



Weitere Algorithmen-Muster

- Randomisierte und heuristische Approximationsverfahren
 - Las-Vegas-Algorithmen
 - Monte-Carlo-Algorithmen
 - Genetische / evolutionäre Algorithmen
 - Simulated Annealing
- Dynamische Programmierung
- Algorithmisches Sammelsurium

Basiert auf Material von:

Kurt Bleisch
Stephan Neuhaus
Karl Rege
Marcela Ruiz
Jürgen Spielberger



Randomisierte und heuristische Approximationsverfahren

Randomisierte und heuristische Approximationsverfahren

Randomisierte Algorithmen: Zufall

- Ein randomisierter Algorithmus versucht, durch die **Wahl von zufälligen Zwischenergebnissen** zu einem (näherungsweise) korrekten Ergebnis zu gelangen.
- Man kann den **Erwartungswert der Rechenzeit und/oder die Fehler- bzw. Versagenswahrscheinlichkeit abschätzen**.
- Nicht deterministisch.
- Randomisierte Algorithmen sind in vielen Fällen einfacher zu verstehen, einfacher zu implementieren und effizienter als deterministische Algorithmen.

Heuristische Algorithmen: Bewertung

- Eine Heuristik in der Informatik ist eine **Bewertung (Fitness-, Kosten-Funktion)**, welche durch eine Berechnung ermittelt wird.
- Bei heuristischen Algorithmen wird versucht, mithilfe von Schätzungen, intuitiv-intelligentem Raten oder unter zusätzlichen Hilfsannahmen eine gute Lösung zu erzeugen, **ohne optimale Eigenschaften garantieren** zu müssen.

Randomisierte und heuristische Approximationsverfahren

Randomisierte Algorithmen:

- Las-Vegas-Algorithmen
- Monte-Carlo-Algorithmen

Heuristische Algorithmen:

- Genetische / evolutionäre Algorithmen
- Simulated Annealing, andere Namen:
 - Statistical Cooling
 - Monte Carlo Annealing
 - Probabilistic Hill Climbing
 - Stochastic Relaxation
 - Probabilistic Exchange Algorithm
 - etc.

→ Wir brauchen für diese Verfahren zunächst gute Zufallszahlen.





Zufallszahlen

Zufallszahlen (echte)

- Gleichverteilt: Jede Zahl innerhalb eines Intervalls kommt gleich häufig vor, Beispiel Zahlen eines Würfels.
- Die Wahl der nächsten Zahl einer Folge muss zufällig (ohne Bildungsgesetz) erfolgen.

Die Folge kann nicht durch Regeln erstellt werden.

→ Es ist schwierig oder sogar unmöglich, echte Zufallszahlen auf einem Computern zu erzeugen.

- Echt zufällige physikalische Prozesse:
 - Würfeln
 - radioaktiver Zerfall
 - Elektronen-Rauschen in einer Diode
 - Braunsche Bewegung
- Echter Zufall in der Mathematik:
 - nächste Zahl an einer beliebigen Stelle der Zahl π 3.1415926535897932384626433832795...

Beweis steht aus ;-)

Zufallszahlen mit Java

- Es werden Folgen von Zahlen erzeugt, die möglichst viele Eigenschaften von Zufallszahlen besitzen.
- Es wird eine Funktion bestimmt, bei der die Auswahl der nächsten Zahl scheinbar zufällig erfolgt: es kommt erst nach langer Zeit zu einer Wiederholung der Folge -> Pseudozufallszahlen

```
long seed;  
double random() {  
    long a = 25214903917; c = 11; p = 248;  
    seed = (a*seed + c) % p;  
    return z/p;  
}
```

a, c und p entsprechen den Werten des Java-Zufallsgenerators^{N1)}.

- gute Wahl von a, c und p ist schwierig; kann nur durch statische Analysen überprüft werden.
- Nachteil: keine «echte» Zufallszahl.
- Vorteil: es kann mehrmals die gleiche Folge erzeugt werden.
- in Java-Klasse Math: `Math.random()` für Bereich [0..1[
- für beliebigen anderen Bereich [0..k[= `(int)(k*Math.random())`

Zufallszahlen mit Java

Die Java-Klasse Random erlaubt grössere Flexibilität (als Math). Dafür muss ein Objekt angelegt werden.

- `Random(long seed)`
Erzeugt einen Zufallszahlengenerator. seed ist der Startwert für die erzeugten Zufallszahlen. Damit kann mehrmals die gleiche Zufallssequenz erzeugt werden.
- `Random()`
Erzeugt einen Zufallszahlengenerator. Der Startwert wird aus der aktuellen Tageszeit in Millisekunden bestimmt.
- `nextInt(int n)`
liefert eine pseudozufällige, gleichverteilte Integer-Zahl im Bereich [0..n[
- `nextDouble()`
liefert eine pseudozufällige, gleichverteilte Double-Zahl im Bereich [0..1[
- `nextGaussian()`
liefert eine pseudozufällige, Gauss-verteilte Double-Zahl mit Mittelwert 0.0 und Standard-Abweichung 1.0.



Randomisierte Approximationsverfahren

Randomisierte Approximationsverfahren

Las-Vegas-Algorithmen

Zufall nimmt Einfluss auf Ablauf des Algorithmus.

- Liefern nie ein falsches Ergebnis.
- Allenfalls wird keine Lösung gefunden.
- Beispiele:
 1. Variante von Quick-Sort, Pivotelement zufällig ausgewählt.
Der Erwartungswert der Laufzeit des randomisierten Quick-Sort-Algorithmus für n Elemente ist $O(n \log n)$.
 2. Anstelle des «sturen» Backtracking kann das Königinnen-Problem auf dem Schachbrett auch mit einem Las-Vegas-Algorithmus gelöst werden. Es werden je Schritt (Spalte) die erlaubten Positionen berechnet und eine davon zufällig ausgewählt, bis alle Spalten besetzt sind. Scheitert der Versuch, startet man wieder in Spalte 1 (ohne Backtracking).

Monte-Carlo-Algorithmen:

Zufall bestimmt Qualität des Ergebnisses.

- Liefern ein mehr oder weniger gutes Ergebnis.
- Die Qualität von Monte-Carlo-Algorithmen kann man durch eine obere Schranke für die Fehlerwahrscheinlichkeit beschreiben.
- Beispiele: siehe nächste zwei Folien.

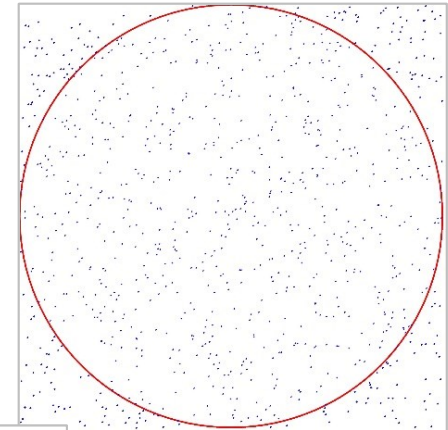
Randomisierte Approximationsverfahren

Monte-Carlo-Algorithmen, 1. Bsp.:

- Wir möchten Pi berechnen.
- Wir können eine Simulation schreiben, die 100'000 Punkte zufällig in einem Quadrat platziert. Pi ergibt sich aus dem Verhältnis der Punkt in dem Kreis zur Gesamtzahl der Punkte.

```
int nSuccess = 0;
double x, y;

for (int i = 0; i < 10000000 ; i++) {
    x = Math.random();
    y = Math.random();
    if ( x*x + y*y <= 1 ) nSuccess++;
}
System.out.println(4*(double)nSuccess/(double) 10000000);
```



3.1414236

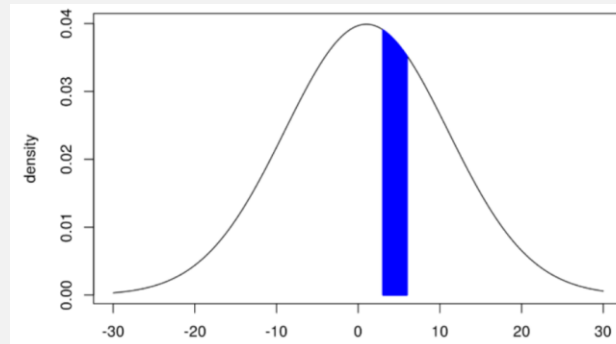
Process finished with exit code 0

Randomisierte Approximationsverfahren

Monte-Carlo-Algorithmen, 2. Bsp.:

Angenommen, wir haben eine Instanz einer Normalverteilung mit einem Mittelwert von 1 und einer Standardabweichung von 10. Wir wollen das **Integral von 3 bis 6** finden:

$$\int_3^6 \frac{1}{10\sqrt{2\pi}} e^{-\frac{(x-1)^2}{2 \cdot 10^2}} dx$$



Wir können eine Simulation schreiben, die 100'000 Proben aus dieser Verteilung entnimmt und sehen, wie viele Werte zwischen 3 und 6 liegen. Das Ergebnis, das wir erhalten haben, ist: 0.1122, was nicht allzu weit von 0.112203 entfernt ist.



Heuristische Approximationsverfahren

Heuristische Approximationsverfahren

Genetische Algorithmen

Idee:

- Eine vereinfachte Vorstellung der Evolution wird in der Informatik idealisiert und künstlich im Computer nachgebildet.
- Die Güte eines Lösungskandidaten wird explizit mit einer **Fitnessfunktion** berechnet, sodass verschiedene Kandidaten vergleichbar sind.

Bewertung

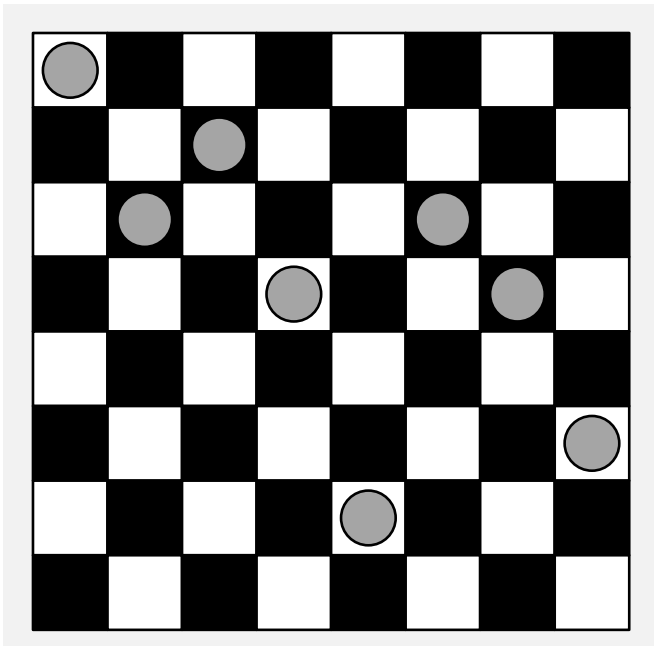
Ablauf:

- Initialisierung: Die erste Generation von Lösungskandidaten wird (meist zufällig) erzeugt.
- Durchlaufe die folgenden Schritte, bis ein Abbruchkriterium erfüllt ist:
 1. **Evaluation Fitness:** Jedem Lösungskandidaten der Generation wird entsprechend seiner Güte ein Wert der Fitnessfunktion zugewiesen. Abbruchkriterium erfüllt?
 2. **Selektion:** Auswahl der neuen Generation und Auswahl von Individuen für die Rekombination.
 3. **Rekombination:** Kombination der ausgewählten Individuen.
 4. **Mutation:** Zufällige Veränderung der Nachfahren.

Heuristische Approximationsverfahren

Genetische Algorithmen, Bsp.:

n-Damen-Problem auf Schachbrett. Speicherung: Folge von Zahlen, i. Zahl: Position von Dame in Spalte i (hatten wir schon mal, z.B. [1, 3, 2, 4, 7, 3, 4, 6]):



Fitness:

Anzahl nicht bedrohender Damenpaare:

- Schlechtester Wert: 0, jede Dame bedroht jede andere .
- Bester Wert: $n*(n-1)/2$.

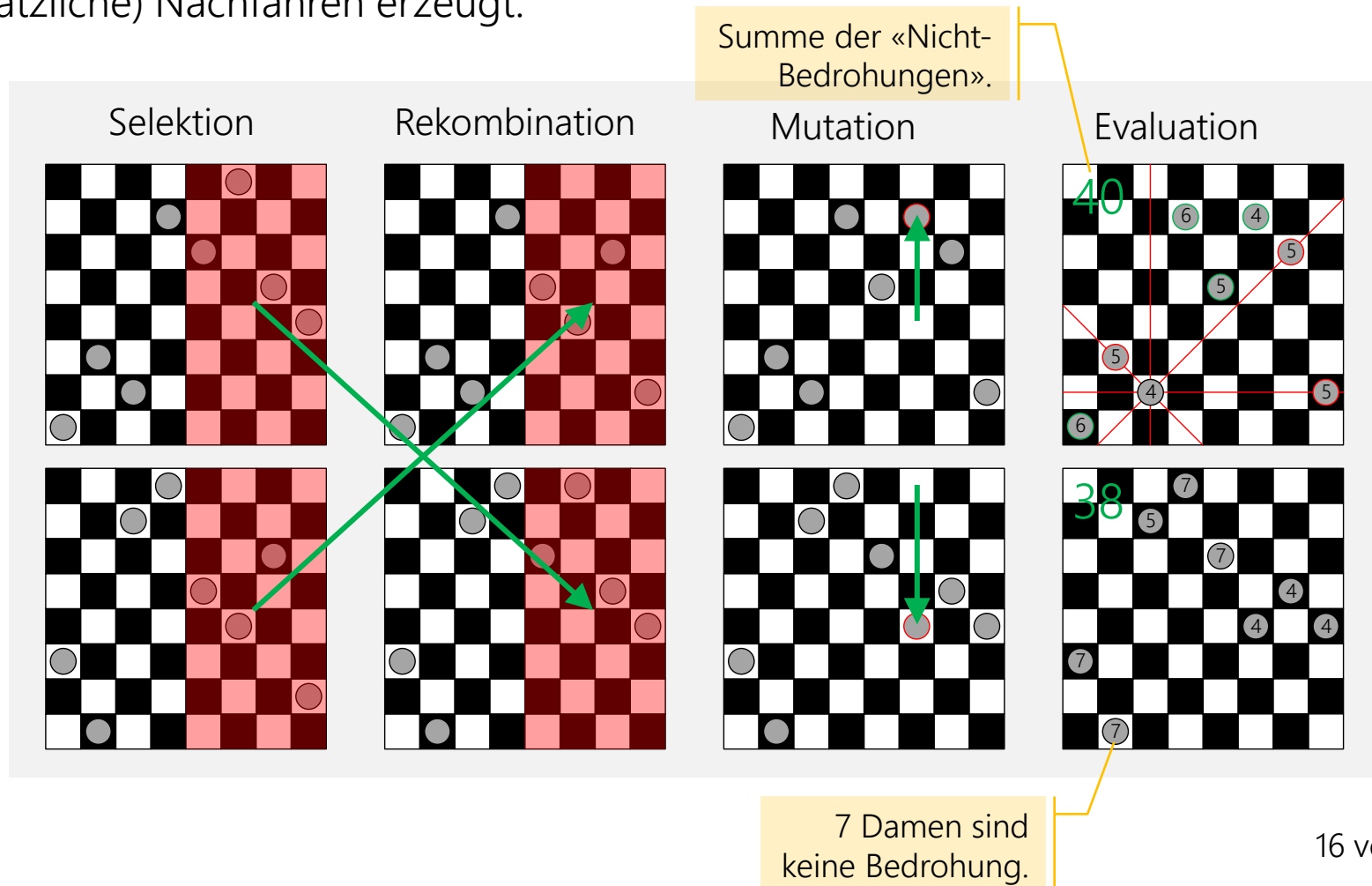
Mutation:

- Ändere einen Wert aus $\{1, \dots, n\}$ zufällig.
- Entspricht: bewege Dame in einer Spalte auf anderen Platz.

Heuristische Approximationsverfahren

Genetische Algorithmen, Bsp.:

Aus den besten Individuen werden mittels Rekombination und Mutation neue (zusätzliche) Nachfahren erzeugt:



Heuristische Approximationsverfahren

Simulated Annealing (= Simuliertes Glühen): Allgemeines Verfahren zur Lösung kombinatorischer Optimierungsprobleme.

Idee:

- Der Algorithmus basiert auf der Simulation eines in der Natur vorkommenden Prozesses, des Abkühlens von Stoffen.
- Metropolis [1953] Untersuchung von Vielkörpersystemen, Fluiden, Gasen, Festkörper auf Mittelwerte, Abweichungen von Temperatur und Druck.
 - Abkühlende Flüssigkeiten streben beim Abkühlen nach einer minimalen Energiebilanz E (stabile, regelmäßige Struktur), nur möglich bei langsamer Kühlung
 - Ist Kühlung zu schnell, ordnen sich die Teilchen unregelmäßig an, schlechte Energiebilanz (Auftreten von Nebenminima).

Heuristische Approximationsverfahren

Simulated Annealing

Prinzip:

1. Start mit der Initial-Konfiguration.
2. Wiederholtes Durchsuchen der Nachbarschaft und Auswahl eines Kandidaten.
3. Evaluiere die Kostenfunktion (oder Fitnessfunktion) und akzeptiere den Kandidaten, wenn er «besser» ist; wenn nicht, wähle einen anderen Nachbarn.
4. Stoppe, wenn die Qualität ausreichend hoch ist, wenn keine Verbesserung gefunden werden kann oder nach einer festgelegten Zeit.

Benötigt werden:

- Eine Methode zur Erzeugung der Anfangskonfiguration.
- Eine Übergangs- oder Generationenfunktion, um einen Nachbarn als nächsten Kandidaten zu finden.
- Eine Kostenfunktion.
- Ein Auswahlkriterium.
- Ein Stopp-Kriterium.

Abhängig von
«Glühtemperatur».

Heuristische Approximationsverfahren

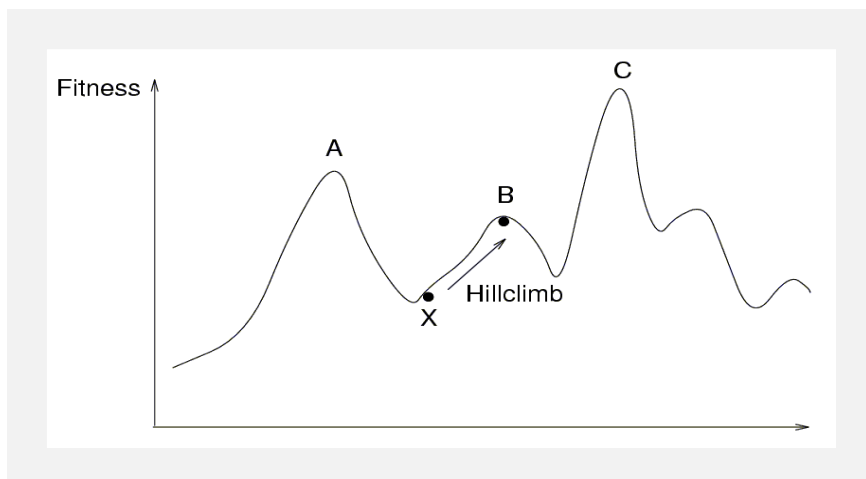
Simulated Annealing

Einfache iterative Verbesserung oder Bergbesteigung:

- Kandidat wird immer und nur dann akzeptiert, wenn die Kosten niedriger (oder die Fitness höher) als die aktuelle Konfiguration sind.

Nachteile:

- Lokales Optimum als bestes Ergebnis.
- Das lokale Optimum hängt von der Anfangskonfiguration ab.
- Im Allgemeinen keine bekannte Obergrenze für die Iterationslänge.

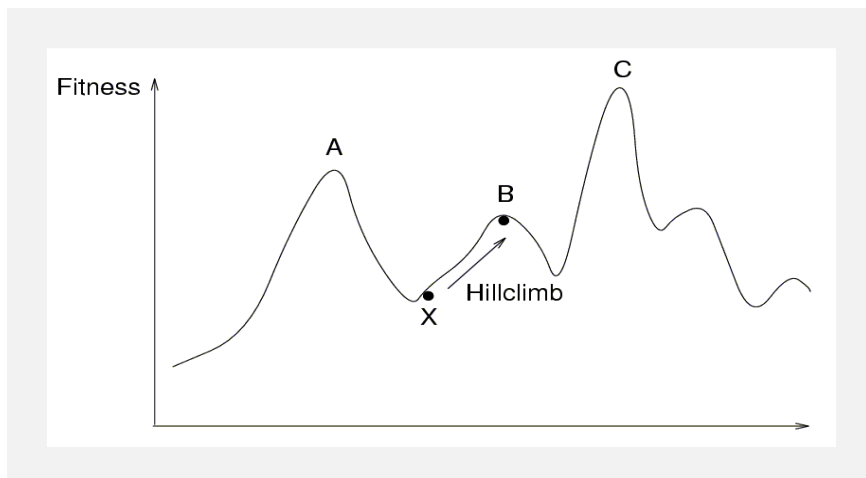


Heuristische Approximationsverfahren

Simulated Annealing

Umgang mit dem Nachteil:

- Algorithmus viele Male mit verschiedenen Anfangskonfigurationen wiederholen
- In früheren Läufen gesammelte Informationen verwenden.
- Verwenden einer komplexeren Generierungsfunktion (Kandidatensuche), um aus dem lokalen Optimum herauszuspringen.
- Verwenden eines komplexeren Bewertungskriteriums, das manchmal (zufällig) auch Lösungen weg vom (lokalen) Optimum akzeptiert:
 - Manchmal Kandidaten mit höheren Kosten akzeptieren, um dem lokalen Optimum zu entkommen.
 - Parameter der Auswertungsfunktion während der Ausführung anpassen.



Heuristische Approximationsverfahren

Simulated Annealing

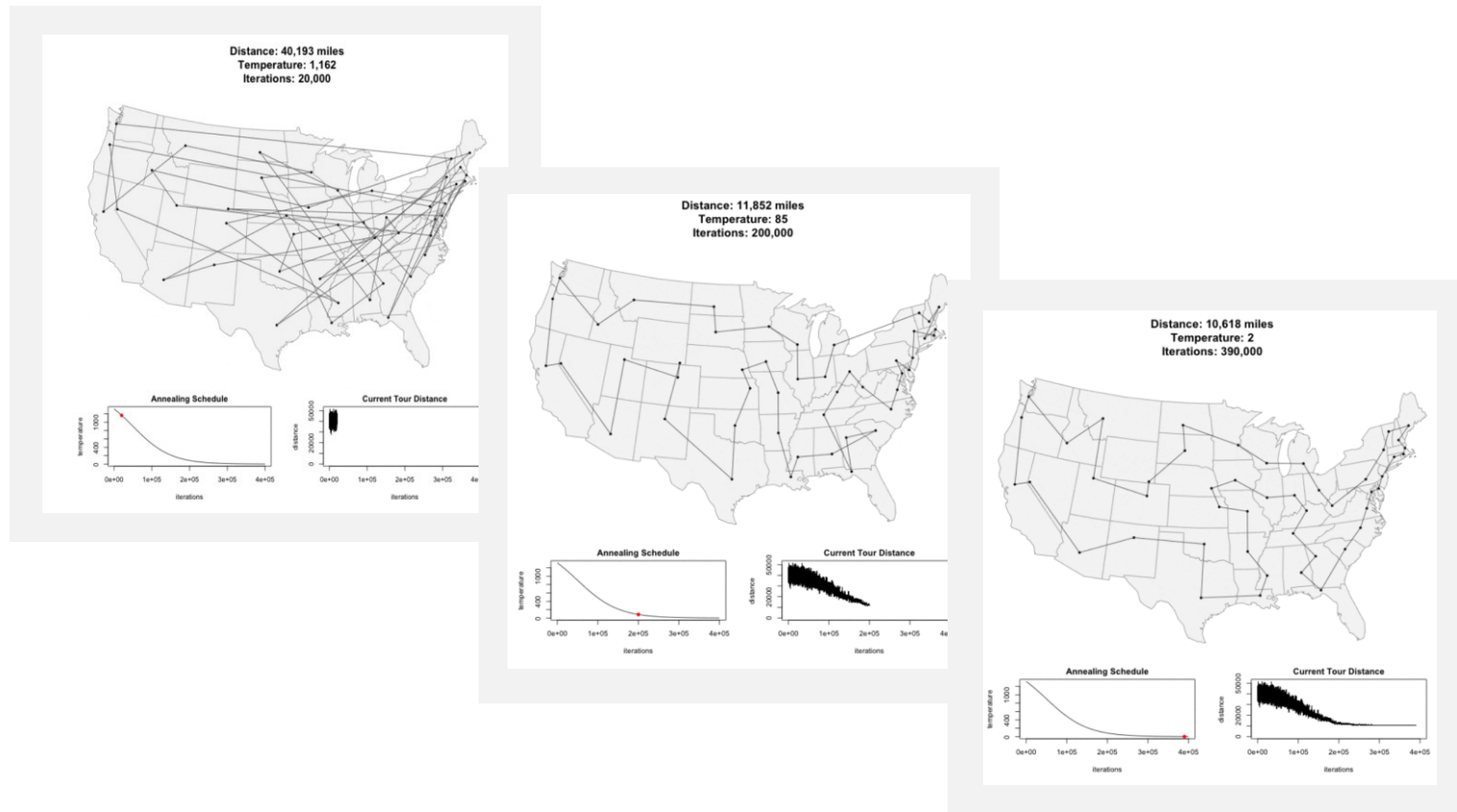
- SA ist eine allgemeine Lösungsmethode, die leicht auf eine große Anzahl von Problemen anwendbar ist.
- Das «Tuning» der Parameter (anfängliches c , Dekrement von c , Stoppkriterium) ist relativ einfach - schwierig, ein Optimum zu finden.
- Im Allgemeinen ist die Qualität der Ergebnisse von SA gut, auch wenn es viel Zeit in Anspruch nehmen kann.
- Ergebnisse sind im Allgemeinen nicht reproduzierbar: ein weiterer Durchlauf kann ein anderes Ergebnis liefern.
- SA kann eine optimale Lösung verlassen und sie nicht wieder finden (versuchen, sich an die beste bisher gefundene Lösung zu erinnern).
- Unter bestimmten Bedingungen wird nachweislich das Optimum gefunden; eine dieser Bedingungen ist, dass man Algorithmus ewig laufen lässt.

Heuristische Approximationsverfahren

Simulated Annealing, Bsp.: Salesman-Traveling-Problem

1. Start mit zufälliger Tour.
2. Wählen nach dem Zufallsprinzip eine neue Kandidatentour aus. Eine Möglichkeit, eine neue Tour auszuwählen, besteht darin, zwei Städte auf der Tour nach dem Zufallsprinzip auszuwählen und dann den Teil der Tour, der zwischen ihnen liegt, umzukehren.
3. Wenn die Kandidatentour besser ist, als die neue Tour akzeptieren.
4. Wenn die Kandidaten-Tour schlechter ist, Tour mit einer gewissen Wahrscheinlichkeit trotzdem akzeptieren:
Die Wahrscheinlichkeit, eine minderwertige Tour zu akzeptieren, hängt davon ab, wie viel länger der Kandidat ist, und von der «**Temperatur des Glühprozesses**». Eine höhere Temperatur macht es wahrscheinlicher, eine minderwertige Tour zu akzeptieren.
5. Gehen zurück zu Schritt 2 und wiederholen den Ablauf viele Male, wobei die Temperatur bei jeder Iteration etwas abgesenkt wird, bis man eine niedrige Temperatur hat und ein (hoffentlich globales, möglicherweise lokales) Minimum erreicht wurde.

Heuristische Approximationsverfahren



Letzte...
02

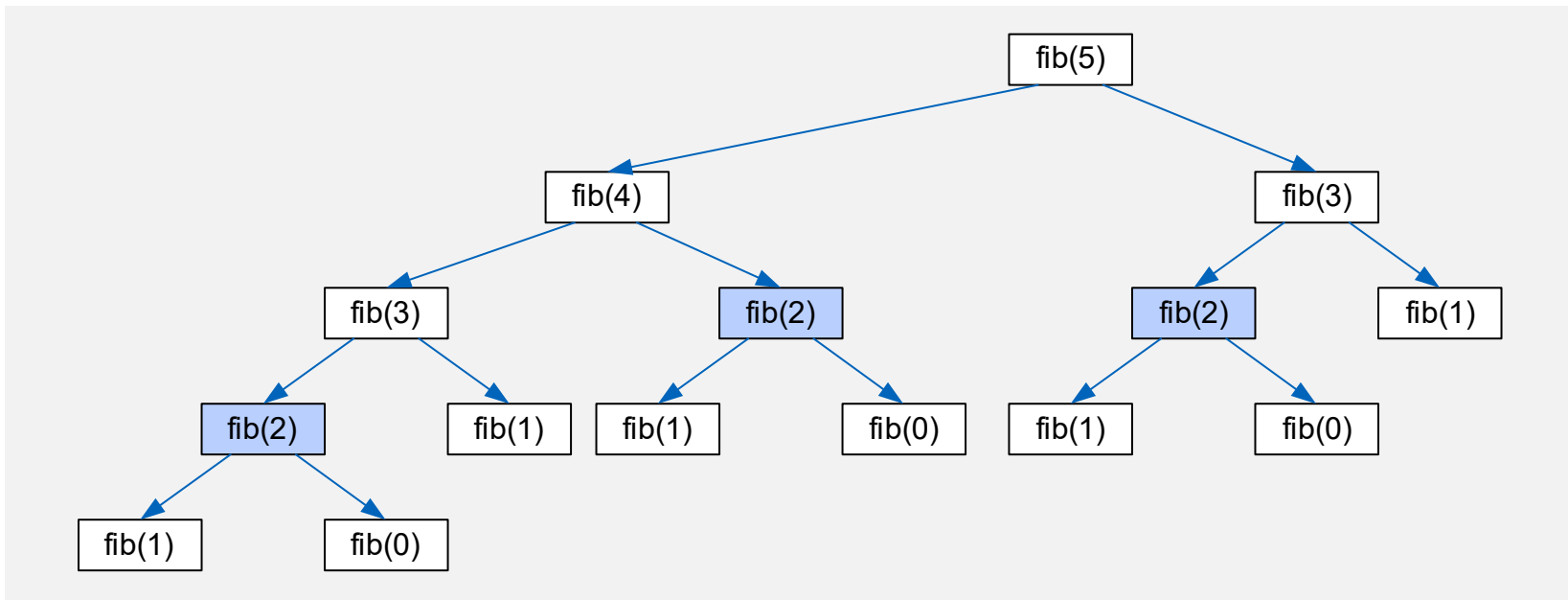


Dynamische Programmierung

Dynamische Programmierung

Idee: Optimierungsproblem durch Aufteilung in Teilprobleme lösen und Zwischenresultaten systematischen speichern.

Beispiel: Berechnung Fibonacci-Zahlen:



→ fib(2) als Zwischenresultat speichern und nur einmal berechnen.

Dynamische Programmierung

Beispiel: Rucksackproblem.

Wir hatten das Problem mit Backtracking gelöst. Wie können wir diesen Ansatz mittels dynamischer Programmierung verbessern?



17l Fassungsvermögen.



Volumen:	1l	2l	7l	8l	8l
Wert:	2'000	3'000	10'000	11'000	17'000

Dynamische Programmierung

Beispiel: Rucksackproblem.

Wir hatten das Problem mit Backtracking gelöst. Wie können wir diesen Ansatz mittels dynamischer Programmierung verbessern?

- Haben wir schon 3 Gegenstände ausprobiert, und im Baum haben zwei Knoten jetzt dasselbe Restvolumen, dann muss nur einer dieser Pfade berechnet werden, um die möglichen Optima zu berechnen.

Beispiel:

- Wir haben die Varianten zu den Gegenständen 3 bis 5 gebildet, dann haben wir zwei Varianten mit demselben Restvolumen (3 und 4, sowie 3 und 5, je 2 Liter).
- Es genügt daher, die optimale Variante mit den Gegenständen 1 und 2 (mit dem gegebenen Restvolumen von 2 Liter) zu berechnen und dieses Resultat in beiden Fällen wiederzuverwenden.

					
Volumen:	1l	2l	7l	8l	8l
Wert:	2'000	3'000	10'000	11'000	17'000



Algorithmisches Sammelsurium

Algorithmisches Sammelsurium

Was es alles **nicht in diese Vorlesung geschafft** hat:

- Parallele Algorithmen (nur kurz)
- Lineare Programmierung
- Fuzzy Logic
- Geometrische Algorithmen
- Signalverarbeitung mit FFT
- Map-Reduce Framework
- Numerische Programmierung
- Bildverarbeitung
- Machine Learning
- Etc.

Zusammenfassung

- Randomisierte und heuristische Approximationsverfahren
 - Las-Vegas-Algorithmen
 - Monte-Carlo-Algorithmen
 - Genetische / evolutionäre Algorithmen
 - Simulated Annealing
- Dynamische Programmierung
- Algorithmisches Sammelsurium



Kontrollfragen Lektion 14
nicht vergessen – heute mit
dem tapfere Schneiderlein.



Im Praktikum heute verwenden
Sie Simulated Annealing um
möglichst viele Guezli zu backen.
Das Praktikum 14 ist **optional**.






Gebrüder Grimm






Tipps zur Prüfungsvorbereitung

Tipps für die Prüfungsvorbereitung

NB: Die Tipps basieren auf den Prüfungen der Vorjahre; sie sollten als Hinweise zur Prüfungsvorbereitung verstanden werden.

 UNBEDINGT...	 ...SLIDES DER VORLESUNG VERSTEHEN INKL. CODE UND NOTIZEN	 ...PRAKTIKA VERSTEHEN MUSTER- LÖSUNGEN ANSCHAUEN
---	--	---

Ausserdem empfohlen:		
 EMPFOHLENE LITERATUR LESENFALLS DIE FOLIEN NICHT VERSTANDEN WERDEN.	 EIGENE ZUSAMMEN- FASSUNG HILFT, VERSTÄNDNIS- LÜCKEN AUFZUDECKEN	 WISSEN TESTEN KONTROLLFRAGEN (MOODLE)

Tipps für die Prüfungsvorbereitung

- Grundkonzepte studieren und verstehen.

Zum Beispiel: Was ist eine Queue? Was sind ihre Basisoperationen? Was ist eine Priority-Queue? Wie funktioniert sie?


- Hauptcharakteristika und Unterschiede verstehen

Zum Beispiel: Was ist der Hauptunterschied zwischen einer Queue und einem Stack?

- Kurze Code-Snippets schreiben

Zum Beispiel: Pseudo-Code zum berechnen des GGTs schreiben. Mit den Code-Snippets der Folien üben!

Einführung: Was ist ein Algorithmus?



- Euklidischer Algorithmus: Iterativ:

```
ggT(a, b) =  
1. a = b → a, resp. b, ist der ggT  
2. a > b → a = a - b  
3. a < b → b = b - a
```

```
while (a != b) {  
    if (a > b) a = a - b;  
    else b = b - a;  
}  
return a;
```

Kann das noch verbessert werden?

21 von 59

Tipps für die Prüfungsvorbereitung

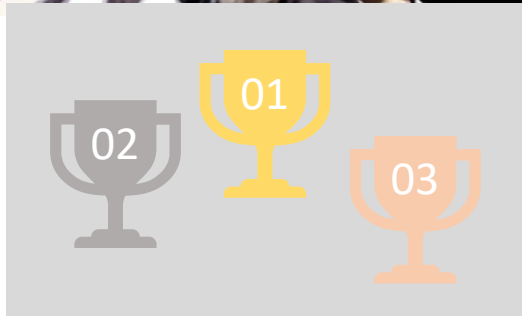
#	Vorlesung	Aufgaben (muss vor der nächsten Lektion abgeschlossen werden)	Ergänzende Unterlagen zur Vorlesung (als Ergänzung, eventuell im Voraus lesen)
01	Die Welt der Algorithmen, ADT, Stacks, Queues	Praktikum 01 (KGV, Stack, Klammer- und XML-Tester) abschliessen und abgeben, Kontrollfragen 01 beantworten.	Sedgewick/Wayne: 1.2 Datenabstraktion (S. 81 - 130)
02	O-Notation, Listen (verkettet, sortiert), Collection-Interface	Praktikum 02 (Erweiterter Klammertester, verkettete Liste, sortierte Liste) abschliessen und abgeben, Kontrollfragen 02 beantworten.	Sakke/Sattler: 13 - 13.5 Grundlegende Datenstrukturen (S. 315 - 342)
03	Sets, Generics, Generics im Collection-Framework	Praktikum 03 (Competitor, Ranking-Server, Rangliste, Namensliste) abschliessen und abgeben, Kontrollfragen 03 beantworten.	Sedgewick/Wayne: 1.3.1 APIs - 1.3.3 Verkettete Listen (S. 140 - 175)
04	Rekursion (direkt, indirekt, End-Rekursion)	Praktikum 04 (Türme von Hanoi, Koch'sche Schneeflocke, Hilbertkurve) abschliessen und abgeben, Kontrollfragen 04 beantworten.	Sakke/Sattler: 2.1.5 Rekursion (S. 27 -30)
05	Bäume, Binärbäume (unsortiert, sortiert), Traversierung	Praktikum 05 (Binärbaum, Traversierung, Rangliste, Suchbaum) abschliessen und abgeben, Kontrollfragen 04 mit Musterlösung abgleichen, Kontrollfragen 05 beantworten.	Sedgewick/Wayne: 3.2 Binäre Suchbäume (S. 424 - 444)
06	Balancierte Bäume, AVL-Baum, B-Baum, 2-3-4-Baum und Rot-Schwarz-Baum	Praktikum 06 (AVL-Baum, Baum-Höhe, Balanced-Check, Remove-Methode) abschliessen und abgeben, Kontrollfragen 06 beantworten.	Sedgewick/Wayne: 3.3 Balancierte Bäume (S. 453 - 479)
07	Graphen, Eigenschaften, Adjazenz-Liste und -Matrix, Traversierung, kürzester Pfad, topologisches Sortieren, maximaler Fluss.	Praktikum 07 (Graphen-Analyse, Adjazent-Matrix und -Liste, Graphen-Implementation, Dijkstra-Algorithmus) abschliessen und abgeben, Kontrollfragen 07 beantworten.	Sakke/Sattler: 16 - 16.4.2 Graphen (S. 445 - 471) und 16.4.5 Maximaler Durchfluss (S. 481 - 483)
08	Trial & Error, Backtracking, Komplexität von Problemen, Ziel- und Bound-Funktion, Pruning, Minimax-Algorithmus	Praktikum 08 (Tiefensuche, Labyrinth zeichnen, Wegsuche Labyrinth) abschliessen und abgeben, Kontrollfragen 08 beantworten.	Sakke/Sattler: 8. Entwurf von Algorithmen (S. 207 - 233)
09	Suchen (binär), Hashing, Kollisionen, Extendible Hashing, Hashing in Java	Praktikum 09 (Hashingtabelle, Competitor mit HashMap, eigene Hashmethode) abschliessen und abgeben, Kontrollfragen 09 beantworten.	Sakke/Sattler: 15. - 15.2 Hashing (S. 419 - 432)
10	Suche in Texten, Such-Algorithmen (Brute-Force, Knuth-Morris-Pratt), Invertierter Index, Levenshtein-Distanz, Trigramm- und phonetische Suche, Pattern-Suche (Regex)	Praktikum 10 (Levenshtein-Distanz, Tel.-Nr. suchen mit Regex, rekursive URL-Suche auf Webseiten) abschliessen und abgeben, Kontrollfragen 11 beantworten.	Sakke/Sattler: 17. Algorithmen auf Texten - 17.2 Knuth-Morris-Pratt (S. 491 - 496) Sakke/Sattler: 17.4.3 Java-Klassen für reguläre Ausdrücke (S. 510 - 511) Sakke/Sattler: 17.5 Ähnlichkeiten von Zeichenketten - 17.5.1 Levenshtein-Distanz (S. 512 - 514)
11	Sortieren 1: Insertion-, Selection- und Bubble-Sort, Stabilität	Praktikum 11 (Bubble-, Selection- und Insertion-Sort, Laufzeitvergleiche und Darstellung) abschliessen und abgeben, Kontrollfragen 10 beantworten.	Sakke/Sattler: 5.1 Suchen in sortierten Folgen - 5.2.4 Sortieren durch Vertauschen - BubbleSort (S. 117 - 131)
12	Sortieren 2: Teile und Herrsche, Quick-, Distribution- und Merge-Sort, Optimierung durch Parallelisierung (Threads)	Praktikum 12 (Beschleunigung und Parallelisierung Quick-Sort) abschliessen und abgeben, Kontrollfragen 12 beantworten.	Sakke/Sattler: 5.2.5 Sortieren durch Mischen: Mergesort - 5.2.7 Sortierverfahren im Vergleich (S. 131 - 142)
13	Fortsetzung Optimierung durch Parallelisierung (Thread-Pool, Fork/Join), Speicherverwaltung, Java-Memory-Modell, Garbage-Collection (einfache und vollautomatische Algorithmen), GC in Java, Tuning	Praktikum 13 (Mark-Sweep-GC, GC mit Generationen) abschliessen und abgeben, Kontrollfragen 13 beantworten.	
14	Randomisierte und heuristische Approximationsverfahren, dynamische Programmierung, Zusammenfassung		

Tipps für die Prüfungsvorbereitung

A large white circle is centered on an orange background. Inside the circle, the text "Q&A" is written in a grey, sans-serif font. To the left of the circle, there is a dashed yellow arc. At the bottom right of the circle, there is a solid blue dot.

Q&A

Ein Jahr im Rückblick...



Eine Frage pro
Woche!

