DIFFERENTIAL EVOLUTION FOR SMALL TSPs WITH CONSTRAINTS

Štefan Peško

Summary: This paper presents evolution algorithm for solving small (up to 32 nodes) traveling salesman problems with constraints. This new differential evolution algorithm with only two parameters - the size of the population and the size of the generations use the Lehmer code of the permutation for the representation of populations. Experience with small instances of TSP with time windows and deadline TSP are discussed.

Key words: evolution algorithm, traveling salesman problem, time constraints.

1 Introduction

The Travelling Salesman Problem (TSP) is the search for the shortest tour that visits a given set of nodes exactly once. The TSP is one of the oldest optimization problem since TSPs are frequent components of optimization problems.

As noted in [1] the existence of exchange neighborhood structures of TSP provides very good heuristics, the existence of various relaxations of the TSP provides very precise lower bounds and structure of TSP polytope can be exploited with powerful branch and cut method solving problems as large as 6000 nodes.

Why are then small TSPs interesting? TSPs are often modeled in the real vehicle routing problems (VRP), which could be seen as multiple TSP when one has to find a tour for a vehicle such that all nodes are visited by one tour. However even though the size of a VRP may be very large, the number of visits that a vehicle can make by one trip is bounded by capacity of vehicle or legal constrains to relative small number. In the case of an insertion heuristics using an exact algorithm for the small TSP instead of a local optimization heuristics translated into a global saves of 1%. On the other hand, one running of the VRP for 1000 nodes...
may translate into 10000 running of the subproblems and thus to solve small TSPs efficiently is important.

The key requirement for resolution of small TSPs is flexibility -- the ability to solve problems with side constraints. One can add time windows to TSP nodes that indicate a interval in which the node must be visited. Other instances include changing travel time between one or many time windows. Interesting case is probabilistic TSP [2] where nodes are visited with given probability.

## 2 TSP with constraints

The TSP with constraints can be stated in simple terms: Given an real \((n+1) \times (n+1)\) distance matrix \(D=(d_{ij})\), find a permutation \(\pi\) of the set \(N=\{1,2,...,n\}\) that minimise the function

\[
c(\pi) = d_{0,\pi(1)} + \sum_{k=1}^{n-1} d_{\pi(k),\pi(k+1)} + d_{\pi(n),0} + K \cdot m(\pi),
\]

(1)

where \(m(\pi)\) is measure of infeasibility of permutation \(\pi\) considering the same type of the side constraints and \(K\) is positive penalty constant. The size of the TSP with constraints we define equal \(n+1\). The travelling salesman tour (TSP tour) is here in the form of the cycle \(0\rightarrow\pi(1)\rightarrow\pi(2)\rightarrow...\rightarrow\pi(n)\rightarrow0\) where \(\pi(k)\) is \(k^{th}\) visited node in this tour from the start node \(0\). Note that for the feasible permutation \(\pi\) we define \(m(\pi)\)=0.

**Example 1.:** For the instance of the TSP with time windows (TSPTW) intervals \([\tau_i^R,\tau_i^D]\) are given for nodes \(i\) from \(N\) where \(\tau_i^R\) is release time and \(\tau_i^D\) is deadlines time. The distance \(d_{ij}\) represents sum of the minimum time of the trip from node \(i\) to node \(j\) and the processing time in node \(j\). The TSPTW tour is feasible (in the simple case) if each arrival \(t_i\) is in interval \([\tau_i^R,\tau_i^D]\). Formally the TSPTW tour is feasible if we find a solution \(t_i\) of following conditions:

\[
\tau_i^R \leq t_i \leq \tau_i^D \quad \text{for } i \in N,
\]

\[
t_0 + d_{o,\pi(1)} \leq t_{\pi(1)},
\]

(2)

\[
t_{\pi(i-1)} + d_{\pi(i-1),\pi(i)} \leq t_{\pi(i)} \quad \text{for } i \in N - \{1\}.
\]

Then we can define the measure of infeasibility \(m(\pi)\) of the TSPTW tour as a minimum total time out of constraints
\[ m(\pi) = \min_{t_1, t_2, \ldots, t_n \geq 0} \sum_{i \in N} (A_i + B_i) \text{ where } \pi(0) = 0 \text{ and } \]
\[ A_i = \max\{0, t_i - \tau_i^D, \tau_i^R - t_i\}, \quad B_i = \max\{0, t_{\pi(i-1)} + d_{\pi(i-1),\pi(i)} - t_{\pi(i)}\}. \]  

**Example 2.** For the instance of the *deadline TSP* (DTSP) only deadlines \(\tau_i^D\) for each visited node \(i\) from \(N\) and the start time \(\tau_0\) for the start node 0 are given. The distance \(d_{ij}\) is interpreted as for the TSPTW. The DTSP tour is feasible if each arrival \(t_i\) to node \(i\) is possible before their deadline \(\tau_i^D\). The DTSP is special case of the TSPTW when \(\tau_i^R = \tau_0\).

3 Lehmer code

The set of all permutation of the set \(N\) we note \(S_n\). In this paper we offer an algorithm where a permutation \(\pi = < \pi(1), \pi(2), \ldots, \pi(n) >\) is represented via its Lehmer code \(L(\pi)\).

General works in discrete mathematics and theoretical computer science deal with effective ways to represent permutation. A pioneer of this matter is Lehmer [3] who associates \(S_n\) and \(L_n\), where \(L_n\) is subset of \{0,1,\ldots,n-1\}^n. There are several ways to establish this one-to-one correspondence. The most classical of them is following: *The Lehmer code of the permutation \(\pi\) is a sequence of the numbers*

\[ L(\pi) = (l_1(\pi), l_2(\pi), \ldots l_n(\pi)), \quad l_i(\pi) = \left| \{ j > i : \pi(j) < \pi(i) \} \right|, \]  

where \(l_i(\pi)\) is the number of entries to the right of \(\pi(i)\), which are smaller. It is not difficult to see how \(\pi\) can be reconstructed from the code \(L(\pi)\):

\[ \pi(k) = N_k [l_k(\pi) + 1], \quad \text{where } N_k = N - \{\pi(1), \pi(2), \ldots, \pi(k-1)\}, \]

with respect to the natural order of the sets \(N_k\).

**Example 3.** From definition (5) we have \(L(<4,6,2,5,3,1,8,7>) = (3,4,1,2,0,1,0)\). We show that \(L^{-1}((3,4,1,2,1,0,1,0)) = (<4,6,2,5,3,1,8,7>)\).

\[ \pi(1)=N_1[3+1]=4, \quad N_1=\{1,2,3,4,5,6,7,8\}, \quad \pi(2)=N_2[4+1]=6, \quad N_2=\{1,2,3,5,6,7,8\}, \]
\[ \pi(3)=N_3[1+1]=2, \quad N_3=\{1,2,3,5,7,8\}, \quad \pi(4)=N_4[2+1]=5, \quad N_4=\{1,3,5,7,8\}, \]
The Lehmer code establishes a bijection between $S_n$ and the set of sequences $L_n$. Now we can define operations $\pm$ with permutations via the Lehmer code. Let $\pi$ and $\psi$ are in $S_n$ with the Lehmer codes $L(\pi)$ and $L(\psi)$. Then we define - for our algorithm – permutations

\[
\pi + \psi = L^{-1}\left([l_1(\pi) + l_1(\psi)] \mod n, \ldots, [l_k(\pi) + l_k(\psi)] \mod (n - k + 1), \ldots, 0\right),
\]
\[
\pi - \psi = L^{-1}\left([l_1(\pi) - l_1(\psi)] \mod n, \ldots, [l_k(\pi) - l_k(\psi)] \mod (n - k + 1), \ldots, 0\right).
\]

4 Differential evolution

Differential evolution (DE) [4] is a new heuristical approach for minimising real-valued multimodal objective functions. In DE algorithm is a population based on algorithm like genetic algorithm using the same operators; crossover, mutation and selection. The main difference in constructing better solution is that genetic algorithm rely on crossover while DE relays on mutation operation. In our DE algorithm no crossover is used.

We now describe our version of the DE algorithm for the TSPs with constraints. The goal is to find $\pi^* = \arg\min \{c(\pi): \pi \text{ in } S_n\}$. The DE works with two populations $P$ and $M$ of size $M$. For an old population $P = \{\pi_1, \pi_2, \ldots, \pi_M\}$ a mutant population $M = \{\mu_1, \mu_2, \ldots, \mu_M\}$ is generated by adding the difference between two elements of $P$ and third element of $P$ according to the rule

\[
\mu_t = \arg\min \{c(\pi_x + (\pi_y - \pi_z)), c(\pi_x + (\pi_y - \pi_e)), c(\pi_e)\},
\]

where $x, y, z$ are integers taken at random from the set $\{1, 2, \ldots, M\}$, mutually different and different from running index $t$. The random permutation $\pi_e$ has Lehmer code $L(\pi_e) = (\text{rand}(n), \text{rand}(n-1), \ldots, \text{rand}(2), 0)$. We assume a uniform probability distribution function $\text{rand}(k)$ for numbers from the set $\{0, 1, \ldots, k-1\}$.

The new population is created by replacing some element of the old $P$ to better element of the mutant population. More formally the DE algorithm is written on the figure 1 as a function DEtsp().
Figure 1: DE algorithm

```
function DEtsp(D, size_pop, size_gen)
(* Initialisation *)
    n = size(D)
    for t = 1 to size_pop do
        L(\pi_t) = (rand(n), rand(n-1), ..., rand(2), 0)
    (* Evolution of generations *)
    for noGen = 1 to size_gen do begin
        generate the mutant population M
        for t = 1 to size_pop do
            if c(\mu_t) < c(\pi_t) then
                \pi_t = \mu_t
        end
    (* Selection of best solution *)
    \pi_{DE} = \pi_1
    for t = 2 to size_pop do
        if c(\pi_t) < c(\pi_{DE}) then
            \pi_{DE} = \pi_t
    return \pi_{DE}
```

Function DEtsp() has two fixed parameters size of the population (size_pop) and the size of the generations (size_gen) and return the best solution of the last generation \(\pi_{DE}\). The initial population is chosen randomly. The Lehmer codes are used in calculation of permutations of mutant population defined by relation (6).

5 Experimental results

Presented DE algorithm was implemented in Python 2.3+. The corresponding program was tested on the small instances of DTSP and TSPTW from selected instances for asymmetric TSPTW [5]. The symmetric instances of the problems (with symmetric distance matrix \(D'\)) are generated by a simple rule from the distance matrix \(D\) of asymmetric instances via rule \(d_{ij}' = \min(d_{ij}, d_{ji})\).

The results of experiment with the size of the population size_pop =2, \(n\), size of the generations size_gen =1000, \(n\) and penalty constant \(K=n\cdot\max\{c_{ij}\}\) are presented for the asymmetric and the symmetric instances of the DTSP and and for the TSPTW in Table 1.
Table 1.: Run time of the DE algorithm (*optimal solution)

<table>
<thead>
<tr>
<th>Instances</th>
<th>Size of TSP</th>
<th>DTSP symmetric</th>
<th>DTSP asymmetric</th>
<th>TSPTW symmetric</th>
<th>TSPTW asymmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbg10a</td>
<td>11</td>
<td>54s</td>
<td>43s</td>
<td>123s</td>
<td>107s*</td>
</tr>
<tr>
<td>rbg017</td>
<td>18</td>
<td>96s</td>
<td>87s</td>
<td>102s</td>
<td>203s</td>
</tr>
<tr>
<td>rbg021</td>
<td>22</td>
<td>128s</td>
<td>111s</td>
<td>256s</td>
<td>245s*</td>
</tr>
<tr>
<td>rbg27</td>
<td>28</td>
<td>468s</td>
<td>438s</td>
<td>558s</td>
<td>561s*</td>
</tr>
<tr>
<td>rbg031a</td>
<td>32</td>
<td>602s</td>
<td>538s</td>
<td>668s</td>
<td>628s*</td>
</tr>
</tbody>
</table>

The first result is that DE solve a small TSP with constraints can be very robust and efficient manner.

This paper is supported by the grant VEGA 1/0490/03

Reference literature