Programmierpraktikum Diskrete Optimierung (Modul P2C1)

Thema: Das Traveling Salesman Problem (TSP)

Veranstaltungshomepage:
`http://www.or.uni-bonn.de/lectures/ss19/praktikum_ss19.html`

1 Programmiersprache

Die Implementierungen müssen in C++ ausgeführt werden und unter Linux mit g++ kompilierbar sein. Sie müssen dem ANSI C++11-Standard genügen. Außer den Standard-Bibliotheken dürfen keine Hilfsbibliotheken verwendet werden, es sei denn es wird in der Aufgabenstellung ausdrücklich erwähnt. Der Code muss verständlich programmiert und kommentiert werden. Selbstverständlich müssen die Programme fehlerfrei funktionieren. Insbesondere werden sie mit dem Programm valgrind (`http://valgrind.org`) überprüft, und valgrind-Fehler führen zu Abzügen. Weitere Hinweise zu gutem Programmierstil finden Sie auch auf der oben angegebenen Internetseite.

2 Das Traveling Salesman Problem (TSP)

Eine Instanz des TSP besteht aus einem vollständigen Graphen auf einer Knotenmenge V sowie einer Funktion $c : V \times V \to \mathbb{R}$. Gesucht ist ein Hamiltonkreis (auch *Tour* genannt) der kleinstmögliche Länge bezüglich c hat. Wir betrachten hier nur das *symmetrische* TSP, d.h. wir setzen voraus, dass $c(a, b) = c(b, a)$ für alle $a, b \in V$ gilt. Falls die Funktion c die Dreiecksungleichung erfüllt, so handelt es sich um eine Instanz des *metrischen* TSP. Bei einer *euklidischen* TSP-Instanz ist die Knotenmenge eine Punktmenge im $\mathbb{R}^2$ und die Funktion c gibt den euklidischen Abstand zwischen den Punkten an.

Das TSP und auch die metrische und euklidische Variante sind NP-schwer. Falls $P \neq NP$, so gibt es für das allgemeine symmetrische TSP keinen polynomiellen Approximationsalgorithmus, der konstante Güte erreicht. Für das metrische TSP erreicht der beste bekannte Approximationsalgorithmus die Güte 3/2 und für das euklidische TSP gibt es ein PTAS.

3 Einführungsaufgabe

Als Teil der Einführungsaufgabe sollen zunächst Routinen geschrieben werden, mit denen TSP-Instanzen im TSPLIB-Format eingelesen werden können. Das TSPLIB-Format ist hier beschrieben:

`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf`

Da wir das TSPLIB-Format nur für symmetrische TSP-Instanzen benutzen, reicht es aus, wenn Ihre Implementierung folgende Schlüsselwörter korrekt umsetzt: `COMMENT`, `DIMENSION`, `EDGE_WEIGHT_TYPE`, `EOF`, `NAME`, `NODE_COORD_SECTION` (entgegen der Beschreibung in der Dokumentation ist `TWOD_COORDS` der Default-Wert für `NODE_COORD_TYPE`) und `TYPE`. Als Wert für

`EDGE_WEIGHT_TYPE` müssen `EUC_2D` und `CEIL_2D` korrekt umgesetzt werden. Als Wert für `TYPE` ist nur `TSP` zulässig.

Geben Sie die Länge einer Tour durch die Knoten in der Reihenfolge $1, 2, \ldots, n$ aus. Darüberhinaus soll die folgende Greedy-Strategie zum Finden einer Tour implementiert werden: Sortiere die Kanten des vollständigen Graphen $G = (V, E)$ aufsteigend nach Länge. Sei $e_1, e_2, \ldots, e_m$ die sortierte Liste. Starte mit einer leeren Menge F und für $i = 1, \ldots, m$ teste, ob $F \cup \{e_i\}$ Teilmenge der Kantenmenge einer Tour ist und falls ja, so füge e_i zu F hinzu. Die Laufzeit Ihrer Implementierung muss $O(n^2 \log n)$ sein. Das Programm soll als Parameter eine Datei im TSPLIB-Format erhalten und als Ausgabe die gefundene TSP Tour im TSPLIB-Format sowie deren Länge ausgeben.

Zu beachten ist, dass das TSPLIB-Format zwar nicht-ganzzahlige Koordinaten erlaubt, alle Kostenfunktionen jedoch ganzzahlig sind. Alle Rechnungen müssen exakt in Integerarithmetik durchgeführt werden.

Spätester Abgabetermin für die Einführungsaufgabe ist der 28.4.2019. Es wird jedoch empfohlen, die Implementierung der Gesamtaufgabe schon während der vorlesungsfreien Zeit weitestgehend abzuschließen, da in der Vorlesungszeit häufig wenig Zeit bleibt.

4 Hauptaufgaben

Zusätzlich zu einer der nachfolgend beschriebenen Aufgaben muss jede Implementierung folgende weitere Schlüsselwörter des TSPLIB-Formats unterstützen: `EDGE_WEIGHT_FORMAT` und `EDGE_WEIGHT_SECTION`. Als Werte für `EDGE_WEIGHT_FORMAT` sind `FULL_MATRIX`, `LOWER_DIAG_ROW`, `UPPER_DIAG_ROW` und `UPPER_ROW` zu unterstützen.

Spätester Abgabetermin für die Hauptaufgabe ist der 30.6.2019.

4.1 Aufgabe 1: Eine effiziente Implementierung des 2-OPT Algorithmus.

Jonas Kobler

Betreuer: Benjamin Rockel

Ein 2-OPT-Schritt lässt sich einfach in $O(n^2)$ implementieren. Man kann jedoch auch eine Laufzeit von $O(n \log n)$ für eine Folge von 2-OPT-Schritten erreichen, wenn man einmalig $O(n^2)$ Preprocessinglaufzeit erlaubt. Beschrieben wird dieser Ansatz in der Arbeit

Mark de Berg, Kevin Buchin, Bart M. P. Jansen, Gerhard Woeginger: *Fine-Grained Complexity Analysis of Two Classic TSP Variants.* arXiv:1607.02725v1 [cs.DS], 2016.

Er basiert auf der Datenstruktur der reversable AVL-trees, die in der Arbeit

M.Chrobak, T.Szymacha, A.Krawczyk: *A data structure useful for finding Hamiltonian cycles.* Theoretical Computer Science, Volume 71, Issue 3, 10 April 1990, pages 419-424.

beschrieben werden. Als Zusatzaufgabe kann eine in obiger Arbeit beschriebene effiziente Implementierung von 3-OPT realisiert werden.

4.2 Aufgabe 2: Eine effiziente Implementierung des Lin-Kernighan-Algorithmus.

Karl Welzel

Betreuer: Markus Ahrens

Es soll der Lin-Kernighan-Algorithmus implementiert werden, wie beschrieben in Kapitel 21 von: B. Korte und J. Vygen: Kombinatorische Optimierung, Springer, 3. Auflage, 2018. Darüber hinaus sollen Verbesserungen von Helsgaun berücksichtigt werden, die in den Arbeiten

Keld Helsgaun: *General k-opt submoves for the Lin-Kernighan TSP heuristic*. Mathematical Programming Computation, October 2009, Volume 1, Issue 2–3, pp 119–163.

sowie

Helsgaun K.: *An effective implementation of the Lin-Kernighan traveling salesman heuristic*. EJOR 12, 106–130 (2000).

beschrieben werden.

4.3 Aufgabe 3: Der Christofides-Algorithmus.

Alexander Esgen

Betreuer: Markus Ahrens

Es soll Christofides Algorithmus implementiert werden, wie beschrieben in Kapitel 21 von: B. Korte und J. Vygen: Kombinatorische Optimierung, Springer, 3. Auflage, 2018.

Zur exakten Lösung des Minimum Weight Perfect Matching Problems darf eine externe Implementierung verwendet werden, z.B. *Blossom-V*:

`http://pub.ist.ac.at/~vnk/software/blossom5-v2.05.src.tar.gz`.

Zusätzlich soll auch ein approximativer Macthingalgorithmus aus

Mirjam Wattenhofer, Roger Wattenhofer: *Fast and Simple Algorithms for Weighted Perfect Matching*. Electronic Notes in Discrete Mathematics, Volume 17, 20 October 2004, Pages 285–291.

implementiert werden.

4.4 Aufgabe 4: Der EAX-Algorithmus.

Philipp Alexander Grzywaczyk

Betreuer: Jannik Silvanus

Nagata hat 2012 einen sehr erfolgreichen genetischen Algorithmus beschrieben, der auf sehr großen Instanzen teilweise bessere Ergebnisse erzielt als LKH-2. Eine verbesserte Version dieses Algorithmus wird in der Arbeit

Yuichi Nagata, Shigenobu Kobayashi, (2013) *A Powerful Genetic Algorithm Using Edge Assem-*

bly Crossover for the Traveling Salesman Problem. INFORMS Journal on Computing 25(2):346-363.

beschrieben und soll implementiert werden. Neben den TSPLIB-Instanzen sollten auch sehr große Testinstanzen von Bill Cooks TSP-Seite getestet werden.

4.5 Aufgabe 5: Das x-und-y-Achsen-TSP.

Anton Lorenzen

Betreuer: Siad Daboul

Für das euklidische TSP gibt es einen exakten $O(n^2)$ Algorithmus, falls alle Punkte auf den Koordinatenachsen liegen. Der Algorithmus stammt aus der folgenden Arbeit:

Eranda Çela, Vladimir Deineko, Gerhard J. Woeginger: *On the x-and-y-axes travelling salesman problem.* European Journal of Operational Research, 2012, Volume 223, pages 333–345.

Der in der Arbeit beschriebene Algorithmus soll implementiert werden. Zudem sollen geeignete Testinstanzen im TSPLIB-Format erzeugt werden und eine grafische Ausgabe der Ergebnisse erfolgen.

4.6 Aufgabe 6: Das k-Linien-TSP.

Paul Degenhardt

Betreuer: Siad Daboul

Für das euklidische TSP gibt es einen exakten $O(n^k)$ Algorithmus, falls alle Punkte auf k parallelen Geraden liegen. Diese Instanzen sind von Interesse, da das größte bekannte Integralitygap des Subtour-LPs auf diesen Instanzen angenommen wird. Der $O(n^k)$ Algorithmus stammt aus der folgenden Arbeit:

Günter Rote: *The N-line Traveling Salesman Problem.* Networks, Vol. 22 (1992) 91–108.

Implementiert werden soll ein Algorithmus, der den Fall $k = 3$ löst. Ein solcher wurde erstmals in

M. Cutler: *Efficient special case algorithms for the N-line planar Traveling Salesman Problem.* Networks, Vol. 10 (1980) 183–195.

beschrieben.

Der in der Arbeit beschriebene Algorithmus soll implementiert werden. Zudem sollen geeignete Testinstanzen im TSPLIB-Format erzeugt werden (z.B. die in der Arbeit

Stefan Hougardy: *On the integrality ratio of the subtour LP for Euclidean TSP.* Operations Research Letters 42 (2014) 495–499.

beschrieben Instanzen $G(n, \sqrt{n-1})$) und eine grafische Ausgabe der Ergebnisse erfolgen.

4.7 Aufgabe 7: Kanteneliminierung: Der geometrische Fall.

In der Arbeit

Stefan Hougardy, Rasmus T. Schroeder: *Edge Elimination in TSP Instances* WG 2014, LNCS 8747, pp. 275–286, 2014.

wir ein kombinatorischer Algorithmus beschrieben, der es erlaubt aus einer TSP-Instanz Kanten zu eliminieren, die in keiner optimalen Lösung vorkommen können. Der Algorithmus hat drei Phasen. Hier soll die erste Phase (Step 1, wie auf Seite 284 der Arbeit beschrieben) implementiert werden, die geometrische Argumente benutzt. Der Algorithmus soll nur auf `EUC_2D` Instanzen arbeiten.

Als Ausgabe soll eine Datei erzeugt werden, die alle nicht eliminierten Kanten enthält. Die Datei soll folgende Struktur haben:

```
n m
Knoten1 Knoten2
...
```

d.h. die erste Zeile der Datei enthält die Anzahl Knoten und Anzahl nicht eliminierter Kanten der Instanz. Danach folgt für jede nicht eliminierte Kante eine Zeile mit den beiden *um 1 reduzierten* (das ist das von Bill Cook verwendete Dateiformat) Knotennummern. Es muss getestet werden, dass die Kantenmenge eine optimale Tour enthält.

4.8 Aufgabe 8: Kanteneliminierung: Der kombinatorische Fall.

In der Arbeit

Stefan Hougardy, Rasmus T. Schroeder: *Edge Elimination in TSP Instances* WG 2014, LNCS 8747, pp. 275–286, 2014.

wir ein kombinatorischer Algorithmus beschrieben, der es erlaubt aus einer TSP-Instanz Kanten zu eliminieren, die in keiner optimalen Lösung vorkommen können. Der Algorithmus hat drei Phasen. Hier sollen die Phasen 2 und 3 (Step 2 und Step 3, wie auf Seite 284 der Arbeit beschrieben) implementiert werden, die einen Backtracking-Ansatz nutzen. Der Algorithmus soll nur auf `EUC_2D` Instanzen arbeiten.

Als Ausgabe soll eine Datei erzeugt werden, die alle nicht eliminierten Kanten enthält. Die Datei soll folgende Struktur haben:

```
n m
Knoten1 Knoten2
...
```

d.h. die erste Zeile der Datei enthält die Anzahl Knoten und Anzahl nicht eliminierter Kanten der Instanz. Danach folgt für jede nicht eliminierte Kante eine Zeile mit den beiden *um 1 reduzierten* (das ist das von Bill Cook verwendete Dateiformat) Knotennummern. Es muss getestet werden, dass die Kantenmenge eine optimale Tour enthält.

4.9 Aufgabe 9: Implementierung des Branch-&-Bound-Algorithmus unter Verwendung der Assignment-Relaxierung.

Es sollen Branch-&-Bound-Varianten implementiert werden, wie beschrieben in Abschnitt 2, "Relaxation I" aus: E. Balas, P. Toth: "Branch and bound methods", The Traveling Salesman Problem, editiert von E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan und D.B. Shmoys, John Wiley & Sons Ltd., 1985, 361–401.

Die dort vorgestellten Branching-Strategien sollen implementiert werden. Dies beinhaltet die Implementierung des Sukzessive-Kürzeste-Wege-Algorithmus zur Lösung des Assignment-Problems.

Die Strategie sollte über einen zusätzlichen Parameter auswählbar sein. Der Aufruf sollte wie folgt erfolgen:

```
tsp  {-s <Strategie-Index>}  <TSPLIB-Dateiname>
```

Wenn keine Strategie übergeben wird, sollte eine Default-Strategie gewählt werden, die sich empirisch als schnell erwiesen hat.

4.10 Aufgabe 10: Implementierung des Branch-&-Bound-Algorithmus unter Verwendung der 1-Baum-Relaxierung.

Martin Drees

Betreuer: Jannik Silvanus

Es sollen Branch-&-Bound-Varianten implementiert werden, wie beschrieben in Abschnitt 3, "Relaxation I" aus: E. Balas, P. Toth: "Branch and bound methods", The Traveling Salesman Problem, editiert von E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan und D.B. Shmoys, John Wiley & Sons Ltd., 1985, 361–401.

Die dort vorgestellten Branching-Strategien sollen implementiert werden. Dies beinhaltet die Implementierung des 1-Baum Algorithmus (insb. eines Minimum-Spanning-Tree-Algorithmus).

Die Strategie sollte über einen zusätzlichen Parameter auswählbar sein. Der Aufruf sollte wie folgt erfolgen:

```
tsp  {-s <Strategie-Index>}  <TSPLIB-Dateiname>
```

Wenn keine Strategie übergeben wird, sollte eine Default-Strategie gewählt werden, die sich empirisch als schnell erwiesen hat.

4.11 Aufgabe 11: Implementierung eines Branch-&-Cut Algorithmus für das TSP.

Jan Malte Schürks

Betreuer: Benjamin Rockel

Es soll ein Branch-&-Cut Algorithmus für das TSP basierend auf der LP-Relaxierung implementiert werden. Dabei sollen zunächst die Subtour-Elimination-Constraints (Korollar 21.26 in Korte, Vygen) und alle 2-Matching-Constraints (Proposition 21.27 in Korte, Vygen) separiert werden und anschließend mit Branch-&-Bound die optimale Lösung bestimmt werden.

Zum Lösen der LPs sollte ein LP-Solver (QSopt, Cplex oder Gurobi) benutzt werden. Ferner dürfen Implementierungen für das Minimum-Cut-Problem und Gomory-Hu-Bäume aus der Graph-Library **Lemon** (`http://lemon.cs.elte.hu/trac/lemon`) verwendet werden.