Aufgabe 1: LaTeX-Dokument

Teilnahme-ID: ?????

Bearbeiter/-in dieser Aufgabe:
Vor- und Nachname

25. Februar 2023

Inhaltsverzeichnis

Anleitung: Trage oben in den Zeilen 8 bis 10 die Aufgabennummer, die Teilnahme-ID und die/den Bearbeiterin/Bearbeiter dieser Aufgabe mit Vor- und Nachnamen ein. Vergiss nicht, auch den Aufgabennamen anzupassen (statt „LaTeX-Dokument“)!

Dann kannst du dieses Dokument mit deiner LaTeX-Umgebung übersetzen.

Die Texte, die hier bereits stehen, geben ein paar Hinweise zur Einsendung. Du solltest sie aber in deiner Einsendung wieder entfernen!

1 Lösungsidee

1.1 Einleitung

Es ist $\mathcal{NP}$-schwer, das weniger krumme Touren-Problem (WKT) optimal zu lösen. Um das zu zeigen, wird eine Reduktion vom eulerschen Pfad-Problem des Handlungsreisenden (E-PTSP) skizziert, welches bekanntermaßen $\mathcal{NP}$-schwer ist. E-PTSP lautet folgendermaßen: Sei $P \subset \mathbb{R}^2$ eine endliche Menge. Dann wird eine Reihenfolge von P gesucht, bei der die Strecke zwischen aufeinanderfolgenden Punkten minimal ist.
E-PTSP kann nun auf WKT reduziert werden, in dem jeder der Punkte P durch eine Struktur S mit sehr kleinem d ersetzt wird. Wie das Diagramm zeigt, kann Struktur S aus beliebigen Richtungen angeflogen werden. Wenn die Struktur sehr klein ist, verhält sie sich wie ein Punkt im E-PTSP. Die Reihenfolge, in welcher die Strukturen optimal angeflogen werden, ist demnach auch die optimale Reihenfolge der Punkte P für E-PTSP.

Unter der Annahme $\mathcal{P} \neq \mathcal{NP}$ gibt es deshalb keinen Algorithmus, der WKT in polynomieller Zeit optimal lößt. Stattdessen stelle ich einen Algorithmus vor, der eine optimale Lösung in polynomieller Zeit annähert, sowie einen Lösungsansatz, der das Problem in exponentieller Zeit optimal lößt.

1.2 Annäherung

Das Problem wird mit Hilfe des Simulated Annealing gelößt.

procedure SIMULATED ANNEALING(Startlösung S, Temperatur T_0, Abkühlkoeffizient α, Iterationen n)
  $S_{Beste} \leftarrow S$
  $C_{Beste} \leftarrow C(S)$
  $T \leftarrow T_0$
  for $i \in 1, 2, \ldots, n$ **do**
    $S_{Neu} \leftarrow$ Nachbarlösung von S
    $C_{Neu} \leftarrow C(S_{Neu})$
    if $C_{Neu} < C_{Beste}$ **then**
      $C_{Beste} \leftarrow C_{Neu}$
      $S_{Beste} \leftarrow S_{Neu}$
    end if
    $r \leftarrow$ Zufallszahl aus $[0, 1]$
    if $r < \exp(\frac{C(S)-C_{Neu}}{T})$ **then**
      $S \leftarrow S_{Neu}$
    end if
    $T \leftarrow \alpha T$
  end for
  return S_{Beste}
end procedure

Gültige Lösungen sind hier alle möglichen Permutationen von N Landeplätzen. Die Beschränkung, dass eine Lösung keine Spitzen Winkel beinhalten darf, wird über die Kostenfunktion $C(S)$ kodiert. Die Kostenfunktion setzt sich zusammen aus der Länge des von S gebildeten Pfades und einer Gebühr $g \in \mathbb{R}$ für jeden spitzen Winkel. g ist eine obere Schranke der Länge eines Pfades, wodurch für jeden Pfad S' mit weniger spitzen Winkeln als S gilt $C(S) > C(S')$. Diese obere Schranke erschließt sich aus der Überlegung, dass eine mögliche Lösung mindestens $N - 1$ Kanten haben muss. Dieser Pfad kann höchstens die Kosten der teuersten $N - 1$ Kanten haben.
Um Nachbarn einer Lösung zu finden, habe ich die für das klassische TSP bekannten Mutationsoperatoren **Insert**, **Displace**, **Reverse-Displace** //TODO quote mutationsoperatoren und einen eigenen, auf dem 3-Opt-Verfahren basierenden Mutationsoperator, den ich **3-Opt** nenne, verwendet. Es wird zufällig einer der Operatoren angewendet. Die Operatoren basieren auf der Darstellung eines Pfades als Permutation von $(1, 2, \ldots, N)$. In dieser Darstellung gibt das k-te Element der Permutation an, welcher Landeplatz als k-tes besucht wird.

Insert wählt einen zufälligen Landeplatz der Permutation aus und setzt ihn an eine zufällige neue Stelle.

Displace wählt ein zufälliges Segment der Permutation aus und setzt es an eine zufällige neue Stelle.

Reverse-Displace wählt ein zufälliges Segment der Permutation aus und setzt es in umgekehrter Reihenfolge an eine zufällige neue Stelle.

3-Opt teilt den Pfad an zufälligen Stellen in vier Segmente auf. Diese werden in einer zufälligen neuen Reihenfolge zusammengesetzt, wobei sie mit Wahrscheinlichkeit 0.5 umgekehrt werden. Der Operator ähnelt der 3-Opt-Heuristik, welche 3 Kanten einer Lösung löscht und die Segmente in einer Reihenfolge zusammensetzt, welche die Gesamtkosten minimiert, denn er ersetzt auch 3 Kanten durch neue.

1.3 Optimale Lösung

Zur optimalen Lösung von WKT wird dieses als Integer-Programming-Problem formuliert. Integer Programming ist $\mathcal{NP}$-schwer, weshalb dafür nur Algorithmen exponentieller Laufzeit bekannt sind.
Sei also I eine Instanz von WKT mit den Punkten $P \subset \mathbb{R}^2$. Die Landeplätze sind $L = \{1, 2, \ldots, |P|\}$ Die Variable $x_{i,j} \in \{0, 1\}$ it $i, j \in L; i < j$ kodiert, ob die Route die Punkte P_i und P_j direkt nacheinander anfliegt, wobei nicht festgelegt ist, welcher zuerst angeflogen wird. Man sagt auch: Die Lösung hat eine Kante zwischen P_i und P_j. Die Variable $y_i \in \{0, 1\}$ mit $i \in L$ kodiert, ob der Pfad bei P_i anfängt oder

endet. Das Integer-Programming-Problem lautet dann:

$$\text{minimiere} \sum_{i=1}^{|P|} \sum_{j=1, j>i}^{|P|} c_{i,j} x_{i,j} \text{ mit} \tag{1}$$

$$\sum_{j=i+1}^{|P|} x_{i,j} + \sum_{k=1}^{i-1} x_{k,i} + y_i = 2 \qquad \text{Für alle } i \in \{1, 2, \ldots, |P|\} \tag{2}$$

$$\sum_{i=1}^{|P|} y_i = 2 \tag{3}$$

$$x_{\min(i,j),\max(i,j)} + x_{\min(j,k),\max(j,k)} \leq 1 \qquad \text{Für alle } i, j, k \in L; i \neq j; j \neq k; i \neq k; P_i, P_j, P_k \text{ bilden spitzen Winkel} \tag{4}$$

$$\sum_{i \in Q} \sum_{j \in Q; j != i} x_{\min(i,j),\max(i,j)} \leq |Q| - 1 \qquad \text{Für alle } Q \subset L; |Q| \geq 2 \tag{5}$$

(6)

Der zu minimierende Term (1) bedeutet, dass die Gesamtlänge des Pfades möglichst kurz sein soll. (2) sorgt dafür, dass auf jeder Landeplatz zwei Landeplätze hat, die direkt vor und nach ihm im Pfad angeflogen werden, oder genau einen, wenn der Landeplatz an einem Ende des Pfades liegt. (3) sorgt dafür, dass es nur 2 solcher Endlandeplätze gibt. (4) verhindert, dass es im Pfad spitze Winkel gibt. Wenn zwei Kanten einen Spitzen Winkel bilden, dann darf maximal eine dieser Kanten in der Lösung sein. (5) sorgt dafür, dass die Lösung nur ein Pfad ist, und nicht ein Pfad und Kreise durch die restlichen Knoten. Weil (5) exponentiell viele Bedingungen beinhaltet, muss es als Lazy Constraint formuliert werden. Wenn es eine potentielle Lösung gibt, wird ein Algorithmus auf diese Lösung angewendet, der Kreise in ihr sucht, und aus den Kreisen die entsprechenden Bedingungen formuliert. Auf diese Weise kann diese Bedingung auch Schnittebenen aus Relaxationen formulieren, wobei in der Relaxationslösung Kreise gesucht werden.
Dieses Integer-Programming-Problem kann mit dem Branch-and-Cut-Algorithmus gelößt werden. Als Startlösung verwende ich dabei das Ergebnis des beschriebenen Annäherungsalgorithmus.

2 Umsetzung

Hier wird kurz erläutert, wie die Lösungsidee im Programm tatsächlich umgesetzt wurde. Hier können auch Implementierungsdetails erwähnt werden.

3 Beispiele

Genügend Beispiele einbinden! Die Beispiele von der BwInf-Webseite sollten hier diskutiert werden, aber auch eigene Beispiele sind sehr gut – besonders wenn sie Spezialfälle abdecken. Aber bitte nicht 30 Seiten Programmausgabe hier einfügen!

4 Quellcode