufsummieren

$$\sum_{P \in \text{Pixel}: dist(P, M) \leq \text{_radiusAutomaten}} \frac{karte(P)}{\text{Flaeche}(\text{Stadtteil}(P))}$$

Komplexität: $\mathcal{O}(4 \cdot \text{_radiusAutomaten}^2 \cdot P)$
P: Anzahl der Pixel

Delta-Kreis-Methode



Nur Veränderungen zu den benachbarten Pixeln berechnen

Deutlich reduzierte Laufzeit, da nicht jeder Kreis aufaddiert werden muss

$$\mathcal{O}(4_{radiusAutomaten}^2 + P \cdot 2 \cdot \frac{2 \cdot radiusAutomaten \cdot \pi}{2})$$

Gültigkeit und Güte von Lösungen



Lösungsgültigkeit

- Alle Automaten gesetzt
- Koordinaten sind gültig
- Keine Überschneidungen
- $\forall a, b \in \text{Automat} :$
 $a \neq b \Rightarrow \text{Abstand}(a, b) \geq 2r$

quadratische Laufzeit:

$$\theta\left(\frac{\text{Automaten} \cdot (\text{Automaten} - 1)}{2}\right)$$

Güte einer Lösung

Eingabe: Zustand mit Automaten

$$\text{Gute} = \sum_{a \in \text{Stadt}} \sum_{b \in \text{Automat}} \frac{\text{Scheinbarkeit}(a, b) \cdot \text{gewünschter Wert}}{\text{Fluss}(a, b)}$$

$$\text{Gute} = \sum_{a \in \text{Automat}} \text{Gewicht}(a) \cdot \text{wert}$$

lineare Laufzeit: $\mathcal{O}(\text{Automaten})$

Insgesamt: quadratische Laufzeit in |Automat|

Zustand

_automaten : Automat[]

_bewertung : Integer

0..*



besteht aus

Automat

_position : Koordinate

_automatGesperrt : Boolean

Lösungsgültigkeit

- Alle Automaten gesetzt
- Koordinaten sind gültig
- Keine Überschneidungen

$\forall a, b \in \text{Automat} :$

$$a \neq b \Rightarrow \text{Abstand}(a, b) \geq 2r$$

quadratische Laufzeit:

$$\theta\left(\frac{\text{Automaten} \cdot (\text{Automaten} - 1)}{2}\right)$$

Güte einer Lösung

Eingabe: Zustand mit Automaten

$$\text{Guete} = \sum_{s \in \text{Stadtteil}} \sum_{a \in \text{Automat}} \frac{\text{Schnittflaeche}(a, s)}{\text{Flaeche}(s)} \cdot \text{gewichtNachFkt}(s)$$

$$\text{Guete} = \sum_{a \in \text{Automat}} \text{Gewichtskarte}(a)$$

lineare Laufzeit: $\mathcal{O}(|\text{Automat}|)$

insgesamt: quadratische Laufzeit in $|\text{Automat}|$

Eröffnungsalgorithmen

Brute-Force-Ansatz



Zufällige Verteilung

1	0	0	0
0	0	0	0
5	45	0	0
5	45	45	0

Greedy-Algorithmus



Optimierungsalgorithmen

Metaheuristik

- Algorithmus unabhängig von der Problemlösung
- Elementare Operationen: Rendemisste, Verschiebung eines Automaten
- Maximierungproblem (Güte)



Tabu-Suche



Simulierte Abkühlung



Einzelverschiebung

- betrachte Nachbarschaft der einzelnen Automaten
- Verbesserungen sofort übernehmen (gelingt)
- verbessert die Lösung nur selten
- Laufzeitkomplexität: $O(n^2 \cdot m^2)$
- Durchschnittliche Anzahl von Wechseln: $\frac{1}{2} \cdot n \cdot m$

Ausga

Textausgabe

- Erste Zeile: Anzahl i der
- Nächsten i Zeilen: Koordinaten der i platzierten Banken
- Letzte Zeile: Gesamtgewicht
- Gewichte auf Ganzzah

Berechnungsschritt

Ausga

Eröffnungsalgorithmen

Brute-Force-Ansatz



Primitives Brute-Force
 $p = \text{Höhe} \cdot \text{Breite}$
 $\mathcal{O}(p^2)$ Automaten.



Zufällige Verteilung

Greedy-Algorithmus

15	12	6
12	13	4
7	4	0

Gewichtskarte

Vollständige Suche

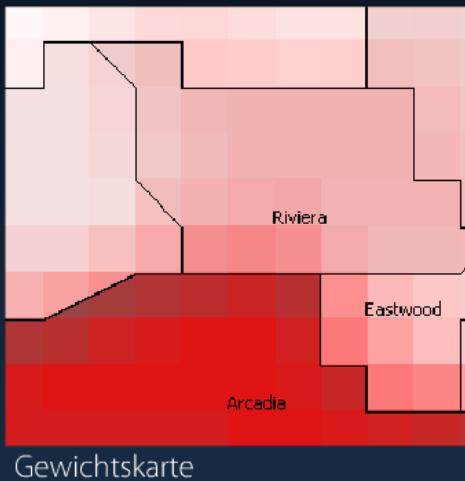
Automat setzen

1. Beste Position ermitteln
 $p = \text{Höhe} \cdot \text{Breite}$
Bei a Automaten: $\mathcal{O}(a \cdot p)$

2. Gültigkeit prüfen
3. Wenn ungültig: gehe zu (1)
4. Setze nächsten Automaten
Laufzeit: $\mathcal{O}(a^2 \cdot p)$

Optimierungsalgorithmen

Brute-Force-Ansatz



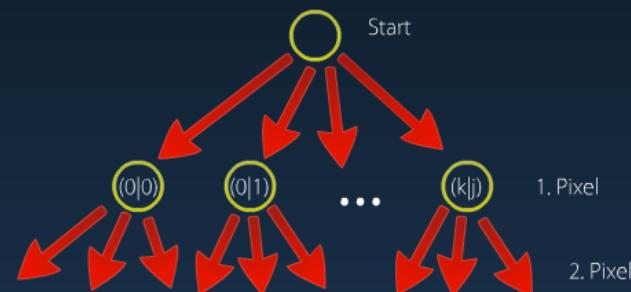
Primitives Brute-Force

Pixel auswählen
 $p = \text{Höhe} \cdot \text{Breite}$
 $\mathcal{O}(p^{\text{Automaten}})$

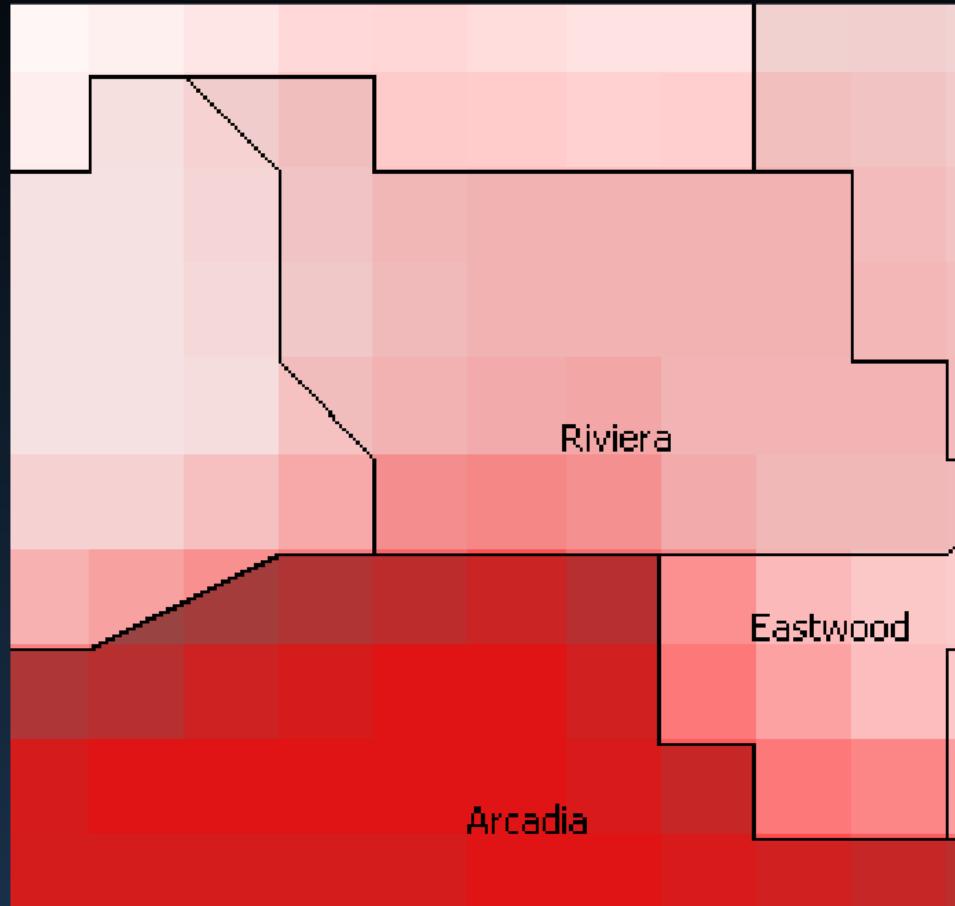
Gültigkeit und Güte der Lösung berechnen

Backtracking

Schlechtere Lösungen möglichst früh verwerfen



STRUCTURE



Gewichtskarte

Primitives Brute-Force

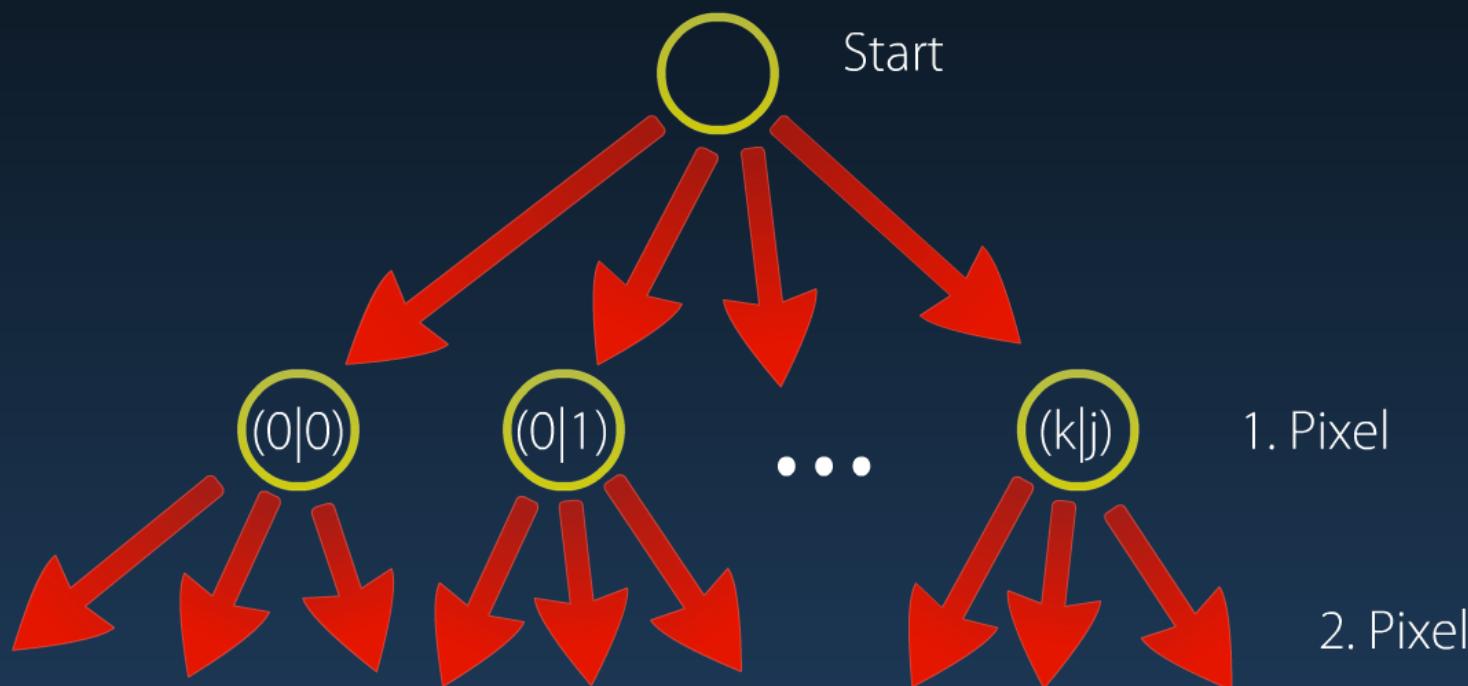
Pixel auswählen

$$p = \text{Höhe} \cdot \text{Breite}$$
$$\mathcal{O}(p^{\text{Automaten}})$$

Gültigkeit und Güte der
Lösung berechnen

Backtracking

Schlechtere Lösungen möglichst
früh verwerfen



Greedy-Algorithmus

Vollständige Suche

15	12	6
12	13	4
7	4	0

Gewichtskarte

Automat setzen

1. Beste Position ermitteln
 $p = \text{Höhe} \cdot \text{Breite}$
Bei a Automaten: $\theta(a \cdot p)$
 2. Gültigkeit prüfen
 3. Wenn ungültig: gehe zu (1)
 4. Setze nächsten Automaten
- Laufzeit: $\mathcal{O}(a^2 \cdot p)$



2. Pixel

Zufällige Verteilung



Greedy-Algorithmus

Vollständige Suche

Automat setzen

- Beste Position ermitteln
 $p = \text{Höhe} - 1$ bei
Bei Automaten: $\theta(a, p)$
- Gültigkeit prüfen
- Wenn ungültig: gehe zu (1)
- Setze nächsten Automaten

Laufzeit: $\mathcal{O}(n^C \cdot p)$

15	12	6
12	13	4
7	4	0

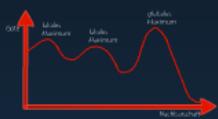
Gewichtsketten

Zufällige Verteilung

Optimierungsalgorithmen

Metaheuristik

- Algorithmen unabhängig von der Problemstellung
- Elementare Operationen: Randomisierte Verschiebung eines Automaten
- Maximierungsproblem (Güte)



Tabu-Suche



Simulierte Abkühlung



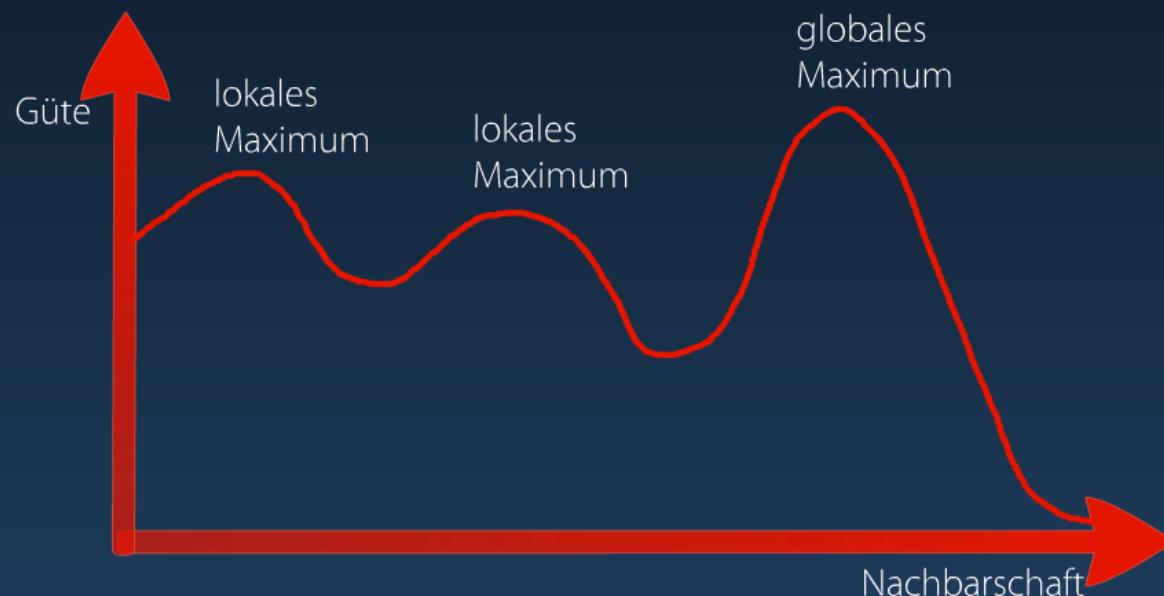
Einzelverschiebung

- beschäftigt Nachbarschaft der einzelnen Automaten
- Verbesserungen sofort übernehmen (gierig)
- verbessert die Lösung nur selten
- Laufzeitkomplexität: $\mathcal{O}(n^C \cdot r^2)$
 $n = \text{Anzahl der Automaten}, r = \text{Radius des Automaten}$

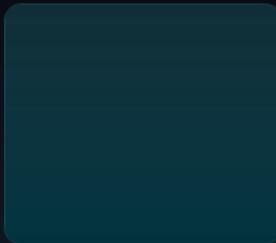
Berechnungsschritt

Metaheuristik

- Algorithmus unabhängig von der Problemstellung
- Elementare Operationen: Randomisierte Verschiebung eines Automaten
- Maximierungsproblem (Güte)



Tabu-Suche



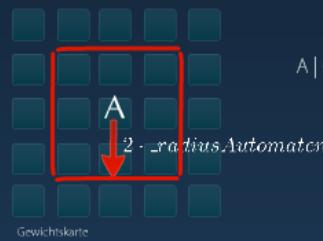
leere Tabu-Liste

Generiere Nachbarschaft:

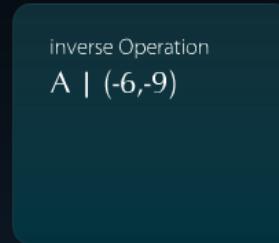
- wähle Automat zufällig
- $A | (0,2) | 0$
- $A | (6,9) | +10$
- $A | (-1,4) | -2$
- alle im Umkreis

Verschiebe A um (6,9)

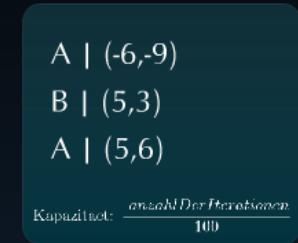
Nachbarschaft eines Automaten A



$A | (2,1) | +10$



inverse Operation
 $A | (-6,-9)$



$A | (-6,-9)$
 $B | (5,3)$
 $A | (5,6)$

Kapazität: $\frac{\text{anzahlDer Iterationen}}{100}$

Generiere Nachbarschaft

Verschiebe Automat

Generiere Nachbarschaft

Verschiebe Automat

...

Generiere Nachbarschaft

- wähle Automat zufällig

• $A | (0,5) | -9$

• $A | (2,3) | -3$

• $A | (-6,-9) | +10$

Verschiebe A um (2,3)

Bei voller Liste: FIFO

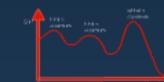
Eingabe: Zustand mit Automaten

Benutzereingabe: Anzahl der Iterationen

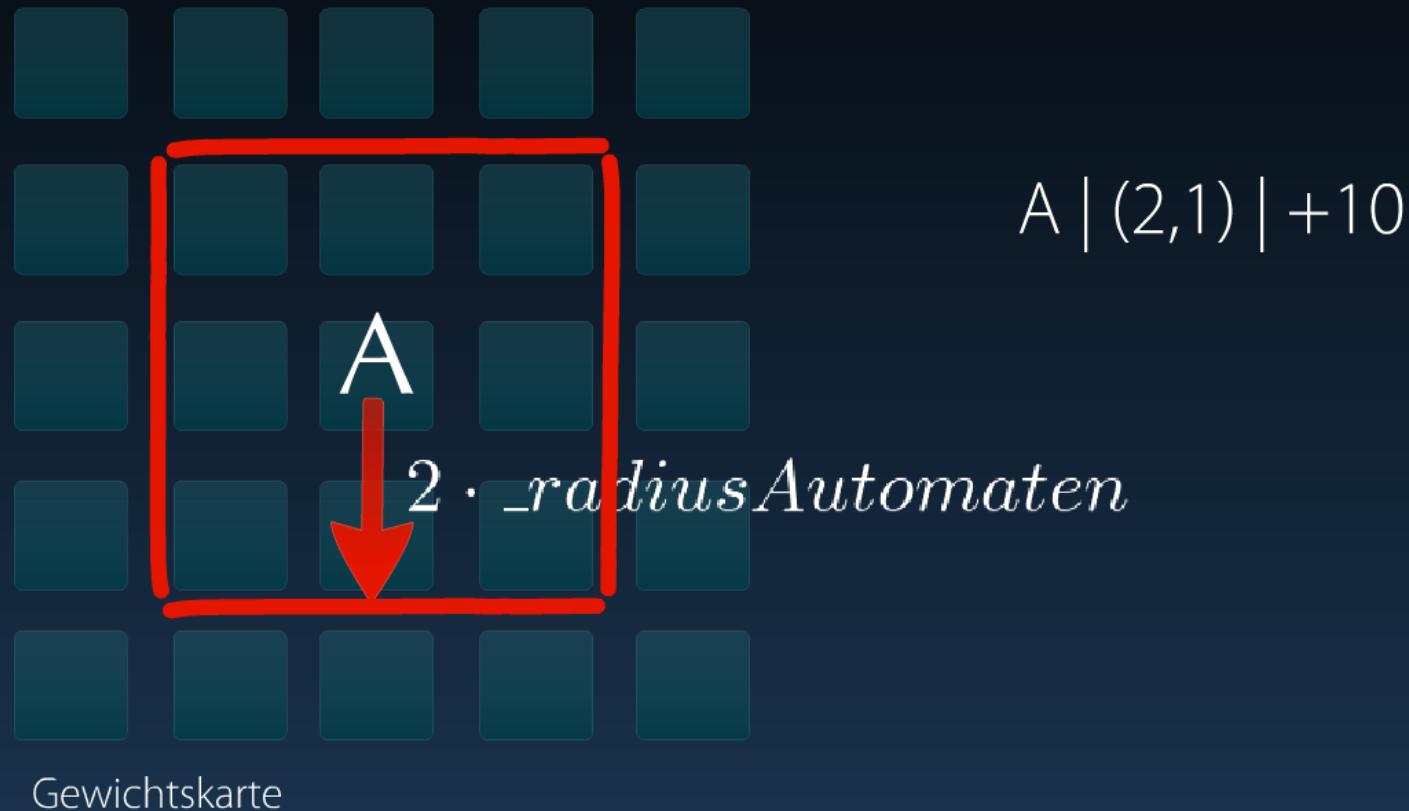
Ausgabe: Zustand mit Automaten

Laufzeitkomplexität:

- Gültigkeit der Lösung: $\Theta(n^2)$
 - Güte der Lösung: $\Theta(1)$
 - Tabu-Liste durchsuchen: $\Theta(\frac{n}{100})$
 - Größe der Nachbarschaft: $\Theta(4r^2)$ ($r = \Theta(1)$)
 - Gesamt: $\Theta(n^2) + \Theta(n^2) + \Theta(\frac{n}{100}) = \Theta(n^2)$
- pro Iteration
- Tabu-Liste als ballandierter Suchbaum
 - Sehr effizientes Generieren neuer Lösungen
 - Ungenauigkeiten durch Gewichtskarte
 - "Blockierung" bei zu vielen Automaten



Nachbarschaft eines Automaten A



inverse Operation
 $A \mid (-6, -9)$

leere Tabu-Liste

Generiere Nachbarschaft:

- wähle Automat zufällig
- $A \mid (0,2) \mid 0$
- $A \mid (6,9) \mid +10$
- $A \mid (-1,4) \mid -2$
- alle im Umkreis

Verschiebe A um $(6,9)$

Generiere Nachbarschaft

Verschiebe Automaten

Generiere Nachbarschaft

Verschiebe Automaten

...

- Liste
- Nachbarschaft:
 - Automat zufällig
 - 2) | 0
 - 9) | +10
 - 4) | -2
- Umkreis
- A um (6,9)

inverse Operation
 $A \mid (-6,-9)$

A | (-6,-9)
B | (5,3)
A | (5,6)

Kapazität: $\frac{-anzahlI}{anzahlL}$

Generiere Nachbarschaft
Verschiebe Automat

Generiere Nachbarschaft
Verschiebe Automat

...

Generiere N

- wähle Au
- A | (0,5)
- A | (2,3)
- A | (-6,-9)

Verschiebe /

Bei voller Lis

tion

)

Nachbarschaft

Automat

Nachbarschaft

Automat

A | (-6,-9)

B | (5,3)

A | (5,6)

Kapazität: $\frac{\text{anzahlDerIterationen}}{100}$

Generiere Nachbarschaft

- wähle Automat zufällig
- A | (0,5) | -9
- A | (2,3) | -3
- A | (-6,-9) | +10

Verschiebe A um (2,3)

Bei voller Liste: FIFO

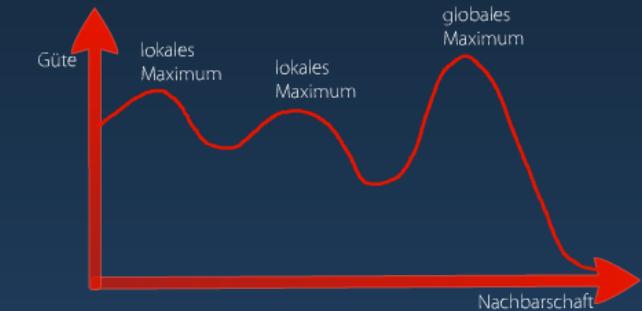
Eingabe: Zustand mit Automaten

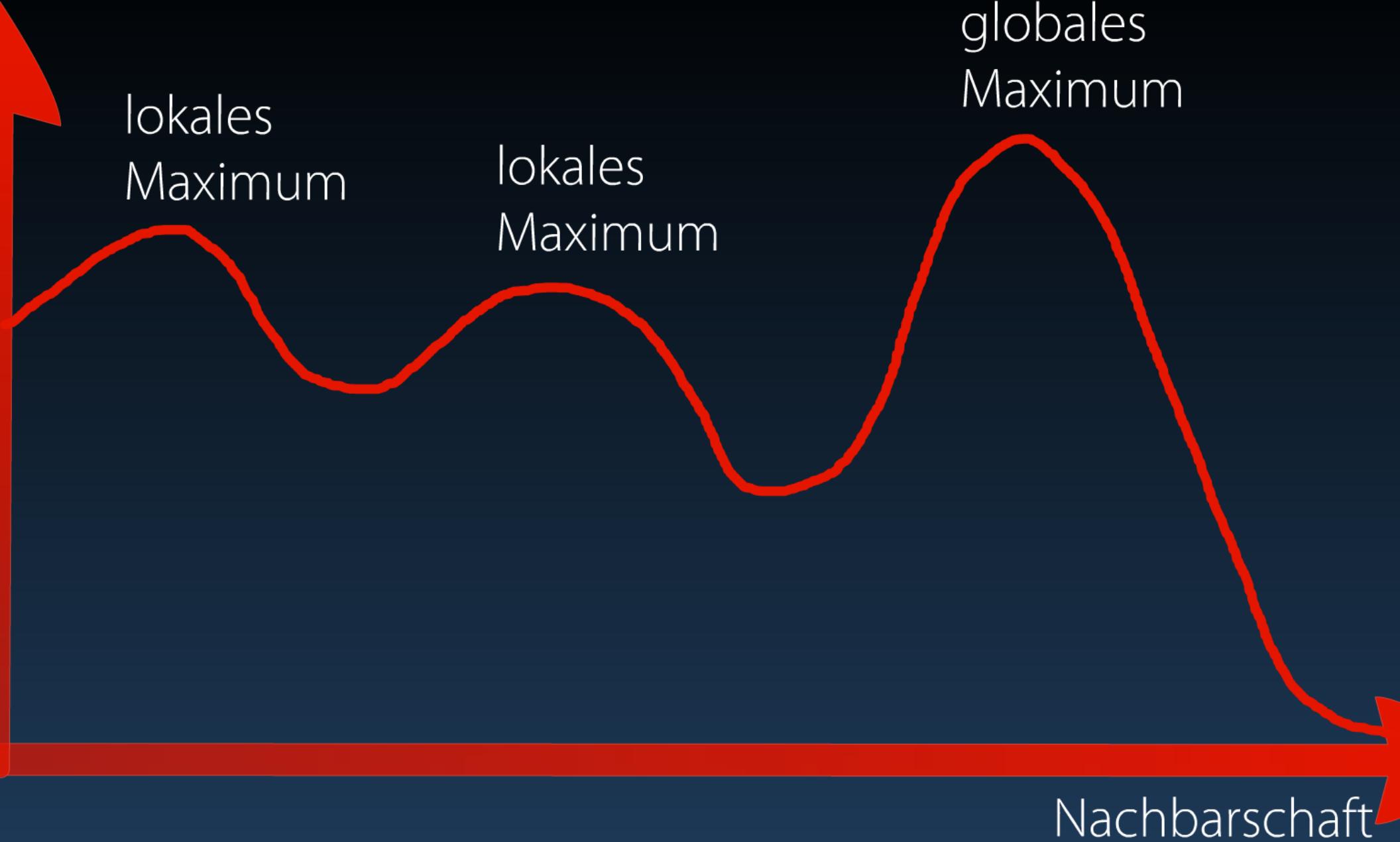
Benutzereingabe: Anzahl der Iterationen

Ausgabe: Zustand mit Automaten

Laufzeitkomplexität:

- Gültigkeit der Lösung: $\theta(a - 1)$
- Güte der Lösung: $\theta(a)$
- Tabu-Liste durchsuchen: $\mathcal{O}(\frac{i}{100})$
- Größe der Nachbarschaft: $\mathcal{O}(4r \cdot 4r) = \mathcal{O}(16r^2)$
- Gesamt: $\mathcal{O}(16r^2 \cdot (\frac{i}{100} + a + a - 1)) = \mathcal{O}(16r^2 \cdot (\frac{i}{100} + 2a))$
pro Iterationsschritt
- Tabu-Liste als ballancierter Suchbaum
- Sehr effizientes Generieren neuer Lösungen
- Ungenauigkeiten durch Gewichtskarte
- "Blockierung" bei zu vielen Automaten





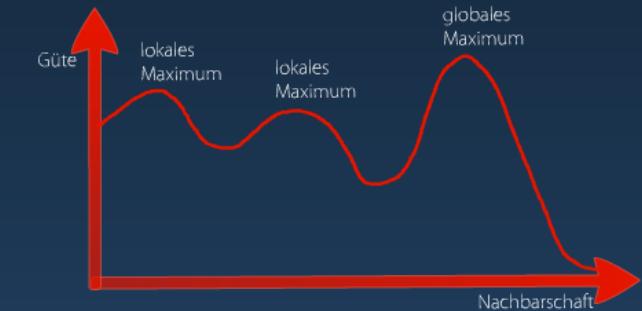
Eingabe: Zustand mit Automaten

Benutzereingabe: Anzahl der Iterationen

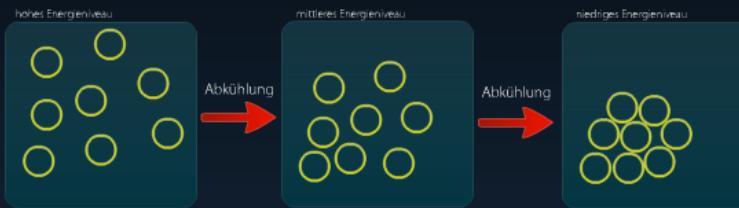
Ausgabe: Zustand mit Automaten

Laufzeitkomplexität:

- Gültigkeit der Lösung: $\theta(a - 1)$
- Güte der Lösung: $\theta(a)$
- Tabu-Liste durchsuchen: $\mathcal{O}(\frac{i}{100})$
- Größe der Nachbarschaft: $\mathcal{O}(4r \cdot 4r) = \mathcal{O}(16r^2)$
- Gesamt: $\mathcal{O}(16r^2 \cdot (\frac{i}{100} + a + a - 1)) = \mathcal{O}(16r^2 \cdot (\frac{i}{100} + 2a))$
pro Iterationsschritt
- Tabu-Liste als ballancierter Suchbaum
- Sehr effizientes Generieren neuer Lösungen
- Ungenauigkeiten durch Gewichtskarte
- "Blockierung" bei zu vielen Automaten



Simulierte Abkühlung

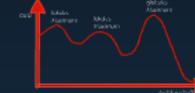


- Bei hoher Temperatur: Atome mit hoher Energie, keine Bindungen
- Schnelle Abkühlung: unregelmäßige Struktur, hartes/sprödes Material, geringe Qualität
- Langsame Abkühlung: regelmäßige Struktur, weiches Material, hohe Qualität (Weichglühen)
- Theorie: Optimum bei unendlich langsamster Abkühlung

Typischer Ablauf

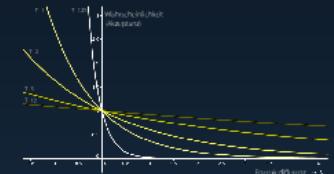
- Initiallösung erzeugen
- Solange $T > 0$
 - Führe i Iterationen aus
 - Generiere zufällige Nachbarlösung
 - Akzeptiere mit Wahrscheinlichkeit $e^{-\frac{\Delta E}{T}}$
 - Erniedrige Temperatur
- Beste Lösung ausgeben

$$\text{Laufzeitkomplexität: } \Theta(T \cdot i \cdot a(u-1))$$



(Erfüllungsverfahren)

Automat(en) verschieben
Gültigkeit sicherstellen
Güte (+Differenz) berechnen
 $\frac{\partial E - E_0}{E}$
Gesamt: $\lambda \left(\frac{\partial E - E_0}{E} + \frac{\partial E_0 - E_0}{E_0} \right) + \left(\frac{\partial E_0 - E_0}{E_0} \right)$



Eingabe: Zustand mit Automaten
Benutzereingabe: Maximale Temperatur
Ausgabe: Zustand mit Automaten

- Variable Größe der elementaren Operationen
- Mehrfache Anwendung des Algorithmus
- Speichern der global besten Lösung
- Effizientes Generieren neuer Lösungen
- Ungenauigkeiten durch Gewichtskarte

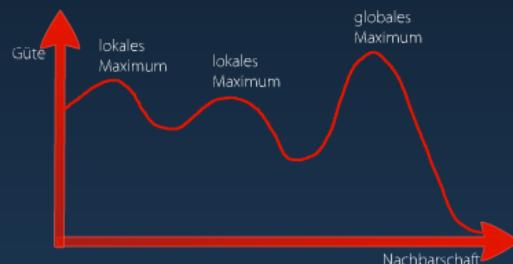
Struktur CC



- Bei hoher Temperatur: Atome mit hoher Energie, keine Bindungen
- Schnelle Abkühlung: unregelmäßige Struktur, hartes/sprödes Material, geringe Qualität
- Langsame Abkühlung: regelmäßige Struktur, weiches Material, hohe Qualität (Weichglühen)
- Theorie: Optimum bei unendlich langsamer Abkühlung

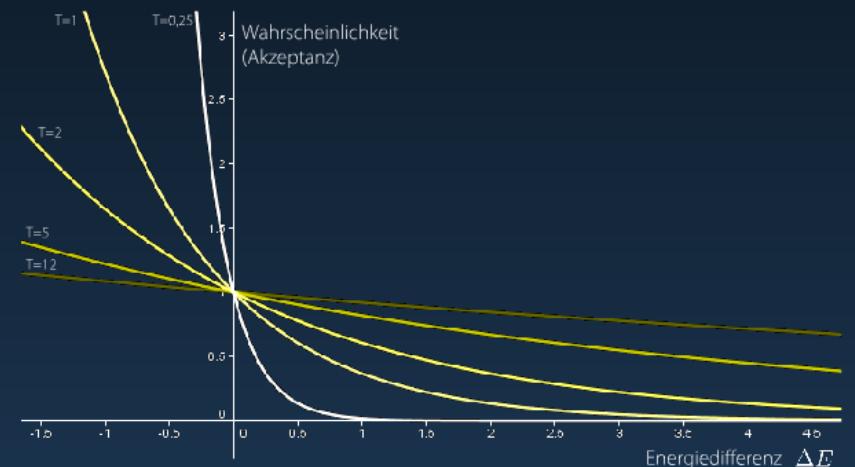
Typischer Ablauf

1. Initiallösung erzeugen
(Eröffnungsverfahren)
 2. Solange $T > 0$
 - a) Führe i Iterationen aus
 - i) Generiere zufällige Nachbarlösung
 - ii) Akzeptiere mit Wahrscheinlichkeit $e^{-\frac{\Delta E}{T}}$
 - c) Erniedrige Temperatur
 3. Beste Lösung ausgeben
- Laufzeitkomplexität: $\theta(T \cdot i \cdot \frac{a(a+1)}{2})$



(Eröffnungsverfahren)

- Automat(en) verschieben Gültigkeit sicherstellen $\theta(\frac{a \cdot (a-1)}{2})$
- Güte (+Differenz) berechnen $\theta(a)$
- Gesamt: $\theta\left(\frac{a(a-1)}{2} + \frac{2a}{2}\right) = \theta\left(\frac{a(a+1)}{2}\right)$



- Automat(en) verschieben

- Gültigkeit sicherstellen

$$\theta\left(\frac{a \cdot (a - 1)}{2}\right)$$

- Güte (+Differenz) berechnen

$$\theta(a)$$

Gesamt: $\theta\left(\frac{a(a - 1)}{2} + \frac{2a}{2}\right) = \theta\left(\frac{a(a + 1)}{2}\right)$

T=1

T=0,25

3

Wahrscheinlichkeit

Lösungswahl
Wahrscheinlichkeit $e^{-\frac{\Delta E}{T}}$



Gültigkeit
 $\theta\left(\frac{a \cdot (a - 1)}{2}\right)$

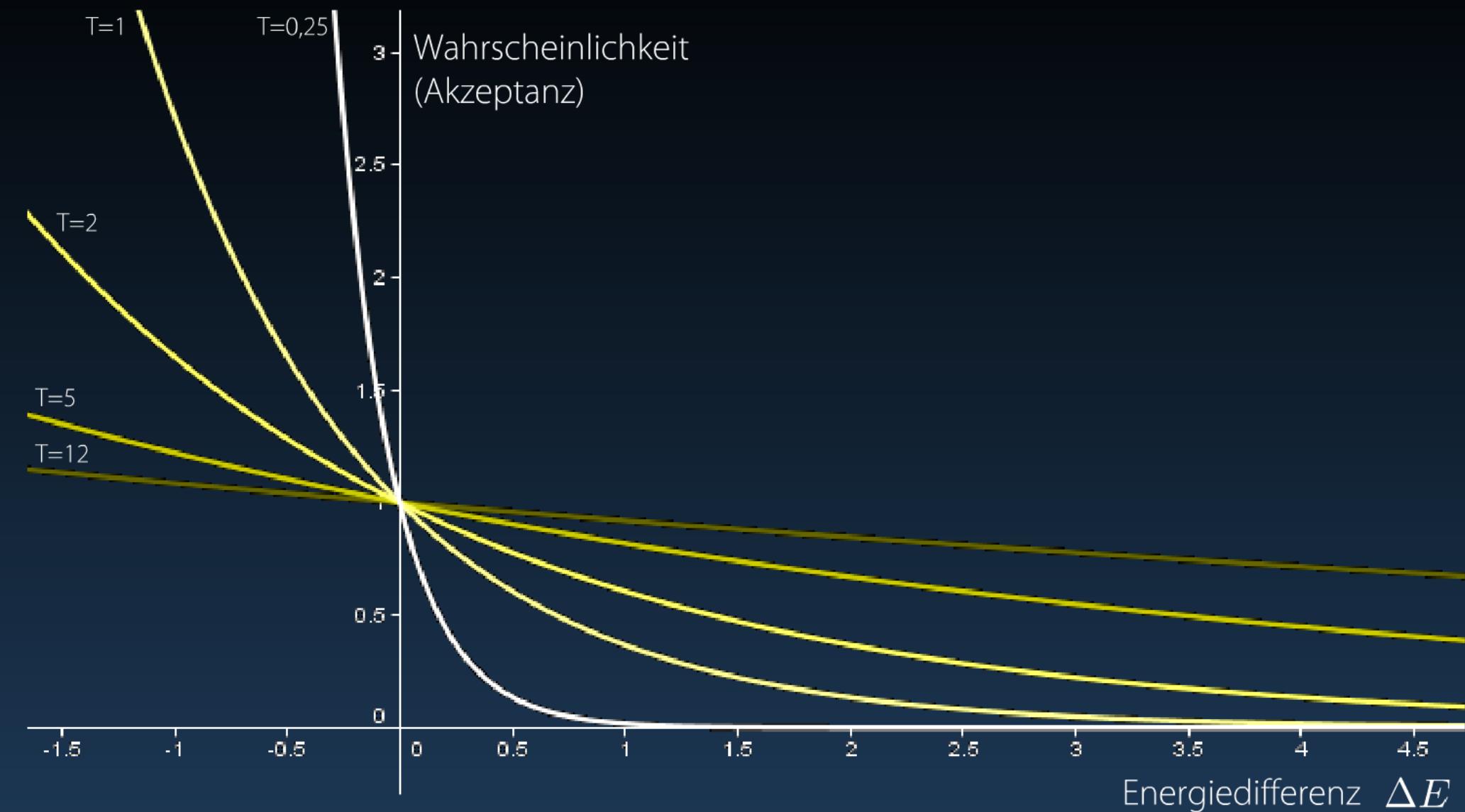
Güte (-)
 $\theta(a)$

Gesam

$- 1\right)$)



Gesamt: $\theta\left(\frac{a(a-1)}{2} + \frac{2a}{2}\right) = \theta\left(\frac{a(a+1)}{2}\right)$



Eingabe: Zustand mit Automaten

Benutzereingabe: Maximale Temperatur

Ausgabe: Zustand mit Automaten

- Variable Größe der elementaren Operationen
- Mehrfache Anwendung des Algorithmus
- Speichern der global besten Lösung
- Effizientes Generieren neuer Lösungen
- Ungenauigkeiten durch Gewichtskarte

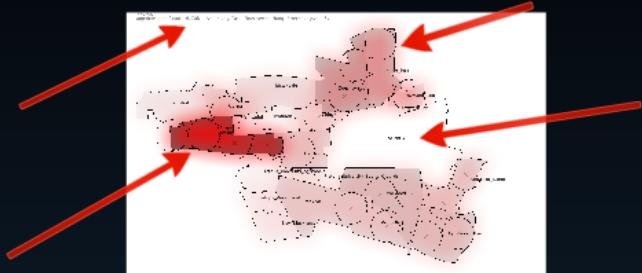
Einzelverschiebung

- betrachte Nachbarschaft der einzelnen Automaten
- Verbesserungen sofort übernehmen (gierig)
- verbessert die Lösung nur selten
- Laufzeitkomplexität: $\mathcal{O}(a^2 \cdot 4r^2)$

a: Anzahl der Automaten, r: Radius der Automaten

Berechnung der Gültigkeit: $O(a)$

Grafische Ausgabe



Ausgabedatei erzeugen

Einzelverschiebung

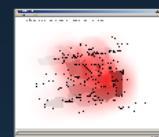
- betrachte Nachbarschaft der einzelnen Automaten
- Verbessern sofern übernehmen (gleich)
- verteilen die Lösung nur selten auf Nachbarn (Kompatibilität)

Textausgabeformat:

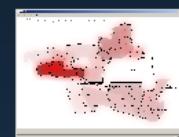
- Erste Zeile: Anzahl i der platzierten Automaten
- Nächsten i Zeilen: Koordinaten und Gewicht w_i der i platzierten Bankautomaten
- Letzte Zeile: Gesamtgewicht der Automaten
- Gewichte auf Ganzahlen gerundet

Beispiele:

Germiston



Hatfield



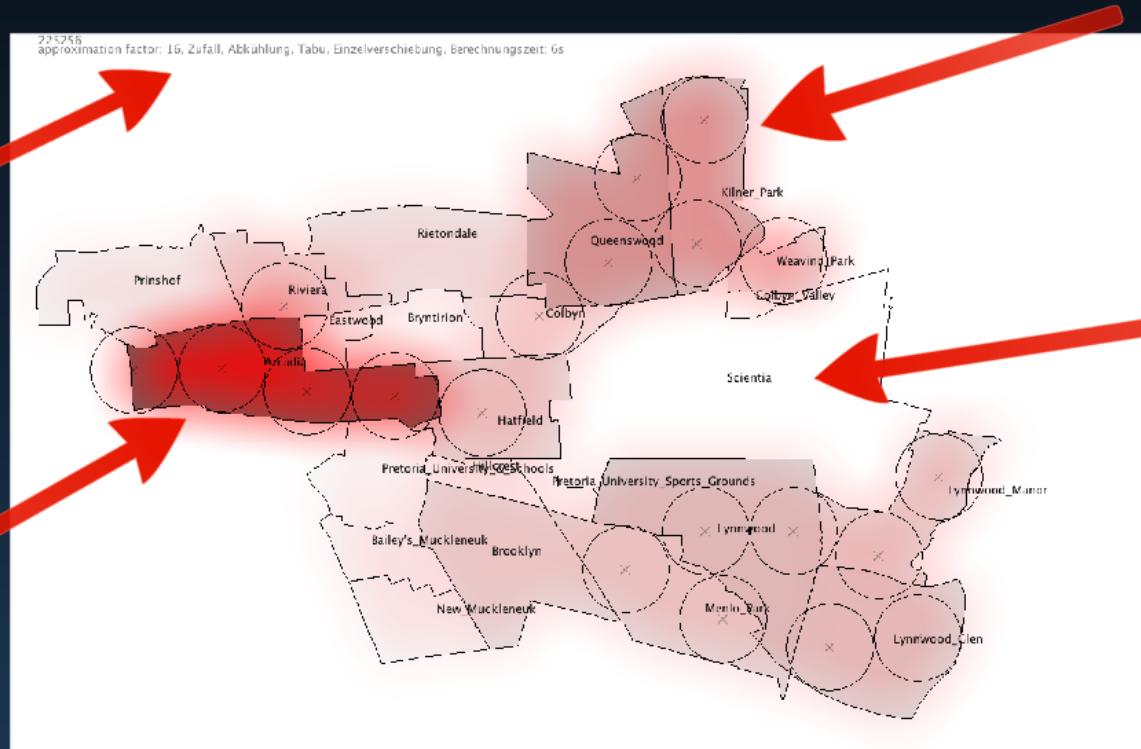
Soweto



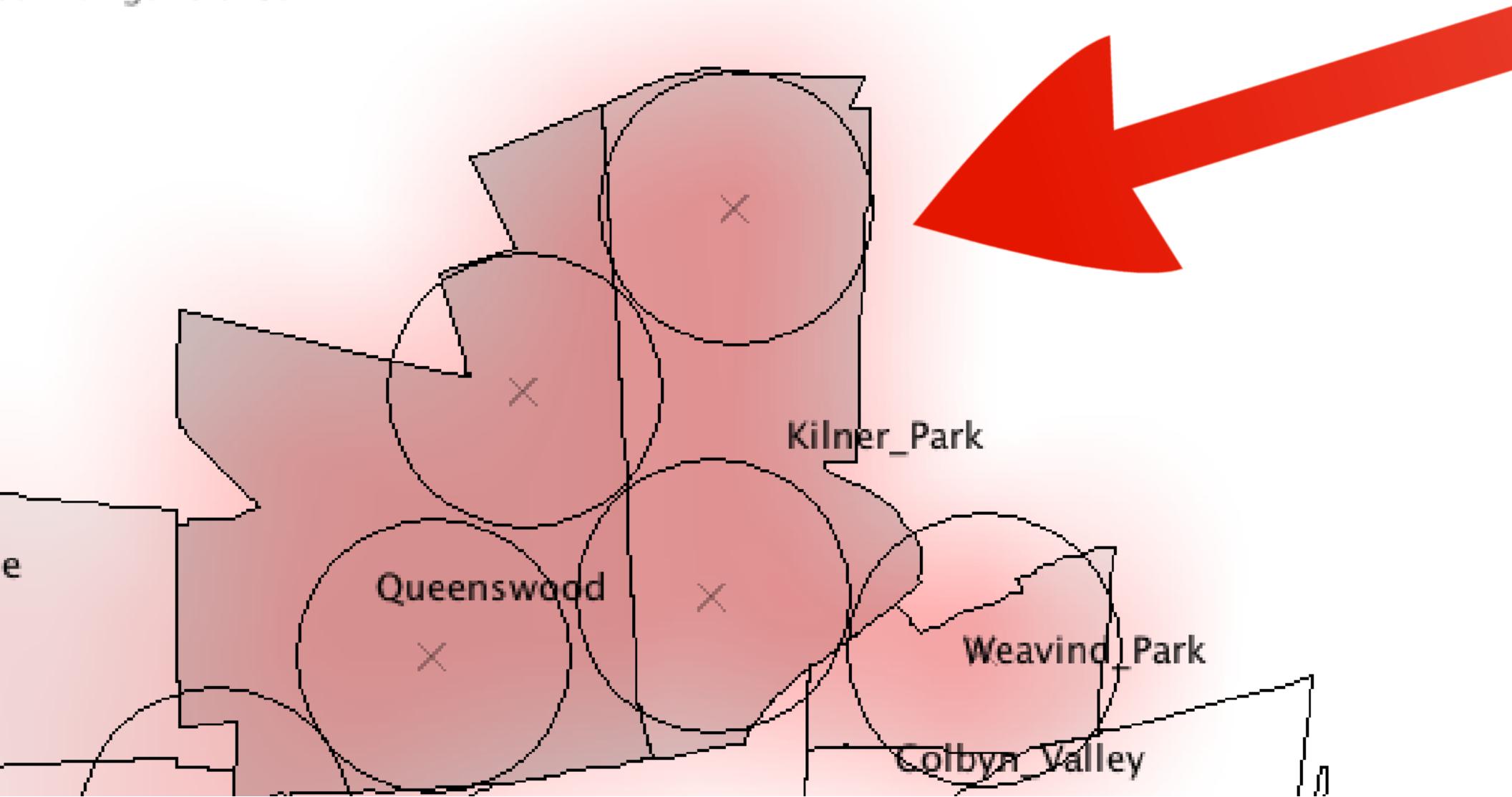
hritt

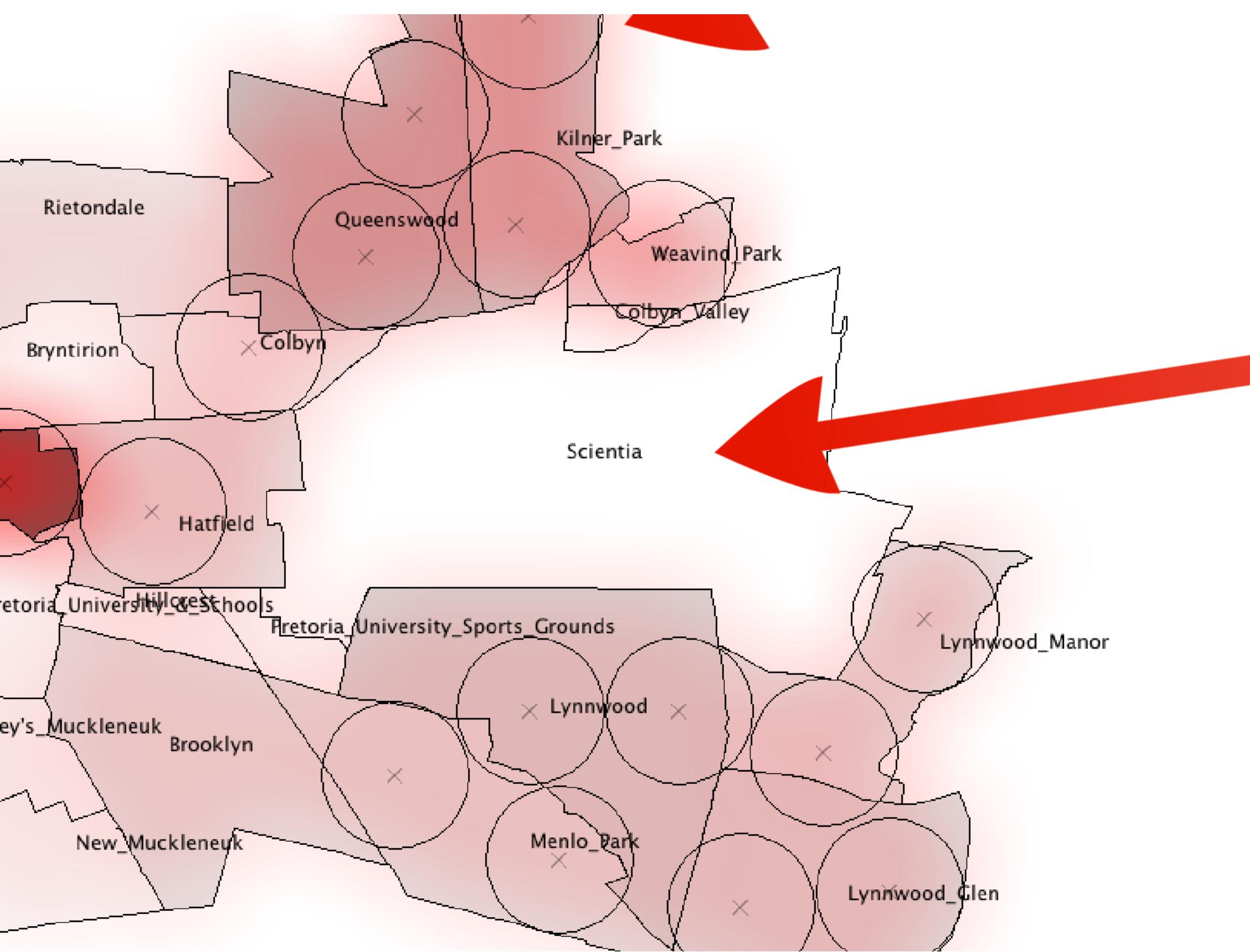
Ausgabe der Lösung

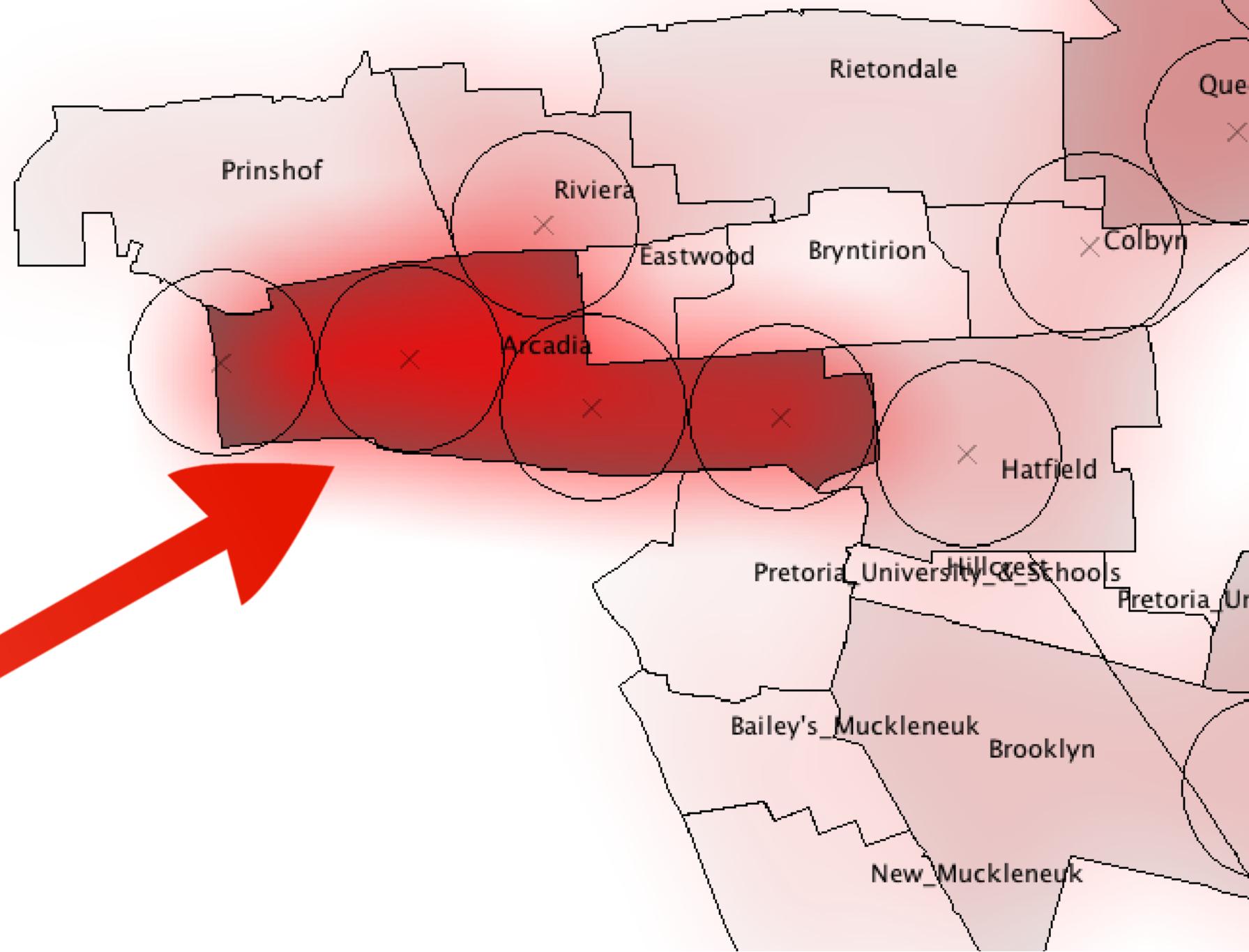
Grafische Ausgabe



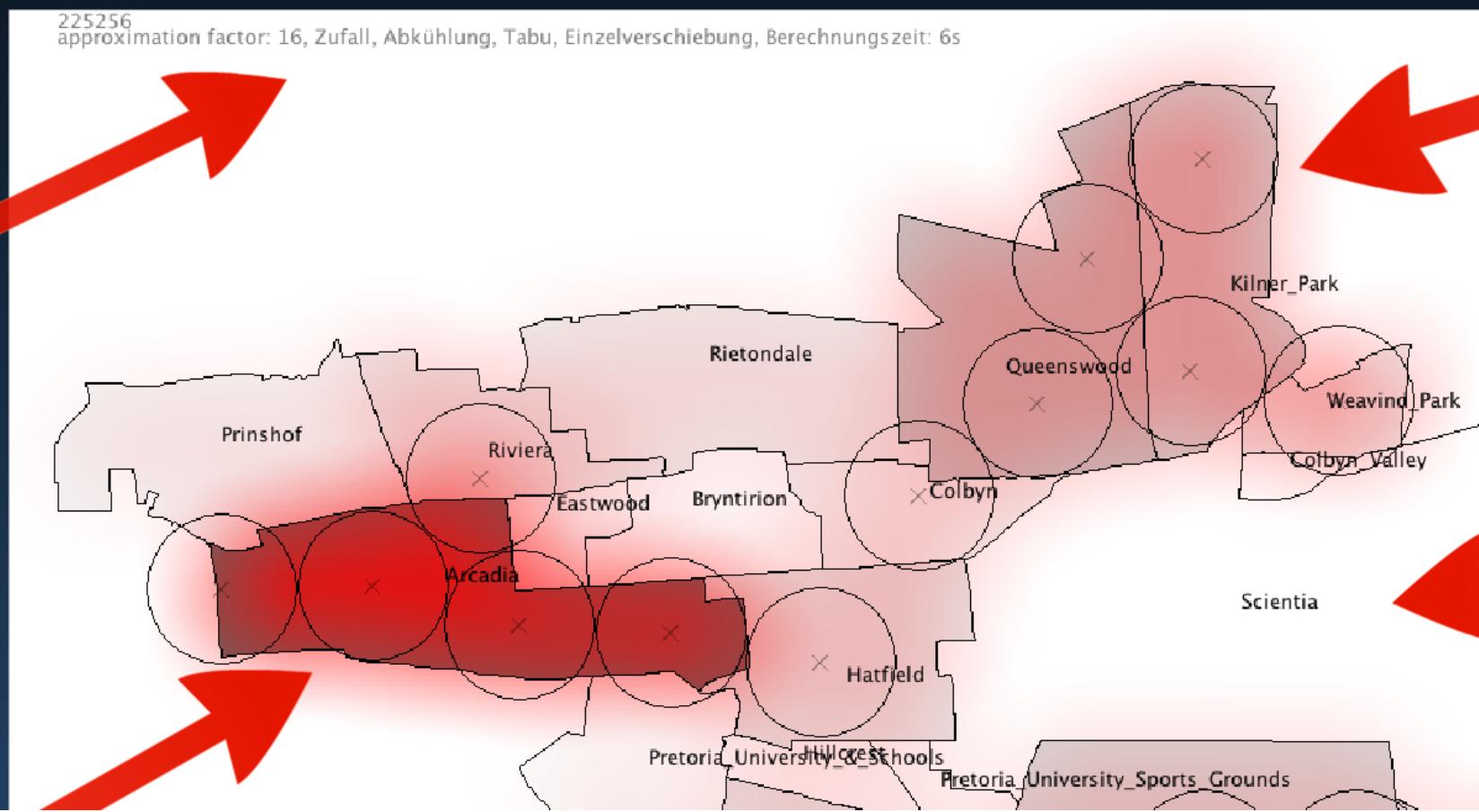
rechnungszeit: 6s







Grafische A





Ausgabedatei erzeugen

Textausgabeformat:

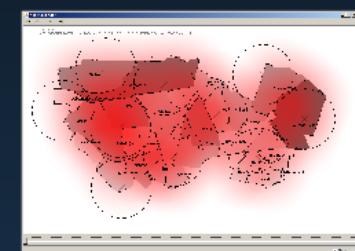
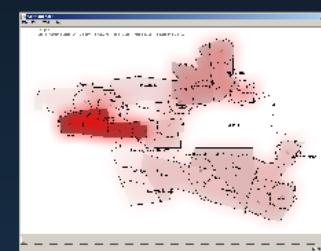
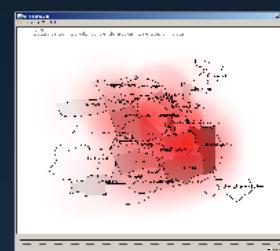
- Erste Zeile: Anzahl i der platzierten Automaten
- Nächsten i Zeilen: Koordinaten und Gewicht w der i platzierten Bankautomaten
- Letzte Zeile: Gesamtgewicht der Automaten
- Gewichte auf Ganzzahlen gerundet

Beispiele:

Germiston

Hatfield

Soweto



Ausgabe der Lösung

Ausgabeara

Textausgabeformat:

- Erste Zeile: Anzahl i der platzierten Automaten
- Nächsten i Zeilen: Koordinaten und Gewicht w
der i platzierten Bankautomaten
- Letzte Zeile: Gesamtgewicht der Automaten
- Gewichte auf Ganzzahlen gerundet



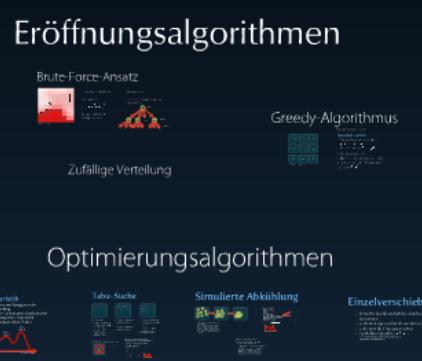
Eingabedatei einlesen



Probleminstanz vorbereiten



Berechnungsschritt



Ausgabe der Lösung



Grafische Ausgabe

Ausgabedatei erzeugen



Typischer Programmablauf

Vorteile

GUI	simpel, funktional
Algorithmen	viele verschiedene, schnell (v.a. dank Gewichtskarte, kein Polygon Clipping)
Leistungsfähigkeit	große Probleminstanzen mit vielen Automaten möglich
Sonstiges	negative Koordinaten erlaubt

Nachteile

manchmal etwas langsam
Rundungsfehler durch Pixelkarte
hoher Speicherbedarf

B&B - Bayern&Brandenburg

1 Mitglied an der HS Regensburg
3 Mitglieder am HPI in Potsdam.

Wie geht das?

Einheitliche Software
Java → LaTeX

LaTeX → TeX Live 2010

Collaborational Computing
Kommunikation



Prezi
Persönliche Treffen
Gute Konzeption

Vorteile

GUI simpel, funktional

Algorithmen viele verschiedene, schnell
(v.a. dank Gewichtskarte,
kein Polygon Clipping)

Leistungsfähigkeit große Probleminstanzen
mit vielen Automaten
möglich

Sonstiges negative Koordinaten
erlaubt

Nachteile

manchmal etwas langsam
Rundungsfehler durch
Pixelkarte

hoher Speicherbedarf

B&B - Bayern&Brandenburg

1 Mitglied an der HS Regensburg

3 Mitglieder am HPI in Potsdam.

Wie geht das?

Einheitliche Software



LATEX → TEX Live 2010

Collaborational Computing
Kommunikation



Prezi



Persönliche Treffen
Gute Konzeption



Quellen

- de.wikipedia.org (Metaheuristiken)
- Metaheuristics for Hard Optimization
by Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: (Tabu-Suche, Sim. Abkühlung)
- Taschenbuch der Algorithmen
(Simulierte Abkühlung)

Demot