VNS for the TSP and its variants

Nenad Mladenović, Dragan Urošević

BALCOR 2011, Thessaloniki, Greece

September 23, 2011

Problem formulation

- Given a set of locations V with travel distances between them.
- Locations are numbered with numbers from 1 to $n = |V|$.
- Location with label 1 is *depot*.
- All other locations are identified with customers which can be divided into two groups
  - *Pickup customers* or producers
  - Delivery customers or consumers.
- It is known a quantity of commodity produced/requested by each of customers;
- A vehicle with given capacity starts and finish at depot and must visit each customers exactly once;
- 1-PDTSP consists of finding a minimum length tour for the vehicle which satisfies all customers.

Previous work

- Problem proposed by Hernández–Pérez and Salazar–Gonzáles
- There are a number of proposed methods for solving 1-PDTSP
  - Hernández–Pérez, Rodríguez–Martín and Salazar–Gonzáles proposed method based on GRASP and VND.
  - Zhao et al. proposed method method based on Genetic algorithm
  - Hernández–Pérez and Salazar–Gonzáles proposed exact method based on Branch and Bound able to solve instances with up to $n = 60$ locations.

Notation

- q_i denotes quantity of commodity produced/demaned by customer at location i;
- If $q_i > 0$ then customer i is pickup customer, otherwise it is delivery customer;
- Depot can be considered as customer with demand

$$q_1 = -\sum_{k=2}^{n} q_k$$

- Hamiltonian tours can be divided into *feasible* and *non-feasible*
- Let $x = x_1, x_2, ..., x_n$ is Hamiltonian tour ($x_1 = 1$);
- We define load of vehicle after visiting customer x_i in the following way

$$L_1(x) = q_{x_1}, \quad L_i(x) = L_{i-1} + q_{x_i}$$

- Tour x is feasible if and only if

$$\max_{i \in \{1,2,...,n\}} L_i - \min_{i \in \{1,2,...,n\}} L_i \leqslant Q$$

- Tour x is infeasible if and only if

$$\max_{i \in \{1,2,...,n\}} L_i - \min_{i \in \{1,2,...,n\}} L_i > Q$$

- Value

$$\max_{i \in \{1,2,...,n\}} L_i - \min_{i \in \{1,2,...,n\}} L_i - Q,$$

we call the measure of infeasibility.

GVNS

GVNS$(x, \ell_{max}, k_{max}, t_{max})$

1 **repeat**
2 $k \leftarrow 1$
3 **repeat**
4 $x' \leftarrow Shake(x, k)$
5 $x'' \leftarrow VND(x', \ell_{max})$
6 NeighborhoodChange(x, x'', k, k_{max})
7 **until** $k > k_{max}$
8 $t \leftarrow CPUTime()$
9 **until** $t > t_{max}$

Sequential VND

- The final solution of `Seq-VND` should be a local minimum with respect to all ℓ_{max} neighborhoods.
- The chances to reach a global minimum are larger than with a single neighborhood structure.
- The total size of `Seq-VND` is equal to the union of all neighborhoods used.
- If neighborhoods are disjoint (no common element in any two) then the following holds

$$|\mathcal{N}_{\texttt{Seq-VND}}(x)| = \sum_{\ell=1}^{\ell_{max}} |\mathcal{N}_\ell(x)|,\ x \in X.$$

Nested VND

- Assume that we define two neighborhood structures ($\ell_{max} = 2$). In the nested VND we in fact perform local search with respect to the first neighborhood in any point of the second.
- The cardinality of neighborhood obtained with the nested VND is product of cardinalities of neighborhoods included, i.e.,

$$|\mathcal{N}_{\texttt{Nest-VND}}(x)| = \prod_{\ell=1}^{\ell_{max}} |\mathcal{N}_\ell(x)|,\ x \in X.$$

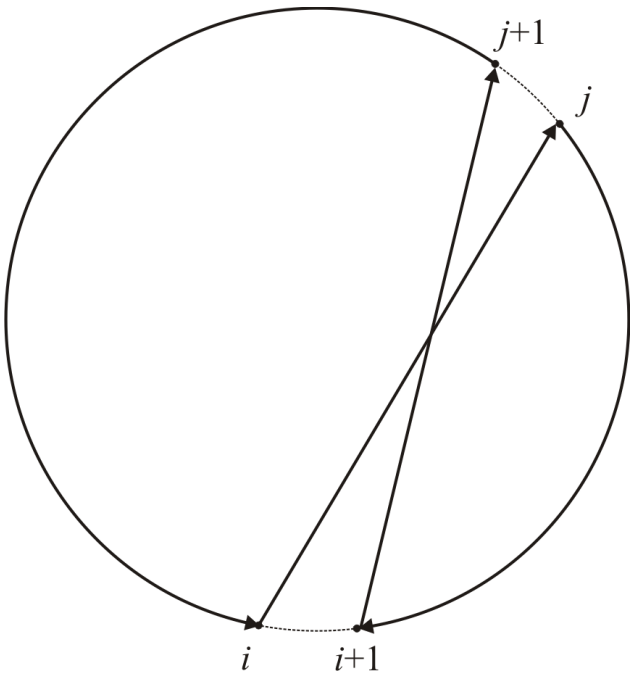
- The pure `Nest-VND` neighborhood is much larger than the sequential one.
- The number of local minima w.r.t. `Nest-VND` will be much smaller than the number of local minima w.r.t. `Seq-VND`.

Mixed nested VND

- After exploring b (a parameter) neighborhoods, we switch from a nested to a sequential strategy.
- We can interrupt nesting at some level b ($1 \leq b \leq \ell_{max}$) and continue with the list of the remaining neighborhoods in sequential manner.
- If $b = 1$, we get `Seq-VND`. If $b = \ell_{max}$ we get `Nest-VND`.
- Since nested VND intensifies the search in a deterministic way, **boost parameter** b may be seen as a balance between intensification and diversification in deterministic local search with several neighborhoods.
- Its cardinality is clearly

$$|\mathcal{N}_{\texttt{Mix-VND}}(x)| = \sum_{\ell=b}^{\ell_{max}} |\mathcal{N}_\ell(x)| + \prod_{\ell=1}^{b-1} |\mathcal{N}_\ell(x)|, \; x \in X.$$

2-opt



- Change of tour length is

$$df = d(x_i, x_j) + d(x_{i+1}, x_{j+1}) - d(x_i, x_{i+1}) - d(x_j, x_{j+$$

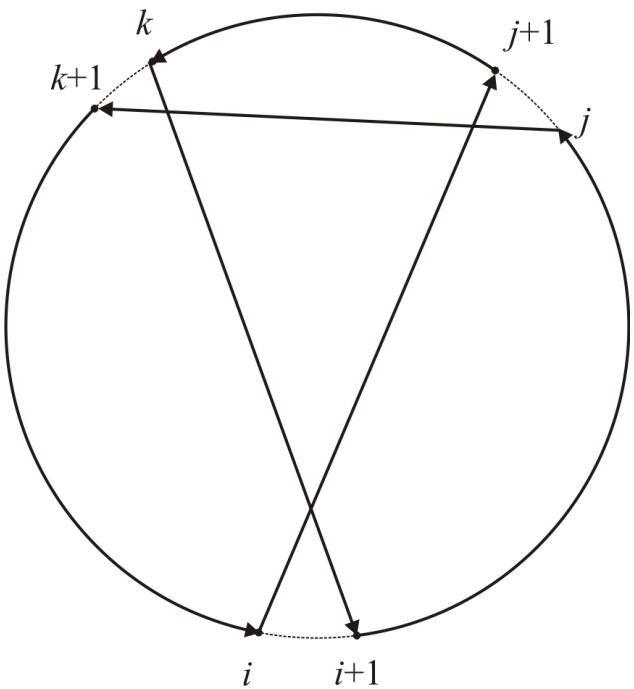
- Set of candidate pairs can be reduced to pairs (i, j) satisfying

$$d(x_i, x_j) < d(x_i, x_{i+1}) \text{ or } d(x_{i+1}, x_{j+1}) < d(x_i, x_{i+1})$$

or

$$d(x_i, x_j) < d(x_j, x_{j+1}) \text{ or } d(x_{i+1}, x_{j+1}) < d(x_j, x_{j+1})$$

3-opt



- Checking of new solution

$$df = d(x_i, x_{j+1}) + d(x_k, x_{i+1}) + d(x_j, x_{k+1}) - (d(x_i, x_{i+1}) + d(x_j, x_{j+1}) + d(x_k, x_{k+1}))$$

- We also reduce the set of candidate moves

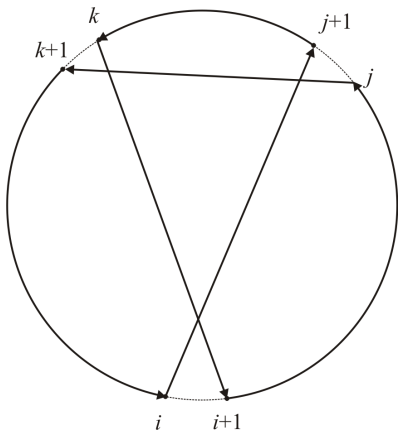
$$d(x_i, x_{j+1}) < d(x_i, x_{i+1})$$

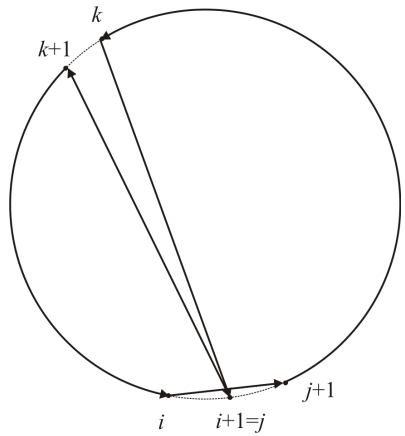
and

$$d(x_i, x_{j+1}) + d(x_j, x_{k+1}) < d(x_i, x_{i+1}) + d(x_j, x_{j+1})$$

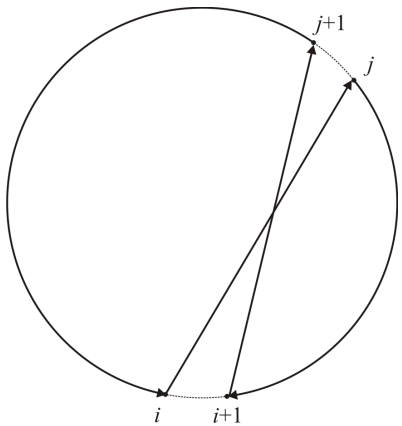
Special cases

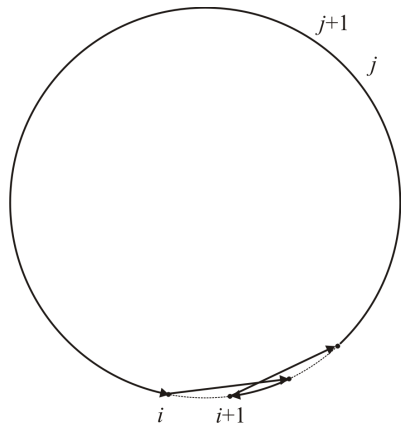
- If index j in 3-opt is set to $i + 1$, we get 2.5-opt:





- If index j in 2-opt is set to $i + 2$, we get 1-opt:





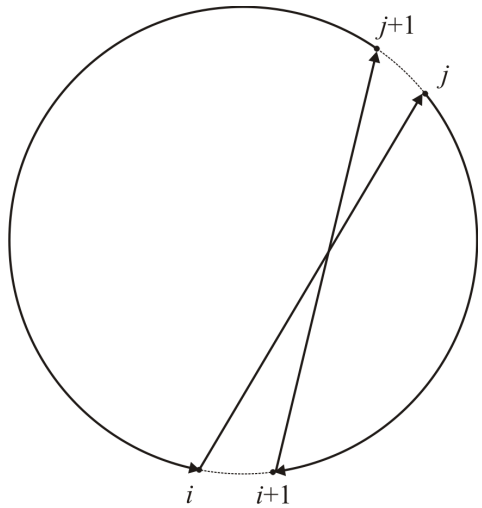
1-PDTSP

- In 1-PDTSP, beside the length of the tour, we should check its feasibility;
- For example, in the solution x' that belongs to 2–opt neighborhood of x, the links $(i, i+1)$ and $(j, j+1)$ are deleted and capacity of the vehicle is changed following the reverse order (from j to $i+1$);
- We need to calculate capacities after each visit and then find their minimum and maximum.

2-opt Feasibility checking

- Let us denote with L loads for initial tour and with L' loads after 2-opt move
- New load after visiting location x_k (any location between x_{i+1} and x_j) is

$$\begin{aligned} L'(x_k) &= L_i + q_{x_j} + q_{x_{j-1}} + \cdots + q_{x_k} \\ &= q_{x_1} + \cdots + q_{x_j} - (q_{x_1} + \cdots + q_{x_{k-1}}) + \\ &\quad (q_{x_1} + \cdots + q_{x_i}) = L_j + L_i - L_{k-1} \end{aligned}$$



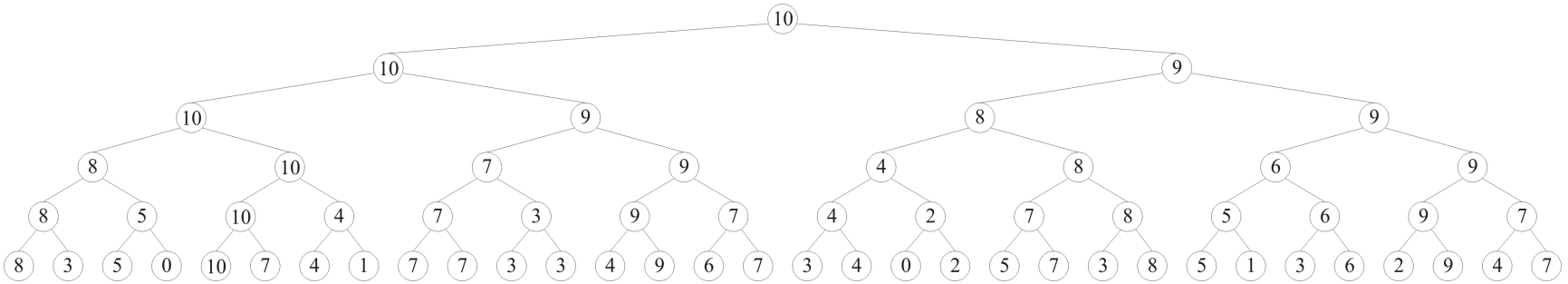
- For all other locations new load is same as previous load
  - Because of that we have:

$$\max\{L'_1, L'_2, ..., L'_n\} = \max\{\max\{L_1, ..., L_i\}, \max\{L_{j+1}, ..., L_n\}, L_j + L_i - \min\{L_i, ..., L_{j-1}\}\}$$

Application of advanced data structure

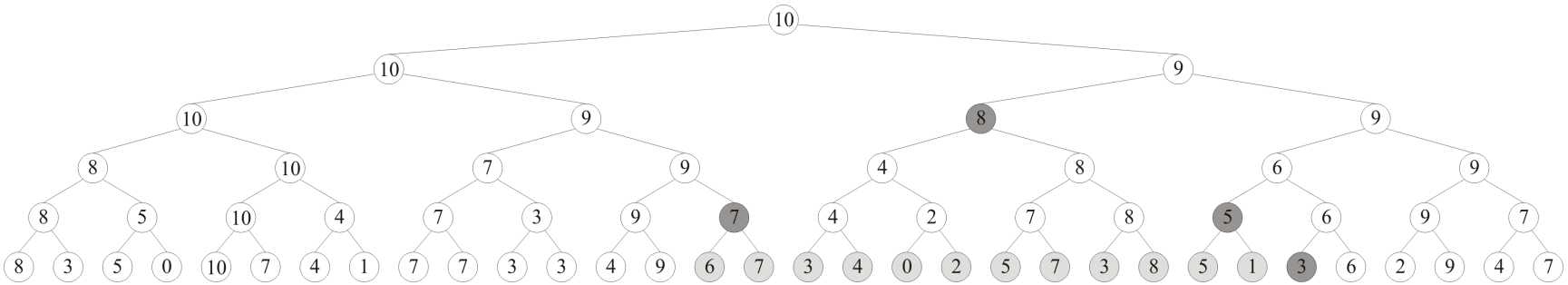
- In order to speedup calculating minimum and/or maximum, we use structure *Binary Indexed Tree, BIT*;
- This is structure providing efficiently computing minimum (maximum) of subsequence of any sequence whose elements may be changed during computation;
- This structure provides two type of operations on such array
  - Changing value of any element of a sequence;
  - Finding minimum (or maximum) of subsequence consisting of adjacent element of a sequence (*query*).

Binary Indexed Trees (BIT)



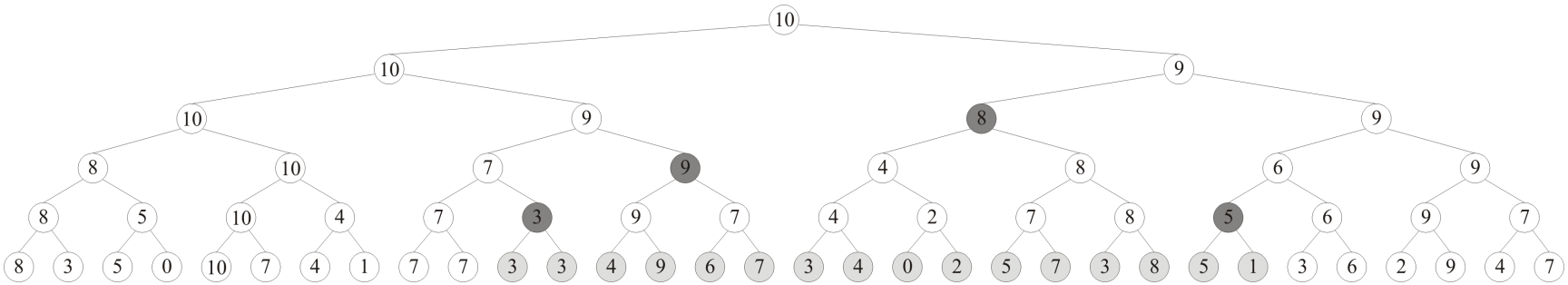
- BIT is efficient data structure introduced by Fenwick (1994) for maintaining cumulative frequencies
- In our case it is used for efficient calculating local minima or local maxima of subsequence of a sequence whose elements change their values during computation
- (Almost) complete binary tree
- Leafs of the tree contain values of array;
- Each non-leaf vertex u contains minimum (maximum) of values stored in leafs of subtree rooted in u
- It is easy to conclude that height of tree is $\lceil \log_2 n \rceil$ (n is sequence cardinality).

BIT - Finding minimum (maximum) of subsequence

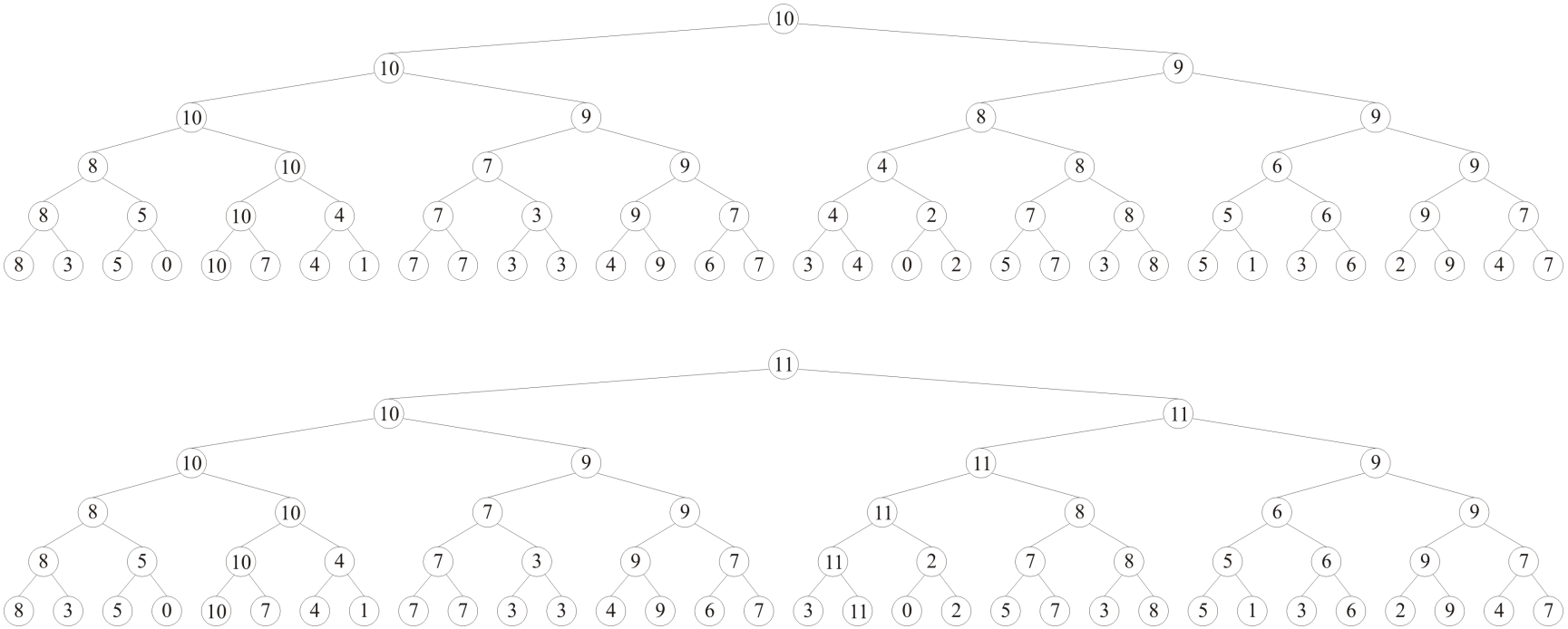


- We find maximum of sub-sequence containing light gray elements;
- Instead of comparing each of these elements with current maximum we compare only values stored in dark gray nodes
- There are at most two dark colored nodes at each level of the tree (depending of subsequence)
- So, complexity of calculating maximum of subsequence with k elements is $\Theta(\log k) = \Theta(\log n)$.

BIT - Find maximum, second example



Updating tree after setting value of any element



Updating tree after setting value of any element

- Change the value of the element a_i of an array a can influence the change in value stored only in nodes which are roots of subtrees containing leaf storing element a_i
- In the previous example leaf containing value 4 change value, and new value is 11 (dark gray colored leaf)
- Nodes containing this leaf are on path from this leaf to root of BIT (other gray colored nodes)
- There are $\lceil \log n \rceil$ nodes on the path and because of that complexity of updating is $\Theta(\log n)$

Using BIT in 2-opt for PDTSP

- For the current solution x we calculate loads of vehicle and create BIT with loads stored in leafs;
- For each 2-opt move (i, j) $(i < j)$ we perform the following steps
  - Check length of the tour after this move
  - Check its feasibility by calculating maximal and minimal load
  - Maximal load is calculated in the following way

    $$\max(\max\{L_1, ..., L_i, \max\{L_{j+1}, ..., L_n\}, L_j + L_i - \min\{L_i, L_{i+1}, ..., L_{j-1}\})$$

  - Minimal load is found in similar way

    $$\min(\min\{L_1, ..., L_i\}, \min\{L_{j+1}, ..., L_n\}, L_j + L_i - \max\{L_i, L_{i+1}, ..., L_{j-1}\})$$

Properties of BIT for solving PD-TSP

Proposition

Updating the binary index tree after setting the element L_i to the new value is executed in $O(\log n)$.

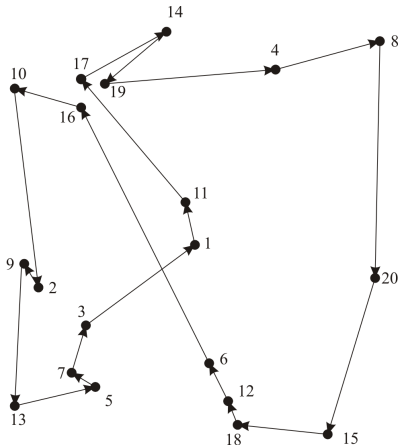
Proposition

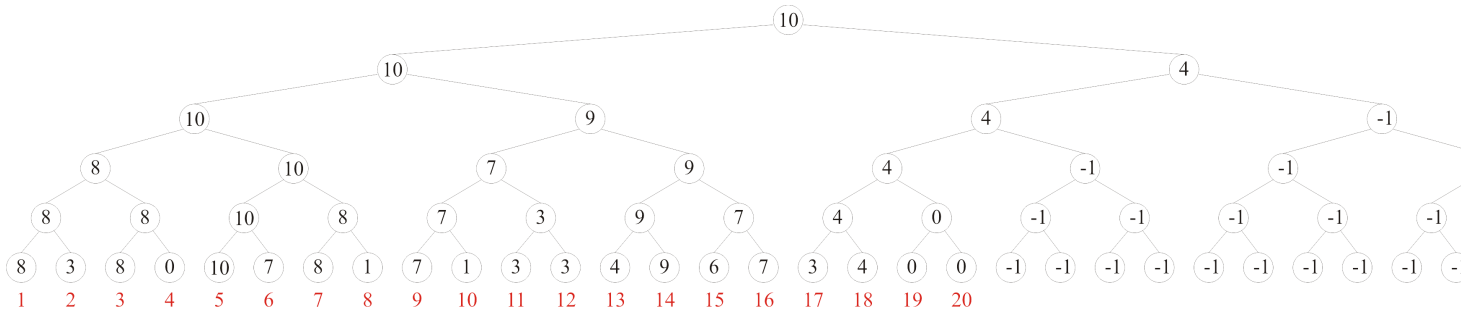
Calculating the maximum value in interval $[L_i, L_j], j > i$ is in $O(\log n)$ time with BIT structure.

Proposition

Checking the feasibility of the 2-opt move for 1-PDTSP with BIT structure is in $O(\log n)$.

Illustrative example





Initial solution

- Choose the first customer x_2 at random;
- Choose x_{i+1} among $c = 20$ closest customers of x_i;
- Consider only feasible sub-tours;
- Among c closest customers, search for those who could be feasibly added at the end of the tour T but not yet visited; select the customer with the largest demand;
- If such a customer does not exist, we search for all customers who have not appeared in the sub-tour T;
- Let S be the set of customers who can be feasibly added to the sub-tour;
  - Select the nearest customer from S with a probability of 0.9, or
  - select a random customer from S with a probability of 0.1
- If there is no customer that can be added ($S = \emptyset$), we add a random customer and continue.

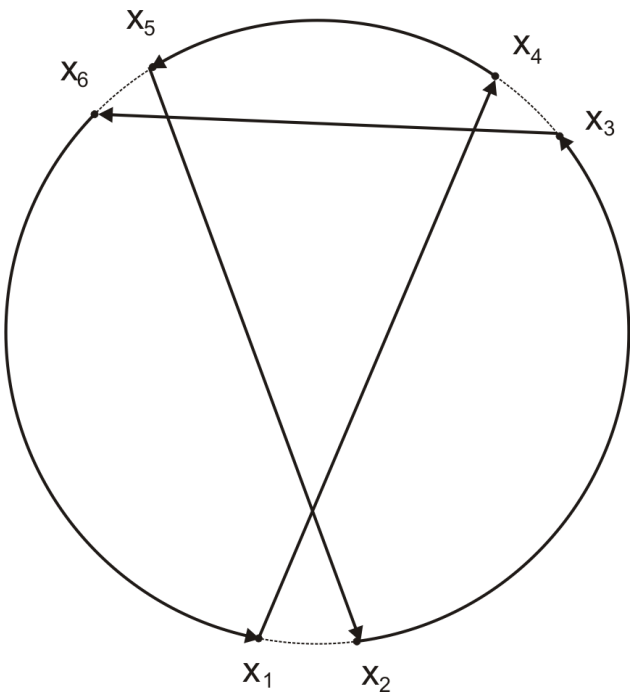
Local search for 1-PDTSP

- Local search is Seq VND thorough the following neighborhoods
  1. 1-opt
  2. 2-opt
  3. Forward and backward insertion

Shaking

- The simplest variant for shaking is to perform sequence of k moves (1–opt, 2–opt or insertion)
- But in that case we often do not get feasible solution
- We decide to make 'smart' moves in order to produce feasible solution after perturbation.

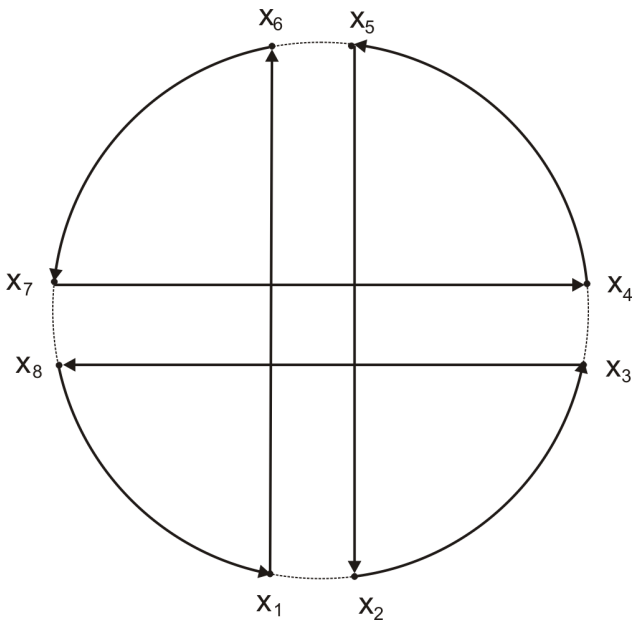
Maintaining feasibility of 3–opt



- If we select indices i_1, i_3 i i_5 such that $L_{i_1} = L_{i_3} = L_{i_5}$ and perform modification as presented on figure we obtain feasible tour (if previous tout is feasible)
- We can prove that vehicle loads after visiting customers x_{i_2}, x_{i_4} and x_{i_6} are unchanged
- For example

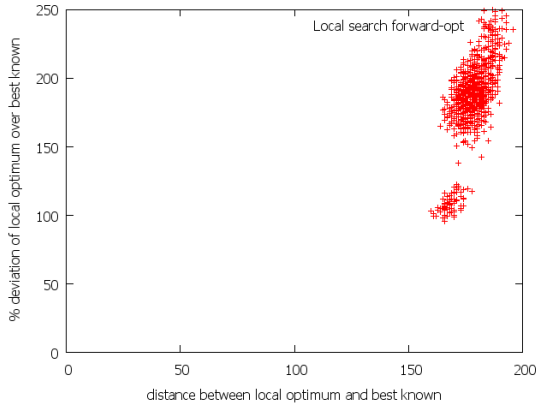
$$L'(x_{i_4}) = q_{x_1} + q_{x_2} + \cdots + q_{x_{i_1}} + q_{x_{i_4}} =$$
$$= L(x_{i_1}) + q_{x_{i_4}} = L(x_{i_3}) + q_{x_{i_4}} = L(x_{i_4})$$

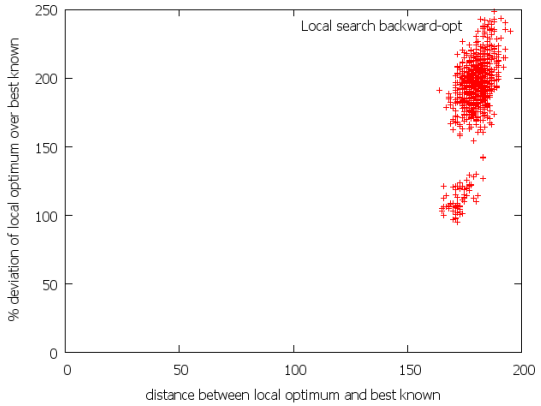
Maintaining feasibility of double–bridge move

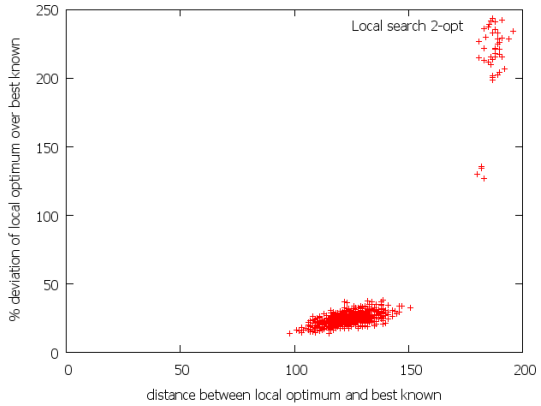


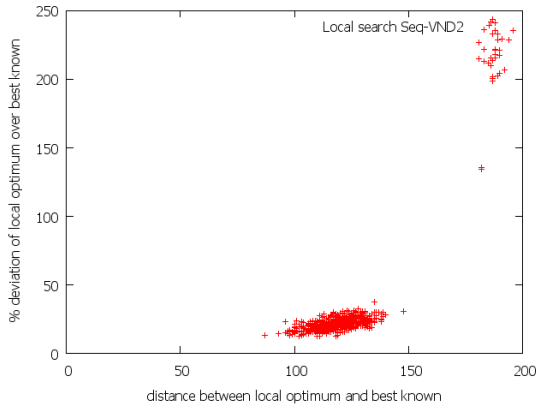
- Indices i_1, i_3, i_5 and i_7 are selected such that loads after visiting corresponding customers are same.

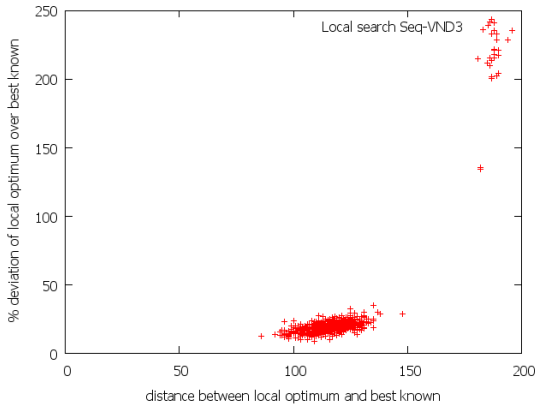
Comparison of local search (neighborhoods)

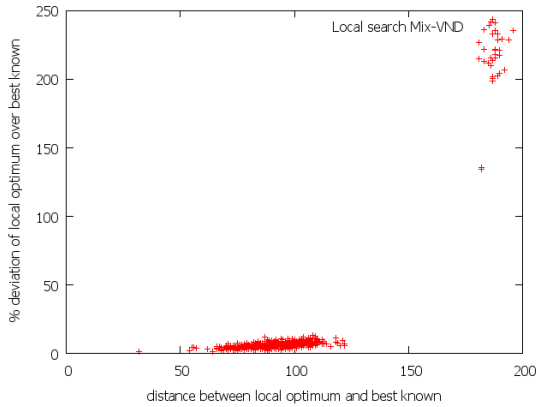




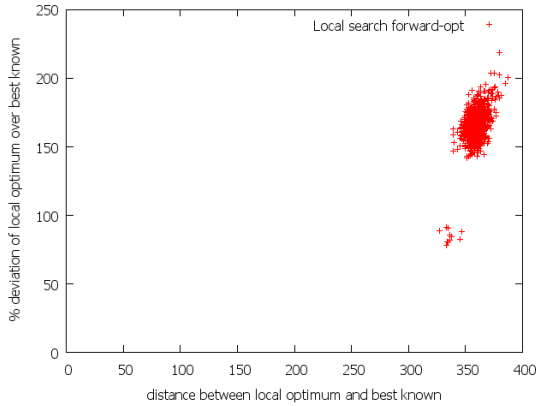


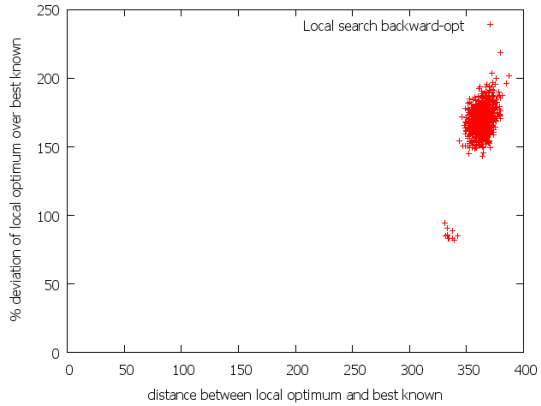


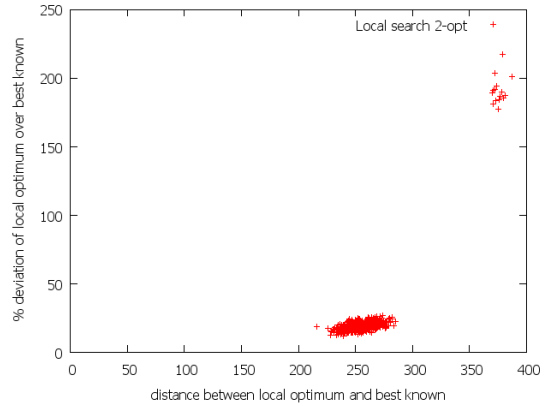


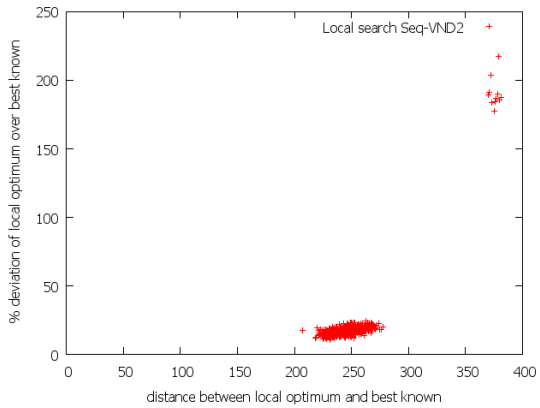


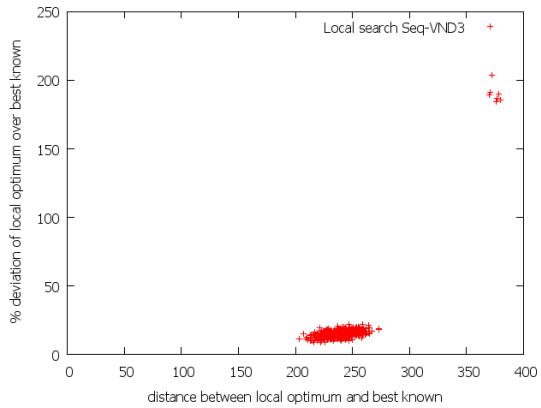
Comparison of local search

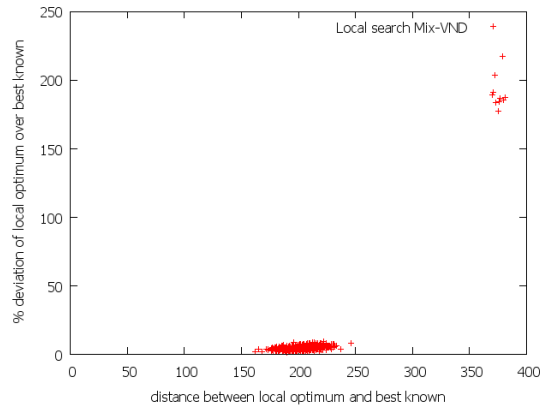












n	Q	Local search	Min. % dev	Max. % dev	Avg. % dev	Av time
200	10	Forward-insertion	95.650	255.517	181.195	0.088
		Backward-insertion	94.753	252.827	186.318	0.081
		2–opt	13.882	242.910	32.433	0.138
		Seq–VND–2	12.275	242.910	27.808	0.163
		Seq–VND–3	8.991	242.910	24.309	0.478
		Mix–VND	1.269	242.910	12.958	2.989
400	10	Forward-insertion	78.320	218.770	165.603	0.385
		Backward-insertion	81.881	218.770	169.416	0.317
		2–opt	12.078	217.104	21.852	0.831
		Seq–VND–2	10.684	217.104	18.954	0.769
		Seq–VND–3	8.951	203.738	16.035	4.062
		Mix–VND	1.492	217.104	6.573	26.569

Table: Comparison of local search

Comparison of shaking

Parameters		VNS (Fine shaking)			VNS (Classical shaking)		
n	Q	Best	Avg.	Time	Best	Avg.	Time
200	10	18699.1	18989.62	49.74	19658.3	24353.16	103.98
200	20	13385.1	13627.38	37.60	13879.1	14347.13	87.17
200	40	11223.8	11323.93	18.00	11279.4	11422.59	73.47
400	10	25545.1	25962.14	165.95	28176.1	34302.72	230.58
400	20	18518.9	18786.59	69.41	20128.5	20943.97	129.57
400	40	15680.0	15803.19	48.67	16101.8	16332.48	146.67

VND without and with 3-opt

Parameters		GVNS (with VND-2)			GVNS (with VND-3)		
n	Q	Best	Avg.	Time	Best	Avg.	Time
200	10	18699.1	18989.62	49.74	18709.1	19000.88	51.70
200	20	13385.1	13627.38	37.60	13391.4	13637.40	40.10
200	40	11223.8	11323.93	18.00	11236.4	11338.02	19.73
400	10	25545.1	25962.14	165.95	25555.6	25974.50	167.58
400	20	18518.9	18786.59	69.41	18530.7	18801.92	71.05
400	40	15680.0	15803.19	48.67	15687.2	15821.80	50.62

Table: Comparison of GVNS with VND-2 and VND-3 as local searches respectively

Seq-VND or Mix-VND

Parameters		VNS with Seq-VND			VNS with Mix-VND		
n	Q	Best	Average	Time	Best	Average	Time
200	10	18699.1	18989.62	49.74	18578.8	18774.22	75.69
200	20	13385.1	13627.38	37.60	13319	13439.86	67.58
200	40	11223.8	11323.93	18.00	11214.8	11249.22	42.32
400	10	25545.1	25962.14	165.95	25467.2	25752.63	165.44
400	20	18518.9	18786.59	69.41	18407	18647.13	152.36
400	40	15680.0	15803.19	48.67	15602.9	15711.91	144.29

Table: Comparison of Seq-VND and Mix-VND

Parameters		Best	VNS 1		VNS 2		GRASP VND		GA		CPU time		
n	*Q*	known	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	VNS 1	VNS 2	GRASP
100	10	12718.60	0.29	1.52	0.0	0.35	1.88	4.62	0.96	1.85	9.96	23.79	8.85
100	20	9357.60	0.02	0.96	0.00	0.18	0.65	2.55			5.89	14.95	2.22
100	40	8165.40	0.00	0.20	0.00	0.04	0.00	0.57			1.84	5.97	0.69
200	10	18578.80	0.65	2.21	0.00	1.05	4.81	7.50	2.70	4.09	49.74	75.69	41.77
200	20	13319.00	0.50	2.32	0.00	0.91	4.58	6.86			37.60	67.58	17.37
200	40	11214.80	0.08	0.97	0.00	0.31	1.34	3.18			18.00	42.32	4.35
300	10	22935.30	0.83	2.39	0.00	1.47	5.30	7.67	4.20	5.62	104.61	122.83	117.86
300	20	16313.40	0.88	2.60	0.00	1.43	6.60	8.62			38.74	115.93	50.90
300	40	13671.40	0.41	1.40	0.00	0.56	2.85	4.80			24.91	88.63	12.89
400	10	25467.20	0.31	1.94	0.00	1.12	5.66	7.68	4.02	5.82	165.95	165.44	220.40
400	20	18407.00	0.61	2.06	0.00	1.30	6.49	8.61			69.41	152.36	91.73
400	40	15602.90	0.49	1.28	0.00	0.70	3.41	5.20			48.67	144.29	23.92
500	10	28774.20	0.00	1.54	0.10	1.28	5.80	7.76	5.57	7.37	124.14	209.76	391.01
500	20	20927.00	0.17	1.70	0.00	1.38	6.53	8.43			107.01	194.76	164.77
500	40	17495.50	0.41	1.50	0.00	0.86	4.47	6.10			89.81	193.52	43.98
1000	10	44744.20	0.96	19.74	0.00	1.52					349.59	393.22	
1000	20	31661.10	1.64	3.57	0.00	1.56	7.69	8.95			478.08	441.03	618.33
1000	40	25450.00	1.20	2.66	0.00	1.28	6.64	8.14			474.72	430.16	440.00

Table: Results on large instances

Conclusions and future work

- We suggest GVNS for solving 1-PD-TSP which contains two NP-hard problems: find minimum TSP tour and find feasible tour;
- Classical k-opt neighborhoods are adapted;
- Binary index tree data structure used for efficient feasibility checking of 2-opt move;
- Both Sequential and mixed nested VND are used within GVNS;
- **All best known solution improved** on large benchmark instances (with up to 500 customers);
- We are applying similar approach for solving Travelling deliveryman problem (with and without profit);
- We are also working on PD-VRP.

Thank you for your attention

nenad.mladenovic@brunel.ac.uk