Optimierungsalgorithmen (OptAlgos)
Programmieraufgabe
Wintersemester 2024/25
Version 01

Programmiersprache:

Realisieren Sie am Besten die Aufgabe auf Ihrem eigenen Notebook, das Sie dann auch zum Testatstermin mitbringen. Java, C, C++, C#, Python, Scala sind ok; für andere Programmiersprachen bitte ich vorab um Anfrage, sollte aber kein Problem sein.

Algorithmen:

Implementieren Sie die folgenden Algorithmen aus der Vorlesung:

- Lokale Suche
- Greedy

Die lokale Suche soll auf dezidiert schlechten Startlösungen aufsetzen, damit die Algorithmen überzeugende Verbesserungen erzielen können.

Generische Implementation:

Achtung: Die im Folgenden genauer erläuterte Vorgabe, die Algorithmen generisch zu implementieren, ist entscheidend wichtig für die Abnahme Ihres Programms!

Die Implementationen der Algorithmen sollen so generisch sein, wie sie in der Vorlesung dargestellt werden. Das heißt, die Implementationen der Algorithmen enthalten keinerlei Information darüber, auf welches Optimierungsproblem und mit welcher Nachbarschaftsdefinition (lokale Suche) bzw. mit welcher Auswahlstrategie (Greedy) der jeweilige Algorithmus angewandt wird. Anders formuliert: Ihre Implementation eines Algorithmus muss in keinster Weise verändert werden, um sie auf ein gänzlich anderes, unvorhergesehenes Optimierungsproblem oder mit einer anderen Nachbarschaftsdefinition / Auswahlstrategie auf dasselbe Optimierungsproblem anzuwenden.

Im selben Sinne darf in der Implementation des Optimierungsproblems nichts zu finden sein, was für die verwendeten Algorithmen oder für die Nachbarschaftsdefinition / Auswahlstrategie spezifisch ist.

Beispielsweise in objektorientierten Sprachen könnte diese Anforderung auf Polymorphie hinauslaufen, das heißt, alle Details des Optimierungsproblems und der gewählten Nachbarschaft sind innerhalb der Implementation eines Algorithmus hinter Interfaces „versteckt“, also implementierende Klassen für die verschiedenen Optimierungsproble-

me und Nachbarschaften / Auswahlstrategien, und die implementierenden Klassen für ein Optimierungsproblem enthalten keine Information über Nachbarschaften / Auswahlstrategien. In vielen Sprachen (u.a. Java und C++) gibt es vergleichbar gute Möglichkeiten mittels Generizität. Generics/Templates sind in manchen Sprachen wie C++ eine gute Alternative zu Polymorphie, in Java nach meiner Erfahrung vielleicht eher doch nicht. Aber die Entscheidung bleibt Ihnen überlassen.

Algorithmische Problemstellung:

Sie wenden Ihre Algorithmen auf folgendes konkretes Optimierungsproblem an: Gegeben ist eine endliche Menge von Rechtecken mit ganzzahligen Seitenlängen sowie eine ganzzahlige *Boxlänge L*. Sie können sich auf den Fall beschränken, dass keine Seitenlänge eines Rechtecks größer als *L* ist.

Gesucht ist eine achsenparallele Platzierung aller dieser Rechtecke in der Ebene, so dass je zwei Rechtecke offen disjunkt platziert sind, das heißt, zwei Rechtecke dürfen nur Eckpunkte und (Segmente der) Randkanten gemeinsam haben, keine inneren Punkte. Rechtecke dürfen in beiden möglichen achsenparallelen Orientierungen platziert sein, also auch um 90 Grad rotiert werden. Die Rechtecke dürfen nicht beliebig platziert sein, sondern jedes Rechteck muss vollständig in einem Quadrat der Länge *L* platziert sein. Diese Quadrate heißen *Boxen* im Folgenden. Zu minimieren ist die Anzahl der Boxen, die Sie benötigen, um alle Rechtecke darin zu platzieren.

Generierung von Instanzen:

Ihr Instanzengenerator erhält als Eingaben neben *L* die Anzahl der zu generierenden Rechtecke sowie für jede der beiden Seitenlängen jeweils eine obere und eine untere Grenze. Er generiert entsprechend viele Rechtecke mit uniform zufällig gewählten Seitenlängen in dem gegebenen Intervall (alle Zufallsentscheidungen sind stochastisch unabhängig).

Nachbarschaften für lokale Suche:

Sie implementieren folgende Nachbarschaften:

- *Geometriebasiert*: Ein Nachbar lässt sich erzeugen, indem Rechtecke direkt verschoben werden, sowohl innerhalb einer Box als auch von einer Box zur anderen. Im Prinzip sind Sie frei darin, wie Sie das machen. Sie sind auch frei darin, die Zielfunktion so abzuändern, dass Nachbarn auch dann besser sind, wenn die eigentlich zu minimierende Zielfunktion nicht besser ist, der Nachbar aber nach heuristischen Überlegungen näher an einer Verbesserung dran ist. Zum Beispiel könnte es sinnvoll sein, einen Schritt zu belohnen, bei dem die Anzahl Rechtecke

in einer Box, in der ohnehin nur wenige Rechtecke in dieser Box sind, weiter verringert wird, auch wenn die Box damit (noch) nicht leer ist.

- *Regelbasiert*: Anstelle von zulässigen Lösungen, arbeitet die lokale Suche hier auf Permutationen von Rechtecken. Analog zum Greedy-Algorithmus werden die Rechtecke in der Reihenfolge der Permutation in den Boxen platziert. Als Regel Die Nachbarschaft definieren Sie durch kleine Modifikationsschritte auf der Permutation. Auch hier könnte es sinnvoll sein, Rechtecke in relativ leeren Boxen anderswo in der Permutation zu platzieren.
- *Überlappungen teilweise zulassen*: Die geometriebasierte Nachbarschaft wird angepasst auf die Situation, dass Rechtecke sich zu einem gewissen Prozentsatz überlappen dürfen. Die Überlappung zweier Rechtecke ist dabei die gemeinsame Fläche geteilt durch das Maximum der beiden Rechteckflächen. Dieser Prozentsatz ist zu Beginn 100 (so dass eine Optimallösung einfach zu finden ist). Im Laufe der Zeit reduziert sich der Prozentsatz, und Verletzungen werden hart in der Zielfunktion bestraft. Am Ende müssen Sie natürlich dafür sorgen, dass schlussendlich eine garantiert überlappungsfreie Lösung entsteht.

Auswahlstrategien für Greedy:

Sie überlegen sich und implementieren zwei grundsätzlich verschiedene Auswahlstrategien, nach denen Sie die Rechtecke reihen, das heißt, zwei Reihenfolgen, in denen die Rechtecke hergenommen und platziert werden. Natürlich müssen Sie sich auch überlegen und implementieren, wie die einzelnen Rechtecke dann offen disjunkt zu den bisher platzierten Rechtecken platziert werden sollen.

GUI:

Weiter implementieren Sie eine einfache GUI mit Visualisierung, in der der Nutzer beliebig häufig eine Instanz bestehend aus einer zufälligen Menge von Rechtecken mit ganzzahligen Seitenlängen und einer ganzzahligen Boxlänge L erzeugen und einen Algorithmus darauf anwenden kann. Der Nutzer kann die Anzahl Rechtecke, die beiden minimalen und maximalen Seitenlängen für die zufällige Erzeugung sowie die Boxlänge L individuell für jede zu generierende Instanz festlegen. Ihre GUI erlaubt es, wiederholt Instanzen zu generieren und beide Algorithmen beliebig oft und mit verschiedenen Nachbarschaften / Auswahlstrategien auf dieselbe Instanz anzuwenden.

Die Platzierung der Rechtecke zwischen zwei Iterationen eines Algorithmus wird so als Standbild visualisiert, dass (1) die Platzierung selbst übersichtlich und ausreichend groß dargestellt ist und (2) der Wechsel zwischen zwei Standbildern einsichtsreich und nicht verwirrend ist. Ihre Visualisierung überspringt automatisch Schritte des Algorithmus geeignet, um zu vermeiden, dass der Zuschauer mit minimalen Änderungen in schneller Abfolge konfrontiert wird.

Allgemeine Hinweise zur Visualisierung von Informationen:

- Vermeiden Sie nach Möglichkeit Überlappungen von Informationen.

- Worauf Sie das Augenmerk des Betrachters lenken wollen, das sollte größer/dicker und mit starken Farben gezeichnet sein. Bildelemente, die momentan nicht so wichtig sind, sollten hingegen idealerweise in neutralem, eher hellem Grau gezeichnet werden, auf jeden Fall eher blasse Farben. Und das Wichtige sollte als letztes gezeichnet werden, damit beim Zeichenprozess nicht Unwichtiges darüber gezeichnet wird.
- Farbkontraste sollten eher groß sein, wenn Sie verschiedene Bildelemente strikt voneinander unterscheiden oder sogar einander gegenüberstellen wollen, und eher klein, wenn Sie Verläufe oder Abstufungen darstellen wollen. Abstufungen sollten monochromatisch oder bichromatisch sein. Wenn positive *und* negative Stufen darzustellen sind, sollten entsprechend zwei Farben verwendet werden (und eine neutrale Farbe für die Null).
- Bedenken Sie, dass eine nicht vernachlässigbare Zahl von Menschen farbenfehlsichtig ist. Vor allem Rot/Grün-Blindheit ist verbreitet. Zum Beispiel Gelb/Blau wäre besser, da die beiden reinen Farben im RGB-Schema unterschiedlich hell sind.
- Nutzen Sie die Möglichkeiten zur Gestaltung Ihrer Visualisierung so, dass auch kleine Details sofort erkennbar und unmissverständlich sind. Diese Anforderung lässt sich leider nicht unbedingt in Regeln fassen, sondern erfordert jedes Mal auf's Neue eine gewisse Kreativität.
- Denken Sie daran, dass der Zuschauer - zum Beispiel der Prüfer – im Gegensatz zu Ihnen selbst nicht intensiv mit Ihrer GUI gearbeitet hat und daher nicht auf Anhieb alles so schnell findet und überblickt wie Sie, sondern dafür eine visuelle Gestaltung entsprechend dieser Hinweise benötigt.

Testumgebung:

Schließlich schreiben Sie unabhängig von der GUI noch eine kleine Testumgebung, die mit einer Folge von Tupeln (Anzahl Instanzen, Anzahl Rechtecke, zwei minimale Seitenlängen, zwei maximale Seitenlängen, Boxlänge) parametrisiert ist. Für jedes Tupel werden mit der ersten der beiden oben beschriebenen Instanzgeneratoren so viele Instanzen generiert und durchgerechnet, wie es das erste Element des Quintupels jeweils besagt. Jeder Algorithmus wird auf jede Instanz angewandt, und die erreichten Zielfunktionswerte sowie die aufgewendeten Laufzeiten (CPU thread time) werden geeignet mitprotokolliert, um die Güte der Algorithmen miteinander zu vergleichen.

Ihre Testumgebung sollte in zwei Versionen aufrufbar sein:

1. Mit ausreichend wenigen und ausreichend kleinen Instanzen, so dass die Testumgebung in wenigen Minuten fertig wird und daher bei der Abnahme vorgeführt werden kann.
2. Mit ausreichend vielen und ausreichend großen Instanzen, so dass ein Durchlauf durch die Testumgebung aussagekräftig für die Korrektheit der Algorithmen sein sollte. Zur Abnahme bringen Sie ein Protokoll eines solchen Durchlaufs mit.

Tuning der Algorithmen und der Nachbarschaften:

Da die zu implementierenden Algorithmen in der Reinform, wie sie in der Vorlesung vorgestellt werden, doch recht langsam sind, müssen Sie sie optimieren. Soweit notwendig, können Sie sich von der Reinform lösen. Das kann zum Beispiel darin bestehen, dass Sie nicht die ganze Nachbarschaft durchgehen, sondern nur relativ wenige, nach einer von Ihnen zu bestimmenden heuristischen Regel besonders aussichtsreich erscheinende Nachbarn. Natürlich können Sie auch die Nachbarschaften selbst „tunen“. Falls Sie im Zweifel sind, ob eine angedachte Tuning-Maßnahme ok ist, fragen Sie bei mir nach.

Ein Ziel, das für Kundengespräche, Vorstandpräsentationen usw. wesentlich ist, soll durch Ihre Implementation auf Ihrem Notebook erfüllt werden:

Jeder der Algorithmen soll bei Instanzen bis zu tausend Rechtecken in 10 Sekunden zu einer Lösung kommen können, die mit bloßem Auge nicht mehr verbessert werden kann.