EC - The use of move evaluations (deltas) from previous iterations in local search
Assignment 5

Adam Korba - 151962 Łukasz Sztukiewicz - 151959

PSEUDOCODE

Input:

n - number of points in the problem

D - distance function that returns distance between two points: *D(x, y)*

- costs of visiting node are incorporated into the function

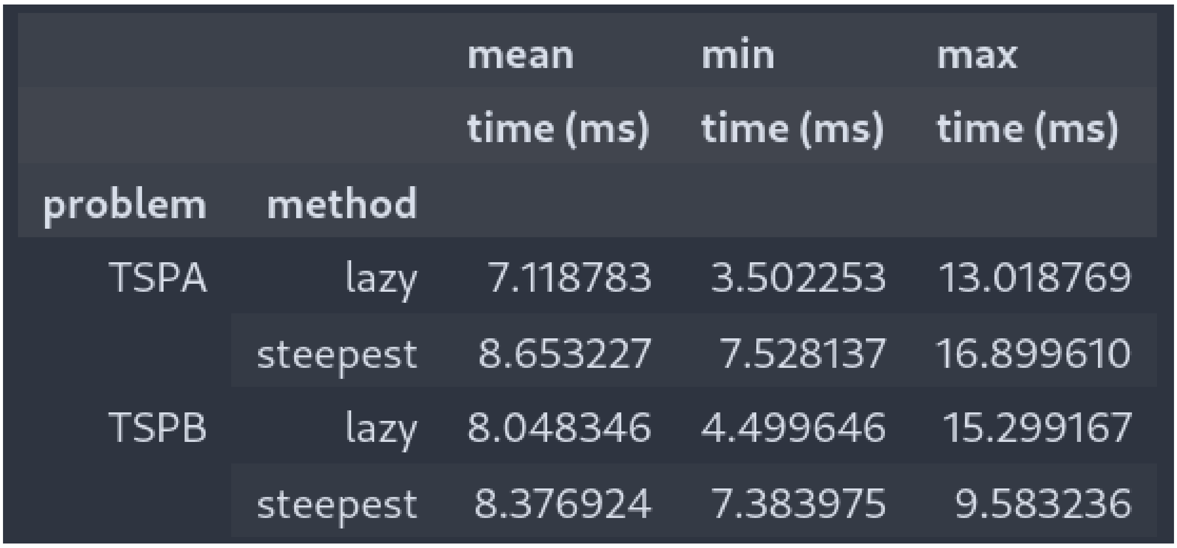
sol - starting solution

```
EDGES = [[...]] // 2d hashmap to check if edge is still present
UNSELECTED = [...] // hashmap to check if node is still not selected
EDGES and UNSELECTED are generated when constructing starting solution
```

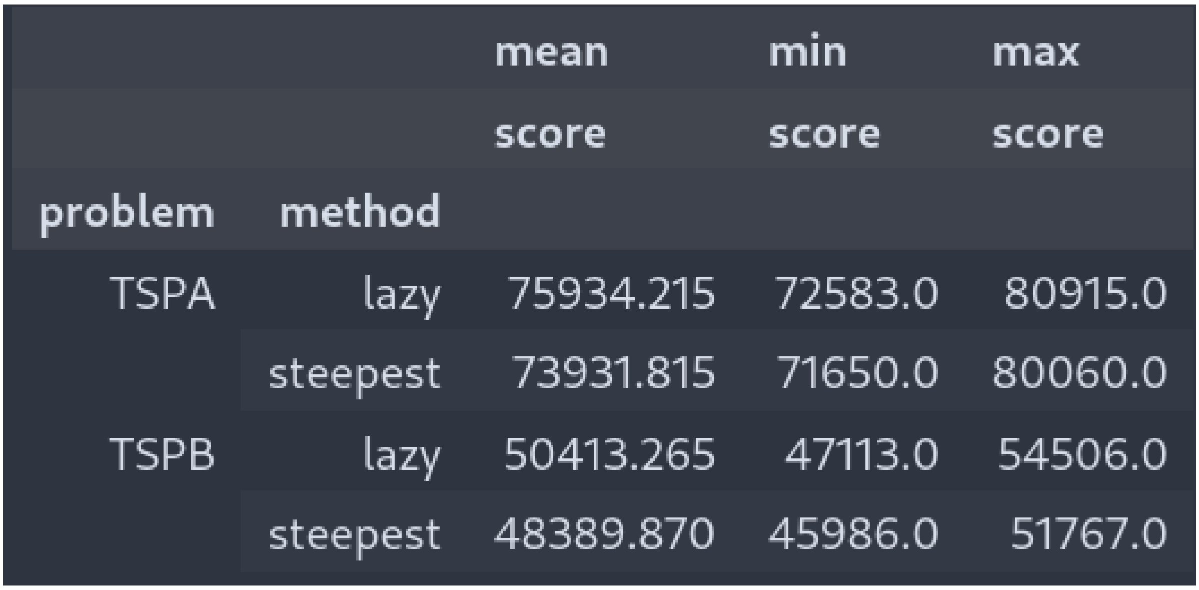
```
function LazySteepestLocalsearch(n, D,sol, EDGES, UNSELECTED)
   1. PQ = MinHeap()
   2. for move in all_possible_moves(sol)
         - delta = evaluate_move(sol, move)
         - if delta < 0: PQ.add(priority=delta, value=move)
   3. while PQ not empty:
         a. delta, move = PQ.pop()
         b. if not is_still_valid(move, sol, EDGES, UNSELECTED):
              i. continue
         c. make_move(move, sol, EDGES, UNSELECTED)
         d. evaluate_new_moves(PQ, move, sol)

function is_still_valid(move, sol, E, U)
   1. if move.type == “edge_exchange”
         a. return move.first_edge in E and move.second_edge in E // case 1
   2. else: // node_exchange
         a. first = move.first_node
         b. return (first.prev, first) in E and (first, first.next) in E
            and move.second_node in U
```

```
function evaluate_new_moves(PQ, move, sol):
      if move.type == “edge_exchange”:
            for edge in move.new_edges:
                  add_all_possible_edge_exchanges(PQ, edge, sol)
      else: // node_exchange
            first = move.first_node
            // add edge exchanges in both directions case 2 and case 3
            add_all_possible_edge_exchanges(PQ, (first.prev, first), sol)
            add_all_possible_edge_exchanges(PQ, (prev, first.next), sol)
            add_all_possible_node_exchanges(PQ, first, sol)
```

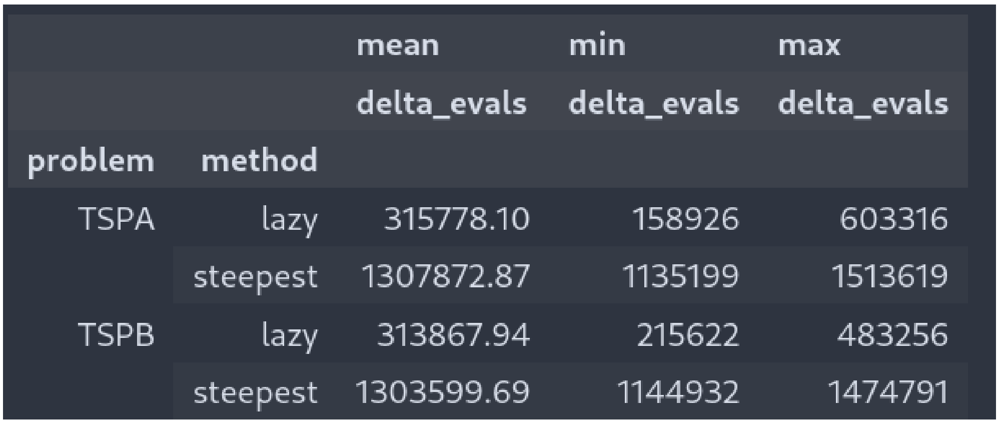
RESULTS



		mean	min	max
		time (ms)	time (ms)	time (ms)
problem	method			
TSPA	lazy	7.118783	3.502253	13.018769
	steepest	8.653227	7.528137	16.899610
TSPB	lazy	8.048346	4.499646	15.299167
	steepest	8.376924	7.383975	9.583236

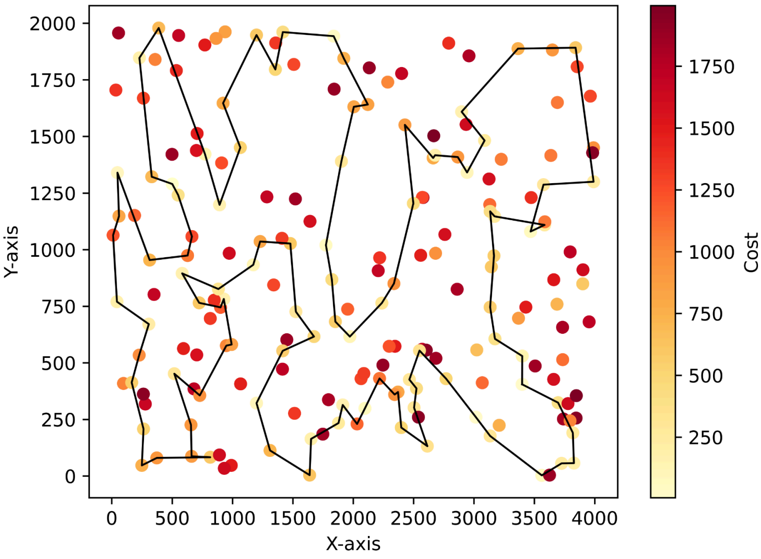


		mean	min	max
		score	score	score
problem	method			
TSPA	lazy	75934.215	72583.0	80915.0
	steepest	73931.815	71650.0	80060.0
TSPB	lazy	50413.265	47113.0	54506.0
	steepest	48389.870	45986.0	51767.0

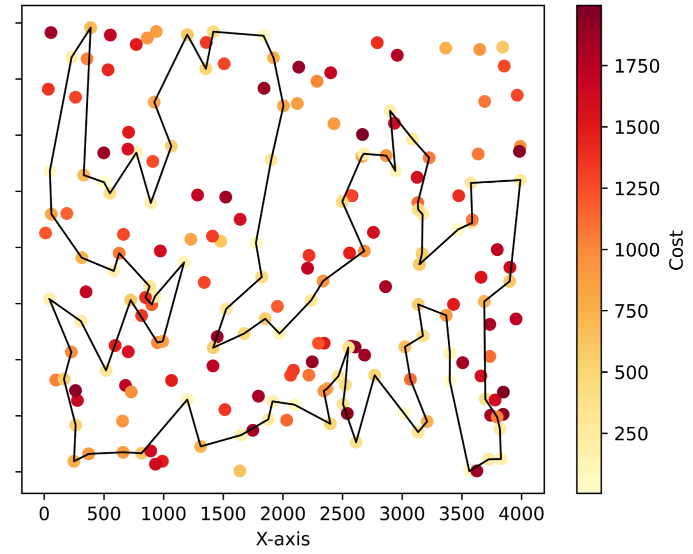


		mean	min	max
		delta_evals	delta_evals	delta_evals
problem	method			
TSPA	lazy	315778.10	158926	603316
	steepest	1307872.87	1135199	1513619
TSPB	lazy	313867.94	215622	483256
	steepest	1303599.69	1144932	1474791

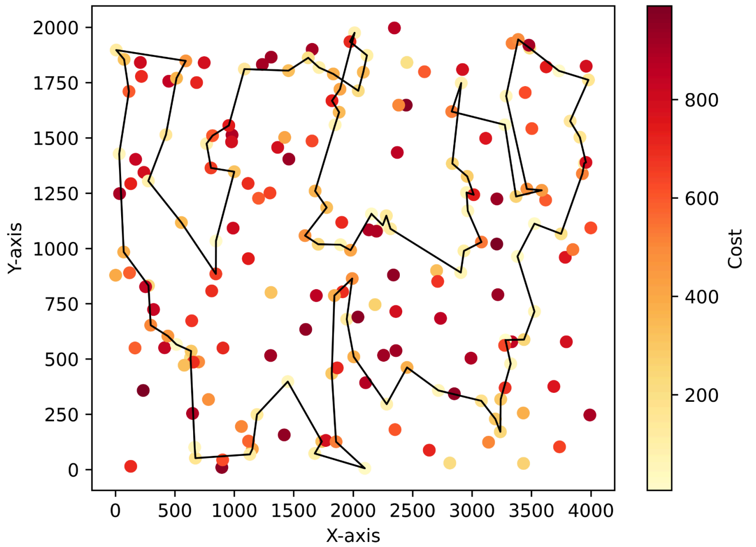
Number of delta evaluations



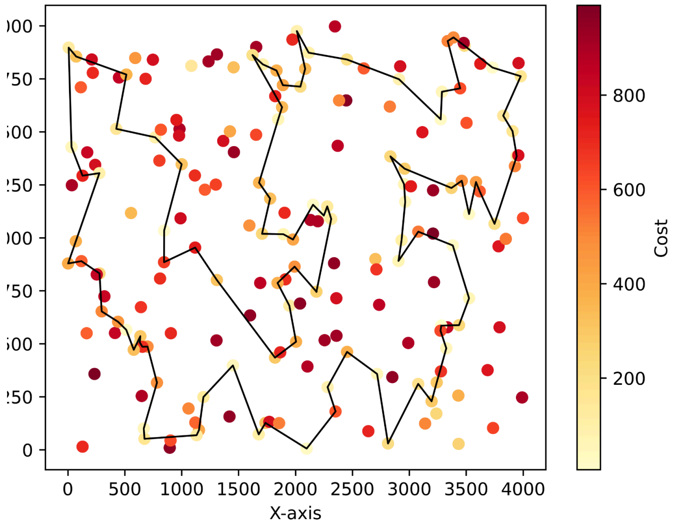
TSPA - Steepest localsearch - lazy evaluation (score: 72583.0)



TSPA - Steepest localsearch (score: 71650.0)



TSPB - Steepest localsearch - lazy evaluation (score: 47113.0)



TSPB - Steepest localsearch (score: 45986.0)

Conclusions

Method utilizing the move evaluations (deltas) from previous iterations in local search proved to drastically decrease the number of move evaluations that have to be calculated during the execution of the algorithm. However in our experiments there was no noticed time improvement, which might be caused by overhead that this method requires:

- priority queue storing all moves,
- keeping track of which moves are still possible

CODE

https://github.com/BbqGamer/TSP/