Partikelschwarmoptimierung für diskrete Probleme

Yushan Liu
Fakultät für Mathematik
TU München

26. Oktober 2014

Ferienakademie im Sarntal - Kurs 1
Moderne Suchmethoden der Informatik: Trends und Potenzial

1 Einführung

1.1 Klassische Partikelschwarmoptimierung (PSO)

Die klassische Partikelschwarmoptimierung wurde 1995 von Kennedy und Eberhart eingeführt. PSO ist ein oft in der Informatik benutztes metaheuristisches Verfahren, um das Minimum einer numerischen Funktion zu finden. Es ist ein naturanaloges Optimierungsverfahren, welches vom biologischen Schwarmverhalten inspiriert wurde.
Die Grundidee ist, dass der Schwarm, bestehend aus einer Population von Partikeln, so lange durch einen Suchraum bewegt wird, bis eine gute Lösung gefunden wird. Dabei stellt jedes Partikel eine potentielle Lösung dar. Die Position eines Partikels wird in jedem Zeitschritt neu berechnet. Es ist äußerst wichtig, dass die Partikel in einem Schwarm kooperieren, um die bestmögliche Lösung zu finden.

1.2 Eigenschaften eines Partikels

Jedes Partikel hat folgende Eigenschaften:

- es hat eine Position im Suchraum
- es bewegt sich und hat daher eine Geschwindigkeit
- es sucht das Optimum
- es kennt seine eigene Position und den Zielfunktionswert an dieser Position
- es erinnert sich an seine bisher beste Position (lokaler Attraktor)
- es kennt seine Nachbarn und deren Zielfunktionswerte
- es kennt die beste bisher erreichte Position aller Partikel (globaler Attraktor)

1.3 Nachbarschaften

Es gibt zwei verschiedene Arten von Nachbarschaften. In einer *physischen* Nachbarschaft werden die Nachbarn durch Abstände charakterisiert. Diese Abstände müssen allerdings in jedem Zeitschritt neu berechnet werden, sodass die Nutzung von physischen Nachbarschaften recht teuer ist.
In der *sozialen* Nachbarschaft dagegen hat jedes Partikel von Anfang an eine definierte Nachbarschaft, die sich im Laufe der Zeit nicht verändert.

2 Diskrete Partikelschwarmoptimierung (DPSO)

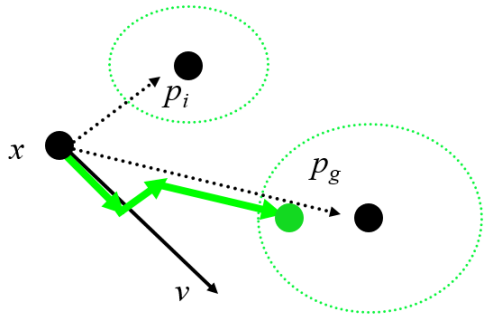
Für die diskrete Partikelschwarmoptimierung brauchen wir zuerst einen Suchraum S, der nur aus endlich vielen Zuständen besteht. Außerdem definieren wir eine Zielfunktion oder Fitnessfunktion $f\colon S \longrightarrow C$, wobei C eine endliche Menge ist. Es existiert eine Halbordnung auf C, d.h. es gilt entweder $c_i < c_j$ oder $c_i \geq c_j$ für alle $i, j \in C$. Nun können wir einen Algorithmus aufstellen, der in jedem Schritt die neue Position jedes Partikels berechnet.

$v_{t+1} = c_1 v_t + c_2(p_{i,t} - x_t) + c_3(p_{g,t} - x_t)$
$x_{t+1} = x_t + v_{t+1}$

Die Variablen haben hier folgende Bedeutung:
x_t = Position zum Zeitschritt t
$p_{i,t}$ = lokaler Attraktor zum Zeitschritt t
$p_{g,t}$ = globaler Attraktor zum Zeitschritt t
$c_{1,2,3}$ = soziale/kognitive Vertrauenskoeffizienten, $c_{1,2,3} \in \mathbb{R}$
$p_{i,t} - x_t$ = lokale Attraktion
$p_{g,t} - x_t$ = globale Attraktion

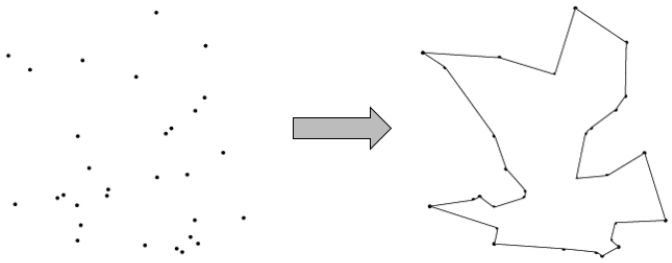
Die Vertrauenskoeffizienten geben an, wie sehr die Partikel sich selbst, ihren eigenen Erfahrungen und den Erfahrungen ihrer Nachbarn vertrauen.

Dieser Algorithmus wird nun in jedem Schritt angewandt und es erfolgt ein Kompromiss zwischen den drei möglichen Wegen: den Weg zum lokalen Attraktor, zum globalen Attraktor und dem eigenen Weg, der durch die Geschwindigkeit v vorgegeben ist:



3 Traveling Salesperson Problem (TSP)

Da wir die diskrete Partikelschwarmoptimierung auf das Traveling Salesperson Problem, ein klassisches Problem der diskreten Optimierung, anwenden wollen, möchte ich zuerst einmal das Problem vorstellen.
Wir haben eine gewisse Anzahl von Städten gegeben und die Aufgabe ist es, eine Rundreise zu finden, sodass jede Stadt genau einmal besucht wird. Natürlich müssen die Städte am Anfang und am Ende gleich sein.



Dieses Problem können wir durch einen Graphen $G = (V, E)$ modellieren. Die Knoten 1 bis N sind die Städte und die gewichteten Kanten stellen die Länge, Reisezeit etc. zwischen zwei Städten dar. Die Gewichtung kann aus einer Abstandsmatrix abgelesen werden.
Normalerweise ist der Graph vollständig, d.h. zwischen je zwei verschiedenen Knoten existiert auch eine Kante. Nun muss man einen Hamiltonkreis finden (also einen geschlossenen Weg, der jeden Knoten genau einmal enthält), sodass die Summe der gewichteten Kanten minimal ist.

4 Anwendung von DPSO auf TSP

4.1 Definitionen

Wir definieren eine **Position** als $x = (n_1, n_2, ..., n_N, n_{N+1})$. Jede Position stellt eine Rundreise als Permutation der N Städte dar. Dabei sollte man beachten, dass $n_1 = n_{N+1}$ gilt, damit der Weg auch wirklich geschlossen ist. Der Suchraum besteht also aus allen Permutationen der N Städte.

Als nächstes definieren wir die **Zielfunktion** f: $f(x) = \sum_{i=1}^{N} dist_{\pi(i),\pi(i+1)}$. $\pi(i)$ ist hier die i-te Stadt in der Tour. Die Zielfunktion berechnet also jeweils die Länge einer gegebenen Rundreise x. Falls der Graph nicht vollständig sein sollte, vervollständigen wir den Graphen, indem wir nicht-existente Kanten hinzufügen und diese mit einem l_{sup} gewichten derart, dass die gefundene Lösung diese Kante nicht enthalten kann. Eine geeignete Bedingung wäre zum Beispiel:
$l_{sup} > l_{max} + (N-1)(l_{max} - l_{min})$.

Als **Geschwindigkeit** v wählen wir eine Liste von Transpositionen, die eine Rundreise in eine andere überführt. (i_k, j_k) bedeutet, dass man die Städte i_k und j_k vertauscht. Somit haben wir:
$v = ((i_k, j_k)), k = 1, ..., \|v\|$.
Die Negation davon ist
$\neg v = ((i_k, j_k)), k = \|v\|, ..., 1$ und es gilt $\neg\neg v = v$.

Die **Länge** $\|v\|$ ist die Anzahl der Transpositionen.

4.2 Operatoren

4.2.1 Addition

Es gibt zwei Arten von Addition, die ich an Beispielen veranschaulichen werde.

1. Position + Geschwindigkeit = Position
   Seien $x = (1, 2, 3, 4, 5, 1)$ und $v = ((1, 2), (2, 3))$. $x + v = (3, 1, 2, 4, 5, 3)$.

Die Liste der Transpositionen wird auf die Position x angewandt, d.h. zuerst werden die Zahlen 1 und 2 vertauscht, danach die Zahlen 2 und 3, sodass sich eine neue Tour $(3, 1, 2, 4, 5, 3)$ ergibt.

2. Geschwindigkeit + Geschwindigkeit = Geschwindigkeit
   Seien $V_1 = v_1^1, ..., v_1^k$ und $V_2 = v_2^1, ..., v_2^k$. $V_1 + V_2 = v_1^1, ..., v_1^k, v_2^1, ..., v_2^k$.

Die Summe ist also die Hintereinanderausführung der Transpositionen.
Es gelten $V + \neg V = \emptyset$ und $\|V_1 + V_2\| \leq \|V_1\| + \|V_2\|$.

4.2.2 Subtraktion

Position - Position = Geschwindigkeit
Für $x_1 - x_2 = v$ findet man durch einen Algorithmus ein v, sodass $x_1 + v = x_2$ gilt. Dieses v ist eine kürzeste Folge von Transpositionen, die jedoch nicht eindeutig ist.
Falls $x_1 = x_2$, gilt $v = \emptyset$.

4.2.3 Multiplikation

Koeffizient $\cdot$ Geschwindigkeit = Geschwindigkeit. Seien $c \in \mathbb{R}$ und $v = v_1, ..., v_k$.

1. $c = 0$: $cv = \emptyset$
2. $c \in]0, 1]$: $cv = v_1, ..., v_{\lfloor ck \rfloor}$
3. $c > 1$: $c = k + c'$ mit $k \in \mathbb{N}, c' \in [0, 1[$. $cv = v + v + ... + v + c'v$
4. $c < 0$: $cv = (-c)\neg v$

4.3 Algorithmus

Wir nehmen jetzt die vorher aufgestellten Gleichungen

$v_{t+1} = c_1 v_t + c_2(p_{i,t} - x_t) + c_3(p_{g,t} - x_t)$

$x_{t+1} = x_t + v_{t+1}$

und vereinfachen diese, indem wir $c_2 = c_3$ setzen und einen Mittelpunkt $p_{ig,t}$ definieren:

$p_{ig,t} = p_{i,t} + \frac{1}{2}(p_{g,t} - p_{i,t})$.

Wir erhalten einen aktualisierten Algorithmus:

$v_{t+1} = c_1 v_t + c_2'(p_{ig,t} - x_t)$

$x_{t+1} = x_t + v_{t+1}$

Dies sind nun unsere endgültigen Gleichungen, die wir später auf ein Beispiel anwenden werden.

4.4 NoHope Tests

Manchmal ist es dem Algorithmus nicht möglich, den erwünschten oder erwarteten Wert zu erreichen . In diesem Fall ist der Schwarm in einem no-hope-Zustand. Es gibt dafür vier Kriterien.

Kriterium 0: Es gibt keine Bewegung im Schwarm und somit können die Positionen nicht verbessert werden. Falls ein Partikel den Abstand 1 zu einem anderen Partikel hat, gibt es zwei Möglichkeiten: Entweder das Partikel bewegt sich gar nicht oder es geht zu der exakt gleichen Position wie das andere Partikel (abhängig vom Vertrauenskoeffizienten).

Kriterium 1: Der Schwarm ist zu klein. Falls der Abstand zwischen zwei Partikeln zu klein ist, werden diese beide irgendwann identisch und es gibt nur noch einen reduzierten Schwarm.

Kriterium 2: Der Schwarm ist zu langsam und es gibt keine effektiven Bewegungen, also kommt der Schwarm dem Optimum nicht näher.

Kriterium 3: Es ist lange Zeit ohne Verbesserung vergangen. Auch in absehbarer Zeit gibt es keine Aussicht auf jegliche Verbesserung.

Ist der Schwarm in einem no-hope-Zustand, kann man das Ergebnis entweder akzeptieren oder einen ReHope-Prozess starten.

4.5 ReHope-Prozesse

ReHope-Prozesse prüfen, ob es noch Hoffnung auf eine bessere Lösung gibt. Als erster Schritt wird der Schwarm immer expandiert. Es gibt vier Arten von ReHope-Prozessen.

Lazy Descent Method (LDM): Jedes Partikel geht zurück zu seinem lokalen Attraktor und bewegt sich zufällig und langsam. Es stoppt, sobald es eine bessere Position gefunden hat oder wenn eine maximale Anzahl an Schritten erreicht ist.

Energetic Descent Method (EDM): Jedes Partikel geht zurück zu seinem lokalen Attraktor und bewegt sich langsam, solange es eine bessere Position findet in einer maximalen Anzahl von Schritten. Dieses Verfahren ist teurer als LDM und wird nur benutzt, wenn die Kombination aus Algorithmus und LDM nicht funktioniert.

Local Iterative Levelling (LIL): Es gibt im Allgemeinen unendlich viele Zielfunktionen mit demselben globalen Minimum. Hier wird folgende Funktion benutzt: $f_t(y) = \frac{f(y_{min})+f(x)}{2}$. Für ein Partikel findet man nun den besten Nachbarn und rechnet für diesen den Wert $f(y_{min})$ aus. Danach sucht man alle Nachbarn mit Abstand 1 und setzt diese in die neue Funktion $f_t(y)$ ein. Das Partikel bewegt sich dann zum besten Nachbarn.
Dieses Verfahren ist teurer als die vorherigen beiden Verfahren und wird deswegen nur benutzt, wenn EDM nicht funktioniert und man weiß, dass eine bessere Lösung existiert.

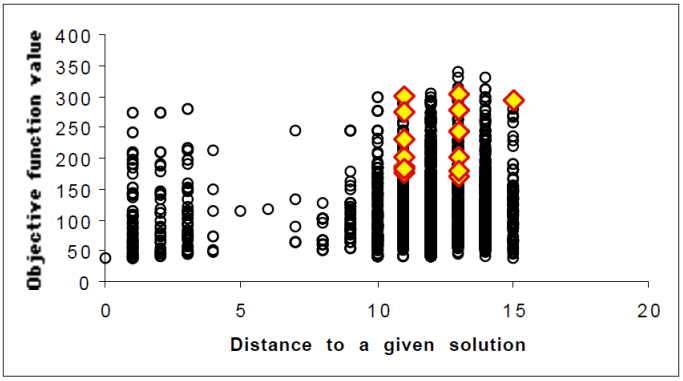
Adaptive ReHope Method (ARM): Diese Methode ist eine Kombination aus den drei anderen Verfahren. Je nach Anzahl der Schritte ohne Verbesserung wird entschieden, welches Verfahren man anwendet, um eine bessere Lösung zu finden.

4.6 Experiment

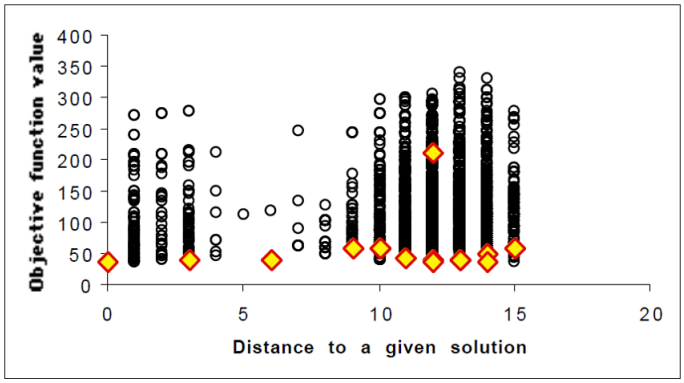
Für das Experiment setzen wir $c_1 \in]0,1[$ und $c_2 \in [0,2]$, die Schwarmgröße S sei $N-1$ und die Nachbarschaftsgröße sei 4. Die Nachbarschaft schließt das Partikel selbst mit ein.
Wir haben ein Problem mit 17 Städten, wobei die folgende Abstandsmatrix gegeben ist. Dafür wollen wir eine minimale Tour finden.

0	3	5	48	48	8	8	5	5	3	3	0	3	5	8	8	5
3	0	3	48	48	8	8	5	5	0	0	3	0	3	8	8	5
5	3	0	72	72	48	48	24	24	3	3	5	3	0	48	48	24
8	48	74	0	0	6	6	12	12	48	48	48	48	74	6	6	12
8	48	74	0	0	6	6	12	12	48	48	48	48	74	6	6	12
8	8	50	6	6	0	0	8	8	8	8	8	8	50	0	0	8
8	8	50	6	6	0	0	8	8	8	8	8	8	50	0	0	8
5	5	26	12	12	8	8	0	0	5	5	5	5	26	8	8	0
5	5	26	12	12	8	8	0	0	5	5	5	5	26	8	8	0
3	0	3	48	48	8	8	5	5	0	0	3	0	3	8	8	5
3	0	3	48	48	8	8	5	5	0	0	3	0	3	8	8	5
0	3	5	48	48	8	8	5	5	3	3	0	3	5	8	8	5
3	0	3	48	48	8	8	5	5	0	0	3	0	3	8	8	5
5	3	0	72	72	48	48	24	24	3	3	5	3	0	48	48	24
8	8	50	6	6	0	0	8	8	8	8	8	8	50	0	0	8
8	8	50	6	6	0	0	8	8	8	8	8	8	50	0	0	8
5	5	26	12	12	8	8	0	0	5	5	5	5	26	8	8	0

Das Minimum ist 39 und es gibt sogar mehrere Möglichkeiten für eine Rundreise mit der Länge 39. Wenn wir annehmen, dass wir die Lösung x_{lsg} kennen, können wir einen Graphen $d(x_{lsg}, f(x))$ plotten. Der x-Wert ist der Abstand zur gegebenen Lösung und der y-Wert der Zielfunktionswert, also die Länge dieser Tour.
Wir nehmen einen Schwarm bestehend aus 16 Partikeln.
Hier ist der Anfangszustand des Graphen:



Man kann sehen, dass es Äquivalenzklassen von Partikeln gibt, die denselben Abstand zur Lösung, aber unterschiedliche Zielfunktionswerte haben.
Nach einigen Iterationen erhalten wir folgenden Graph:



Der Schwarm bewegt sich in Richtung der Lösung. Manche Partikel bewegen sich auch zum Minimum der jeweiligen Äquivalenzklasse. Es ist also möglich, dass man mehrere Lösungen gleichzeitig findet, wenn der Schwarm groß genug ist.
In der Tabelle sind die Ergebnisse dieses Experiments aufgelistet:

Swarm size	Hood size	c_1	Random c_2	ReHope type	Best solution	Hood type	Tour evaluations	Arithmetical /logical operations
16	4	0.5	]0,2]	ARM	39 (3 solutions)	social	7990	4.8 M
16	4	0.5	]0,2]	ARM	39	physical	7742	6.3 M
8	4	0.5	]0,2]	ARM	39	social	4701	2.8 M
1	1	0.5	]0,2]	ARM	44	social	926 to ∞	0.6 M
128	4	0.999	]0,2]	no	47	social	41837 to ∞	33.2 M to ∞
32	4	0.999	]0,2]	no	66	social	14351 to ∞	10.8 M to ∞
16	4	0.999	]0,2]	no	86	social	2880 to ∞	1.9 M to ∞

Man kann nun folgende Schlussfolgerungen ziehen:
Bei Nutzung einer physischen Nachbarschaft braucht man weniger Tourevaluationen, dafür ist es teurer (mehr arithmetische Operationen), da man die Abstände in jedem Schritt neu berechnen muss.
ReHope-Prozesse zu benutzen ist keine Garantie dafür, dass man die optimale Lösung findet (siehe vierte Zeile).
Wenn man nur den Algorithmus ohne jegliche ReHope-Prozesse benutzt, reicht es meistens nicht, um den besten Wert zu finden.
Je größer der Schwarm, desto mehr Tourevaluationen und arithmetische Operationen braucht man, aber es ist auch wahrscheinlicher, das Minimum zu finden.

5 Kritik

Wie man sieht, funktioniert der Algorithmus und es ist möglich, damit das Minimum einer Funktion zu finden. Nach Experimenten haben jedoch die lokale und globale Attraktion zu einem späteren Zeitpunkt viele gemeinsame Transpositionen, die sich gegenseitig aufheben, was dazu führt, dass

die Partikel sich dem Minimum nicht nähern können, sondern der Abstand gleich bleibt. Die Konvergenzrate verringert sich also.
Diese Tatsache wird im Folgenden mit dem LTD Modell bewiesen.

5.1 Long Term Discrete PSO (LTD)

Wir differenzieren $c_{1,2,3}$ aus dem diskreten Algorithmus auf folgende Weise:
$v_{t+1} = a \cdot v_t + r_{loc} \cdot b_{loc} \cdot (p_{i,t} - x_t) + r_{glob} \cdot b_{glob} \cdot (p_{g,t} - x_t)$ mit
a = Trägheit
b_{loc}, b_{glob} = Beschleunigungskoeffizienten
r_{loc}, r_{glob} = zufällige Zahlen aus [0,1].
Für die Long Term Discrete PSO haben wir vier Bedingungen:

- $p := p_{i,t} = p_{g,t}$ für alle Partikel
- $a = 0$
- r_{loc} und r_{glob} sind gleichverteilt
- Geschwindigkeit zwischen Positionen ist eine Folge von Transpositionen

5.2 Satz

Seien $s \in [0, 1]$ und $b_{loc} = b_{glob} = b$.
Die Wahrscheinlichkeit, dass im LTD Modell ein bestimmtes Partikel seinen Abstand zu p in einer Iteration um einen Faktor von mindestens $b \cdot s$ reduziert, ist $(1 - s)^2$.

Beweis:
Mit den Voraussetzungen und $x_{t+1} = x_t + v_{t+1}$ erhält man die Gleichung
$x_{t+1} = x_t + r_{loc} \cdot b \cdot (p - x_t) + r_{glob} \cdot b \cdot (p - x_t)$.
Sei d die Anzahl der Transpositionen in $(p - x_t)$. Dann sind die ersten $min(r_{loc}, r_{glob}) \cdot b \cdot d$ Transpositionen gleich. Wenn man diese Transpositionen das erste Mal anwendet, bewegt sich das Partikel näher zur erwarteten Lösung. Wenn man diese allerdings das zweite Mal verwendet, bewegen sich Elemente, die schon an der richtigen Stelle waren, wieder an einen anderen Platz. Somit entfernt sich das Partikel wieder von der erwünschten Lösung.
Die Anzahl der effektiven Transpositionen ist also nur $\|r_{loc} - r_{glob}\| \cdot b \cdot d$, das sind nämlich diejenigen Transpositionen, die nur einmal angewandt werden. Nur die effektiven Transpositionen tragen zur Konvergenz bei.
Die Wahrscheinlichkeit, dass der Anteil der effektiven Transpositionen größer als $b \cdot s$ ist, ist:
$(P(\frac{\|r_{loc} - r_{glob}\| \cdot b \cdot d}{d}) \geq b \cdot s) = (P(\|r_{loc} - r_{glob}\|) \geq s)$.
Da r_{loc} und r_{glob} gleichverteilt sind, erhalten wir:
$(P(\|r_{loc} - r_{glob}\|) \geq s) = (1 - s)^2$.

□

6 Verbesserung

Da die Intepretation von Geschwindigkeit als eine Folge als Transpositionen die Konvergenzrate verringert, wollen wir sowohl die Geschwindigkeit als auch die Addition neu definieren, um diesen Verlangsamungseffekt zu verhindern.

6.1 Addition

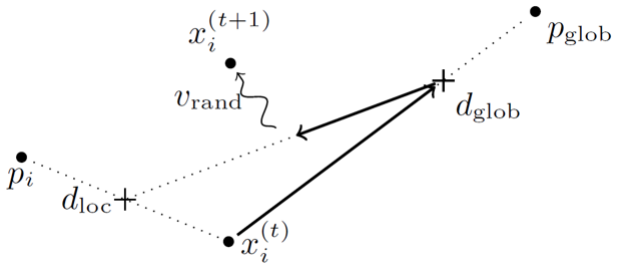
Statt einer Hintereinanderausführung von Transpositionen führen wir eine schwerpunktsbasierte Bewegung ein. Wir setzen

$d_{loc} = x_t + r_{loc} \cdot b_{loc} \cdot (p_{i,t} - x_t)$ und

$d_{glob} = x_t + r_{glob} \cdot b_{glob} \cdot (p_{g,t} - x_t)$ und definieren

$x_{t+1} = d_{glob} + \frac{1}{2} \cdot (d_{loc} - d_{glob}) + v_{rand}$ mit $v_{rand} = r_{rand} \cdot b_{rand} \cdot (p_{rand} - x_t)$.

Die zufällige Bewegung am Ende verhindert ein zu schnelles Konvergieren.



6.2 Geschwindigkeit

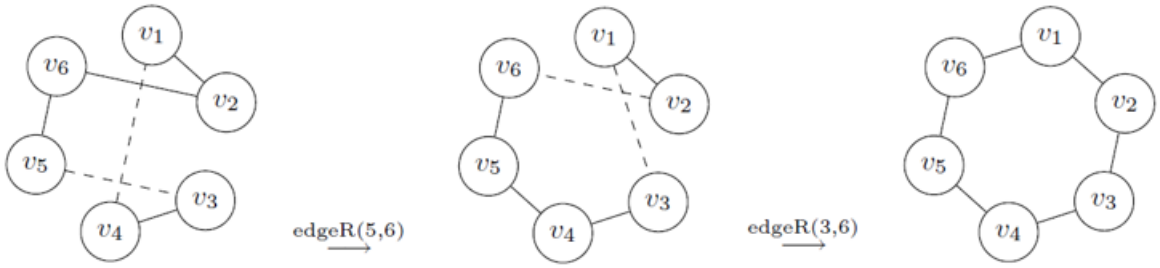
Bisher haben wir die Geschwindigkeit als eine Liste von Transpositionen dargestellt. Nun wollen wir statt der Städte die Kanten vertauschen, sodass eine kürzere Tour entsteht.

Dafür führen wir einen neuen Operator ein:

$l \circ edgeR(i,j) = (c_1 ... c_{i-1} c_j c_{j-2} ... c_{i+2} c_{i+1} c_i ... c_n)$, wobei l eine Tour $(c_1 ... c_n)$ ist.

Dieser Operator invertiert also die Folge der Städte zwischen den Indizes i und j und vertauscht die dafür benötigten Kanten.

Man kann sich dies am folgenden Beispiel veranschaulichen:



(a) (1 2 6 5 3 4) (b) (1 2 6 5 4 3) (c) (1 2 3 4 5 6)

6.3 Experimentelle Ergebnisse

Dieses neue Modell wurde für mehrere Probleme getestet. Die Ergebnisse sind überzeugend, dass man mit der neuen Definition von Addition und Geschwindigkeit einen viel besseren Wert erzielt. In den Klammern stehen die jeweiligen Minima.

Die Zeilen bedeuten von oben nach unten gelesen:

- relativer Fehler
- maximaler Wert
- Durchschnittswert

- Standardabweichung
- bestes Ergebnis

Move type	Composition	Centroid	Centroid	Centroid
Distance Repr.	Transposition	Adj. Transposition	Transposition	edgeR
Problem				
berlin52 (7542)	**104.6%**	**194.6%**	**70.5%**	**22.5%**
	18780	23193	15103	10233
	15431.9 ±1046.8	22216.8 ±531.6	12862.3 ±863.0	9240.4 ±410.8
	12588	20755	10977	8259
pr76 (108159)	**220.9%**	**317.7%**	**156.5%**	**88.9%**
	416630	468479	347091	249993
	347107.5 ±18966.6	451771.6 ±6919.1	277404.3 ±21902.2	204322.5 ±19535.6
	304195	431144	228652	158676
gr96 (55209)	**310.3%**	**430.4%**	**220.8%**	**128.5%**
	251167	302477	204837	149542
	226531.6 ±9631.1	292815.8 ±4398.8	177107.8 ±14565.0	126126.1 ±8618.0
	203617	278382	138255	107835

Move type	Composition	Centroid	Centroid	Centroid
Distance Repr.	Transposition	Adj. Transposition	Transposition	edgeR
Problem				
kroA100 (21282)	**377.2%**	**529.2%**	**238.0%**	**111.2%**
	115730	137761	83682	55383
	101552.2 ±4468.6	133913.6 ±2291.3	71933.4 ±4702.6	44945.0 ±3498.7
	91137	125127	60524	37289
kroC100 (20749)	**386.7%**	**537.4%**	**256.2%**	**133.9%**
	112320	135960	84903	58291
	100988.8 ±4189.6	132261.1 ±2107.8	73903.8 ±5234.0	48538.7 ±3093.7
	90375	125228	63107	41237
kroD100 (21294)	**364.2%**	**503.1%**	**239.0%**	**127.7%**
	106556	131549	83922	58077
	98847.9 ±4650.5	128438.0 ±1902.7	72194.1 ±4865.6	48487.5 ±3227.4
	90347	121195	62665	41828

7 Literatur

- M. Clerc: *Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem*, 2000.
- M. Hoffmann, M. Mühlenthaler, S. Helwig, R. Wanka: *Discrete Particle Swarm Optimization for TSP: Theoretical Results and Experimental Evaluations.* in: *Proc. International Conference on Adaptive and Intelligent Systems (ICAIS)*, pp. 416-427, 2011.