The Traveling Deliveryman Problem in a specific application scenario: Meta-heuristic vs. ILP approach

Jan Mikula
Thursday 14:30-16:00
Cybernetics and Robotics
mikulj14@fel.cvut.cz

I. Introduction and motivation

Suppose a set of customers in a city waiting for their deliveries, and travel times between each pair of them to be known. The Traveling Deliveryman Problem (TDP) asks for a sequence of visits such that each customer is served exactly once, and the sum of all waiting times is minimized. This problem can be viewed as customer-oriented, as the one who provides the service seeks to satisfy the customers rather than minimizing own travel expenses [1]. A closely related and well-studied is the Traveling Salesman Problem (TSP), which aims in the just-mentioned opposite direction, i.e., it is so-called server-oriented. Both problems are known to be NP-hard for general metric spaces [2], and as their range of applications is multidisciplinary and wide, they have received much attention in the operations research literature in past decades (although the TSP significantly more than the TDP). For an exhaustive overview of the TSP and its applications, see [3]; practical applications of TDP that are traditionally mentioned by operations research community are, e.g., customer-centered routing such as pizza-delivery or repairs of appliances [4], data retrieval in computer networks [5] or emergency logistics [6]. Quite recently, a different side of the scientific spectrum started to take notice of the problem and seek its efficient solution; that is - robotics. The TDP and its generalized version Graph Search Problem (GSP) introduced in [7] can be used to formulate a mobile robot search, another problem which is shown to be NP-hard in [8]. The robotic problem can be approximated by one of the two more straightforward combinatorial optimization problems, as shown in [9], [10], for either known or unknown environment in which the robot operates. The solution can be used, e.g., as an efficient planner in a rescue scenario in which the mobile agent searches for victims after some catastrophic event.

This short paper-like report is a semestral project assigned by the Combinatorial Optimization course at Czech Technical University in Prague, which I attend in 2020. Here, I present a meta-heuristic method called Ms-GVNS, which I designed for solving the TDP in a specific context of mobile robotics introduced in [9], [10], which is characterized by the need to periodically replan a route while a robot moves and senses an environment. The method's design and comparison with state-of-the-art heuristics from the literature were made as part of my on-going diploma theses work and therefore are not part of this report. The previous work aimed to find a solution of the best possible quality while bounding the computational time by a given hard limit so that the replanning can be done in real-time with a fixed frequency. However, the method was not yet assessed in other computational contexts, such as the ability to find an optimal solution for smaller instances (< 100 customers), which is the domain of Integer Linear Programming (ILP) solvers. The primary motivation of this work is to evaluate the method in the latter context. Assume a scenario where the optimal solution of TDP is desired but need not be guaranteed (i.e., a non-optimal solution can be rarely accepted), and we are interested in the pros and cons of using either the meta-heuristic or an ILP model. More specifically, as follows. What is the maximal statistical error of meta-heuristic not finding the optimal for instances where the ILP model can provide it? How much faster (or slower) is the meta-heuristic compared to the ILP model? Do there exist instances that can be solved by the ILP (in a reasonable time), but the meta-heuristic struggles to find the optimum? How hard is it to implement one or the other? To answer these questions, I present for the TDP an ILP formulation as well as a combinatorial formulation used initially with the meta-heuristic. As the literature is abundant with various ILP models, their review and a justified choice of the suitable ones take significant importance in this report. The chosen models from Fischetti et al. [4] and Naeni and Salehipour [11] are implemented in C++ using the Gurobi$^{\text{TM}}$ alongside the proposed method and a reference meta-heuristic introduced in Silva et al. [12]. For the computational evaluation, I use several sets of benchmark instances [13].

II. Problem Statement

A. Combinatorial formulation

The Traveling Deliveryman Problem, also known as the Repairman Problem, or the Minimum Latency Problem, is formally described by

- $G := (V, E)$: a complete undirected graph with N vertices $V := \{v_1, \dots, v_N\}$ in which every pair of distinct ones $v_i \neq v_j$ is connected by a unique edge $e_{i,j} = (v_i, v_j) \in E$,
- $d : E \to \mathbb{R}_0^+$: a non-negative cost $d(i,j)$ associated with each edge $e_{i,j}$ representing a length of the shortest path (or travel time) from v_i to v_j,
- $s \in V$: a starting vertex (depot) of the deliveryman; all other vertices are the customers.

Let the sequence of vertices $\mathcal{X} = \langle\, x_0 = s, x_1, \dots, x_n \,\rangle$, where $n := N - 1$ is the number of customers, be a Hamiltonian path in G starting from the depot. Furthermore, let $d(x_i, x_j)$ be the cost of an edge between i-th and j-th vertex in $\mathcal{X}$. The cumulative cost to reach k-th vertex in the sequence $\mathcal{X}$ is defined as $\delta_k^{\mathcal{X}} := \sum_{i=1}^{k} d(x_{i-1}, x_i)$. Finally, the total cost of $\mathcal{X}$ is defined as $\texttt{Cost}(\mathcal{X}) := \sum_{k=1}^{n} \delta_k^{\mathcal{X}}$. The objective of the TDP is to find an optimal path $\mathcal{X}^*$ that minimizes the cost, i.e.,

$$\mathcal{X}^\star := \underset{\mathcal{X} \in \mathcal{H}(\pi)}{\arg\min}\, \texttt{Cost}(\mathcal{X}) = \underset{\mathcal{X} \in \mathcal{H}(\pi)}{\arg\min} \sum_{k=1}^{n} \delta_k^{\mathcal{X}} = \underset{\mathcal{X} \in \mathcal{H}(\pi)}{\arg\min} \sum_{k=1}^{n} \sum_{i=1}^{k} d(x_{i-1}, x_i), \tag{1}$$

where $\pi = (G, d, s)$ is an instance of the TDP and $\mathcal{H}(\pi)$ is the set of all Hamiltonian paths in graph G starting in s. Note that I assume an open variant of the problem, i.e., the travel from the last customer back to the depot, is not considered. However, a Hamiltonian circuit instead of the path is often deemed in the literature. In that case, the cost would be defined as $\texttt{Cost}(\mathcal{X}) := \left(\sum_{k=1}^{n} \delta_k^{\mathcal{X}}\right) + \delta_n^{\mathcal{X}} + d(x_n, x_0)$.

B. ILP formulation

Creating and solving an ILP model is a very common way of dealing with many combinatorial optimization problems. The TDP can be formulated as a ILP model in different ways. Here I state as a reference an early intuitive model introduced in Fischetti et al. [4]. First, denote the set of customers with labels $1, \dots, n$, and the depot with label 0. Additionally, note that the expression $\sum_{k=1}^{n} \delta_k^{\mathcal{X}}$, which figures in the objective function defined above, can be rewritten as $\sum_{k=0}^{n-1} (n-k) d(x_k, x_{k+1})$. This yields the following selection of variables:

$$X_{ij} = \begin{cases} n-k, & \text{if edge } (i,j) \text{ is } k\text{-th in the sequence} \\ 0, & \text{if } (i,j) \text{ is not used.} \end{cases} \tag{2}$$

$$Y_{ij} = \begin{cases} 1, & \text{if the deliveryman traverses edge } (i,j) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Then the MILP formulation of the TDP is as follows:

$$\min \textstyle\sum_{i=0}^{n} \sum_{j=0}^{n} d(i,j) X_{ij} \tag{4}$$

$$\text{s. t.} \quad 1 = \textstyle\sum_{j=0, i\neq j}^{n} Y_{ij} \quad i = 0, 1, \dots, n \tag{5}$$

$$1 = \textstyle\sum_{i=0, i\neq j}^{n} Y_{ij} \quad j = 0, 1, \dots, n \tag{6}$$

$$n = \textstyle\sum_{i=1}^{n} X_{0i} \tag{7}$$

$$1 = \textstyle\sum_{i=0, i\neq k}^{n} X_{ik} - \sum_{j=0, j\neq k}^{n} X_{kj} \quad k = 1, 2, \dots, n \tag{8}$$

$$X_{i0} = 0 \quad i = 1, 2, \dots, n \tag{9}$$

$$X_{0j} \leq n Y_{0j} \quad j = 1, 2, \dots, n \tag{10}$$

$$X_{ij} \leq (n-1) Y_{ij} \quad i, j = 1, 2, \dots, n; i \neq j \tag{11}$$

$$Y_{ij} \in \{0, 1\} \quad i, j = 1, 2, \dots, n; i \neq j \tag{12}$$

$$X_{ij} \in \{0, 1, \dots, n\} \quad i, j = 1, 2, \dots, n; i \neq j \tag{13}$$

Constraints (5) and (6) assure each customer is visited and left once and once only. Constraints (7) and (8) guarantee zero occurrence of subpaths that do not include all customers. Constraints (9)-(11) bind together Y_{ij} and X_{ij} variables (e.g., $Y_{ij} = 0 \Rightarrow X_{ij} = 0$). Overall, this model contains $2n(n+1)$ variables and $n^2 + 4n + 3$ constraints.

III. Related works

Two noticeable leading courses are apparent in solving the TDP in the operations research community. The first one aims to find the optimal solution to the problem; however, it is limited to small instance sizes due to infeasible computing times. The second major course, a more relaxed one, seeks a solution that is only close enough to optimum in exchange for much lower computational times. Heuristics, often based on some more general search strategies (meta-heuristics), are the core solution methods in this area. Approximation algorithms, which lie somewhere in the middle, are also known for the TDP. These methods give approximate solutions, but, unlike heuristics, with a theoretically proven guarantee of performance. The researchers who develop approximation algorithms focus mostly on lowering the approximation factor or computational complexity of their algorithms; however, computational results on benchmark instances are usually not present in their works.

A. Exact algorithms

Early ones proposed by Lucena [14], and Bianco et al. [15] rely on non-linear integer formulations in which a Lagrangian relaxation is used to derive lower bounds. Fischetti et al. [4] develop Integer Linear Programming (ILP) formulation and new theoretical results on the matroidal structure of a class of combinatorial problems. The results are used to derive lower bounds for the TDP and are embedded into an enumerative algorithm capable of solving 60-vertices instances to optimality. Some other ILP formulations, and exact algorithms are proposed in [16], [17], [18], [19], [11]. In the last mentioned, authors Naeni and Salehipour develop an ILP model that benefits from position-based variables. On the set of 70 randomly generated instances of sizes 10-50, they show their model can deliver the largest number of the best solutions in a shorter time compared to the most of the already mentioned models [4], [16], [17], [18].

B. Approximation approach

Approximation algorithms for the TDP on a tree or on a general metric graph are developed, e.g., in [1], [5], [20], [21], [22], [23], [24]. The lowest approximation factors in the literature are 3 [24] and 3.59 [21] for the tree and the general case respectively.

C. Meta-heuristics

The heuristic approach mostly relies on more general search strategies (meta-heuristics), especially Variable Neighborhood Search (VNS) [25] and Greedy Randomized Adaptive Search Procedures (GRASP) [26]. Salehipour et al. [13] propose a GRASP that embeds either Variable Neighborhood Descent (VND) [27] or VNS and evaluate both variants on a set of randomly generated benchmark instances of sizes in a range from 10 to 1000. Silva et al. [12] later present a simple and effective meta-heuristic called GILS-RVND, which is based on the combination of GRASP, Iterated Local Search (ILS) [28], and Randomized VND (RVND) [29]. It improves all the results obtained by [13] on their instances and finds new best solutions for two of TSPLIB [30] instances. Mladenović et al. [31] propose a Generalized VNS (GVNS), which is also able to improve the previous results obtained by [13]. Ban et al. [19] suggest a meta-heuristic algorithm combining between Tabu Search (TS) [32] and VNS and show that it compares well with the state-of-the-art algorithms [13], [12] in the quality of obtained solutions. However, the TS-VNS does not improve the bar set by the GILS-RVND [12] in the matter of computing time. To the best of our knowledge, the approach by Silva et al. [12] is the one producing the best trade-off between solution quality and computational time in the literature.

IV. Problem solution

A. Ms-GVNS description

Intro: Variable Neighborhood Search (VNS) initially proposed by Mladenović and Hansen [25] is a single-start stochastic meta-heuristic based on the idea of improving a single solution temporally even by some non-improving steps. In its scheme, two phases alternate: a shake which allows escaping local optimum and a local search phase, which descents towards one. Additionally, a systematic change of neighborhoods within the search is applied. General VNS (GVNS) is a variant that uses Variable Neighborhood Descent (VND) in the local search phase. VND can be seen as a deterministic variant of VNS, which explores a solution space using several neighborhood structures, usually in sequential order. Greedy Randomized Adaptive Search Procedure (GRASP), unlike VNS, is a multi-start process that has been developed and established within the research community by the works of many authors, e.g., [33], [34], [35]. Greedy Randomized Adaptive (GRA) construction heuristic is applied to each restart to create a new solution, which is then improved by VND, and the best overall solution is returned at the end. GVNS and GRASP have similarities and also significant differences. They both use VND as a local search method, and both are stochastic to be able to escape local optima - but in a different way. While GVNS randomly perturbates the best current solution (in the shaking phase), GRASP creates an entirely new one in a randomized fashion and starts the search from the beginning.

I originally designed the method presented here as a hybrid GRASP-GVNS meta-heuristic. However, after thorough testing and parameter tuning, it turned out to provide the best results with a deterministic pure-greedy construction instead of GRA construction as first intended. This fact simplified its description from more complicated GRASP-GVNS to plain GVNS, which restarts after further attempts to improve a solution are no longer effective. The method is presented in Algorithm 1. It takes a triple of stopping criterion parameters i_{max}, c_{goal}, t_{max}, and then three extra j_{max}, $\mathcal{P}$, and $\boldsymbol{\mathcal{N}}$ qualitative parameters. Given the constant i_{max}, the algorithm stops and returns a valid solution after a fixed number of iterations i_{max} (restarts), if not stopped earlier by other criteria. Given the CPU time limit, the algorithm finishes, at worst, after t_{max} seconds plus some negligibly small delta. At last, the algorithm can also stop after it has found a solution with cost smaller or equal to given goal c_{goal}. Regarding the qualitative paramaters: j_{max} determines the number of inner-loop iterations, $\mathcal{P}$ is a sequence of $|\mathcal{P}|$ positive integers $\langle\, p_1, p_2, \dots \,\rangle$, and $\boldsymbol{\mathcal{N}}$ is a sequence of $|\boldsymbol{\mathcal{N}}|$ neighborhood structures $\langle\, \mathcal{N}_1, \mathcal{N}_2, \dots \,\rangle$.

The Algorithm 1 is first initialized (lines 2-3), and then the main (restart) loop follows (lines 4-19). Inside the restart loop, a greedy solution is assigned to a current solution (line 5), then follows the main GVNS loop (lines 6-16), and finally the current solution is assigned to incumbent if it is improving (lines 17-18). Inside the GVNS loop, index k is initialized to one (line 8), and the inner loop follows (lines 9-16). The algorithm is parameterized by a sequence of positive integers $\mathcal{P}$, which take a role in the perturbation phase (line 10), where the k-th member of the sequence is passed to the `Shake` procedure. The procedure is

Algorithm 1: Multi-start General Variable Neighborhood Search (Ms-GVNS)

```
Function Ms-GVNS($i_{max}$, $c_{goal}$, $t_{max}$, $j_{max}$, $\mathcal{P}$, $\boldsymbol{\mathcal{N}}$):
    $\mathcal{X}_0 \leftarrow$ Construct(); $\mathcal{X}^\star \leftarrow \mathcal{X}_0$;
    $i \leftarrow 1$; $stop \leftarrow$ false;
    while $stop$ = false and $i \leq i_{max}$ do          ▷ Loop 1: main Ms-GVNS loop ⇝ restart loop
        $\mathcal{X} \leftarrow \mathcal{X}_0$;
        $j \leftarrow 1$;
        while $stop$ = false and $j \leq j_{max}$ do      ▷ Loop 2: inner Ms-GVNS loop ⇝ main GVNS loop
            $k \leftarrow 1$;
            while $stop$ = false and $k \leq |\mathcal{P}|$ do          ▷ Loop 3: inner GVNS loop
                $\mathcal{X}' \leftarrow$ Shake($\mathcal{X}$, $p_k$);
                ($\mathcal{X}'$, $stop$) $\leftarrow$ Improve($\mathcal{X}'$, $t_{max}$, $c_{goal}$, $\boldsymbol{\mathcal{N}}$);
                if Cost($\mathcal{X}'$) < Cost($\mathcal{X}$) then
                    $\mathcal{X} \leftarrow \mathcal{X}'$; $k \leftarrow 1$; $j \leftarrow 1$;
                else
                    $k \leftarrow k + 1$;
            $j \leftarrow j + 1$;
        if Cost($\mathcal{X}$) < Cost($\mathcal{X}^\star$) then
            $\mathcal{X}^\star \leftarrow \mathcal{X}$;
        $i \leftarrow i + 1$;
    return $\mathcal{X}^\star$
```

Algorithm 2: Variable Neighborhood Descent (VND)

```
Function Improve($\mathcal{X}$, $t_{max}$, $c_{goal}$, $\boldsymbol{\mathcal{N}}$):
    $i \leftarrow 1$; $stop \leftarrow$ false;
    while $i \leq |\boldsymbol{\mathcal{N}}|$ do
        Denote the $i$-th neighborhood structure in sequence $\boldsymbol{\mathcal{N}}$ as $\mathcal{N}_i$.;
        $\mathcal{X}' \leftarrow \arg\min_{\widetilde{\mathcal{X}} \in \mathcal{N}_i(\mathcal{X})}$ Cost($\widetilde{\mathcal{X}}$);
        if Cost($\mathcal{X}'$) < Cost($\mathcal{X}$) then
            $\mathcal{X} \leftarrow \mathcal{X}'$; $i \leftarrow 1$;
            if Cost($\mathcal{X}$) $\leq c_{goal}$ then
                $stop \leftarrow$ true; break;
        else
            $i \leftarrow i + 1$;
        Get the total CPU time $t$ since start.;
        if $t \geq t_{max}$ then
            $stop \leftarrow$ true; break;
    return $\mathcal{X}$, $stop$
```

applied to the current solution and results in a new lower-level current solution $\mathcal{X}'$, which is improved (line 11) and evaluated (lines 12-13). If the cost of the $\mathcal{X}'$ is less than the cost of the current solution, then $\mathcal{X}'$ is assigned to the current, and k is reset back to 1. Else, k is incremented, and the loop starts over with the next parameter in $\mathcal{P}$. Note the *stop* flag returned from the improving procedure. If it is true, then the whole procedure quickly comes to an end and returns the incumbent solution. If not stopped by the flag, the algorithm performs i_{max} iterations, terminates, and returns the incumbent.

An initial greedy solution $\mathcal{X}_0$ is created in function `Construct`. In this solution, every vertex on position i is the nearest neighbor of preceding vertex on position $i-1$, with the exception of the depot vertex s, which is always placed at first position.

The basic VNS employs a mechanism that prevents getting stuck in local optima. In our more complex meta-heuristic, the mechanism is implemented in function `Shake`. It removes and re-inserts p_k edges from and to the given path such that a new feasible path is generated. The edges to be removed are chosen randomly, and the way the path is glued back together is randomized as well.

The last component of Ms-GVNS is a local search procedure implemented as the VND algorithm in function `Improve`. VND explores a solution space using several neighborhood structures. Its success relies on the following facts: a local optimum for one neighborhood structure is not necessarily a local optimum with respect to another neighborhood structure, and a global optimum is a local optimum with respect to all considered neighborhood structures. The pseudo-code of VND is shown in Algorithm 2. Note, that besides attempting to improve the given solution, function `Improve` is as well responsible for checking termination criteria given by goal cost c_{goal} and CPU time limit t_{max}.

A stably good performance of the Ms-GVNS is ensured by fixing the following parameters at values: $j_{max} = \lceil n/5 \rceil$, $\boldsymbol{\mathcal{N}} = \langle \mathcal{N}_{\texttt{2-opt}}, \mathcal{N}_{\texttt{swap}}, \mathcal{N}_{\texttt{or-opt2}}, \mathcal{N}_{\texttt{or-opt3}}, \mathcal{N}_{\texttt{or-opt4}} \rangle$, and $\mathcal{P} = \langle 4, 8, 12 \rangle$. Here $n = size(i)$ is the size of considered TDP instance i, and `2-opt`, `swap`, `or-opt2`, `or-opt3`, and `or-opt4` are standard operators used traditionally for TSP and related problems. All of the considered neighborhoods can be explored in $\mathcal{O}(n^2)$, if some smart pre-computed structures are used in the implementation.

B. ILP models

Here, I want to discuss the choice of ILP models of the TDP as well as some troubles that I encountered. Originally, I intended to implement and evaluate two ILP models of the TDP from the literature. I selected one from Fischetti et al. [4], as an intuitive reference model, and one from Naeni and Salehipour [11], as the state-of-the-art model. Let me call the models FI, and NS respectively. Model FI is fully described in Sec. II-B and its understanding, as well as implementation, is rather straightforward. The main improving idea of NS model is to introduce extra position-based variables

$$Z_{ij} = \begin{cases} 1, & \text{if vertex j is at i-th position of the solution} \\ 0, & \text{if else,} \end{cases} \tag{14}$$

and link them with the existing ones from the FI model. The authors of [11] report that model with this modification provides the best results in the shortest computational time. However, when first encountered it, I was having trouble both understand and implement some parts of it. After discussing with my supervisor, we have concluded that some constraints in the paper are most probably mistakenly written. Therefore, instead of showing here the NS model from the paper, I introduce a version of it, corrected by me, with the help of my supervisor. I call this model NS', and it expands the FI model by Z_{ij} variables (14), and the following constraints:

$$\tfrac{1}{2}n(n+1) = \textstyle\sum_{i=0}^{n} \sum_{j=0, j\neq i}^{n} X_{ij} \tag{15}$$
$$1 = Z_{00} \tag{16}$$
$$1 = \textstyle\sum_{i=0}^{n} Z_{ij} \quad j = 0, 1, \ldots, n \tag{17}$$
$$1 = \textstyle\sum_{j=0}^{n} Z_{ij} \quad i = 0, 1, \ldots, n \tag{18}$$
$$0 = \textstyle\sum_{i=0}^{n} (n-k+1) Z_{ik} - \sum_{i=0, i\neq k}^{n} X_{ik} \quad k = 1, \ldots, n \tag{19}$$
$$Z_{ij} \in \{0, 1\} \quad i, j = 0, 1, 2, \ldots, n \tag{20}$$

Constraint (15) sets the total sum of $X_{i,j}$ variables, and (16) ensures, that vertex 0 is at position 0. Constraints (17)-(18) guarantee that every vertex is at exactly one position, and each position is occupied by exactly one vertex. Eq. (19) binds together Z_{ij} and X_{ij} variables. Constraints shared with the FI model, (15)-(18), and (20) are as well stated in the paper [11] together with some extra constraints which were removed and replaced by constraint (19) in the NS' model. I contacted the authors of [11] and asked them to clarify their original NS model, and at the time of writing this report, I am waiting for their reply. Therefore, for practical reasons, from now on, the corrected model NS' is considered instead of NS in this work.

Note that constraint (15) does not contain Z_{ij} variables, and therefore it can be easily solely added to FI model to create a new model FI+.

V. Computational evaluation

A. Implementation

Ms-GVNS, the proposed meta-heuristic, and GILS-RVND [12], the reference meta-heuristic, are both implemented in C++, sharing all possible parts of code to ensure fairness. The implementation is single-threaded. The authors of [12] were so kind to provide me with their code so I could ensure that my implementation of GILS-RVND is not worse, in any sense, than their implementation.

All ILP models discussed in Sec. IV-B are implemented as well in C++ using Gurobi$^{\text{TM}}$ general MILP solver. The setting of variables and constraints into the models is rather straightforward, but some extra implementation effort is required for additional features, which I discuss next.

1) When computing, Gurobi$^{\text{TM}}$ uses all available CPUs by default. To ensure fairness when comparing to the meta-heuristics, I have to enforce using only a single thread by setting `GRB_IntParam_Threads` parameter to 1.
2) Next, by setting `GRB_DoubleParam_TimeLimit` parameter, I impose an upper bound on computational time.
3) If Gurobi$^{\text{TM}}$ finishes and returns the optimal solution, it reports the total time t_{total} of computation. However, $t_{total} = t_{opt} + t_{proof}$, where t_{opt} is the time the optimal solution was found, and t_{proof} is the time it took to proof its optimality. Since we are only interested in the former, I had to add a custom callback, which saves t_{opt} to memory to be recovered later. Additionally, this callback also performs a check on finding a target solution (if some given), and if yes, then it immediately terminates and records the so-far computation time.
4) Lastly, to provide better starting conditions for the solver, I code a way to set an initial feasible solution into the model.

B. Benchmark instances and methodology

The computational evaluation is done on six sets of benchmark instances with 5, 10, 15, 20, 25, and 50 customers (+ 1 depot). Each set contains 20 randomly generated problems of equal size and different costs of travel between the customers. Instances with 10, 20, and 50 customers are provided by the authors of [13]. Instances with five customers are custom created from 10-customer instances by ignoring five random customers. Analogously, 15-customer instances are created from 20-customer and 25 from 50.

The experiments follow a methodology that is explained next. Each considered method is run n_{run}-times on a fixed instance. The runs are assumed to be independent. The cost of the optimal solution (optimal objective value) is provided as a target to the method, and if the optimum is found, the method terminates immediately, and the running time rt is recorded. Alternatively, the running time may reach a provided upper bound t_{max}, in which case the experiment is terminated and marked as unsuccessful. After the n_{run} runs, the minimal, mean, and maximal value of rt, and its standard deviation from all successful runs are recorded.

Furthermore, all the times are used to produce a time-to-target plot (TTT-plot). TTT-plot is an approximation of the cumulative Run-Time Distribution (RTD), and its construction is well-described in Resende and Ribeiro [26], and we follow the same methodology shown by the authors. After finishing the last run, the recorded rts of successful runs are sorted in increasing order and the probability $p_i = (i - 1/2)/n_{run}$ is associated with each i-th sorted rt_i, for $i = 1, \dots, n_{run}$. The meaning of p_i can be understood as follows: p_i is the probability that the algorithm finds a solution at least as good as the target in at most rt_i seconds. Finally, a TTT-plot is constructed by plotting all points (rt_i, p_i). One drawback of TTT-plots is that whenever two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ are evaluated on the same instance, and their TTT-plots are superimposed, it might not be clear on the first sight, which algorithm performs better and by how much. To compare the algorithms productively, some adequate metric must be introduced. Let RT_1 and RT_2 be random variables representing the time needed by algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively to find a solution as good as the given target value. Let $p_{12} \equiv P(RT_1 \leq RT_2)$ be the probability, that the random variable RT_1 takes a value smaller or equal to RT_2. Assuming that both algorithms stop when (and only if) they find a solution at least as good as the target, we can say that $\mathcal{A}_1$ performs better than $\mathcal{A}_2$ if $p_{12} > 0.5$. An iterative procedure to compute p_{12} with arbitrary small approximation error for two algorithms following general cumulative probability distributions is introduced in [36]. Later, the authors [37] developed a program to compute the approximation of p_{12} from provided TTT-plots of the two algorithms. In this work, I use a computation inspired by their program.

C. Results and their discussion

The experimental part of this work is done in two phases. The purpose of the first, preliminary phase, is to select the best ILP model among those discussed in Sec. IV-B. Then, in the second phase, the chosen model is compared to the performance of two considered meta-heuristics. The meta-heuristics are also compared to each other, which was not done before in this computational context, where optimal solutions are required, and considered instances are small. All experiments were executed on a personal computer with Intel® Core$^{\text{TM}}$ i7-6700 CPU (3.40 GHz), 16 GB of RAM, and Ubuntu 18.04.4 LTS.

In the first phase, we select a subset of 10 instances of size 21 (20 customers + 1 depot), and perform ten runs for each with models FI, FI+, and NS'. The average running times are 530, 46, and 83 seconds respectively. Based on these results, we select FI+ as the best model. Next, we try to provide better starting conditions for the model by setting it an initial feasible solution at the start. The initial solution is the same as used in Ms-GVNS. This trick improves the average running time of FI+ from 46 to 30 seconds on the considered instances. In this configuration, the model is used as well in the next phase.

In the second experimental phase, we consider all $6 \cdot 20 = 120$ instances of sizes ranging from 6 to 51. Here, the rt upper bound is $t_{max} = 600\,s$, i.e., 10 minutes. For each configuration consisting of a problem instance and a solver/method, $n_{run} = 100$ runs is performed, except of FI+ and 50-customer instances, where $n_{run} = 30$. Overall, the experiments consist of the total of $3 \cdot 120 \cdot 100 - 20 \cdot 70 = 34600$ independent runs, and are summarized in Tab. I.

In the Table, each row corresponds to n_{run} runs of the method shown in the second column on 20 instances of the size shown in the first column. The meta-heuristics are capable of finding the optimum within 10 minutes every time. The ILP model, on the other hand, either finds the optimum before the time limit in every run on a particular instance or in zero runs (i. e., all runs are unsuccessful). The latter case happens for four instances of size 26 and 18 instances of size 51, as it is shown in the third column of Tab. I. Columns 4-8 are all averaged over all successful 20 instances of the particular size. The averaged minimal, mean, and maximal value of running time rt in milli-seconds are shown in columns 4, 5, and 7, respectively. A percentage

Size	Method	Fail	Averaged over 20 instances				
			Time-To-Target (TTT) / running time rt				Probability [%]
			min [ms]	mean [ms]	sd [%]	max [ms]	
$5+1$	FI+	0	3.243	3.498	5.3	4.353	0.0
$5+1$	Ms-GVNS	0	0.005	0.006	39.4	0.022	17.9
$5+1$	GILS-RVND	0	0.004	0.006	35.3	0.020	–
$10+1$	FI+	0	93.466	94.837	1.2	100.629	0.0
$10+1$	Ms-GVNS	0	0.006	0.010	57.5	0.047	60.2
$10+1$	GILS-RVND	0	0.006	0.012	61.0	0.049	–
$15+1$	FI+	0	1284.897	1304.477	0.8	1346.302	0.0
$15+1$	Ms-GVNS	0	0.008	0.037	89.4	0.174	67.8
$15+1$	GILS-RVND	0	0.011	0.060	85.3	0.275	–
$20+1$	FI+	0	18978.081	19230.071	0.6	19492.618	0.0
$20+1$	Ms-GVNS	0	0.015	0.138	92.7	0.706	69.0
$20+1$	GILS-RVND	0	0.021	0.178	87.2	0.802	–
$25+1$	FI+	4	85327.237	85873.465	0.8	87686.563	0.0
$25+1$	Ms-GVNS	0	0.025	0.327	88.4	1.587	67.9
$25+1$	GILS-RVND	0	0.041	0.456	93.2	2.304	–
$50+1$	FI+	18	113293.538	113798.582	0.3	114328.255	0.0
$50+1$	Ms-GVNS	0	0.435	7.226	95.8	35.890	62.0
$50+1$	GILS-RVND	0	0.673	10.298	85.4	43.431	–

TABLE I: Results of experiments performed on 120 instances of 6 different sizes for ILP model FI+, custom meta-heuristic Ms-GVNS, and reference meta-heuristic GILS-RVND.

standard deviation of rt relative to the mean value is shown in the 6th column. The averaged probability that the considered method returns optimum in a shorter time then a reference method (as discussed in Sec. V-B) is shown in the last column. Here, the reference method is the reference meta-heuristic GILS-RVND.

As seen on the first sight, running times of the meta-heuristics are significantly lower than those of the ILP model. Actually, even the maximal values of meta-heuristics are better, by several orders, than the minimal values of FI+. These results are very expected for large and medium-size instances. However, it might be surprising that even for the simplest problems, the two approaches' computational times are hardly comparable. The meta-heuristics can find the optimum faster than ILP by at least three orders in 100% experimental cases and still have 100% success rate even for instances where the ILP solver fails to finish in feasible computing time.

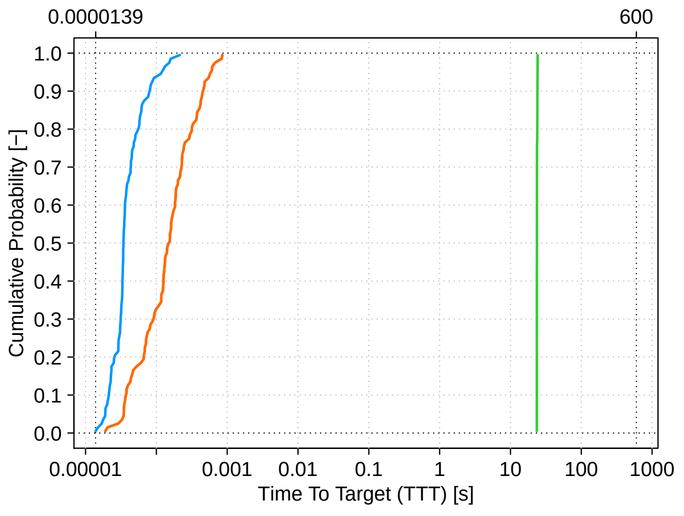
Regarding the probability computed from TTT-plots, for FI+, all the probabilities are zero, and for GILS-RVND, the values are not available since it makes no sense to compare it to itself. However, interesting results can be seen in the case of Ms-GVNS. For instances with the number of customers more or equal to 10, the average probability is always above 60%. In other words, Ms-GVNS is able to find the optimum, on average, faster then GILS-RVND with probability 60% (or more). The exceptions to this observation are the smallest instances (size 6), where GILS-RVND performs much better than Ms-GVNS.

Another thing to notice is the relative standard deviation of running times of the meta-heuristics, which is much higher than for the ILP solver. This observation is the result of the stochastic approach used with the heuristics.

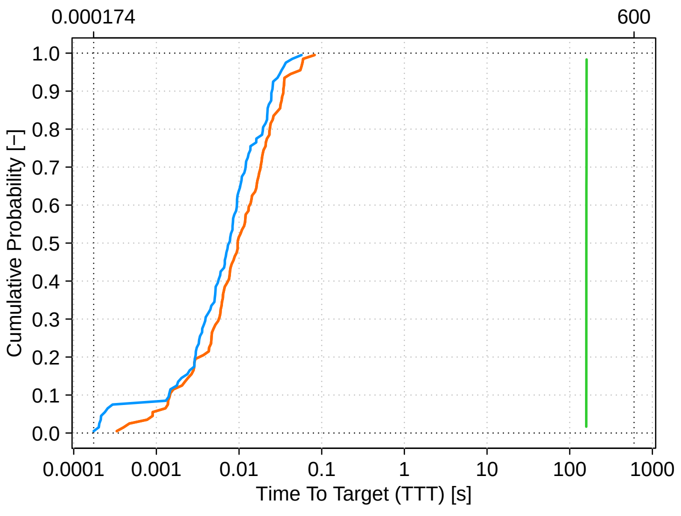
Finally, to fully illustrate the run-time behavior of all considered algorithms, in Fig. 1 we present the full TTT-plots on typical examples of the tested instances.

VI. Conclusion

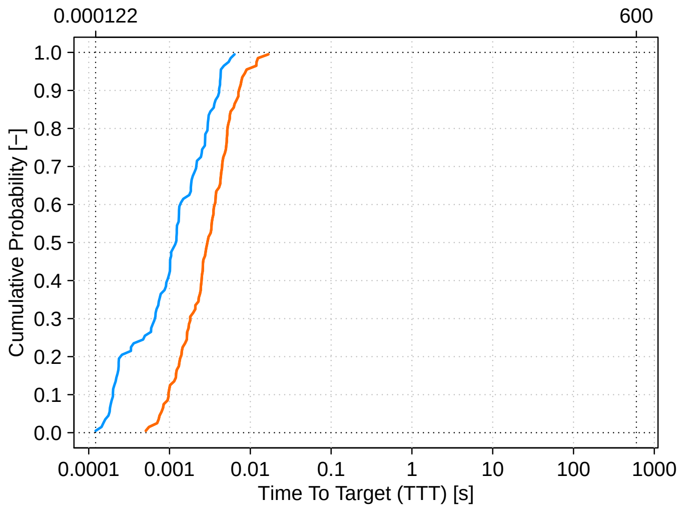
In my experimentally oriented semestral project, I took a different view on a familiar combinatorial optimization problem, the TDP. While previously, I focused on large instances for which I designed a fast meta-heuristic returning good-enough solutions; this time, I pursued optimality on smaller instances. I have shown, that my method is in performance statistically superior to the state-of-the-art mathematical model [4], [11] implemented in commercial ILP solver. Although there are no theoretical guarantees on the quality of the solution provided by the meta-heuristic, there are no solid arguments against using it in practice. My method found the optimum in 100% cases and by several orders faster than the ILP solver. At the same time, it stably provides better performance than the state-of-the-art meta-heuristics GILS-RVND [12]. Among 120 tested instances, I did not manage to find any instance that could be solved by the ILP solver, but the meta-heuristics would struggle to find the optimum. Regarding the



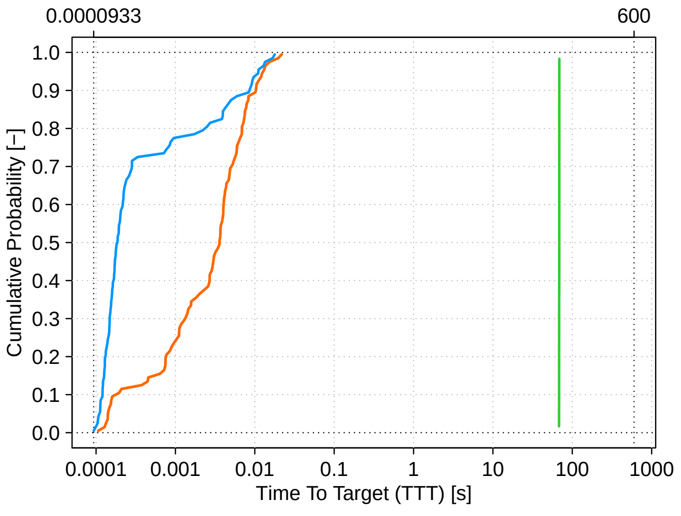
(a) Size 26, inst R8.



(b) Size 51, inst R9.



(c) Size 51, inst R15.



(d) Size 51, inst R19.

Fig. 1: Examples of TTT-plots for FI+ (green), Ms-GVNS (blue), and GILS-RVND (orange) on one instance with 26 customers and three instances with 51 customers.

implementation, the mathematical models are quite straightforward, if correctly formulated. The API provided by Gurobi$^{\text{TM}}$ is well documented and easy to use. The meta-heuristics, on the other hand, are not so easy to implement from scratch. While the basic algorithm may seem simplistic, some of its components like perturbation, VND, or efficient calculation of improvements after applying several operators, can be delicate and problematic.

References

[1] A. Archer and D. P. Williamson, "Faster approximation algorithms for the minimum latency problem," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2003, pp. 88–96.

[2] S. Sahni and T. Gonzalez, "P-Complete Approximation Problems," *Journal of the ACM (JACM)*, vol. 23, no. 3, pp. 555–565, jul 1976. [Online]. Available: http://dl.acm.org/doi/10.1145/321958.321975

[3] W. J. Cook, *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2012. [Online]. Available: http://www.jstor.org/stable/j.ctt7t8kc

[4] M. Fischetti, G. Laporte, and S. Martello, "The Delivery Man Problem and Cumulative Matroids," *Operations Research*, vol. 41, no. 6, pp. 1055–1064, dec 1993. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/opre.41.6.1055

[5] G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela, "On Salesmen, Repairmen, Spiders, and Other Traveling Agents," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2000, pp. 1–16. [Online]. Available: http://link.springer.com/10.1007/3-540-46521-9{_}1

[6] A. M. Campbell, D. Vandenbussche, and W. Hermann, "Routing for Relief Efforts," *Transportation Science*, vol. 42, no. 2, pp. 127–145, may 2008. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/trsc.1070.0209

[7] E. Koutsoupias, C. Papadimitriou, and M. Yannakakis, "Searching a fixed graph," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1996, pp. 280–289. [Online]. Available: http://link.springer.com/10.1007/3-540-61440-0{_}135

[8] A. Sarmiento, R. Murrieta-Cid, and S. Hutchinson, "A multi-robot strategy for rapidly searching a polygonal environment," in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 2004.

[9] M. Kulich, L. Přeučil, and J. J. Miranda-Bront, "Single robot search for a stationary object in an unknown environment," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 5830–5835. [Online]. Available: http://ieeexplore.ieee.org/document/6907716/

[10] M. Kulich, J. J. Miranda-Bront, and L. Přeučil, "A meta-heuristic based goal-selection strategy for mobile robot search in an unknown environment," *Computers & Operations Research*, vol. 84, pp. 178–187, aug 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0305054816300995

[11] L. M. Naeni and A. Salehipour, "A New Mathematical Model for the Traveling Repairman Problem," in *2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, dec 2019, pp. 1384–1387. [Online]. Available: https://ieeexplore.ieee.org/document/8978898/

[12] M. M. Silva, A. Subramanian, T. Vidal, and L. S. Ochi, "A simple and effective metaheuristic for the Minimum Latency Problem," *European Journal of Operational Research*, vol. 221, no. 3, pp. 513–520, sep 2012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S037722171200269X

[13] A. Salehipour, K. Sörensen, P. Goos, and O. Bräysy, "Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem," *4OR*, 2011.

[14] A. Lucena, "Time-dependent traveling salesman problem the deliveryman case," *Networks*, vol. 20, no. 6, pp. 753–763, 1990. [Online]. Available: http://doi.wiley.com/10.1002/net.3230200605

[15] L. Bianco, A. Mingozzi, and S. Ricciardelli, "The traveling salesman problem with cumulative costs," *Networks*, vol. 23, no. 2, pp. 81–91, mar 1993. [Online]. Available: http://doi.wiley.com/10.1002/net.3230230202

[16] C. A. van Eijl, "A polyhedral approach to the delivery man problem," Technische Universiteit Eindhoven (Memorandum COSOR), Eindhoven, Tech. Rep. 1995, 1995. [Online]. Available: https://research.tue.nl/en/publications/a-polyhedral-approach-to-the-delivery-man-problem

[17] I. Méndez-Díaz, P. Zabala, and A. Lucena, "A new formulation for the Traveling Deliveryman Problem," *Discrete Applied Mathematics*, vol. 156, no. 17, pp. 3223–3237, oct 2008. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0166218X08002163

[18] I. O. E. Ezzine, H. Chabchoub, and F. Semet, "New formulations for the traveling repairman problem," in *Proceedings of the 8-th International Conference of Modeling and Simulation*. Hammamet, Tunisia: MOSIM, 2010.

[19] H.-B. Ban, K. Nguyen, M.-C. Ngo, and D.-N. Nguyen, "An efficient exact algorithm for the Minimum Latency Problem," *Progress in Informatics*, no. 10, p. 167, mar 2013. [Online]. Available: http://www.nii.ac.jp/pi/n10/10{_}167.html

[20] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, "The minimum latency problem," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing - STOC '94*. New York, New York, USA: ACM Press, 1994, pp. 163–171. [Online]. Available: http://portal.acm.org/citation.cfm?doid=195058.195125

[21] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, "Paths, trees, and minimum latency tours," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* IEEE Computer. Soc, 2003, pp. 36–45. [Online]. Available: http://ieeexplore.ieee.org/document/1238179/

[22] A. Archer, A. Levin, and D. P. Williamson, "A Faster, Better Approximation Algorithm for the Minimum Latency Problem," *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1472–1498, jan 2008. [Online]. Available: http://epubs.siam.org/doi/10.1137/07068151X

[23] A. Archer and A. Blasiak, "Improved Approximation Algorithms for the Minimum Latency Problem via Prize-Collecting Strolls," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA: Society for Industrial and Applied Mathematics, jan 2010, pp. 429–447. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611973075.36

[24] G. N. Frederickson and B. Wittman, "Approximation Algorithms for the Traveling Repairman and Speeding Deliveryman Problems," *Algorithmica*, vol. 62, no. 3-4, pp. 1198–1221, apr 2012. [Online]. Available: http://link.springer.com/10.1007/s00453-011-9515-4

[25] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, nov 1997. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0305054897000312

[26] M. G. C. Resende and C. C. Ribeiro, *Optimization by GRASP*. New York, NY: Springer New York, 2016. [Online]. Available: http://link.springer.com/10.1007/978-1-4939-6530-4

[27] A. Mjirda, R. Todosijević, S. Hanafi, P. Hansen, and N. Mladenović, "Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem," *International Transactions in Operational Research*, 2017.

[28] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated Local Search: Framework and Applications," 2010, pp. 363–397. [Online]. Available: http://link.springer.com/10.1007/978-1-4419-1665-5{_}12

[29] D. Satyananda and S. Wahyuningsih, "Sequential order vs random order in operators of variable neighborhood descent method," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 17, no. 2, pp. 801–808, 2019.

[30] G. Reinelt, "TSPLIB - A Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, nov 1991. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/ijoc.3.4.376

[31] N. Mladenović, D. Urošević, and S. Hanafi, "Variable neighborhood search for the travelling deliveryman problem," *4OR*, vol. 11, no. 1, pp. 57–73, 2013.

[32] F. Glover, "Tabu Search - Part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, aug 1989. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/ijoc.2.1.4http://pubsonline.informs.org/doi/abs/10.1287/ijoc.1.3.190

[33] J. P. Hart and A. W. Shogan, "Semi-greedy heuristics: An empirical study," *Operations Research Letters*, vol. 6, no. 3, pp. 107–114, jul 1987. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0167637787900216

[34] T. A. Feo and M. G. C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol. 8, no. 2, pp. 67–71, apr 1989. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0167637789900023

[35] T. A. Feo, M. G. C. Resende, and S. H. Smith, "A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set," *Operations Research*, vol. 42, no. 5, pp. 860–878, oct 1994. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/opre.42.5.860

[36] C. C. Ribeiro, I. Rosseti, and R. Vallejos, "On the use of run time distributions to evaluate and compare stochastic local search algorithms," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009.

[37] C. C. Ribeiro and I. Rosseti, "tttplots-compare: a perl program to compare time-to-target plots or general runtime distributions of randomized algorithms," *Optimization Letters*, vol. 9, no. 3, pp. 601–614, mar 2015. [Online]. Available: http://link.springer.com/10.1007/s11590-014-0760-8