

VRP - Vehicle Routing Problem

Michael Bögl, 0055390

Projektpraktikum

Inhaltsverzeichnis

| | | |
|----------|---|----------|
| 1 | Einleitung | 3 |
| 2 | Implementierung | 3 |
| 2.1 | Penalty Operatoren | 3 |
| 2.1.1 | Overload Penalty | 3 |
| 2.1.2 | Tardiness Penalty | 3 |
| 2.1.3 | Routetime Penalty | 3 |
| 2.1.4 | Traveltimeexcess Penalty | 4 |
| 2.1.5 | Distance Penalty | 4 |
| 2.2 | Initialisierung | 4 |
| 2.3 | Push-Forward Insertion Heuristic nach [Tha99] | 4 |
| 2.4 | Random | 4 |
| 3 | Operatoren | 4 |
| 3.1 | Crossoveroperatoren | 4 |
| 3.1.1 | Sequence Based Crossover - SBX | 5 |
| 3.1.2 | Route Based Crossover - RBX | 5 |
| 3.1.3 | One Point Crossover - OPX | 5 |
| 3.1.4 | Simple Route Based Crossover - SRBX | 5 |
| 3.2 | Mutationsoperatoren | 5 |
| 3.2.1 | M1 - one level exchange | 5 |
| 3.2.2 | M2 - two level exchange | 5 |
| 3.2.3 | LSM | 6 |

1 Einleitung

Bei der Implementierung hielt ich mich an [PB96] (Mutations- und Crossoveroperatoren), [Tha95] (Penaltyoperatoren) und [Tha99] (Initialisierungsroutine).

Man kann sowohl Probleme mit (Solomon) und ohne (TSPLib) Zeitfenster lösen. Sind keine Zeitfenster gegeben, werden intern die *ReadyTime* auf 0 und *DueDate* auf *int.MaxValue* gesetzt.

Es stehen 2 verschiedene Importer zur Verfügung:

SolomonImporter: Für Dateien im Solomon Format.

TSPLibImporter: Für Dateien im TSPLib Format. Hier können allerdings nur Dateien mit einer x,y Positionsangabe (EUC_2D) importiert werden. Stehen mehrere Depots zur Auswahl, wird immer das erste gewählt.

In der Problemansicht kann, wie auch beim TSP, eine *best known solution* angegeben werden (Routenlänge und minimale Anzahl an Fahrzeugen). Auch die Art der Initialisierung und welche Penaltyoperatoren mit welchen Gewichten verwendet werden wird hier eingestellt.

Zudem gibt es die Option *only allow feasible solutions*. Diese Option verwirft ungültige Individuen nach dem Crossover. Das geschieht, indem die ungültige Route wieder durch ein zufälliges Elternteil ersetzt wird. Dies entspricht dem Verhalten von GENEROUS ([PB96]).

2 Implementierung

2.1 Penalty Operatoren

Um gültige Routen von ungültigen zu unterscheiden wurden Penalty Operatoren implementiert. Die Operatoren sind aus [Tha95]. Das Gewicht mit dem die Penalty Operatoren einfließen sind einstellbar. Im Original sind folgende Gewichte eingestellt:

- Overload Penalty: 50
- Tardiness Penalty: 25
- Routetime Penalty: 0,05
- Traveltimexcess Penalty: 50
- Distance Penalty: 0,5

2.1.1 Overload Penalty

Für jede Route wird das gesamte zu transportierende Gewicht errechnet. Übersteigt das Gewicht die Maximallast des Fahrzeugs wird das überschüssige Gewicht als Penalty gesehen und mit dem entsprechenden Gewicht multipliziert.

2.1.2 Tardiness Penalty

Für jeden Kunden in jeder Route wird bestimmt, ob das Fahrzeug innerhalb des Zeitfensters eintrifft. Die Fahrzeit ist die Distanz zwischen den einzelnen Kunden. Kommt das Fahrzeug nach dem Ende des Zeitfensters, wird die von der aktuellen Ankunftszeit die spätest erlaubte Ankunftszeit abgezogen und ergibt das Tardiness Penalty.

2.1.3 Routetime Penalty

Entspricht der kompletten Reisezeit der Fahrzeuge (Ankunftszeit + Servicezeit + Reisezeit/Distanz).

2.1.4 Traveltimeexcess Penalty

Bei jedem Kunden wird überprüft ob und wie weit die längste Routenzeit bereits überschritten ist. Diese Überschreitung wird zum Penalty gezählt.

2.1.5 Distance Penalty

Entspricht der Distanz zwischen 2 aufeinanderfolgenden Kunden.

2.2 Initialisierung

Es stehen 2 Initialisierungsroutinen implementiert:

- Push-Forward Insertion Heuristic
- Random

2.3 Push-Forward Insertion Heuristic nach [Tha99]

Die Kunden werden entsprechend der Formel: $C_i = -\alpha d_{0i} + \beta l_i + \gamma(p_i/360)d_{0i}$ sortiert.

C_i : Kosten des Kunden i .

d_{0i} : Distanz zwischen Depot und Kunde i .

l_i : Spätester Ankunftszeitpunkt bei Kunde i .

p_i : Polarkoordinatenwinkel von Kunde i .

Entsprechend dem Artikel ist $\alpha = 0,7$, $\beta = 0,1$ und $\gamma = 0,2$.

Es wird mit einer leeren Route begonnen und der erste Kunde in der Liste in diese Route eingefügt. Jeder noch nicht eingefügte Kunde wird zwischen jede Kante eingefügt und der Umweg errechnet. Nun wird der Kunde zwischen die Knoten eingefügt, für den der Umweg minimal ist und die Nebenbedingungen (Überladung, Zeitbedingungen) erfüllt sind. Ist kein Einfügen mehr möglich, dann wird eine neue Route eröffnet. Dies wird solange wiederholt, bis alle Kunden geroutet sind.

2.4 Random

In den Problemdateien ist die maximale Anzahl der zur Verfügung stehenden Transportfahrzeuge festgelegt. In den Dateien im Format TSPLib sind keine Angaben über die maximale Anzahl von Fahrzeugen vorhanden, deshalb wird die Anzahl der maximalen Fahrzeuge gleich der Anzahl der Städte gesetzt.

3 Operatoren

Bei der Implementierung der Operatoren hielt ich mich stark an [PB96]. Es beschreibt 2 Crossover- (SBX, RBX) und 3 Mutationsoperatoren (1M, 2M, LSM).

Zusätzlich zu den 2 Crossover Operatoren habe ich 2 eigene Operatoren implementiert (OPX, SRBX).

3.1 Crossoveroperatoren

Die Crossoveroperatoren erzeugen auch ungültige Routen. Nach einem Crossover wird eine Reparaturfunktion angewendet, die doppelt vorkommende und nicht vorkommende Kunden entfernt bzw. an geeigneter Position einfügt. Auch nach dieser Reparaturfunktion ist es möglich, dass die Route ungültig ist, da Nebenbedingungen (Zeitfenster, Kapazität der Fahrzeuge) verletzt sind. Ist die Option *only allow feasible routes* aktiviert, werden diese Routen verworfen und durch ein zufällig gewähltes Elternteil ersetzt.

3.1.1 Sequence Based Crossover - SBX

Es wird ein zufälliger Kunde in einer zufälligen Route in Elternteil 1 (*customer1* in *route1*) und Elternteil 2 (*customer2* in *route2*) ausgewählt. Alle Routen bis auf *route1* von Elternteil 1 werden in das Kind übernommen. Nun wird *route1* von Anfang bis *customer1* von Elternteil 1 und *route2* von *customer2* bis zum Ende der Route von Elternteil 2 in eine Route zusammengefasst und in das Kind kopiert.

Anschliessend wird die Reparaturfunktion ausgeführt. Diese Reparaturfunktion fügt nicht vorhandene Kunden an einer passenden Stelle ein und entfernt Kunden, wenn sie mehrfach besucht werden.

Ist die Option *only allow feasible routes* aktiviert wird am Ende der Funktion die Gültigkeit der Route überprüft und gegebenenfalls wieder verworfen.

3.1.2 Route Based Crossover - RBX

Es wird eine zufällige Route (*route1*) in Elternteil 1 und eine zufällige Route (*route2*) in Elternteil 2 gewählt. Alle Routen bis auf *route1* von Elternteil 1 werden in das Kind übernommen. Nun wird *route2* von Elternteil 2 in das Kind kopiert.

Anschliessend wird die Reparaturfunktion ausgeführt.

Ist die Option *only allow feasible routes* aktiviert wird am Ende der Funktion die Gültigkeit der Route überprüft und gegebenenfalls wieder verworfen.

3.1.3 One Point Crossover - OPX

Es wird ein zufälliger Kunde (*customer1*) in Elternteil 1 und ein zufälliger Kunde *customer2* in Elternteil 2 gewählt. Vom Beginn bis *customer1* werden alle Routen aus Elternteil 1 und von *customer2* bis zum Ende werden alle Routen aus Elternteil 2 in das Kind kopiert.

Anschliessend wird die Reparaturfunktion ausgeführt.

Ist die Option *only allow feasible routes* aktiviert wird am Ende der Funktion die Gültigkeit der Route überprüft und gegebenenfalls wieder verworfen.

3.1.4 Simple Route Based Crossover - SRBX

Funktioniert gleich dem RBX nur werden die gesamten Routen aus Elternteil 1 übernommen und der Reparaturfunktion überlassen, welche Kunden gelöscht werden und welche erhalten bleiben.

3.2 Mutationsoperatoren

Die Mutationsoperatoren wählen eine Route aus und versuchen die Kunden dieser Route in anderen Routen unterzubringen. M1, M2 und LSM bevorzugen kürzere Routen. RandomNeighbour-Exchange und RandomSwap sind nur zum Vergleich enthalten.

3.2.1 M1 - one level exchange

Wählt zufällig eine Route (*route1*) aus und versucht für jeden Kunden eine Position zu finden, bei dem der Umweg am geringsten ist und die Nebenbedingungen nicht verletzt werden. Ist eine solche Position gefunden, wird der Kunde dort eingefügt.

3.2.2 M2 - two level exchange

Wählt zufällig eine Route (*route1*) aus, versucht jeden Kunden (*customer1*) aus dieser Route anstelle eines anderen Kunden (*customer2*) in den restlichen Routen zu ersetzen und für *customer2* an einer passenden Stelle (*position1*) einzufügen. Ist dies möglich, wird *customer2* durch *customer1* ersetzt und *customer2* an *position1* eingefügt. Dabei wird immer darauf geachtet, dass keine Nebenbedingungen verletzt werden.

3.2.3 LSM

Entspricht einer lokalen Suche und hat nur Sinn, wenn die Lösung bereits gültig ist. Es wird eine Sequenz von Kunden (3, 2, 1) ausgewählt und versucht diese Sequenz an einer anderen Stelle einzufügen. Diese lokale Suche wird für alle Kunden in einem Individuum ausgeführt. LSM ist der zeitintensivste Operator.

Literatur

- [PB96] Jean-Yves Potvin and Samy Bengio. The Vehicle Routing Problem with Time Windows - Part II: Genetic Search. *INFORMS Journal of Computing*, 8:165–172, 1996.
- [Tha95] Sam R. Thangiah. Vehicle Routing with Time Windows using Genetic Algorithms. *Application Handbook of Genetic Algorithms: New Frontiers, Volume II*, pages 253–277, 1995.
- [Tha99] Sam R. Thangiah. A Hybrid Genetic Algorithms, Simulated Annealing and Tabu Search Heuristic for Vehicle Routing Problems with Time Windows. *Practical Handbook of Genetic Algorithms, Volume III: Complex Structures*, pages 347–381, 1999.