A Large Neighborhood Search for a Cooperative Optimization Approach to Distribute Service Points in Mobility Applications*

Thomas Jatschka[1], Tobias Rodemann[2], and Günther R. Raidl[1]

[1] Institute of Logic and Computation, TU Wien, Austria
{tjatschk,raidl}@ac.tuwien.ac.at
[2] Honda Research Institute Europe, Germany
tobias.rodemann@honda-ri.de

Abstract. We present a large neighborhood search (LNS) as optimization core for a *cooperative optimization approach* (COA) to optimize locations of service points for mobility applications. COA is an iterative interactive algorithm in which potential customers can express preferences during the optimization. A machine learning component processes the feedback obtained from the customers. The learned information is then used in an optimization component to generate an optimized solution. The LNS replaces a mixed integer linear program (MILP) that has been used as optimization core so far. A particular challenge for developing the LNS is that a fast way for evaluating the non-trivial objective function for candidate solutions is needed. To this end, we propose an *evaluation graph*, making an efficient incremental calculation of the objective value of a modified solution possible. We evaluate the LNS on artificial instances as well as instances derived from real-world data and compare its performance to the previously developed MILP. Results show that the LNS as optimization core scales significantly better to larger instances while still being able to obtain solutions close to optimality.

1 Introduction

The traditional approach for solving service point placement problems, such as distributing charging stations for electric vehicles or vehicle sharing stations in a geographic area, essentially is to first estimate the demand that may be fulfilled at potential locations and then to select actual locations either manually or by some computational optimization. However, estimating the customer demand that may be fulfilled by certain stations is an intricate task in which erroneous assumptions may result in heavy economic losses for the service point provider. Also, estimating demand upfront requires specific data which can be challenging and/or expensive to collect. As an alternative approach, in [1] we introduced a *cooperative optimization approach* (COA) for optimizing the locations of service points in mobility applications. In contrast to the traditional approach, COA is an iterative interactive algorithm that solves the demand data acquisition and optimization in a single process by allowing customers to express their preferences intertwined with the optimization. A machine learning component processes the feedback obtained from the customers and provides a surrogate objective function. This surrogate objective is then used in an optimization component to generate an optimized solution. This solution is then a basis for further interaction with the users to obtain more relevant knowledge, and the whole process is repeated until some stopping criterion is met. So far, COA uses a mixed integer linear program (MILP) in the optimization core for determining solutions [2] or, in a former version [3], basic metaheuristic approaches that treated the problem as black box model and hence do not make significant use of structural properties of the problem. For an exact optimization core, the generated solutions are optimal w.r.t. to the so far known information derived from the customer feedback. However, this optimality comes at the cost of large computation times, especially for large-scale instances with thousands of customers and hundreds of potential service point locations. In contrast, a heuristic optimization core may feature better scalability towards larger instances. To this end we present here a large neighborhood search (LNS) that can reduce computation times by orders of magnitudes with only small

* Thomas Jatschka acknowledges the financial support from Honda Research Institute Europe.

losses in final solution quality. Due to the nature of the non-trivial objective function of our service point distribution problem, an efficient way for evaluating said objective is necessary to make this speedup possible. Therefore, our LNS features a data structure, referred to as *evaluation graph* for modeling the evaluation of solutions. We show how the evaluation graph can be used to efficiently keep track of small changes in the solution, such as opening or closing a service point. Based on this evaluation graph, the LNS is able to quickly repair partially destroyed solutions in a promising heuristic way. We evaluate the LNS on artificial instances as well as instances derived from real-world data and compare its performance to the previously developed MILP-based approach.

In the next section we review related work. Section 3 formally defines the General Service Point Distribution Problem (GSPDP), as it is referred to, while an overview on the COA framework is given in Section 4. Our main contribution, the LNS with its evaluation graph, is presented in Section 5. Section 6 explains the benchmark scenarios, and Section 7 discusses experimental results. Finally, Section 8 concludes this article and gives an outlook on future work.

2 Related Work

The basic concept of COA was presented in [1]. In interactive optimization algorithms, such as COA, humans are used to (partially) evaluate the quality of solutions and to guide the optimization process. For a survey on interactive optimization, see [4]. Interactive algorithms are often combined with surrogate-based approaches [5, 6], in which a machine learning model is trained to evaluate intermediate solutions approximately in order to reduce user interactions and to avoid user fatigue [7]. In contrast to COA, most approaches from literature only allow a single user to interact with the algorithm, e.g., [8, 9]. Hence, in [10] COA's surrogate function is based on a matrix factorization model [11], a popular collaborative filtering technique [12] in which unknown ratings of items are derived from users with similar preferences.

In [3] two heuristic black box optimization approaches were suggested for COA to generate new candidate solutions w.r.t. to the current surrogate model: a variable neighborhood search as well as a population-based iterated greedy approach. In [2] COA was substantially extended to also be applicable in use cases where the satisfaction of demands relies on the existence of two or more suitably located service stations, such as car and bike sharing systems.

More generally, there exists a vast amount of literature regarding the location planning of service points for mobility applications, see, e.g., [13] for electric vehicle charging stations or [14] for stations of a bike sharing system. However, to the best of our knowledge no further work on interactive optimization approaches for location planning in mobility applications exists.

3 The General Service Point Distribution Problem

In this section we give a formal description of the *Generalized Service Point Distribution Problem* (GSPDP) introduced in [2], which is the problem to be solved at the core of COA and for which we will then propose the LNS. Service points may be set up at a subset of locations $V = \{1, \dots, n\}$. Establishing a service point at a location $v \in V$ is associated with costs $z_v^{\text{fix}} \geq 0$ and the total setup costs of all stations must not exceed a maximum budget $B > 0$. Additionally, the expected costs for maintaining this service point over a defined time are $z_v^{\text{var}} \geq 0$. Given a set of users U, each user $u \in U$ has a certain set of *use cases* C_u, such as going to work, visiting a recreational facility, or going shopping.

Each user's use case $c \in C_u$ is associated with a demand $D_{u,c} > 0$ expressing how often the use case is expected to happen within some defined time period. The demand of each use case may possibly be satisfied by subsets of service points to different degrees, depending on the concrete application and the customer's preferences. Hence, we associate each use case c of a user u with a set of *Service Point Requirements* (SPR) $R_{u,c}$ with which a user can express the dependency on multiple service points to fulfill the needs of the use case. For example, for the use case of visiting a fitness center using a bike sharing system, one SPR may represent the need of a rental station close to home or work and a second SPR a rental station close to some fitness center. We denote the set of all different SPRs over all use cases of a user u by $R_u = \bigcup_{c \in C_u} R_{u,c}$. Moreover, let $R = \bigcup_{u \in U} R_u$ be the set of all SPRs over all users.

For now, let us further assume we know values $w_{r,v} \in [0, 1]$ indicating the suitability of a service point at location $v \in V$ to satisfy the needs of user $u \in U$ concerning SPR $r \in R_{u,c}$ in the use case

$c \in C_u$. A value of $w_{r,v} = 1$ represents perfect suitability while a value of zero means that location v is unsuitable; values in between indicate partial suitability. For each unit of satisfied customer demand a prize $q > 0$ is earned.

A solution to the GSPDP is a subset of locations $X \subseteq V$ indicating where service points are to be set up. It is feasible if its total fixed costs do not exceed the maximum budget B, i.e.,

$$z^{\text{fix}}(X) = \sum_{v \in X} z_v^{\text{fix}} \leq B. \tag{1}$$

The objective function of the GSPDP is to maximize

$$f(X) = q \cdot \sum_{u \in U} \sum_{c \in C_u} D_{u,c} \cdot \min_{r \in R_{u,c}} \left(\max_{v \in X} w_{r,v} \right) - \sum_{v \in X} z_v^{\text{var}}. \tag{2}$$

In the first term, the obtained prize for the expected total satisfied demand is determined by considering for each user u, each use case c, and each SPR r a most suitable location $v \in V$ at which a service point is to be opened. Over all SPRs of a use case, the minimum of the obtained suitability values is taken. The second term of the objective function represents the total maintenance costs for the service stations. In [2] we have shown that the GSPDP is NP-hard.

By linearizing the objective function, the GSPDP can be modeled by the following MILP.

$$\max \quad q \cdot \sum_{u \in U} \sum_{c \in C_u} D_{u,c}\, y_{u,c} - \sum_{v \in V} z_v^{\text{var}}\, x_v \tag{3}$$

$$\sum_{v \in V} o_{r,v} \leq 1 \qquad \forall r \in R \tag{4}$$

$$o_{r,v} \leq x_v \qquad \forall v \in V,\ r \in R \tag{5}$$

$$y_{u,c} \leq \sum_{v \in V} w_{r,v} \cdot o_{r,v} \qquad \forall u \in U,\ c \in C_u,\ r \in R_{u,c} \tag{6}$$

$$\sum_{v \in V} z_v^{\text{fix}}\, x_v \leq B \tag{7}$$

$$x_v \in \{0, 1\} \qquad \forall v \in V \tag{8}$$

$$0 \leq y_{u,c} \leq 1 \qquad \forall u \in U,\ c \in C_u \tag{9}$$

$$0 \leq o_{r,v} \leq 1 \qquad \forall r \in R,\ v \in V \tag{10}$$

Binary variables x_v indicate whether or not a service point is deployed at location $v \in V$. Continuous variables $o_{r,v}$ are used to indicate the actually used location $v \in V$ for each SPR $r \in R$; these variables will automatically become integer. The degree to which a use case $c \in C_u$ of a user $u \in U$ can be satisfied is expressed by continuous variables $y_{u,c}$. The objective value is calculated in (3). Inequalities (4) ensure that at most one location with the highest suitability value is selected for each SPR. Inequalities (5) and (6) ensure that use cases are only satisfied if there are suitable locations with opened service points for each SPR of the respective use case. Inequalities (6) additionally determine the degree to which a use case is satisfied. Last but not least, Inequality (7) ensures that the budget is not exceeded.

4 Cooperative Optimization Algorithm

A crucial aspect of COA's general approach is that the suitability values $w_{r,v}$ are not explicitly known a priori. A complete direct questioning would not only be extremely time consuming but users would easily be overwhelmed by the large number of possibilities, resulting in incorrect information. For example, users easily tend to only rate their preferred options as suitable and might not consider certain alternatives as also feasible although they actually might be on second thought when no other options are available.

Hence, interaction with users needs to be kept to a minimum and should be done wisely to extract as much meaningful information as possible. Therefore, COA does not ask a user to directly provide best suited station locations for the SPRs but creates meaningful *location scenarios*, i.e., subsets of locations, and asks the users to evaluate these. More specifically, a user u returns as

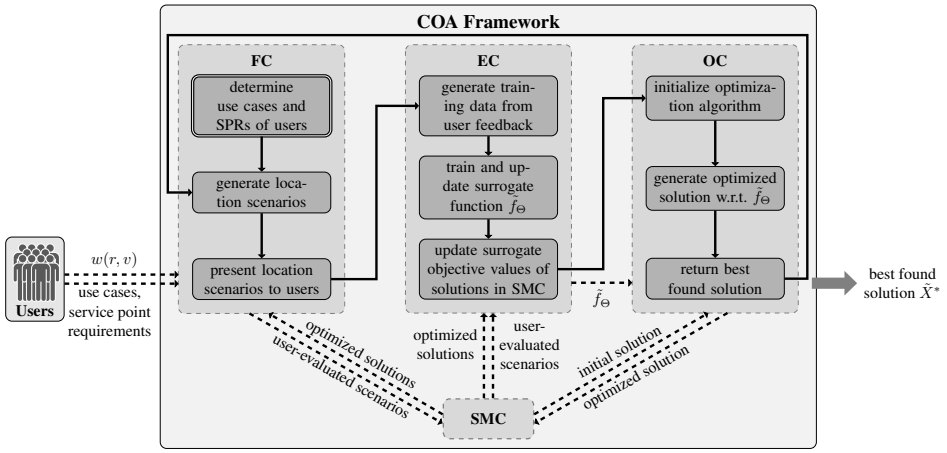

Fig. 1: Components of COA and their interaction.

evaluation of a location scenario S w.r.t. one of the user's SPRs $r \in R_u$ a best suited location $v_{r,S} \in S$ and the corresponding suitability value $w(r, v_{r,S}) > 0$ or the information that none of the locations of the scenario S is suitable. We assume here that the suitability of a location w.r.t. an SPR can be specified on a five valued scale.

The COA framework consists of a *Feedback Component (FC)*, an *Evaluation Component (EC)*, an *Optimization Component (OC)*, and a *Solution Management Component (SMC)*. Figure 1 illustrates the fundamental principle and communication between these components. During an initialization phase, the FC first asks each user $u \in U$ to specify her or his use cases C_u with their associated SPRs $R_{u,c}$, as well as corresponding demands $D_{u,c}$, $c \in C_u$. Then, the FC is responsible for generating individual location scenarios for each user which are presented to the user in order to obtain her/his feedback.

The obtained feedback is processed in the EC. A crucial assumption we exploit is that in a large user base some users typically have similar preferences about the locations of service points w.r.t. to some of their use cases. Hence, by identifying these similarities and learning from them, the EC maintains and continuously updates a *surrogate suitability function* $\tilde{w}_\Theta(r, v)$ approximating the real and partially unknown suitability values $w_{r,v}$ of service point locations $v \in V$ w.r.t. SPR $r \in R$ without interacting with the respective user. Based on this surrogate function, the EC also provides the surrogate objective function

$$\tilde{f}_\Theta(X) = q \cdot \sum_{u \in U} \sum_{c \in C_u} D_{u,c} \cdot \min_{r \in R_{u,c}} \left(\max_{v \in X} \tilde{w}_\Theta(r, v) \right) - \sum_{v \in X} z_v^{\text{var}} \tag{11}$$

with which a candidate solution X can be approximately evaluated.

A call of the OC is supposed to determine an optimal or close-to-optimal solution to the problem with respect to the EC's current surrogate objective function $\tilde{f}_\Theta$. In [2] this is achieved by solving the MILP (3)–(10) in which the suitability values are approximated by the surrogate suitability function $\tilde{w}_\Theta$.

The SMC stores and manages information on all generated solutions as well as suitability values obtained by the FC.

The whole process is repeated until some termination criterion is reached. In the end, COA returns a solution $\tilde{X}^*$ with the highest surrogate objective value of all of the so far generated solutions. For more details, in particular on how meaningful solution scenarios are derived in the FC and how a matrix factorization is utilized to determine the approximated values $\tilde{w}_\Theta(r, v)$ in the EC, we refer the interested reader to [2].

5 Large Neighborhood Search

We now propose a large neighborhood search (LNS) as a faster replacement for the original MILP-based optimization core in COA. The LNS follows the classical scheme from [15]. The key idea

of LNS is to not search neighborhoods in a naive enumerative way but instead to identify via some problem-specific more effective procedure either best or promising solutions within larger neighborhoods. To this end, LNS frequently follows an iterative destroy and repair scheme: First, a given solution is partially destroyed, typically by freeing a subset of the decision variables and fixing the others to their current values. Afterwards this partial solution is repaired by finding best or at least promising values for the freed variables. If the obtained solution is better than the previous one, it is accepted, otherwise the previous solution is kept.

In our LNS a solution to a GSPDP instance is destroyed in a uniform random fashion by adding k^{dest} new locations to the solution, where k^{dest} is a parameter that is varied.

To repair a solution X, we make use of a randomized greedy approach: Let $\Delta(v, X)$ denote by how much the objective value of a solution X would decrease when removing location v from X. Note that, it is discussed later how $\Delta(v, X)$ can be efficiently calculated for all $v \in X$. In each iteration we first generate a restricted candidate list of k^{rep} locations $v \in V$ for which $\Delta(v, X)$ is lowest, i.e., the candidate list contains the locations that have the lowest impact on objective value of X. Hereby, k^{rep} is another strategy parameter. Ties are broken randomly. A location is then chosen uniformly at random from this restricted candidate list and removed from X.

To construct an initial solution in the first iteration of COA, we also make use of the repair heuristic, starting from $X = V$ and then sequentially removing locations from X for which $\Delta(v, X)$ is lowest until the solution becomes feasible, i.e. $k^{\text{rep}} = 1$ for constructing an initial solution. In subsequent iterations of COA, the LNS is warm-started with COA's current best solution $\tilde{X}^*$.

Our LNS makes use of two destroy operators with $k^{\text{dest}} = 10$ and $k^{\text{dest}} = 20$, respectively, and two repair operators with $k^{\text{rep}} = 2$ and $k^{\text{rep}} = 4$, respectively. These settings have shown to yield a robust convergence behavior across the kinds and sizes of instances in our benchmark sets. In each iteration a repair and destroy operator is chosen uniformly at random. Moreover, each LNS run terminates after 40 iterations without improvement.

A crucial aspect for developing an effective heuristic for solving the GSPDP is that computing the surrogate objective value $\tilde{f}_\Theta$ of a solution in a straight-forward way from scratch is time consuming. Hence, in order to accelerate this task we maintain for a GSPDP instance a directed graph $G = (LL \cup SL \cup CL \cup \{l_{\text{obj}}\}, A_{LL} \cup A_{SL} \cup A_{CL})$ referred to as *evaluation graph*. This graph represents the objective function calculation and stores intermediate results for a current solution, allowing for an effective incremental update in case of changes in the solution. The evaluation graph consists of four layers of nodes, which are the location layer (LL), the SPR layer (SL), the use case layer (CL), and the evaluation layer containing a single node l_{obj}. The location layer contains n nodes corresponding to the locations in V, i.e., $LL = \{l_v \mid v \in V\}$. The use case layer consists of one node for each use case C_u of each user $u \in U$, i.e., $CL = \{l_c \mid c \in C_u,\ u \in U\}$, and the SPR layer contains one node for each SPR in $\in R_{u,c}$, for each use case $c \in C_u$ and user $u \in U$, i.e., $SL = \{l_{u,r} \mid r \in R_{u,c},\ c \in C_u, u \in U\}$.

There exists an arc in G from a node of the location layer l_v to a node of the SPR layer $l_{u,r}$ if $\tilde{w}_\Theta(v, r) > 0$, i.e., $A_{LL} = \{(l_v, l_{u,r}) \mid l_v \in LL,\ l_{u,r} \in SL,\ \tilde{w}_\Theta(v, r) > 0\}$. A node of the SPR layer is connected to a node of the use case layer if the corresponding SPR is an SPR of the corresponding use case, i.e., $A_{SL} = \{(l_{u,r}, l_c) \mid l_{u,r} \in SL,\ l_c \in CL,\ r \in R_{u,c}\}$. Finally, each node l_c of the use case layer is connected to l_{obj}, i.e., $A_{CL} = \{(l_c, l_{\text{obj}}) \mid l_c \in CL\}$.

The location layer gets as input a binary vector $(x_v)_{v \in V}$ with $x_v = 1$ if $v \in X$ and $x_v = 0$ otherwise, w.r.t. a solution X. Moreover, each node in G has an activation function $\alpha()$ that decides its output value which is propagated to its successor nodes in the next layer as their input, i.e.,

$$\alpha_{LL}(l_v, X) = \begin{cases} 1 & \text{if } v \in X \\ 0 & \text{otherwise,} \end{cases} \qquad \forall l_v \in LL, \tag{12}$$

$$\alpha_{SL}(l_{u,r}, X) = \max_{(l_v, l_{u,r}) \in A_{LL}} (\alpha_{LL}(l_v, X) \cdot \tilde{w}_\Theta(v, r)) \qquad \forall l_{u,r} \in SL, \tag{13}$$

$$\alpha_{CL}(l_c, X) = \min_{(l_{u,r}, l_c) \in A_{SL}} \alpha_{SL}(l_{u,r}, X) \qquad \forall l_c \in C_L, \tag{14}$$

$$\alpha_{eval}(l_{\text{obj}}, X) = \sum_{(l_c, l_{\text{obj}}) \in A_{CL}} \alpha_{SL}(l_c, X) - \sum_{v \in X} z_v^{\text{var}}. \tag{15}$$

The evaluation graph stores all output of the activation functions from the last evaluated solution and is therefore especially efficient for evaluating subsequent solutions that only differ in a single

location $v \in V$ as not everything needs to be calculated from scratch but just the modified value v w.r.t. the current solution X needs to be propagated. Note that A_{LL} needs to be updated in each iteration of COA as the EC recalculates the surrogate suitability values $\tilde{w}_\Theta$ in each iteration with newly obtained user feedback.

Additionally, the evaluation graph also makes it possible to efficiently keep track of how much each location v contributes to the objective value of a solution. For this purpose, we introduce the following new notations. Let X be a current solution and $c \in C_u$ be a use case of a user $u \in U$ that is satisfied (to some degree) in X, i.e., for each $r \in R_{u,c}$ there exists at least one location $v \in X$ such that $\tilde{w}_\Theta(r, v) > 0$. Let $v^{\max}(r, X)$ refer to a location in the solution for which $\tilde{w}_\Theta(r, v^{\max}(r, X)) = \max_{v \in X} \tilde{w}_\Theta(r, v)$. For the sake of readability we further refer to $\tilde{w}_\Theta(r, v^{\max}(r, X))$ as $\tilde{w}^{\max}_\Theta(r, X)$. Additionally, let $\tilde{w}^{\text{fallback}}_\Theta(r, X)$ denote the second highest suitability value for an SPR r w.r.t. to the locations in X, i.e., $\tilde{w}^{\text{fallback}}_\Theta(r, X) = \max\{\tilde{w}_\Theta(r, v) \mid \ v \in X \setminus \{v^{\max}(r, X)\} \cup \{0\}\}$. Note that $\tilde{w}^{\text{fallback}}_\Theta(r, X)$ is zero if $X \setminus \{v^{\max}(r, X)$ is empty. Finally, let $\tilde{w}^{\min}_\Theta(u, c, X) = \min_{r \in R_{u,c}} \tilde{w}^{\max}_\Theta(r, X)$.

From the definition of the surrogate objective function, it follows that the degree to which a use case c is satisfied in a solution X is only determined by the set of locations $\{v^{\max}(r, X) \mid r \in R_{u,c}\}$. Hence, let $\Delta(u, c, v, X)$ denote by how much the degree to which a use case $c \in C_u$ of a user $u \in U$ is satisfied w.r.t. a solution X would decrease when removing v from X, i.e.,

$$\Delta(u, c, r, X) = \begin{cases} q \cdot D_{u,c} \cdot (\tilde{w}^{\max}_\Theta(r, X) - \tilde{w}^{\text{fallback}}_\Theta(r, X)) & \tilde{w}^{\text{fallback}}_\Theta(r, X) < \tilde{w}^{\min}_\Theta(u, c, X) \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

$$\Delta(u, c, v, X) = \max\{\Delta(u, c, r, X) \mid r \in R_{u,c}, v = v^{\max}(r, X)\} \cup \{0\} \tag{17}$$

Generally speaking, the removal of a location v from a solution X only has an impact on a use case $c \in C_u$ if it results in a change of $\tilde{w}^{\min}_\Theta(u, c, X)$. Additionally, note that the GSPDP also allows cases in which one service point location can be associated to multiple SPRs of the same use case. Such a case would for example correspond to situations in which a customer returns a vehicle at the same station at which the vehicle was picked up. Therefore, the removal of a location from X may affect a use case w.r.t. more than one of its SPRs. However, only the change that affects $\tilde{w}^{\min}_\Theta(u, c, X)$ the most is relevant for calculating by how much the degree to which a use case is satisfied changes.

Hence, the amount $\Delta(v, X)$ by how much the objective value of a solution would decrease when removing location v from X is calculated as

$$\Delta(v, X) = -z^{\text{var}}_v + \sum_{u \in U} \sum_{c \in C_u} \Delta(u, c, v, X). \tag{18}$$

Note that the time required for determining $w^{\max}, w^{\text{fallback}}$, and $w^{\min}$ is negligible if the domain of the rating scale by which users can specify suitability values is small. Moreover, $\Delta(v, X)$ does not need to be calculated from scratch every time a location is added or removed from the solution. Let $X \circ \{v\}$ refer to the modification of a solution, by either adding or removing a location $v \subseteq V$ to/from X. Then $\Delta(v', X \circ \{v\})$ with $v' \in X$ can be determined from $\Delta(v', X)$ as follows:

$$\Delta(v', X \circ \{v\}) = \Delta(v', X) - \sum_{u \in U} \sum_{c \in C_u} \Delta(u, c, v', X) + \Delta(u, c, v', X \circ \{v\}). \tag{19}$$

Additionally, $\Delta(v, X)$ needs to be updated only w.r.t. use cases that are actually affected by the modification of the solution, i.e., only if $\tilde{w}^{\max}_\Theta, \tilde{w}^{\text{fallback}}_\Theta$, or $\tilde{w}^{\min}_\Theta$ of a use case change. Finally, for each use case $c \in C_u$ at most $2 \cdot |R_{u,c}|$ locations need to updated in the worst case.

6 Benchmark Scenarios

Benchmark scenarios for our experiments were generated as described in detail in [2] and are available at `https://www.ac.tuwien.ac.at/research/problem-instances/#spdp`.

The considered test instances are of two groups. One group of instances is inspired by the location planning of car sharing systems and hence referred to as CSS. Locations are randomly generated on a grid in the Euclidean plane. The number of use cases for each user is chosen randomly, but each use case always has two SPRs. To generate suitability values for locations

w.r.t. SPRs, ten *attraction points* are randomly placed on the grid, and each SPR is then associated with a geographic location sampled from a normal distribution centered around a randomly chosen attraction point. The actual suitability value is then calculated via a sigmoid function based on the distance between the SPR's geographic location and the respective service point location and afterwards perturbed by Gaussian noise. Six sets of 30 benchmark instances were generated for CSS, considering different combinations of the number of potential service point locations and the number of users.

The second group of instances is derived from real-world taxi trip data of Manhattan and referred to as MAN. The underlying street network of the instances corresponds to the street network graph of Manhattan provided by the Julia package LightOSM[3]. The Taxi trips have been extracted from the 2016 Yellow Taxi Trip Data[4]. For the generation of the instances all trips within the ten taxi zones with the highest total number of pickups and drop-offs of customers were considered, resulting in a total of approximately two million taxi trips. The set of potential service point locations has been chosen randomly from vertices of the street network that are located in the considered taxi zones. Each use case of a user is associated with two SPRs representing the origin and destination of a trip chosen uniformly at random. Suitability values for locations w.r.t. SPRs are again calculated via a sigmoid function based on the distance between the SPR's geographic location and the respective service point location. The MAN benchmark group also consists of 30 instances in total with each instance having 100 potential service point locations and 2000 users. Additionally, each instance will be evaluated with different budget levels $b\,[\%] \in \{30, 50, 70\}$ such that about b percent of the stations can be expected to be opened.

7 Computational Results

All test runs have been executed on an Intel Xeon E5-2640 v4 2.40GHz machine in single-threaded mode. Gurobi 9.1[5] was used to solve the MILP models in the OC. We compare our COA with the LNS, denoted in the following as COA[LNS], to the COA from [2] that uses the MILP (3)–(10) as optimization core and henceforth denoted as COA[MILP]. Since COA[LNS] always uses the current best solution $\tilde{X}^*$ as initial solution, we also set $\tilde{X}^*$ as starting solution in the MILP solver.

We present the results of COA by providing snapshots at different levels of performed user interactions. In [2] we have argued that at most $I_u^{\mathrm{UB}} = \sum_{r \in R_u} (|\{v \mid w(r,v) > 0\}| + 1)$ interactions per user are required to completely derive all suitability values of user $u \in U$. Let I_u be the number of user interactions of user $u \in U$ performed within COA to generate some solution. Then, $I = 100\% \cdot (\sum_{u \in U} I_u / I_u^{\mathrm{UB}})/m$, refers to the relative average number of performed user interactions relative to I_u^{UB} over all users. Results are presented in an aggregated way at various *interaction levels* ψ by selecting for each instance the COA iteration at which I is largest but does not exceed ψ.

First, we provide some general information about the performance of COA[LNS]. Table 1 shows for each instance group at different interaction levels the average number of performed destroy and repair iterations n_{iter}, the average time in seconds required for finding the best solution $t^*[s]$, and the average total time in seconds until the LNS terminated $t[s]$. We can see that the LNS terminates within 43 to 80 iterations on average and usually terminates within three seconds for the CSS instances and within eight seconds for the MAN instances. While the total number of iterations is relatively low, we later show in Table 2 that the solutions generated by the LNS are almost optimal w.r.t. the presented instances. The number of iterations performed tends to decrease as the number of performed user interactions increases while the total runtime increases in each iteration for the MAN instance but stays almost constant for the CSS instances. The decreasing number of iterations can be explained by the LNS being warm-started with the so far best found solution $\tilde{X}^*$. Moreover, as the number of user interactions increases, COA is able to identify more locations relevant to the SPRs of the use cases of the users, resulting in a higher number of arcs between the nodes in the service point layer and the nodes in the SPR layer of the respective evaluation graph. Therefore, the number of iterations until the LNS converges decreases while the time for performing one iteration increases.

Next, we investigate COA runs in which we apply in each iteration both, the LNS and the MILP, for solving the exact same GSPDP instances w.r.t. $\tilde{w}_\Theta$ as well as the initial solution $\tilde{X}^*$.

[3] `https://github.com/DeloitteDigitalAPAC/LightOSM.jl`

[4] `https://data.cityofnewyork.us/Transportation/2016-Yellow-Taxi-Trip-Data/k67s-dv2t`

[5] `https://www.gurobi.com/`

Table 1: Results of COA[LNS].

	CSS																	
(n,m)	(100, 500)			(100, 1000)			(200, 1000)			(200, 2000)			(300, 1500)			(300, 3000)		
ψ	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$
40	50	0.21	0.87	62	0.96	2.08	60	0.21	0.61	76	1.37	2.31	59	0.35	0.66	75	1.32	1.99
50	51	0.27	1.09	67	1.18	2.82	67	0.48	1.05	68	1.03	2.42	65	0.50	0.94	71	1.32	2.34
60	46	0.22	1.17	58	1.09	3.09	58	0.41	1.17	59	1.01	2.74	66	0.61	1.19	65	1.47	2.96
70	47	0.25	1.39	53	0.79	3.09	50	0.30	1.27	58	1.18	3.07	64	0.56	1.22	64	1.45	3.27
80	45	0.18	1.51	48	0.44	2.78	45	0.16	1.16	50	0.59	2.80	56	0.47	1.33	59	1.14	2.98
90	43	0.10	1.48	44	0.25	2.64	45	0.17	1.19	49	0.60	2.73	46	0.22	1.17	44	0.43	2.51

	MAN								
b	30%			50%			70%		
ψ	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$	n_{iter}	$t^*[s]$	$t[s]$
40	78	2.19	3.85	74	1.58	3.35	59	0.65	1.76
50	80	3.70	6.12	75	2.76	5.25	55	1.10	3.30
60	78	4.22	8.20	72	3.72	7.21	63	2.00	5.02
70	65	3.40	8.18	64	3.10	7.74	54	1.51	5.62
80	55	2.62	7.65	58	2.93	8.12	54	1.93	6.74
90	49	1.40	7.24	48	1.27	7.35	46	0.73	6.09

The MILP solver is able to find optimal solution in all cases, but at the expense of typically much longer running times. Note however that only the solution generated by the LNS is further used for the next iteration in COA. Table 2 shows the average percentage gaps between the objective values of the best solutions found by the LNS and respective optimal solutions w.r.t. $\tilde{f}_\Theta$, denoted by $\text{gap}_{\tilde{f}_\Theta}[\%]$, the average total running times in seconds of the LNS $t[s]$, the average times $t^\circ_{\text{M}}[s]$ needed by the MILP solver required for reaching a solution with at most the same objective value as the solution obtained by the LNS, as well as the average total times $t_{\text{M}}[s]$ in seconds of the MILP solver for determining a proven optimal solution. Bold values indicate best times w.r.t. t, t°_{M}, and t_{M}. First, we can see that the solutions generated by the LNS are on average only about 1% worse than an optimal solution for most instance groups. Next, the table shows that for CSS instances with a n/m ratio of $1/10$, the MILP solver needs significantly more time for finding good solutions. Note that these instances have been designed in such a way that users behave less similar resulting in more complex instances. Nonetheless, the LNS significantly outperforms the MILP w.r.t. all instance groups. For all instance groups the LNS requires significantly less time on average to terminate than the MILP needs to reach a solution of the same quality as the solution obtained by the LNS. Additionally, Table 2 especially highlights how much more time the MILP requires for improving a solution at the same quality as the best found LNS solution to a provable optimal solution. Moreover, further tests have shown that most of the time the LNS is able to identify its best found solution while the MILP solver has still not yet solved the root relaxation in the same amount of time.

Finally, we want to compare independent COA[MILP] and COA[LNS] runs, and thus the impact of the in general slightly worse intermediate solutions of the LNS on the overall results of the two COA variants. For this purpose Table 3 shows for each interaction level the average optimality gaps between the best found solution during the optimization to an optimal solution w.r.t. the original objective f for COA[LNS] ($\text{gap}_{\text{L}}[\%]$) as well as COA[MILP] ($\text{gap}_{\text{M}}[\%]$). The table shows that small differences in the solution quality w.r.t. $\tilde{f}_\Theta$ translate to slightly larger differences w.r.t. f. With the exception of the MAN instance group with $b[\%] = 30$, the solutions generated by COA[LNS] are usually at most 3% off from the values obtained by COA[MILP]. In most cases, the average differences are around 1% or less. Hence, in general it can be concluded that the LNS substantially outperforms the MILP in terms of computation time while still being able to generate almost optimal solutions.

Table 2: Times required by the LNS, times the MILP solver needed to obtain a solution with at least the same quality as the solution of the LNS, as well as the total time required by the MILP to find a proven optimal solution. Additionally, the optimality gaps between the LNS solutions and respective optimal solutions are also shown.

	CSS											
	(100, 500)				(200, 1000)				(300, 1500)			
ψ	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$
40	**0.87**	4.58	6.50	0.91	**0.61**	2.41	3.90	0.65	**0.66**	2.97	4.01	0.08
50	**1.09**	4.03	7.68	0.90	**1.05**	3.60	5.27	0.27	**0.94**	3.59	4.31	0.10
60	**1.17**	5.50	7.47	0.78	**1.17**	3.32	5.12	0.19	**1.19**	3.67	4.62	0.07
70	**1.39**	6.65	8.10	0.64	**1.27**	3.75	4.81	0.12	**1.22**	3.14	3.74	0.07
80	**1.51**	5.74	7.04	0.44	**1.16**	4.48	5.97	0.08	**1.33**	3.28	4.40	0.04
90	**1.48**	5.48	6.73	0.33	**1.19**	4.11	5.06	0.06	**1.17**	4.58	5.30	0.03

	CSS											
	(100, 1000)				(200, 2000)				(300, 3000)			
ψ	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$
40	**2.08**	21.87	37.42	2.15	**2.31**	32.79	92.15	1.24	**1.99**	26.04	85.61	0.81
50	**2.82**	28.03	50.51	1.97	**2.42**	37.61	90.11	1.07	**2.34**	39.84	101.52	0.57
60	**3.09**	35.60	59.04	1.45	**2.74**	36.47	126.67	0.89	**2.96**	38.34	130.05	0.47
70	**3.09**	42.95	67.34	1.74	**3.07**	40.48	111.96	0.84	**3.27**	43.41	136.93	0.36
80	**2.78**	43.57	69.94	1.83	**2.80**	40.98	120.07	0.90	**2.98**	43.78	137.76	0.37
90	**2.64**	40.09	74.98	1.37	**2.73**	41.33	123.32	0.78	**2.51**	63.56	149.78	0.37

	MAN											
	30%				50%				70%			
ψ	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$	$t[s]$	$t^{\circ}_{\mathrm{M}}[s]$	$t_{\mathrm{M}}[s]$	$\mathrm{gap}_{\tilde{f}_{\Theta}}[\%]$
40	**3.85**	67.21	326.46	2.15	**3.35**	17.24	53.54	0.87	**1.76**	6.36	10.71	0.21
50	**6.12**	80.31	328.53	1.36	**5.25**	16.76	95.43	0.59	**3.30**	10.20	15.29	0.11
60	**8.20**	131.28	368.28	1.19	**7.21**	24.36	89.15	0.43	**5.02**	14.59	21.54	0.07
70	**8.18**	140.34	375.46	1.06	**7.74**	24.86	108.59	0.35	**5.62**	13.22	21.73	0.06
80	**7.65**	160.13	414.39	1.12	**8.12**	27.70	108.01	0.34	**6.74**	18.00	24.43	0.05
90	**7.24**	154.44	411.55	1.29	**7.35**	43.43	102.70	0.27	**6.09**	13.03	17.46	0.03

Table 3: Quality of solutions generated by COA[LNS] and COA[MILP].

	CSS											
(n,m)	(100, 500)		(100, 1000)		(200, 1000)		(200, 2000)		(300, 1500)		(300, 3000)	
ψ	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$
40	3.46	**2.98**	11.92	**9.57**	1.64	**1.21**	4.34	**2.81**	0.54	**0.51**	2.81	**2.36**
50	2.06	**1.50**	7.34	**4.72**	0.81	**0.62**	2.86	**1.90**	0.43	**0.31**	1.87	**1.27**
60	1.62	**0.63**	4.31	**2.29**	0.45	**0.36**	2.21	**1.20**	0.30	**0.20**	1.31	**0.72**
70	1.20	**0.27**	4.11	**1.61**	0.22	**0.12**	1.56	**0.47**	0.19	**0.11**	0.81	**0.28**
80	0.66	**0.18**	2.72	**0.92**	0.15	**0.05**	1.30	**0.18**	0.09	**0.04**	0.58	**0.15**
90	0.43	**0.01**	1.95	**0.08**	0.08	**0.02**	0.95	**0.05**	0.06	**0.01**	0.44	**0.03**

	MAN					
b	30%		50%		70%	
ψ	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$	$\mathrm{gap_L}[\%]$	$\mathrm{gap_M}[\%]$
40	8.61	**3.46**	3.58	**3.46**	**1.32**	3.46
50	5.14	**1.88**	2.19	**1.88**	**0.77**	1.88
60	3.32	**1.14**	1.41	**1.14**	**0.46**	1.14
70	2.53	**0.63**	0.86	**0.63**	**0.25**	0.63
80	2.03	**0.26**	0.54	**0.26**	**0.12**	0.26
90	1.77	**0.10**	0.33	**0.10**	**0.05**	0.10

8 Conclusion and Future Work

We presented a large neighborhood search (LNS) to be used as optimization core in a cooperative optimization approach (COA) for the general service point distribution problem (GSPDP) in

mobility applications. While the LNS follows the traditional destroy and repair principle, a major challenge was to (a) effectively guide the repair heuristic to produce promising new solutions and to (b) efficiently calculate the surrogate objective function for modified solutions in an incremental way. Both was achieved by introducing the evaluation graph, which stores relevant intermediate results allowing efficient updates when stations are added to or removed from the current solution. In particular, the evaluation graph provides an effective way to keep track of how much impact each location in the solution has on its respective objective value. The efficient update possibility also allows to consider a larger amount of locations during the destroy procedure. The performance of the LNS within COA was tested on artificial instances as well as instances derived from real-world data and was compared to the original COA with its MILP-based optimization core. Results show that at the cost of a slight deterioration of usually not more than one percent in the quality of the solutions, the LNS can outperform the MILP w.r.t. to computation times by orders of magnitudes. In future work it seems promising to also consider other metaheuristic approaches, such as an evolutionary algorithm that uses the evaluation graph for efficiently recombining solutions. Moreover, the GSPDP it is still a rather abstract problem formulation, and it would be important to extend it as well as the solving approach to cover further relevant practical aspects such as capacities of stations and time dependencies of users.

References

1. Jatschka, T., Rodemann, T., Raidl, G.R.: A cooperative optimization approach for distributing service points in mobility applications. In Liefooghe, A., Paquete, L., eds.: Evolutionary Computation in Combinatorial Optimization. Volume 11452 of LNCS., Springer (2019) 1–16
2. Jatschka, T., Raidl, G., Rodemann, T.: A general cooperative optimization approach for distributing service points in mobility applications. Technical Report AC-TR-21-006, TU Wien, Vienna, Austria (2021) submitted.
3. Jatschka, T., Rodemann, T., Raidl, G.R.: VNS and PBIG as optimization cores in a cooperative optimization approach for distributing service points. In Moreno-Díaz, R., et al., eds.: Computer Aided Systems Theory – EUROCAST 2019. Volume 12013 of LNCS., Springer (2020) 255–262
4. Meignan, D., Knust, S., Frayret, J.M., Pesant, G., Gaud, N.: A review and taxonomy of interactive optimization methods in operations research. ACM Transactions on Interactive Intelligent Systems **5** (2015) 17:1–17:43
5. Sun, X., Gong, D., Jin, Y., Chen, S.: A new surrogate-assisted interactive genetic algorithm with weighted semisupervised learning. IEEE Transactions on Cybernetics **43** (2013) 685–698
6. Sun, X.Y., Gong, D., Li, S.: Classification and regression-based surrogate model-assisted interactive genetic algorithm with individual's fuzzy fitness. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, ACM (2009) 907–914
7. Llorà, X., Sastry, K., Goldberg, D.E., Gupta, A., Lakshmi, L.: Combating user fatigue in iGAs: Partial ordering, support vector machines, and synthetic fitness. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, ACM (2005) 1363–1370
8. Kim, H.S., Cho, S.B.: Application of interactive genetic algorithm to fashion design. Engineering applications of artificial intelligence **13** (2000) 635–644
9. Dou, R., Zong, C., Nan, G.: Multi-stage interactive genetic algorithm for collaborative product customization. Knowledge-Based Systems **92** (2016) 43–54
10. Jatschka, T., Rodemann, T., R. Raidl, G.: Exploiting similar behavior of users in a cooperative optimization approach for distributing service points in mobility applications. In Nicosia, G., et al., eds.: Machine Learning, Optimization, and Data Science. Volume 11943 of LNCS., Springer (2019) 738–750
11. Bell, R.M., Koren, Y., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42** (2009) 30–37
12. Ekstrand, M.D., Riedl, J.T., Konstan, J.A.: Collaborative filtering recommender systems. Foundations and Trends in Human–Computer Interaction **4** (2011) 81–173
13. Frade, I., Ribeiro, A., Gonçalves, G., Antunes, A.: Optimal location of charging stations for electric vehicles in a neighborhood in Lisbon, Portugal. Transportation Research Record: Journal of the Transportation Research Board **2252** (2011) 91–98
14. Kloimüllner, C., Raidl, G.R.: Hierarchical clustering and multilevel refinement for the bike-sharing station planning problem. In: International Conference on Learning and Intelligent Optimization. Volume 10556 of LNCS., Springer (2017) 150–165
15. Gendreau, M., Potvin, J.Y., et al.: Handbook of Metaheuristics. Volume 3. Springer (2019)