Bi-objective orienteering for personal activity scheduling

Piotr Matl[a], Pamela C. Nolz[b], Ulrike Ritzinger[b], Mario Ruthmair[c], Fabien Tricoire[a]

[a] *University of Vienna, Department of Business Administration, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria*

[b] *Austrian Institute of Technology, Mobility Department - Dynamic Transportation Systems, Giefinggasse 2, 1210 Vienna, Austria*

[c] *University of Vienna, Department of Statistics and Operations Research, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria*

Abstract

We propose and solve a rich, bi-objective extension of the orienteering problem with time windows (OPTW) to model a combined routing and scheduling problem. Our research is motivated by the problem faced by mobile freelancers who have to integrate irregular appointments and tasks into their daily routines. Those people have a number of tasks which they need to perform at various locations (e.g. meetings with different clients), subject to varying time constraints (e.g. opening hours), and with different levels of importance or urgency (e.g. submitting a deliverable versus cleaning the home office). Furthermore, sets of related tasks may be subject to precedence relations and timely dependencies between them. We explicitly consider the trade-off between planning more tasks and enjoying more free time by means of a bi-objective model. The extension of the OPTW and the bi-objective formulation result in the Personal Planning Problem (PPP). In this paper we present a mathematical formulation of the PPP and a metaheuristic based on Large Neighborhood Search (LNS) is developed to generate a set of non-dominated solutions to the problem. We analyze solution quality on real-world-inspired test instances. Exact reference sets based on a linear single-commodity flow model are used as benchmarks. Computational experiments show that the proposed metaheuristic generates near-optimal solution fronts and scales well to larger instances.

Keywords: bi-objective orienteering, large neighborhood search, personal planning

1. Introduction

The present study originates from a research project which aims at investigating the optimal planning of flexible activities and complex trip sequences, and at developing a personal schedule optimizer with built-in routing functionality. The problem to be solved is motivated by the challenges faced, for example, by mobile freelancers with complex routines of multiple projects and different clients. People in this target group have a variety of activities which they may need to perform at various locations (e.g. meetings with different clients, project work at different venues). These activities may be subject to timing constraints (e.g. arranged appointment times, opening hours), and they may have varying priorities or urgency (e.g. submitting a deliverable versus cleaning the home office). Hence, those people need to plan their time wisely to strike a balance between their professional activities and their private leisure time.

If many activities of private and professional nature need to be performed at diverse locations within varying time frames, determining when, where, and in what order to plan them can be overwhelming for a person due to the large number of possibilities. A sophisticated schedule optimizer can help to overcome this complexity by filtering out inefficient options and presenting only a few of the most streamlined plans. If trip sequences and daily schedules are improved a better use of leisure time as well as an increase in the number of performed activities can be achieved. This can lead to a better quality of life and satisfaction by facilitating a better work-life balance. Thus, a personal activity scheduler should offer suggestions for where, when, and in which order activities can efficiently be planned. Four aspects need to be considered: potential locations of an activity (where to perform an activity), potential time windows for an activity (when to perform the activity), the sequence of activities (does another activity have to be performed before or after this activity, and does a time lag have to be respected), and the travel times between locations.

We propose to model this problem with an extension of the well-known orienteering problem with time windows (OPTW) by Kantor and Rosenwein (1992). We consider a graph, with nodes and directed arcs, where the nodes are used to represent the activities and their locations, the profits at the nodes correspond to the relative importance of these activities, and the service times measure how much time the activities require. The arcs in the graph represent the travel time between two nodes depending on the given mode of transport.

Several extensions to the classic OPTW are needed to model additional real-life aspects. First, activities may have several possible locations to choose from. For example, groceries can be bought at various supermarkets, eating out is possible at different restaurants, and packages can be sent from any post office. Moreover, efficient schedules should not be short-sighted and should be based on a planning horizon longer than just one day. As a result, tasks and locations may have multiple time windows during the planning horizon. For instance, a pharmacy may be open only on weekdays, and its opening hours may be split due to a midday break. Similarly, tasks such as having lunch at noon may have their own time windows independent of their potential locations. In addition, sets of tasks may be connected by precedence relations. For example, the subsequent stages of a project (e.g. music composition, practice, and recording), may be done individually at any time, but not in any order. Finally, it may be necessary to respect a certain time delay between related tasks. For example, if the person wishes to exercise three times per week, they may wish to allocate a day of rest between each session. This can be modeled by imposing a minimum time delay constraint between the individual sessions. Similarly, a maximum time delay constraint may be added to ensure that tasks are not too far apart (e.g. buying groceries and bringing them home into the fridge).

The subjective quality of a schedule depends not only on its profit in terms of the tasks planned, but also on the efficiency of their timing and routing. These two aspects are conflicting, and people may also have varying individual preferences with regard to the trade-off between between professional activities and leisure. This means that two conflicting objectives have to be considered when determining optimal schedules: (i) the profit of planned activities and (ii) the amount of leisure time. For these reasons, we solve a bi-objective formulation of the problem, where the aim is to find a set of Pareto-optimal compromise solutions, i.e. solutions for which it is impossible to improve either objective without worsening the other. This allows the decision makers to choose among a set of diverse schedules the one which best matches their preferences.

We call this problem the Personal Planning Problem (PPP). We present a metaheuristic based on Large Neighborhood Search (LNS) in order to generate solutions within very short computation times. We analyze solution quality on real-world-inspired test instances. For this purpose, we also formulate the problem as a mixed-integer linear programming model based on single-commodity flows and solve it with CPLEX embedded in an epsilon-constraint framework. The sets of solutions generated by our LNS are compared to the exact solutions and evaluated with different performance measures for multi-objective optimization.

The remainder of the article is organized as follows. In the next section, a literature review is presented. In Section 3, a detailed mathematical formulation for the PPP is provided. In Section 4, the solution approach is introduced and explained. Different computational experiments are presented in Section 5 to show the effectiveness of our solution approach and discuss the impact of the envisaged objectives. Concluding remarks are provided in Section 6.

2. Literature Review

A growing body of research has been published on the Orienteering Problem (OP) and its variants. A recent review is presented in Vansteenwegen et al. (2011). The OP is related to the traveling salesman problem with profits (TSPP), of which a slightly older review is provided by Feillet et al. (2005). As noted by Vansteenwegen et al. (2011), not much research has been published specifically on the OPTW originally proposed by Kantor and Rosenwein (1992), but the team orienteering problem with time windows (TOPTW) is a generalization that has been given notable attention.

2.1 *Single-objective OPs*

An exact solution method to orienteering problems including the TOPTW is proposed in Boussier et al. (2007). However, due to the problem's difficulty and the instance sizes of real-life applications, most research on the TOPTW has focused on heuristic approaches (Vansteenwegen et al., 2011). Vansteenwegen et al. (2009) propose a fast and deterministic Iterated Local Search (ILS) to solve the TOPTW within a few seconds and introduce a set of benchmark instances. Following this paper, more elaborate algorithms are introduced (in order of publication) in Montemanni and Gambardella (2009) (Ant Colony Optimization, ACO), Montemanni et al. (2011) (enhanced ACO), Labadie et al. (2011) (GRASP+variable neighborhood descent), Labadie et al. (2012) (granular variable neighborhood search), and Lin and Yu (2012) (simulated annealing). Most authors report a set of new best solutions at the time of publication. To the extent of our knowledge, the most recent contribution is that of Hu and Lim (2014), who combine heuristic and exact methods within an iterative three-component heuristic which finds 35 new best solutions and appears to outperform previous methods in terms of average performance.

Some rich extensions to the OP have also been proposed. Tricoire et al. (2010) model the combined problem of route planning and scheduling for field workers and sales representatives to visit both regular and potential new clients. The authors introduce the multi-period OP with multiple time windows (MuPOPTW), and solve it with an exact algorithm embedded within a variable neighborhood search (VNS). The VNS in Tricoire et al. (2010) also produces high quality results on the (T)OPTW benchmark instances. Another rich variant is presented by Souffriau et al. (2013), who introduce the multi-constraint TOP with multiple time windows (MCTOPMTW). This approach considers resource constraints. By doing so, a number of usual constraints are tackled, for instance multiple time windows. Souffriau et al. (2013) are motivated by a tourism application where the different attributes can for instance represent entry costs (to limit total spend) or point-of-interest categories (to accommodate "max-n-type constraints" such as visiting at most n museums).

2.2 *Multi-objective OPs*

Despite interest in orienteering problems in general, multi-objective formulations of the problem have only received attention in the last ten years. This is somewhat unexpected since the OP is characterized by an inherent conflict between the profit collected and the distance traveled. Most researchers solve the problem in a single-objective way, with a hard constraint on a resource (usually time), while some approaches maximize the difference between profit and cost (Feillet et al., 2005).

Jozefowiez et al. (2008) develop a multi-objective evolutionary algorithm for the traveling salesman problem with profits (TSPP). An exact method for the bi-objective TSPP, based on the ϵ-constraint framework, is introduced in Bérubé et al. (2009). Another exact approach, following the two-phase method, is provided by Filippi and Stevanato (2012). In a following article, the same authors provide an approximation method which enumerates a subset of the Pareto set (Filippi and Stevanato (2013)).

A different bi-objective orienteering problem is introduced by Schilde et al. (2009), with applications in the tourism sector. The objectives refer in this case to the different categories of points of interest (e.g. culture, leisure, dining), with each such point offering different degrees of benefits for each category. The aim is to find a variety of tours offering different degrees of focus on each category, but without neglecting too much any particular category. The authors develop an ACO algorithm and a VNS to generate non-dominated fronts of potential solutions. Both procedures are tested on bi-objective OP (BOOP) benchmark instances as well as on real-life instances. An important point of difference between the BOOP presented by Schilde et al. (2009) and the bi-objective PPP is that the objective functions in the PPP are systematically negatively correlated. Also motivated by tourism applications, Rodríguez et al. (2012) present a formulation with multiple objectives, including: minimizing distance traveled, minimizing the cost of scheduled activities, and maximizing the utility gained from those activities. Constraints include visiting hours, lunch and dinner breaks, and preference information on the types of activities. The problem is solved using an interactive tabu search (TS) metaheuristic in which the search is guided by the decision maker's choices. Tricoire (2012) proposes a general algorithmic framework (multi-directional local search, MDLS) for solving multi-objective problems including the BOOP. The non-dominated set is iteratively improved using single-objective improvement algorithms for each objective. The framework is tested on the BOOP instances of Schilde et al. (2009) and improved results are obtained.

Most recently, a generic bi-objective branch-and-bound algorithm is presented by Parragh and Tricoire (2015) and applied specifically to the bi-objective TOPTW. MDLS is used to compute upper bound sets, and column generation for lower bound sets. Exact sets are obtained through branch-and-price.

2.3 Routing problems with multiple time windows

The presence of multiple time windows can complicate the design of optimisation algorithms. In general, time windows have an impact on route duration. In turn, minimizing route duration can be required to ensure feasibility or to reduce a route's contribution to the objective function. In the case of single time windows, route duration can easily be minimized in linear time using forward time slack, as explained by Savelsbergh (1992). However when multiple time windows are involved, there is the extra decision of selecting a time window for each customer. In certain cases, this involves increased time complexity.

In some cases, minimizing route duration is required to assess route feasibility. Tricoire et al. (2010) tackle the case where (i) each customer has multiple time windows, (ii) route duration is limited and (iii) the time window at the depot does not match the limit on route duration. For a given sequence of customers (route), the route duration can then be minimized by selecting optimal times for starting service at each customer, which is done by an algorithm running in $O(m^2)$, where m is the total number of time windows in the tour. Therefore, determining whether a route is feasible or not takes $O(m^2)$. Souffriau et al. (2013) use a different approach to the same problem of determining route feasibility, in the context of solving a TOP with resource constraints: customers with multiple time windows are duplicated, each duplicate being subject to only one of the time windows at that customer. An artificial resource is also created for each customer requiring duplication. This resource can only be consumed once, so it is not possible to visit the same customer multiple times using different time windows. This way, there is no added complexity when checking the time feasibility of a route; however, this requires the handling of resource constraints, so there is still extra complexity to ensure route feasibility, albeit in a different part of the algorithm.

In other cases, route duration has an impact on objective value. Belhaiza et al. (2014) consider such a problem; they use a similar approach as the one from Tricoire et al. (2010), but with a different algorithm for tour duration minimization. Christiansen and Fagerholt (2002) optimize the robustness of routes, measured by the risk of arriving at certain times. In general, optimising the scheduling of a given sequence of visits in a vehicle route is similar to scheduling a given sequence of tasks on a machine: the decision variables are the start of service at each customer / for each task. A recent overview of existing work in scheduling single routes with multiple time windows, as well as scheduling a given sequence of tasks on a given machine, is present in Vidal et al. (2015).

The work presented in this article follows an approach similar to the one by Souffriau et al. (2013): locations with multiple time windows are duplicated and at most one of them may be selected in a given solution.

3. Problem Definition and Mathematical Model

In the PPP described in Section 1 we are given a set of tasks and associated locations where the tasks can be performed. Both tasks and locations have (possibly multiple) associated time windows within the planning horizon. To facilitate discussions in the remainder of the paper we introduce the notion of "potential visits" which are feasible combinations of a task, an associated location, and one of the task's and location's time windows which overlap by at least the task's service time. By appropriately combining the two time windows we end up with a single time window for each potential visit in which the corresponding task can start. Multiple periods within the planning horizon (e.g. days) can be implicitly encoded in multiple time windows associated to a task (leading to replicated potential visits) and therefore do not need to be explicitly considered in the problem defined below.

3.1 Problem Definition

In the following we define the PPP in a formal way: We are given a directed graph $G = (V, A)$ with set $V = \{0, ..., n+1\}$ of potential visits, and arc set $A \subset V \times V$. Visit 0 and $n+1$ correspond to the fixed start and end visit, respectively. Set T denotes the tasks including set $\overline{T} \subseteq T$ of mandatory tasks. Potential visits V are partitioned into subsets $V_1, ..., V_{|T|}$ with all potential visits in V_k corresponding to the same task $k \in T$. Arc $(i, j) \in A$ represents the situation when visit j is performed directly after visit i, and t_{ij} denotes the travel time when going from i to j. Each potential visit $i \in V$ is assigned a profit $p_i \geq 0$, a service time $s_i \geq 0$, and a time window $[e_i, l_i], 0 \leq e_i \leq l_i$. We denote by $\tilde{T}_k \subseteq T \setminus \{k\}$ the set of tasks preceding task k. Further, we are given a minimum time gap $a_{k'k} \geq 0$ and a maximum time gap $b_{k'k} \geq a_{k'k}$ between each preceding task $k' \in \tilde{T}_k$ and task $k \in T$. We refer the reader to Section 1 for the motivation of the two gap bounds. In case there is no minimum and/or maximum gap between two related tasks we simply set the corresponding values to $a_{k'k} = 0$ and/or $b_{k'k} = \infty$, respectively. Let $V_{k'}^{jk} \subseteq V_{k'}$ be the set of visits i for which task $k' \in \tilde{T}_k$, visit $j \in V_k$, $a_{k'k} \leq l_j - e_i - s_i$ and $e_j - l_i - s_i \leq b_{k'k}$ holds, i.e. the set of visits of which exactly one has to be chosen if visit j is performed. We summarize the input data in Table 1.

Table 1: Input data

V	set of potential visits
$0, n+1$	start and end visit
A	set of arcs
t_{ij}	travel time on arc $(i,j) \in A$
p_j	profit for visit $j \in V$
s_j	service time for visit $j \in V$
$[e_j, l_j]$	time window for starting visit $j \in V$
T	set of tasks
V_k	set of visits corresponding to the same task $k \in T$
$\overline{T} \subseteq T$	set of mandatory tasks
$\tilde{T}_k \subseteq T \setminus \{k\}$	set of predecessor tasks which must be performed before task k
$a_{k'k}, b_{k'k}$	minimum and maximum gap between task $k' \in \tilde{T}_k$ and task k
$V_{k'}^{jk} \subseteq V_{k'}, k' \in \tilde{T}_k$	set of visits of which exactly one has to be chosen if visit $j \in V_k$ is performed

A feasible solution for the PPP is a sequence of visits which starts with 0, performs a subset of potential visits V, and ends with $n + 1$. Without loss of generality, we assume that visit 0 starts at time 0 and visit $n + 1$ starts at time l_{n+1} (to maximize the free time in-between) which denotes the end of the planning horizon. For each mandatory task $k \in \overline{T}$ exactly one visit in V_k and for each optional task $k \in T \setminus \overline{T}$ at most one visit in V_k has to be performed. If a visit in V_k starts at time t then for each preceding task $k' \in \tilde{T}_k$ exactly one visit in $V_{k'}$ has to be performed before and needs to be finished within the given minimum and maximum time gaps $a_{k'k}$ and $b_{k'k}$, respectively, i.e. within the time interval $[t - b_{k'k}, t - a_{k'k}]$. Due to time windows and minimum gaps between tasks with precedence relations, free time might occur between consecutively performed visits. Without loss of generality, we assume that free time is always consumed directly after finishing a visit before traveling to the next visit.

We consider two conflicting objective functions: f_1 maximizes the total profit of all performed visits, while f_2 maximizes the total free time not consumed by service or travel time.

3.2 Mathematical Model

We use the following variables in our mathematical model:

- $x_{ij} \in \{0, 1\}, \forall (i, j) \in A$, denote binary decision variables for performing visit j directly after i.
- $y_j \in \{0, 1\}, \forall j \in V$, denote binary decision variables for performing visit j.
- $w_j \geq 0, \forall j \in V \setminus \{n + 1\}$, denote continuous variables for the free time after visit j.
- $z_{ij} \geq 0, \forall (i, j) \in A$, denote continuous variables for the start time of visit j when coming directly from visit i.
- $z_j \geq 0, \forall j \in V$, denote continuous variables for the start time of visit j.

Our model for the PPP is defined as follows:

$$f_1 = \max \sum_{j \in V} p_j y_j \tag{1}$$

$$f_2 = \max \sum_{j \in V} w_j \tag{2}$$

$$\text{subject to} \quad \sum_{j \in V_k} y_j = 1 \qquad \forall k \in \overline{T} \tag{3}$$

$$\sum_{j \in V_k} y_j \leq 1 \qquad \forall k \in T \setminus \overline{T} \tag{4}$$

$$\sum_{(0,j) \in A} x_{0j} = y_0 = 1 \tag{5}$$

$$\sum_{(j,n+1) \in A} x_{j,n+1} = y_{n+1} = 1 \tag{6}$$

$$\sum_{(i,j) \in A} x_{ij} = \sum_{(j,i) \in A} x_{ji} = y_j \qquad \forall j \in V \setminus \{0, n+1\} \tag{7}$$

$$\sum_{(i,j) \in A} z_{ij} = z_j \qquad \forall j \in V \setminus \{0\} \tag{8}$$

$$z_j + s_j y_j + w_j + \sum_{(j,h) \in A} t_{jh} x_{jh} = \sum_{(j,h) \in A} z_{jh} \qquad \forall j \in V \setminus \{n+1\} \tag{9}$$

$$z_0 = 0 \tag{10}$$

$$z_{n+1} = l_{n+1} \tag{11}$$

$$\max\{e_i + s_i + t_{ij}, e_j\} x_{ij} \leq z_{ij} \leq l_j x_{ij} \qquad \forall (i,j) \in A \tag{12}$$

$$\sum_{i \in V_{k'}^{jk}} y_i \geq y_j \qquad \forall j \in V_k, \forall k' \in \tilde{T}_k, \forall k \in T \tag{13}$$

$$\sum_{i \in V_{k'}^{jk}} z_i + s_i y_i \geq z_j - b_{k'k} y_j \qquad \forall j \in V_k, \forall k' \in \tilde{T}_k, \forall k \in T \tag{14}$$

$$\sum_{i \in V_{k'}^{jk}} z_i + s_i y_i \leq z_j - a_{k'k} y_j + M_{k'}^{jk}(1 - y_j) \qquad \forall j \in V_k, \forall k' \in \tilde{T}_k, \forall k \in T \tag{15}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{16}$$

$$y_j \in \{0,1\} \qquad \forall j \in V \tag{17}$$

$$w_j \geq 0 \qquad \forall j \in V \setminus \{n+1\} \tag{18}$$

The two objective functions (1) and (2) maximize the total profit and freetime, respectively. Constraints (3) and (4) ensure that each mandatory task is done exactly once and each optional task at most once, respectively. Equalities (5)–(7) describe the tour which starts at visit 0, performs a subset of visits, and ends at visit $n + 1$. Subtours are eliminated by the single-commodity flow system (8)–(12) on the start time variables: Equalities (8) link arc with visit time variables. Flow conservation constraints (9) state that the start time of a visit plus service time, free time, and the following travel time equals the start time of the next visit. Equalities (10) and (11) set the times at the start and end visit, respectively. Linking constraints (12) ensure the visits' time windows. Inequalities (13) guarantee that all predecessor tasks of j have to be performed before that task, while constraints (14) and (15) preserve the maximum and minimum time gap between the tasks, respectively. The tightest value for the big-M constant in (15) is defined as:

$$M_{k'}^{jk} = \max_{i \in V_{k'}^{jk}} \{l_i + s_i\}$$

Finally, integrality constraints (16) and (17), and non-negativity constraints (18) define the domains of the corresponding variables. Note that because of equalities (5)–(8) and (10)–(11) variables y_j and z_j for visits $j \in V$ can be eliminated from the model, but have been kept for better readability.

4. Large Neighborhood Search for the PPP

We propose a bi-objective large neighborhood search (LNS) heuristic (BO-LNS) for solving the PPP. Introduced by Shaw (1998) and later extended by Schrimpf et al. (2000) as *ruin and recreate*, LNS is a metaheuristic framework based on the idea of iteratively destroying and subsequently repairing solutions to explore large parts of the solution space and discover diverse local optima. A destroy operator removes certain parts of a solution according to some predetermined selection strategy. The resulting partial solution is then reconstructed into a full solution with a repair operator, commonly an insertion heuristic. LNS has been shown to perform well on a variety of problem classes, particularly when many constraints must be taken into account. The interested reader can find a more detailed discussion of the related LNS literature in the review by Pisinger and Ropke (2010).

4.1 Outline of the proposed algorithm

Although the original LNS heuristic was conceptualized as a single-objective algorithm, its design offers inherent synergy with bi-objective orienteering-type problems such as the PPP. One LNS iteration corresponds to a single destroy operation removing several tasks, and a subsequent repair operation which (re)inserts tasks one at a time until no further tasks can be planned. Since every insertion operation strictly improves the profit objective and strictly worsens the free time objective, it is possible that *multiple* mutually non-dominated solutions are found during *one* iteration of the LNS. We exploit this property by evaluating Pareto-efficiency after every individual insertion, and updating the archive of non-dominated solutions as necessary. Figure 1 visualizes this process.

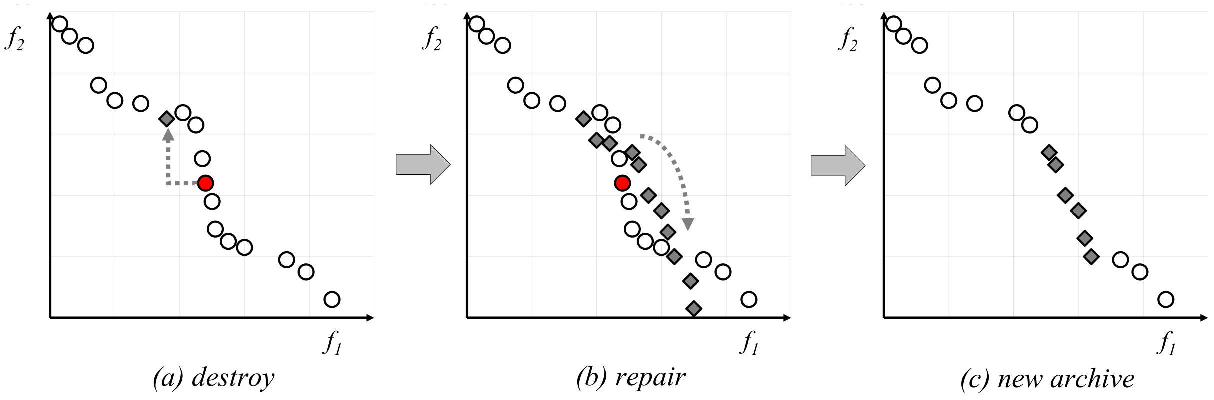

Figure 1: Bi-Objective Large Neighborhood Search

To ensure proper diversification of the search, each of the solutions from the archive is successively used as the starting solution for the LNS. Combined with the property explained above, every such iteration generates a set of mutually non-dominated neighbor solutions N originating from the chosen starting solution. During each iteration, the archive is updated as the Pareto non-dominated union of itself and N. By using the archived solutions as starting points in each iteration, the LNS is able to systematically explore all areas of the Pareto frontier.

Prior to any optimization, we perform some pre-processing of the input data. Specifically, each task is associated with a set of "potential visits", as described previously in Section 3. These sets of visits are then used as the basic units of construction for all solutions. Each task's visits are sorted according to the start time of their time window in order to increase the efficiency of feasibility checks and insertion operations. Likewise, the solutions in the archive are sorted in increasing order of objective f_1 (in our case profit), which helps to speed up dominance checks.

Algorithm 1 outlines the search strategy of the proposed bi-objective LNS. The archive A is initialized with an empty solution (a schedule with no planned tasks). If mandatory tasks are to be scheduled, then this solution is deleted from A after a feasible replacement is found in the first iteration. A parallel archive S is used to keep track of the current set of non-dominated *candidate* solutions which have not yet been used as starting solutions for the LNS. In each iteration, the solution s_0 with the lowest f_1 among the candidate solutions in S is copied, destroyed, and repaired by the LNS. The resulting set N of mutually non-dominated neighbor solutions is examined for inclusion in the archive A. All new and Pareto-efficient solutions are also included in S, and dominated solutions are removed from S – in contrast to a standard queue, this update policy guarantees that the LNS is initialized only with starting solutions which are still non-dominated. If the starting solution s_0 used in the current iteration did not lead to any improvement of the overall archive A, then it is removed from S, otherwise it remains in S and can be re-examined in subsequent iterations as long as it leads to improvement of A. The algorithm continues until all candidate solutions in S have been used by the LNS at least once and did not improve the archive A, at which point S will be empty.

One completion of the inner while loop corresponds to one exhaustive "pass" over the archive A. This can be repeated until some termination criterion (e.g. run time, number of passes, etc.) has been reached. In the practical application motivating our research, the search is continued indefinitely and can be terminated at any time by the user.

We propose a modular framework for the definition of the destroy and repair operators based on implicit permutations of general operator characteristics. These are described in more detail in the following two sections.

Algorithm 1 Bi-Objective Large Neighborhood Search (BO-LNS)

```
$x \leftarrow \emptyset$                          ▷ empty initial solution, a schedule with no tasks planned
$A \leftarrow \{x\}$                              ▷ archive of non-dominated solutions

repeat
    $S \leftarrow A$                              ▷ sorted list of candidate starting solutions
    while $S \neq \emptyset$ do
        $x \leftarrow s_0$                        ▷ the solution with lowest $f_1$ in the sorted list $S$
        $x' \leftarrow destroy(x)$
        $N \leftarrow repair(x')$                 ▷ set of mutually non-dominated neighbor solutions
        for all $n \in N$ do
            $A \leftarrow A \cup_p \{n\}$         ▷ Pareto non-dominated union
            if $n$ is new and Pareto-efficient then
                $S \leftarrow S \cup_p \{n\}$
        if $A$ was not improved then
            $S \leftarrow S \setminus \{s_0\}$
until termination criterion
return $A$
```

4.2 Destroy Operator

The destroy operator removes parts of a solution based on the following parameters:

component	*definition of what constitutes a "part" of the solution*
criterion	*on what basis these parts are evaluated*
quantity	*how much of the solution is to be destroyed*

A randomly selected permutation of these parameters is used in each iteration to determine the behavior of the destroy operator. Pseudocode for the operator is presented in Algorithm 2.

Solution Components

Different definitions of "component" are used so that the destroy operator can work on different levels of aggregation. We use "potential visits" as the lowest level of aggregation for the PPP. With this, the following solution parts may be considered for removal by the destroy operator:

single	*individual, possibly unrelated visits*
consecutive	*sequences of consecutive visits*
same-day	*all visits on a particular planning day*
chain	*visits connected by precedence relations*

Algorithm 2 Modular Destroy Operator

```
input: a solution $x$

$C \leftarrow \emptyset$                                   ▷ list of solution components to evaluate
$R \leftarrow \emptyset$                                   ▷ ranking list of the parts in C
$n, n_{max} \leftarrow 0$

$y_1 \leftarrow selectComponentType$
$y_2 \leftarrow selectRankingCriterion$
$y_3 \leftarrow selectDestroyQuantity$

$C \leftarrow determineSolutionComponents(x, y_1)$

for all components $c$ in $C$ do
    $r \leftarrow evaluateRanking(c, y_2)$
    $R \leftarrow storeRanking(R, r, c)$

$n_{max} \leftarrow numberVisitsRemove(x, y_3)$

while $n \leq n_{max}$ do
    $c \leftarrow takeOutWorstComponent(R)$
    $x \leftarrow removeComponent(c, x)$
    $n \leftarrow n + visitsInPart(c)$
return $x$
```

The *single* component type allows for the most fine-grained selection of parts to remove, but it offers no concept of relatedness as proposed by Shaw (1998). The *consecutive* component type considers sequences of n consecutive visits, scheduled one immediately after the other in the solution. This provides relatedness both in terms of routing (the locations are visited one after another) as well as timing (the time windows either overlap or progress from earlier to later). Smaller values of the component length parameter n allow for a more targeted selection, while larger values focus more on relatedness. If one such consecutive part is selected for removal, *all* of its constituent visits are removed.

The *same-day* component type is more specific to the PPP and operates on a higher level of aggregation. With this, entire days of tasks are considered for removal. Finally, the *chain* component type targets sets of tasks connected through precedence relations and time delay constraints. Although individual visits of a precedence chain may be removed with the previous definitions, this definition enforces the removal and subsequent reinsertion of an entire chain, which may be necessary to find a sufficiently different scheduling of the chain as a whole.

Removal Criteria

Provided that potential components for removal have been identified, the decision still has to be made on which of them to remove. The following removal criteria are proposed to determine removal priorities:

min. forward shift	*how much the visits can be shifted in the current plan*
min. time window flexibility	*how tight/relaxed the time windows are*
min. profit/time ratio	*the trade-off between score and time*
random	*a purely stochastic selection*

The *forward shift* criterion measures how much a visit (and subsequent ones) can be pushed forward without affecting the feasibility of the solution as a whole. A lower forward shift indicates a more constrained part of the solution, and is therefore removed with higher priority. The *time window flexibility* criterion is equal to the visit's time window minus its service time (an average in the case of aggregate parts). Although this figure is constant for each visit, each task has a number of visits to choose from, so this criterion evaluates the visits actually chosen in the current solution. Parts with lower values indicate potential or existing sources of bottlenecks.

The *profit/time ratio* criterion estimates the relative score value of a solution part compared to how it has been scheduled. It is calculated as the total score awarded by the component divided by the sum of the travel times to/from the tasks and their service times. Free time within a segment is not counted toward the total time. Short segments with high profit achieve the best scores, whereas long segments with low profit are the worst. Finally, the *random* criterion assigns to each evaluated component a removal priority randomly chosen from a uniform distribution ranging from 0 to 100.

The destroy operator removes from the solution the n components with the lowest criteria values, where n depends on the destroy quantity parameter.

Destroy Quantity

This parameter determines how much of the solution will actually be destroyed. Since different solutions will have different numbers of visits scheduled, and components of different sizes may be considered for removal, this parameter refers to the percentage of *individual visits* to remove from those currently scheduled. This ensures that the relative effect is more or less the same regardless of instance size and of how empty or full the current solution is.

Experimentation showed that selecting a random value per iteration from a fixed uniform range of 5% to 30% worked well.

4.3 Repair Operator

The repair operator heuristically inserts unscheduled visits, one at a time, into an empty or partial solution until no further insertions are feasible. As with the destroy operator, the behavior of the repair operator is subject to three characteristics, randomly selected in each iteration of the LNS. The repair parameters are as follows:

scope	*the pool of tasks which are considered for insertion*
criterion	*which insertion positions are favored*
order	*the order in which they are inserted*

In order to maintain feasibility, the repair operator gives priority to missing mandatory and/or precedent tasks if any are missing in the partial solution. We emphasize that this is the only form of infeasibility which needs to be corrected in the proposed metaheuristic. It is guaranteed that at least one feasible reinsertion exists for each of these missing tasks (namely the position from before the destroy operation), but other insertion positions may be chosen depending on the repair parameters. The pseudocode for the repair operator is shown in Algorithm 3.

Insertion Scope

The *scope* parameter determines the set of tasks which are considered for insertion by the operator. Three variants are used:

all	*all currently unscheduled tasks*
removed	*only the tasks removed by the previous destroy operation*
neglected	*only the tasks which were unscheduled prior to the destroy operation*

These three options facilitate varying degrees of intensification or diversification. Selecting from the pool of *removed* tasks offers the highest degree of intensification, while the pool of *neglected* tasks imposes diversification. The pool of *all* currently unscheduled tasks represents a mixture of both strategies. Note that even if the set consists of only a single task, this does not imply that it will be reinserted at the same position or using the same visit.

Insertion Criterion

The insertion *criterion* defines the measure used to evaluate and compare the many feasible insertions for each visit. The following three criteria are proposed:

min. shift time	*the extra travel and service time required*
max. total slack	*the potential slack time at the chosen position*
max. time window flexibility	*time window size compared to the service time*

Algorithm 3 Modular Repair Operator

```
input: a solution $x$, set of all tasks $T$

$T' \leftarrow \emptyset$ (set of candidate tasks for insertion)
$I \leftarrow \emptyset$ (ordered list of best insertions of the tasks in C)

$z_1 \leftarrow \textit{selectInsertionScope}$
$z_2 \leftarrow \textit{selectInsertionCriterion}$
$z_3 \leftarrow \textit{selectInsertionOrder}$

$T' \leftarrow \textit{makeCandidateList}(x, T, z_1)$

for all tasks $t$ in $T'$ do
    $i \leftarrow \textit{getBestInsertion}(x, t, z_2)$
    $I \leftarrow \textit{storeInsertion}(i, I)$

$I \leftarrow \textit{sortInsertionList}(I, z_3)$

$N \leftarrow \emptyset$
for all $i \in I$ do
    $x' \leftarrow \textit{tryToInsertVisit}(i, x)$
    if $x' \neq x$ then
        $N \leftarrow N \cup \{x'\}$
return $N$
```

The *shift time* criterion measures by how much time subsequent visits must be shifted in order to accommodate the insertion. It is the sum of the visit's service time and the travel time to and from this visit at the proposed insertion position, minus the current travel time between the visits before and after the insertion position. Shifting visits to later positions may cause bottlenecks as tasks are pushed to the end of their time windows. Smaller scores are thus favored as they reflect a more unobtrusive task and/or timing.

The *total slack* criterion estimates how much the visit itself could be shifted during subsequent insertions of other visits. The score for this criterion is the sum of the necessary waiting time before starting the task, and the time between the end of the task and the visit's closing time. Insertions with lower total slack potential are more likely to become bottlenecks because they cannot be shifted as much.

The *time window flexibility* criterion does not depend directly on the current state of the schedule. It is a static measure for each visit, equal to the visit's time window length minus its service time. Each task generally has many possible visits, so this criterion filters these possibilities for those which likely offer greater temporal flexibility after other visits are removed and/or added in later iterations. If the size of the visit's time window allows it to be inserted at several positions, then one of them is randomly selected. Visits with more flexibility in their time windows will generally have shorter service time, wider time windows, or both. Inserting such visits is unlikely to trap the search prematurely in a local optimum.

All three of these criteria focus exclusively on the timing aspects of the problem. However, the visit's profit can easily be incorporated into the insertion decision. To achieve this, the insertion scores are weighted according to the profit of the visit, by multiplying (or dividing, in the case of minimum shift time) the score by the profit. This yields a total of six insertion criteria.

Insertion Order

The *order* parameter determines in which order visits are inserted into the partial solution. Three possibilities are proposed:

greedy	*the visit with the best insertion score is inserted first*
GRASP	*one of the visits with the best insertion scores is randomly selected*
random	*a purely stochastic ordering*

For each eligible task we identify the visit with the best score according to the selected insertion criterion (detailed in the previous section) forming a list of visits for further processing. In the *greedy* option, the visit with the best score in the list is inserted. With the *GRASP* (Feo and Resende, 1995) option, a restricted candidate list of the *RCL* best visits is formed and one of them is randomly chosen according to a roulette wheel selection (as in the more general GRASP heuristics). Note that for both the *greedy* and *GRASP* variants the list of best visits for each task and the ordering must be recomputed after each insertion. Finally, the *random* option shuffles the list of candidate tasks, and the tasks are then inserted in the best way (with respect to the insertion criterion selected above) in this order until all tasks have been examined.

Note that this parameter has no direct influence on which visits are chosen for each task, or on the positions at which those visits are inserted. Those decisions are determined by the *criterion* parameter.

5. Computational Study

In this section we report and analyze the computational performance of the proposed BO-LNS. Our test instances were generated on real-world-based information and are described in detail in Section 5.1. The test settings used in all experiments are listed in Section 5.2. In Section 5.3 the quality of the metaheuristic is evaluated by comparing the results to the optimal Pareto sets found with an exact method on a set of smaller instances. Section 5.4 presents computational results on a set of larger instances. The complete set of instances used and all detailed computational results are available as part of the electronic version of this paper, or upon request from the authors.

5.1 Instance Data

The benchmark instances for our study are based on real-world information gathered during the course of the research project by an analysis of 259 online interviews and two focus groups with 10 participants each. This analysis resulted in six instance groups (A to F) representing different types of weekly schedule patterns. Five instances are generated for each group, yielding 30 instances which differ in the number of tasks (ranging from 13 to 74), locations (15 to 3000), and precedence relations.

We generated small and large versions of these 30 instances, for 60 instances total. The set of the 30 smaller instances has a planning horizon of 3 days and can be solved to optimality using the mathematical model described in Section 3 embedded in an epsilon-constraint framework. The 30 large instances consist of tasks for an entire week as well as a larger selection of locations.

With regard to the routing aspect of the problem, existing data sets of real locations in the city of Vienna, Austria, were used. Appropriate subsets (e.g. supermarkets, restaurants, specialty stores) are identified as available locations for different types of tasks, obtained from map data provided by OpenStreetMap© contributors. One particular location designates the decision-maker's home, which is always the start and end point of the schedule. Time window data on opening and closing times is either supplied from map data of OpenStreetMap© contributors, or estimated. The underlying distance matrix was computed by *Ariadne*, a routing tool proposed in Prandtstetter et al. (2013) based on floating car data from FLEET, an FCD system continuously working since 2003 and collecting data from about 3000 taxis in Vienna.

5.2 Test Settings

In the practical application, BO-LNS continues until the search is terminated by the user. In order to provide a meaningful basis of comparison between all types of instances, here we allow the algorithm to make only one pass over the archive, i.e. the search terminates once all solutions in the archive have been used as starting solutions for the LNS without subsequently generating new non-dominated solutions and improving the archive. This corresponds to Algorithm 1 without the repeat-until loop.

The parameter settings used for our study are listed in Table 2. Based on preliminary experiments, a fixed range for the destroy intensity was chosen from $d_{\min} = 5\%$ to $d_{\max} = 30\%$ of the visits in the current solution. These are the only parameters which have a global effect on the LNS, the remaining parameters apply only to specific operator modules. For the destroy operator's consecutive component type, the segment length was set to a range between $s_{\min} = 2$ and $s_{\max} = 5$, chosen uniformly at random whenever this component type is used. The repair operator's GRASP insertion uses a restricted candidate list of size $RCL = 5$, i.e. one of the best five insertions is performed uniformly at random in each step.

Parameter	$d_{\min}$	$d_{\max}$	$s_{\min}$	$s_{\max}$	RCL
Value	5	30	2	5	5

Table 2: Default Parameter Values

In general, it is not obvious how to gauge the quality of Pareto set approximations, even in comparison to an optimal reference set. Zitzler et al. (2008) provide an overview of the approaches and indicators proposed in the literature. We use two indicators to evaluate the quality of the Pareto set approximations obtained with BO-LNS: the hypervolume and the multiplicative unary epsilon.

The hypervolume measures the area in objective space dominated by the approximation set. This measure depends on a reference point, the choice of which also impacts the relative performance gaps observed between different approximations. Although the origin is a valid reference point for double-maximization problems such as the PPP, the point given by the minimum objective values among all solutions for a given instance is generally much stricter, and this is what we use in our study. The multiplicative unary epsilon indicator measures the minimal factor ϵ by which all objective values in an approximation set would have to be multiplied (or divided in the case of minimization objectives) so that no solution in the approximation set is dominated by the given reference set (Zitzler et al., 2003). Loosely speaking, the epsilon indicator can be interpreted as the worst-case deviation to the reference set. Since the hypervolume is affected by every solution in the approximation and the epsilon indicator only by the "worst" solution, these two measures are often used together to make a more balanced comparison. For clarity, we report both indicators in percentage terms; the hypervolume as a percentage of the optimal value, and the epsilon indicator as a percentage gap, e.g. 25% corresponds to $\epsilon = 1.25$. We report also the cardinality of the approximation sets as well as the CPU time required to generate them with BO-LNS.

Our algorithm is implemented in Java and all computations were conducted on a personal computer running Windows 10 with an Intel Core i7-4790 3.60 GHz processor with 8GB of RAM. Each of the 60 instances is solved 100 times.

5.3 Comparison with Exact Reference Sets

The subset of small instances is solved to proven optimality by CPLEX 12.6.2 on a single core of an Intel Xeon 2643 with 3.3 GHz and 8 GB memory limit, using the model proposed in Section 3 embedded in the ϵ-constraint framework presented in Mavrotas (2009). We were able to compute optimal reference sets for all 30 small instances with an average run time of three hours, but highly depending on the number of potential visits and especially the number of precedence relations among them.

Table 3 reports for each instance the maximum, average, and minimum values observed for the four performance indicators: hypervolume, multiplicative unary epsilon, approximation set cardinality, and CPU time in seconds. The columns on the left side list instance characteristics: the number of tasks (T), the number of tasks with precedence constraints (P), the number of locations (L), and the resulting number of feasible visits (V). The latter can be seen as a rough indicator of instance size.

Since we have solved all instances 100 times, we also present a more detailed statistical analysis of the variation in performance. Figures 2, 3, and 4 visualize for each instance the variation in attained hypervolume, unary epsilon, and cardinality, respectively. We omit CPU time since BO-LNS terminated within 1 second in all test runs on this set of instances. The instances are sorted in order of improving average-case performance for each indicator.

Hypervolume. The average hypervolume is well above 90% on all 30 instances, above 95% on all but three, and above 97% overall. We can see from Figure 2 that this performance is consistent, as the deviation from the average is generally very small – for all but one instance, the difference between the 25th and 75th percentiles is only two or three percentage points. In addition, the median hypervolume is nearly always higher than the average, which indicates that the distribution is skewed toward higher values, i.e. above-average performance. In fact, Figure 2 reveals that even most "outliers" actually represent hypervolume values of 90% or higher. When it comes to the instances with the weakest hypervolume performance, we notice that these are the instances with the lowest cardinality (D5, D3, D1). This is not a surprise, because as the number of solutions in the approximation decreases, the hypervolume becomes increasingly sensitive to the proportionally larger hypervolume contribution of each individual solution, resulting in greater variability. Nonetheless, even on these instances BO-LNS achieves median hypervolumes of at least 93%.

Inst.	**T**	**P**	**L**	**V**	**Hypervolume (%)**			**Unary Epsilon (%)**			**Cardinality**			**CPU Time (s)**		
					Max	*Ave*	*Min*	*Max*	*Ave*	*Min*	*Max*	*Ave*	*Min*	*Max*	*Ave*	*Min*
A1	22	14	17	85	99.5	98.6	96.4	5.3	2.3	1.2	40	33.9	24	1	1	1
A2	22	13	17	99	98.9	97.4	95.9	13.6	4.5	2.2	47	39.5	32	1	1	1
A3	23	16	18	140	99.6	97.7	90.7	6.0	2.6	0.9	28	23.2	19	1	1	1
A4	19	8	17	74	99.2	97.0	92.7	6.2	2.6	0.9	25	19.8	15	1	1	1
A5	23	14	19	71	99.5	97.9	92.8	5.3	2.2	0.4	26	22.9	17	1	1	1
B1	19	4	17	53	99.9	99.3	97.5	5.3	2.7	0.3	29	26.0	20	1	1	1
B2	18	5	16	60	99.4	98.1	96.5	6.9	3.7	1.1	35	27.8	16	1	1	1
B3	21	7	19	54	99.1	97.7	94.2	8.6	3.0	1.8	29	21.2	12	1	1	1
B4	13	4	15	33	100.0	99.7	99.1	2.4	0.7	0.0	8	7.7	7	1	1	1
B5	16	5	17	40	99.8	99.1	96.8	6.5	1.6	0.3	24	21.1	16	1	1	1
C1	33	22	20	90	97.9	96.9	93.3	62.9	18.4	3.3	43	31.7	20	1	1	1
C2	25	14	17	82	99.5	97.8	92.0	9.7	4.5	1.8	32	26.8	20	1	1	1
C3	24	14	18	74	99.2	98.1	94.3	39.7	6.2	1.9	33	27.7	22	1	1	1
C4	26	13	21	74	99.6	98.8	97.2	5.3	2.9	0.9	40	34.3	28	1	1	1
C5	21	10	18	65	99.3	98.6	96.4	3.9	2.7	1.0	28	24.5	20	1	1	1
D1	23	6	25	57	95.4	93.9	86.1	12.2	7.1	2.2	15	13.6	12	1	1	1
D2	26	6	24	72	97.2	95.3	92.1	17.4	10.1	4.0	44	34.6	26	1	1	1
D3	27	10	23	65	99.2	93.2	86.2	38.2	18.5	0.2	12	8.2	7	1	1	1
D4	23	7	20	47	100.0	97.6	80.6	29.2	5.3	0.0	12	10.5	7	1	1	1
D5	24	4	19	62	99.9	91.5	82.8	12.2	6.2	0.1	6	6.0	5	1	1	1
E1	28	11	20	76	99.0	97.8	94.9	15.2	6.2	1.5	36	27.8	19	1	1	1
E2	24	6	18	65	99.1	97.4	94.3	32.4	7.2	1.6	43	34.7	26	1	1	1
E3	31	12	25	61	98.0	96.8	91.5	7.4	3.6	1.9	41	33.1	22	1	1	1
E4	26	12	19	82	98.9	96.9	93.5	35.5	8.5	2.6	48	38.2	30	1	1	1
E5	26	10	17	89	98.6	97.4	95.2	6.4	3.7	1.6	46	34.3	22	1	1	1
F1	29	8	20	100	98.9	95.3	91.2	50.4	8.1	2.9	35	30.1	25	1	1	1
F2	29	8	17	98	99.3	96.1	91.6	41.7	9.2	3.1	38	33.5	28	1	1	1
F3	28	7	16	101	100.0	99.0	91.0	37.8	3.3	0.0	33	31.1	26	1	1	1
F4	25	5	16	89	99.8	98.9	96.7	3.1	1.9	0.4	43	36.8	31	1	1	1
F5	27	6	18	77	99.1	98.1	94.0	208.9	28.4	1.6	48	33.6	24	1	1	1

Table 3: Performance on instances with known optimal reference sets (T: tasks; P: tasks with precedence relations; L: locations; V: visits)

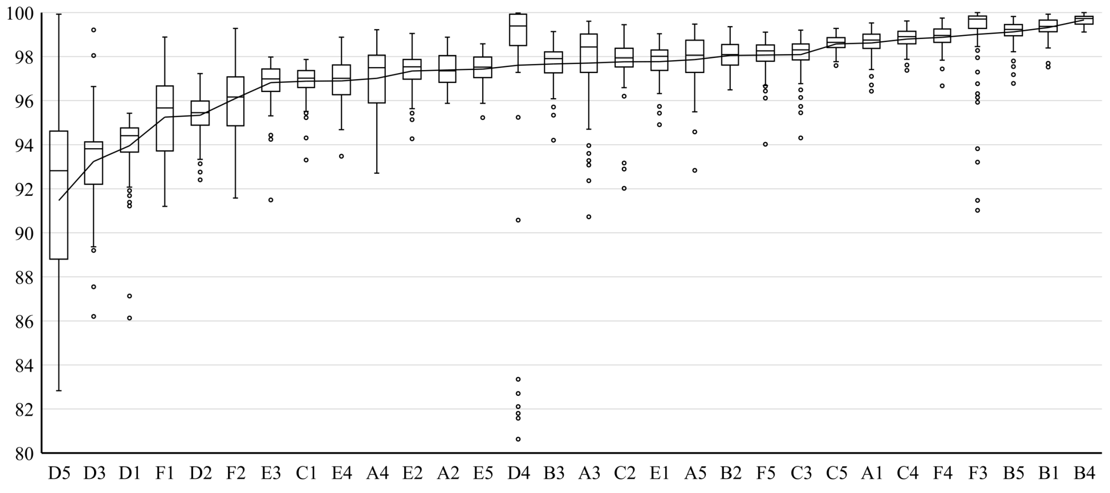

Figure 2: Variation in Attained Hypervolume (%) per Instance (Small Instances)

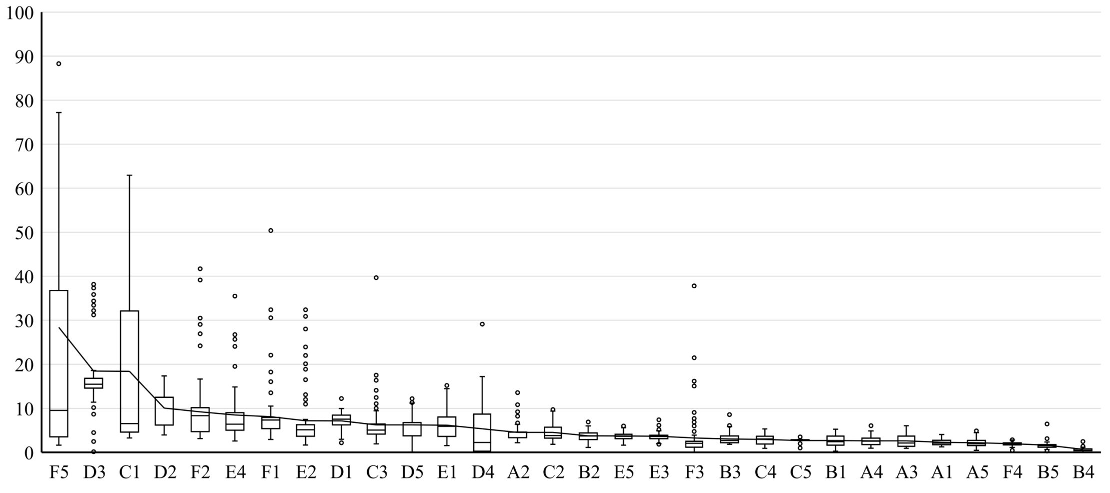

Figure 3: Variation in Attained Unary Epsilon (%) per Instance (Small Instances)

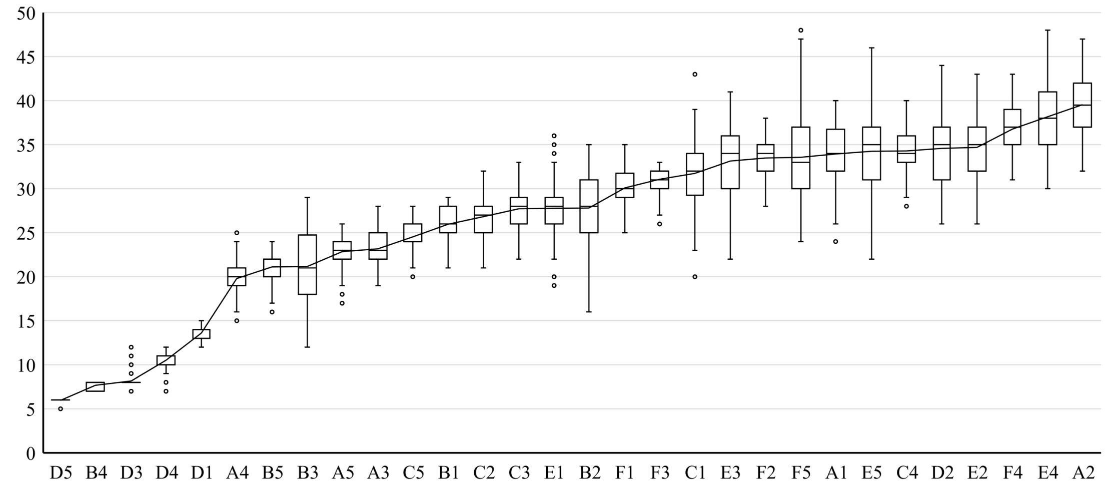

Figure 4: Variation in Attained Cardinality per Instance (Small Instances)

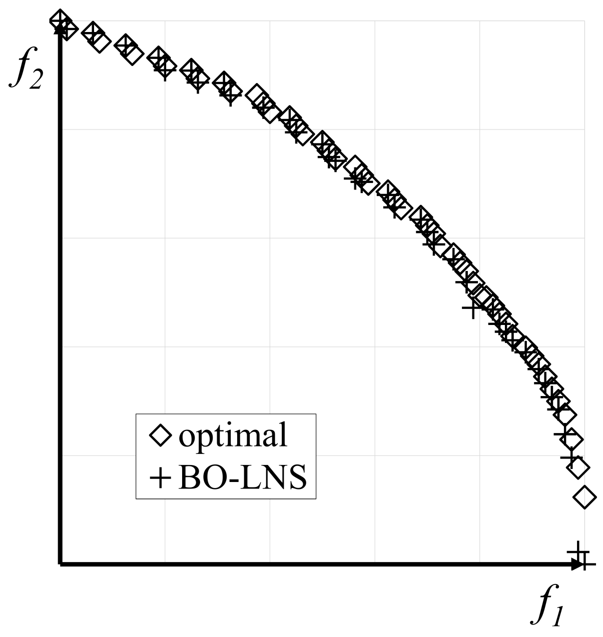

Figure 5: Normalized approximation set of test run with worst unary epsilon value (208.9%, Instance F5, Run 62)

Unary Epsilon. From Figure 3 we see that the median unary epsilon is 10% or lower for all but one instance, and the average 10% or lower on all but three. As with the hypervolume, the deviation from the average is generally very small, with an inter-quartile range of less than five percentage points on nearly all instances. The median unary epsilon is also generally better than the average, which again indicates that the epsilon performance of BO-LNS is skewed toward lower (better) values. This is particularly true for the instances with the largest variability, such as F5 and C1.

With respect to outliers, we note that because the unary epsilon indicator represents a worst-case deviation, its inferential power is strongest when the indicator is small – a low epsilon value implies that *all* solutions in the approximation set are very close to the reference, which is the case for nearly all the instances. In contrast, a large epsilon value can be observed even if just one solution is distant, and indeed the epsilon indicator improves dramatically in those cases when that solution is removed, which is counter-intuitive to the aims of multi-objective optimization. To emphasize this, we visualize in Figure 5 the approximation set with the worst unary epsilon value (208.9%) observed in the thousands of test runs we conducted. The approximation set in question is clearly of high quality, and indeed, achieves a hypervolume of over 97% of the optimum.

Cardinality. A larger number of trade-off solutions does not, on its own, imply a better approximation of the true Pareto set. However, when considered in combination with a low unary epsilon indicator, one may conclude that not only are more compromise solutions identified, *all* of them are close to the reference set and therefore of high quality. And indeed, this observation can be made for most of the instances examined here: BO-LNS generates an average of 20 to 40 trade-off schedules for most instances while maintaining unary epsilon values below 10% or even 5% in many cases. Hence, with BO-LNS a greater variety of compromise solutions does not come at the price of lower average quality.

CPU Time. As noted at the beginning of this section, all test runs on the smaller instances terminated within 1 second. The performance analyzed above is thus all the more satisfactory. However, since the optimal reference sets are known for this set of instances, we have also done additional tests to determine the extent to which BO-LNS is able to converge toward those optima. By changing the termination criterion to a maximum CPU time of just five seconds, BO-LNS was able to reach an average hypervolume of over 99% and an average unary epsilon of just over 4% over all 30 smaller instances, and solved nearly half of these instances to optimality.

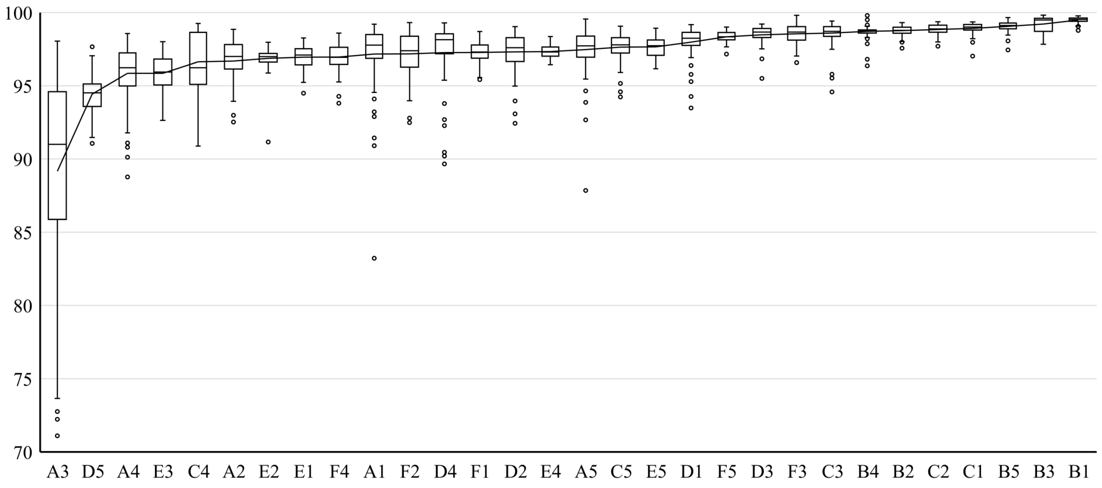

Figure 6: Variation in Attained Hypervolume (%) per Instance (Large Instances)

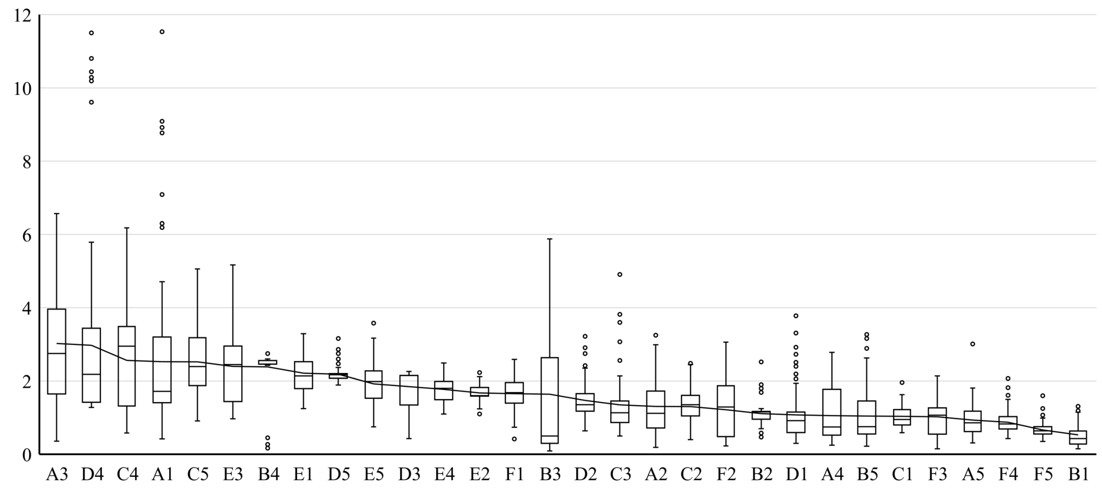

Figure 7: Variation in Attained Unary Epsilon (%) per Instance (Large Instances)

5.4 Instances with Approximate Reference Sets

This section reports computational results of the proposed BO-LNS on large instances for which no exact optimal reference sets are known. We calculated approximate reference sets by running the algorithm without any stopping condition (i.e. looping over the archive indefinitely) other than computation time, which we limited to 1 hour per run. Each instance was solved 10 times with these settings, and the final reference set was constructed as the non-dominated union of all solutions found over all runs combined. These approximate reference sets are used here as the benchmarks for evaluating the relative hypervolume and multiplicative unary epsilon indicators for the large instances.

Computational results for the 30 large instances are presented in Table 4. As before, we conduct 100 runs per instance and report the maximum, average, and minimum indicator values observed, as well as details on the instance characteristics. The variation in performance for the various indicators is visualized in Figures 6, 7, 8, and 9.

Hypervolume. BO-LNS achieves an average hypervolume of over 95% for nearly every instance, and does so in a reliable way – Figure 6 shows that almost all of the inter-quartile ranges are smaller than two or three percentage points. In fact, even the outliers are virtually always above 90%. The single exception to this trend is instance A3, and the reason is again the very small cardinality of the Pareto sets for this instance. In practice, when computation time is the termination criterion, this behavior is offset by the

Inst.	**T**	**P**	**L**	**V**	**Hypervolume (%)**			**Unary Epsilon (%)**			**Cardinality**			**CPU Time (s)**		
					Max	*Ave*	*Min*	*Max*	*Ave*	*Min*	*Max*	*Ave*	*Min*	*Max*	*Ave*	*Min*
A1	32	23	2245	17472	99.21	97.16	83.22	11.53	2.52	0.42	21	17.2	12	17	10.0	5
A2	36	27	2118	19281	98.85	96.69	90.90	3.25	1.31	0.19	18	15.1	9	51	23.1	7
A3	32	28	1931	15949	98.05	89.17	71.10	6.57	3.02	0.36	15	11.4	8	20	9.9	2
A4	28	18	1962	15967	98.57	95.85	88.77	2.78	1.06	0.25	16	13.2	10	20	12.6	4
A5	32	23	223	3928	99.56	97.47	87.85	3.01	0.94	0.31	31	25.7	18	20	11.7	4
B1	27	4	110	1947	99.77	99.49	98.78	1.38	0.53	0.15	53	49.2	41	1	1.0	1
B2	28	4	233	3883	99.31	98.76	97.56	2.52	1.11	0.47	32	29.8	27	1	1.0	1
B3	29	5	2186	8528	99.82	99.21	97.83	5.88	1.64	0.09	18	16.8	14	5	2.9	2
B4	27	4	2187	6589	99.80	98.71	96.37	2.75	2.39	0.17	29	26.4	25	3	2.3	2
B5	26	5	1890	5856	99.66	99.06	97.45	3.27	1.05	0.22	58	50.0	39	4	2.9	2
C1	62	42	1107	4241	99.36	98.92	97.02	2.05	1.04	0.59	107	86.4	64	83	53.9	27
C2	63	41	1316	6164	99.37	98.84	97.70	2.48	1.30	0.40	63	54.1	44	107	67.6	38
C3	63	42	723	3048	99.42	98.59	94.58	4.91	1.35	0.50	71	59.5	46	66	38.4	19
C4	60	38	1101	5260	99.25	96.64	90.88	6.18	2.56	0.58	82	65.0	51	120	67.1	34
C5	62	40	438	2372	99.07	97.64	94.24	5.06	2.52	0.91	60	52.5	45	91	50.0	27
D1	35	9	2599	12775	99.18	97.98	93.49	3.78	1.08	0.30	36	27.5	20	49	26.6	15
D2	40	13	2440	26867	99.04	97.32	92.44	3.22	1.47	0.64	59	48.1	34	233	150.0	59
D3	42	13	2036	65633	99.22	98.48	95.51	2.26	1.85	0.43	43	32.6	19	86	48.7	20
D4	42	13	2034	42228	99.29	97.27	89.66	11.50	2.97	1.28	27	22.5	17	36	24.7	14
D5	40	10	2049	29690	97.66	94.43	91.07	3.16	2.19	1.89	23	20.3	16	31	19.6	10
E1	65	38	809	4267	98.26	96.95	94.50	3.29	2.21	1.25	79	55.0	39	108	58.9	23
E2	71	47	766	4096	97.97	96.88	91.16	2.23	1.67	1.10	92	69.8	44	218	147.6	89
E3	72	45	726	4650	98.01	95.85	92.64	5.17	2.40	0.97	77	57.6	43	266	164.8	81
E4	74	48	1003	5356	98.36	97.33	96.44	2.49	1.77	1.10	117	93.6	74	422	273.3	139
E5	70	43	756	4846	98.93	97.66	96.17	3.58	1.92	0.75	94	67.4	52	220	116.6	55
F1	39	11	2948	13198	98.71	97.28	95.43	2.59	1.66	0.42	61	46.2	35	66	38.8	20
F2	30	7	2488	12997	99.31	97.17	92.49	3.06	1.22	0.23	26	20.4	13	11	6.3	4
F3	29	8	100	1947	99.81	98.55	96.58	2.14	1.02	0.15	17	16.6	15	1	1.0	1
F4	35	6	2630	14776	98.60	96.96	93.81	2.07	0.88	0.43	44	32.8	21	31	17.7	9
F5	32	9	2481	12419	99.01	98.34	97.16	1.60	0.66	0.35	65	49.7	36	26	13.8	6

Table 4: Performance on instances with approximate reference sets (T: tasks; P: tasks with precedence relations; L: locations; V: visits)

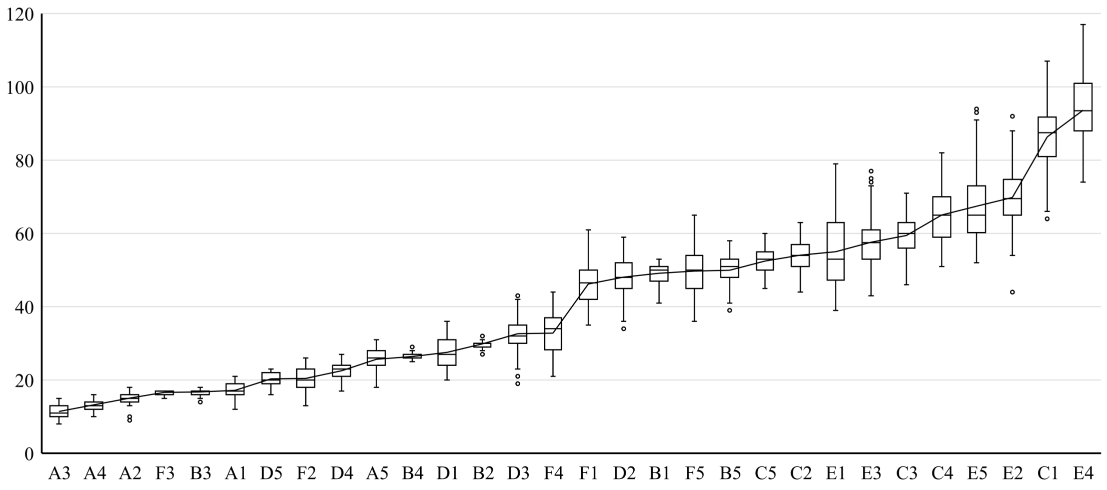

Figure 8: Variation in Attained Cardinality per Instance (Large Instances)

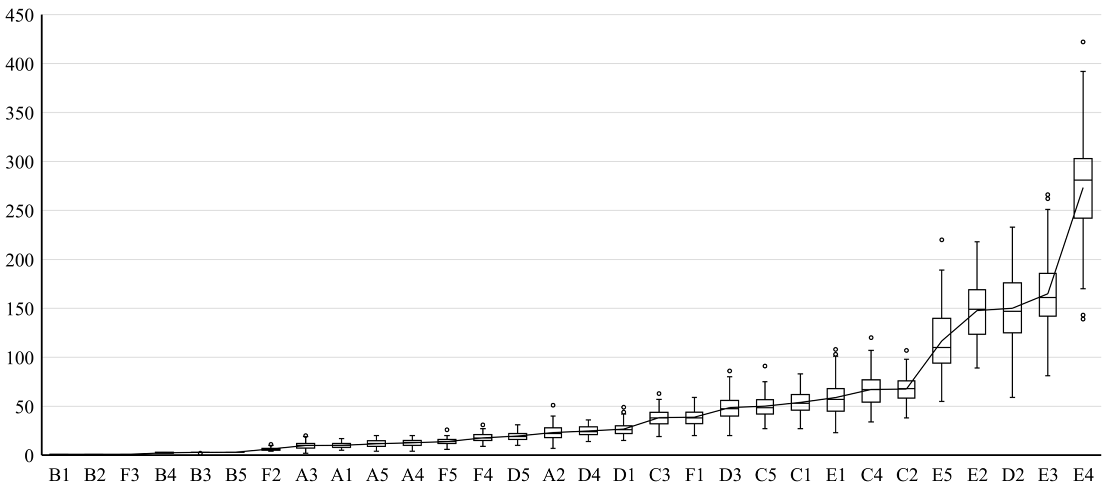

Figure 9: Variation in Required CPU Time per Instance (Large Instances)

very fast run time on such Pareto sets, allowing several searches over a smaller one in the same time as one search over a larger one. Nonetheless, even on this instance BO-LNS still achieves nearly 90% of the hypervolume on average; the main difference is the larger variability when compared to instances with larger Pareto sets.

Unary Epsilon. The average unary epsilon is three percent or lower for every instance. As with the hypervolume, BO-LNS is largely consistent in achieving this level of performance, with nearly all inter-quartile ranges spanning only two percentage points or less. Although performance is generally more variable on the instances with worse average performance, even the outliers virtually always represent epsilon values of less than 10%. Since these values are all quite low in absolute terms, we can conclude that BO-LNS reliably identifies solutions very close to the best known reference sets.

Cardinality. We can see from Figure 8 that the number of trade-off schedules varies widely depending on the instance, ranging from just an average of 11 (instance A3) all the way up to 93 (instance E4). It also appears that the variability increases together with average size. However, in absolute terms this variation is rather small; for any given instance, the inter-quartile range is around 10 solutions or less, which roughly corresponds to differences of ± 5 from one run to the next. These are relatively minor differences when the Pareto sets already have over 40 solutions for half the instances. Given the rather large Pareto sets on these instances and the low values of the unary epsilon indicator, it is fair to say that the quality of the approximations identified by BO-LNS remains similar also when the size of the instances and Pareto sets increases.

CPU Time. For most of the larger instances (25 out of 30), BO-LNS requires less than 60 seconds of CPU time. A closer comparison with the instance characteristics in Table4 reveals that the CPU times are strongly related to the number of tasks and precedence relations, as well as Pareto set cardinality. The most time-consuming instances (subsets C and E) have the most tasks and by far the most precedence relations, and they also yield the largest Pareto sets (cf. Figure 8). The opposite relationship can be observed with the A and B instances.

One counter-intuitive observation is that the more difficult subsets C and E have the fewest feasible visits, with just over 6,000 for the largest instance, compared to well over 10,000 for most of the other instances. It was expected that the number of visits would be a major factor for difficulty, since this figure determines the size of the search space. However, with each subsequent insertion of a task, many of these visits become infeasible due to either time windows or precedence relations. The number of visits therefore has only a moderating effect on run time.

Finally, it was a major objective for the practical application that the algorithm converges rapidly to a representative solution set. Since the search can be stopped at any time by the user, it is important that reasonable trade-off schedules are found in a short time. Individual schedules can then be further optimized if time is available. This was one reason why the standard epsilon-constraint method was not suited to our application.

In order to better illustrate the convergence of our BO-LNS, we plot in Figure 10 the hypervolume and unary epsilon indicators as a function of search progress, averaged over all 60 instances and 6000 test runs. A search progress of 0% corresponds to the first iteration, and 100% to the final approximation set returned upon termination.

We can see that in the first iteration, BO-LNS achieves an average hypervolume of around 81% with a unary epsilon of around 10%. The first 10% of the search then yields the largest improvement, increasing the hypervolume to above 90% of the final value, and reducing the unary epsilon gap to below 7%. The remainder of the search further improves these figures, but at a steadier and less dramatic pace. Even if we ignore the 30 small instances with run times below 1 second, the average time required to reach the 10% point is only around 5 seconds, and virtually never larger than 30 seconds. We therefore conclude that the LNS converges rapidly to a representative set of schedules, and is capable of further improving them as long as extra time is available.

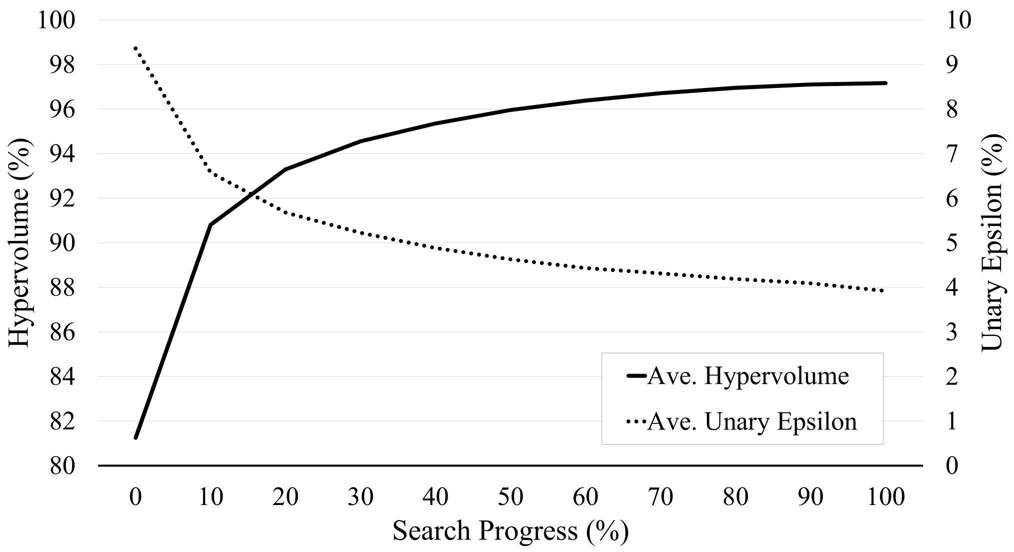

Figure 10: Hypervolume and Unary Epsilon Development as a Function of Search Progress

6. Conclusion

In this article, we tackle personal activity scheduling for mobile freelancers or other users with complex daily schedules and trips. For that purpose we define the Personal Planning Problem (PPP) – a bi-objective generalization of the orienteering problem. Aspects of generalization include multiple possible locations for a given task, multiple time windows, multiple periods, minimum and maximum delays between tasks, and precedence relations between tasks. The two objectives considered are the maximization of profit and of free time; they are naturally conflicting, and capture the natural tradeoff between scheduling more tasks and having more leisure time.

The PPP is modeled as a mixed integer linear program based on single-commodity flows and experiments show that small instances can be solved exactly, using the well-known epsilon-constraint framework. However, no instance of a relevant size for the real-world problem could be solved exactly with this approach. For that purpose, we developed a bi-objective large neighborhood search (BO-LNS) metaheuristic. We exploit the fact one destroy/repair iteration can generate several mutually non-dominated solutions, since partially destroyed solutions are still feasible in orienteering-type problems. By iteratively using non-dominated solutions as starting solutions, our BO-LNS is able to explore and improve the entire Pareto frontier. In addition, we propose a modular framework for the construction of a variety of destroy and repair operators.

Extensive computational experiments show that our BO-LNS can efficiently handle the many side-constraints given by the real-world problem, and produces highly efficient and representative solution sets in a consistent and reliable way. Smaller benchmark instances for which exact solutions are known are solved nearly to optimality within a few seconds. Tests on larger instances confirm the consistency in solution quality and indicate that the algorithm scales well.

BO-LNS is currently implemented in a prototype mobile application. Although a number of realistic constraints are already considered, the underlying model and functionality can still be enriched. For instance, a user may want to specify that lunch locations have to be different every day. Such features can be modeled as resource constraints, and it constitutes a promising perspective in order to make our solution approach more comprehensive to end users.

Computational Results

The instances generated for this study, as well as detailed results for all instances, are available as part of the online version of this article, or upon request from the authors.

Acknowledgments

This work has been partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the program Talents under grant 835532 (project FLASH, project partners: CURE Center for Usability Research and Engineering, FLUIDTIME Data Services, HERRY Consult). M. Ruthmair has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-014. We thank Markus Straub for his help with OpenStreetMap© and generating test instances, all OpenStreetMap© contributors for providing the map, and we gratefully acknowledge the taxi data provided by Taxi 31300. We also thank Alexandra Millonig for providing her expertise in human factors. Finally, thanks are due to two anonymous reviewers, whose comments helped to improve the quality of this paper.

References

Belhaiza, S., Hansen, P., Laporte, G., 2014. A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. Computers & Operations Research 52, 269 – 281.

Bérubé, J., Gendreau, M., Potvin, J., 2009. An exact epsilon-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits. European Journal of Operational Research 194, 39–50.

Boussier, S., Feillet, D., Gendreau, M., 2007. An exact algorithm for team orienteering problems. 4or 5, 211–230.

Christiansen, M., Fagerholt, K., 2002. Robust ship scheduling with multiple time windows. Naval Research Logistics 49, 611–625.

Feillet, D., Dejax, P., Gendreau, M., 2005. Traveling salesman problems with profits. Transportation science 39, 188–205.

Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. Journal of global optimization 6, 109–133.

Filippi, C., Stevanato, E., 2012. A two-phase method for bi-objective combinatorial optimization and its application to the TSP with profits. Algorithmic Operations Research 7.

Filippi, C., Stevanato, E., 2013. Approximation schemes for bi-objective combinatorial optimization and their application to the TSP with profits. Computers & Operations Research 40, 2418 – 2428.

Hu, Q., Lim, A., 2014. An iterative three-component heuristic for the team orienteering problem with time windows. European Journal of Operational Research 232, 276–286.

Jozefowiez, N., Glover, F., Laguna, M., 2008. Multi-objective meta-heuristics for the traveling salesman problem with profits. Journal of Mathematical Modelling and Algorithms 7, 177–195.

Kantor, M.G., Rosenwein, M.B., 1992. The orienteering problem with time windows. Journal of the Operational Research Society , 629–635.

Labadie, N., Mansini, R., Melechovskỳ, J., Wolfler Calvo, R., 2012. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. European Journal of Operational Research 220, 15–27.

Labadie, N., Melechovskỳ, J., Calvo, R.W., 2011. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. Journal of Heuristics 17, 729–753.

Lin, S.W., Yu, V.F., 2012. A simulated annealing heuristic for the team orienteering problem with time windows. European Journal of Operational Research 217, 94–107.

Mavrotas, G., 2009. Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. Applied Mathematics and Computation 213, 455–465.

Montemanni, R., Gambardella, L., 2009. An ant colony system for team orienteering problems with time windows. Foundations of Computing and Decision Sciences 34, 287–306.

Montemanni, R., Weyland, D., Gambardella, L., 2011. An enhanced ant colony system for the team orienteering problem with time windows, in: Computer Science and Society (ISCCS), 2011 International Symposium on, IEEE. pp. 381–384.

Parragh, S.N., Tricoire, F., 2015. Branch-and-bound for bi-objective integer optimization. Under review.

Pisinger, D., Ropke, S., 2010. Large neighborhood search, in: Handbook of metaheuristics. Springer, pp. 399–419.

Prandtstetter, M., Straub, M., Puchinger, J., 2013. On the way to a multi-modal energy-efficient route, in: Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE, IEEE. pp. 4779–4784.

Rodríguez, B., Molina, J., Pérez, F., Caballero, R., 2012. Interactive design of personalised tourism routes. Tourism Management 33, 926–940.

Savelsbergh, M.W.P., 1992. The vehicle routing problem with time windows: Minimizing route duration. INFORMS Journal on Computing 4, 146–154.

Schilde, M., Doerner, K.F., Hartl, R.F., Kiechle, G., 2009. Metaheuristics for the bi-objective orienteering problem. Swarm Intelligence 3, 179–201.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G., 2000. Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159, 139–171.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: Principles and Practice of Constraint ProgrammingCP98. Springer, pp. 417–431.

Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D., 2013. The multiconstraint team orienteering problem with multiple time windows. Transportation Science 47, 53–63.

Tricoire, F., 2012. Multi-directional local search. Computers & Operations Research 39, 3089–3101.

Tricoire, F., Romauch, M., Doerner, K.F., Hartl, R.F., 2010. Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 37, 351–367.

Vansteenwegen, P., Souffriau, W., van Oudheusden, D., 2011. The orienteering problem: A survey. European Journal of Operational Research 209, 1–10.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., van Oudheusden, D., 2009. Iterated local search for the team orienteering problem with time windows. Computers & Operations Research 36, 3281–3290.

Vidal, T., Crainic, T.G., Gendreau, M., Prins, C., 2015. Timing problems and algorithms: Time decisions for sequences of activities. Networks 65, 102–128.

Zitzler, E., Knowles, J., Thiele, L., 2008. Quality assessment of pareto set approximations, in: Multiobjective Optimization. Springer, pp. 373–404.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Da Fonseca, V.G., 2003. Performance assessment of multiobjective optimizers: an analysis and review. IEEE transactions on evolutionary computation 7, 117–132.