Optimizing Onsite Food Services at Scale

Konstantina Mellou
Microsoft Research
kmellou@microsoft.com

Luke Marshall
Microsoft Research
lumarsha@microsoft.com

Krishna Chintalapudi
Microsoft Research
krchinta@microsoft.com

Patrick Jaillet
Massachusetts Institute of Technology
jaillet@mit.edu

Ishai Menache
Microsoft Research
ishai@microsoft.com

ABSTRACT

Large food-service companies typically support a wide range of operations (catering, vending machines, repairs), each with different operational characteristics (manpower, vehicles, tools, timing constraints, etc.). While the advances in Internet-based technologies facilitate the adoption of automated scheduling systems, the complexity and heterogeneity of the different operations hinders the design of comprehensive optimization solutions. Indeed, our collaboration with Compass Group, one of the largest food-service companies in the world, reveals that many of its workforce assignments are done manually due to the lack of scheduling solutions that can accommodate the complexity of operational constraints. Further, the diversity in the nature of operations prevents collaboration and sharing of resources among various services such as catering and beverage distribution, leading to an inflated fleet size.

To address these challenges, we design a *unified* optimization framework, which can be applied to various food-service operations. Our design combines neighborhood search methods and Linear Programming techniques. We test our framework on real food-service request data from a large Compass Group customer, the Puget-Sound Microsoft Campus. Our results show that our approach scales well while yielding fleet size reductions of around 2x. Further, using our unified framework to simultaneously schedule the operations of two different divisions (catering, water distribution) yields 20% additional savings.

CCS CONCEPTS

• **Theory of computation → Optimization with randomized search heuristics**; Linear programming; Integer programming; • **Applied computing →** *Transportation.*

KEYWORDS

route optimization, workforce scheduling, planning

ACM Reference Format:
Konstantina Mellou, Luke Marshall, Krishna Chintalapudi, Patrick Jaillet, and Ishai Menache. 2020. Optimizing Onsite Food Services at Scale. In *28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '20), November 3–6, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3397536.3422266

SIGSPATIAL '20, November 3–6, 2020, Seattle, WA, USA

ACM ISBN 978-1-4503-8019-5/20/11…$15.00
https://doi.org/10.1145/3397536.3422266

1 INTRODUCTION

Operations routing and scheduling are at the core of diverse applications, including technician scheduling [33], health-care personnel assignment [15], airline planning [3] and more. The automation and the optimization of the underlying tasks is crucial both for attracting and retaining customers, as well as for maintaining low operation costs [8, 11, 12]. An increasingly important sector which has hitherto received less attention in the literature is *non-commercial food-services* – distributing and serving food and beverages for large organizations that span over multiple buildings in the same geographical area. Examples for such organizations include large enterprises and university campuses. The market-size for this sector was estimated around 200 billion dollars already in 2012 [32].

The goal of this paper is to find automated and systematic ways to optimize such onsite food-service operations. As we elaborate below, some of the underlying operations give rise to very complicated combinatorial optimization problems that, to the best of our knowledge, have not been explicitly considered in prior work (see §5 for discussion). Further, the diverse nature of the different operations (e.g., food catering vs. water/beverage distribution) often forces the maintenance and management of independent fleets for each service. This leads to inflated fleet sizes, and in turn, high capital expenditures and maintenance costs.

Our study is particularly inspired by observing and analyzing the onsite food services within our own campus, the Microsoft Puget Sound campus, which accommodates tens of thousands of Microsoft employees. Geographically, the campus includes more than 100 buildings spread over an area of around 200 square miles in West Washington state (see Figure 4). The food-service operations in our campus are run by Compass Group – a large field service company, which employs many thousands of people worldwide. Compass Group (or Compass in short) focuses mostly on food services, including enterprise dining, catering events, water and beverage deliveries and refills, vending machines, but also provides other services such as facility repairs.

Due to the scale and complexities involved, the responsibility for the different services across the Microsoft campus (and in other similar-size campuses) is divided across multiple divisions (e.g., a division in charge of catering, and another in charge of purified water distribution). The different divisions operate independently, each managing its own personnel, vehicles and equipment. The planning and scheduling problems for the different divisions share

some common characteristics, such as dispatching "resources" (personnel, food) across geo-distributed locations. However, there are substantial differences in the underlying complexity. For example, the water distribution division manages periodic pickup and delivery of water containers, which can be fulfilled by a fairly static schedule. On the other hand, the catering division manages a diverse set of events, each with its own set of requirements with respect to food, staffing skillset (driver, bartender, etc.), equipment, as well as timing of the pickup, dropoff and cleanup.

Because the various services have such different operation models, each Compass division employs its own managing methodology. In fact, most divisions resort to manual scheduling processes, done by domain-expert dispatchers. We have teamed up with Compass to first analyze the efficiency of their operations. In particular, we have installed IoT-devices on Compass' vehicles, tracking location and speed. Our analysis of the location traces indeed reveals that the manual process results in inefficient utilization of vehicles (§2).

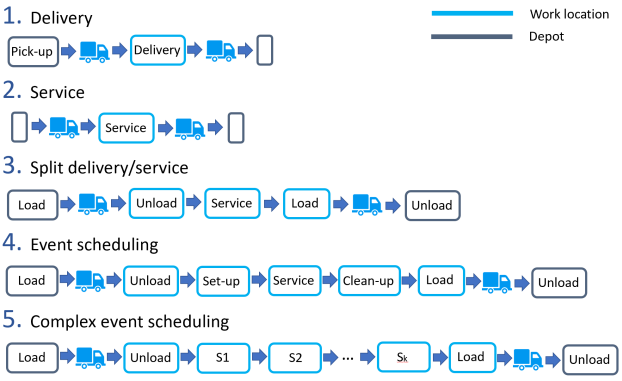


Figure 1: Operation templates.

As a first step towards our solution, we aim to *formulate* the optimization problems pertaining to Compass' operation. Towards that end, we introduce a small number of *templates* (see Figure 1) that characterize the different operations. Each template consists of a sequence of steps (or work-items) that are required to be carried out in a specific order in order to complete a task (delivery, repair, event). Each of the steps may require a combination of vehicles and/or employees with certain characteristics. The generality of the set of templates forms an umbrella that covers the different operation types, where every task of a division can be classified into one of the templates. The idea behind the templates is to approach the underlying optimization problem with a *unified* scheme, without requiring separate solutions for each scenario.

Based on the template specification, we target the design of algorithms that would provide high-quality solutions for the different operations. Our main focus is on the scheduling of catering events, which turns out to be the most challenging optimization problem across the divisions of Compass. Each catering event can be modeled by Template 4 (Figure 1): the first steps consist of delivering the food items and appropriate equipment to the event location; the delivery may require specialized vehicles (e.g., refrigerator vehicles) and employees with appropriate driving licenses. Then, the event should be set up by personnel with proper domain knowledge and experience. During the event itself, multiple employees may need to be present, again with appropriate expertise (servers, bartenders). Finally, clean-up and return of equipment to a depot are carried out after the event is over. Employees may perform one or more of these steps if they have the required expertise. Template 5 is a generalization of this operation, which allows for an arbitrary number of steps for an event.

The scheduling of catering events introduces several challenges. First, there are strict precedence constraints between the different stages, and some stages have strict timing constraints (e.g., an event has to start exactly at 10am, and should be set up between 9:30am and 10am); for some stages, travel time has to be factored in the timing considerations. Vehicles have capacity constraints that need to be taken into account for the item transportation, and employees with appropriate driving licenses must be assigned to each vehicle trip. Further, the event itself requires the synchronization and coordination of vehicles and manpower. For example, an event may require both a bartender and a server to be present for its entire duration. Only a subset of resources (vehicles, employees) can carry out a certain step (e.g., only a small subset of employees is trained to be a bartender). The combination of precedence and timing constraints, along with coordinating and synchronizing a heterogeneous set of resources gives rise to a very complicated combinatorial optimization problem.

To address the optimization challenges, we design a search-based algorithm which we term Unified Food-Service Optimization (UFSO). UFSO builds on Adaptive Large Neighborhood Search (ALNS), which has been shown to be effective for vehicle routing problems [27]. The ALNS method relies on repair and destroy operators, which modify part of the schedule while maintaining feasibility. Due to the timing complexities of our problem, standard repair operators might not maintain a feasible solution. Consequently, we design novel Linear Programming (LP) based repair operators, which ensure feasibility throughout the search process.

We evaluate our algorithm through simulations on real data – resources, vehicles, manpower, and event requests from Compass. Our results suggest that our algorithm can be effectively used by different divisions to schedule their daily operations. In particular, we show that our algorithm obtains high quality solutions, by comparing it against a theoretical lower bound for the problem. From an operational perspective, we obtain a reduction of around 2x in the number of vehicles currently used by the catering division. Finally, we show that optimizing the operation of two divisions simultaneously (catering and water distribution) reduces the fleet size by additional 20%.

In summary, our main contributions are:

- We analyze onsite food-service operations in a campus of a large enterprise §2; the analysis builds on IoT-device data that is collected in real-time from the vehicles.
- We formulate the optimization problem pertaining to onsite food services, and design algorithms that handle the complex coordination of resources, vehicles and employees §3.
- Our simulation results using real request data point to substantial savings that can be obtained by adopting our optimization approach within and across divisions §4.

2 BACKGROUND AND MOTIVATION

In this section, we provide a quantitative study of the operations of onsite food services provided in the Microsoft Campus by Compass.

2.1 Food Services in Microsoft Campus

Our focus is on two food service divisions – *food catering* and *water distribution*. We start by providing their details and requirements.

Catering events. The fleet used for catering comprises a total of 37 vehicles including 30 small vans, each of capacity 100 units of items, 5 sprinters of capacity 500, and 2 large boxtrucks with capacity 1500. Each of these vehicles costs between 50,000-100,000 USD in capital expenditure. We note that this type of vehicles incurs additional operational costs (insurance, maintenance, gas) in the order of $100 per day.

The staff comprises 59 employees with a diverse set of skills and level of expertise. Each individual, depending on their level of experience, might have one out of four catering roles: caterer, senior caterer, caterer lead, or area manager. Skill diversification includes the possession of a bartending license, as well as the driving skills. Each employee may or may not have a driving license; driving the larger trucks requires a special license. Furthermore, some employees might only be eligible to perform event clean-up if they lack the appropriate catering expertise to prepare an event.

Various types of items are required depending on the specifics of the event. For instance, warm meals for lunch events, vs. ice cream for social afternoon events. Since unit items require different capacities in the vehicles, a pre-processing step is performed to transform all item capacities into the same unit, which also facilitates their assignment to vehicles. For instance, warm food items take double the space of the rest of the item types, so each warm food item needs two capacity units of storage in the vehicles.

The division receives over a hundred event orders per day on average with fairly diverse requirements. Figure 2 captures the number of active orders during different hours of the day. As seen from Figure 2, the load peaks around lunch time, but is also substantial at breakfast times. Currently, given the complexity of the operations and non-availability of solutions that accommodate these complex constraints, schedules and assignments for the next day are generated manually every evening incurring approximately ten man-hours; the dispatchers try to increase efficiency by grouping buildings and events that are close by.

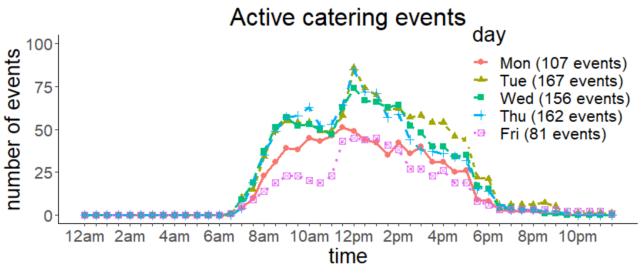


Figure 2: Number of catering events during the day.

Water distribution. This division is responsible for refilling water dispensers in a fixed set of office buildings twice a day - morning and before lunch. New dispensers are unloaded and any old ones are loaded back into the vehicle – a process that takes an average time of approximately 20 minutes. At the end of the day, the dispensers are collected for cleaning in the night. The only skill required for personnel in this division is driving. Consequently, the schedules for this division are usually static. Figure 3 captures the number of water distribution events that takes place over the day.

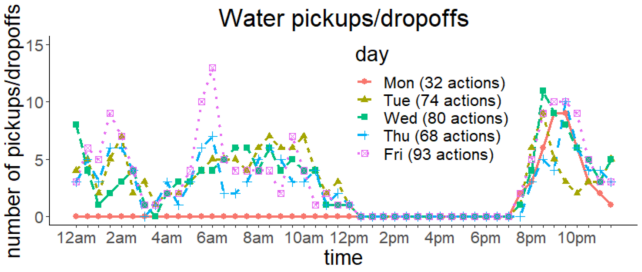


Figure 3: Daily water distribution pickups/dropoffs.

Opportunities for inter-division fleet sharing. As seen from Figures 2–3, the diurnal load distributions are fairly complementary. For example, while catering load peaks at 12 PM, water distribution load is almost lull between 12PM-8PM. Similarly, there are almost no catering events after 8 PM, while water distribution is very common in these hours. Furthermore, we have observed that at any point in time, at least half of the total vehicles for these two divisions remains unused (i.e., parked in the depot). This indicates that having a single pool of vehicles for the two divisions and jointly scheduling them may lead to significant reduction in fleet size.

2.2 Understanding Inefficiencies via Real-Time Location Tracking

The common challenge among Compass' operations lies in efficiently assigning resources to geographically distributed customer locations. Our engagement with Compass started a few years ago, where the initial motivation was to learn more about their operations via a *data-driven* approach. Towards that end, we installed IoT devices on Compass' vehicles that serve the Microsoft Puget Sound campus. The devices generate information periodically about the GPS location and the vehicle speed. The information is transmitted in real-time using novel low-cost long range wireless technology, which is described in [17], allowing us to obtain locations at a very fine granularity (10 second intervals).

Figure 4 shows the corresponding geographic area and the campus locations, which are the targets for the different food-delivery operations. Figure 5 aggregates all vehicle trajectories, where darker colors indicate road segments with more vehicle trips. To gain further insights about the efficiency of Compass' operations, we have analyzed utilization data obtained from the IoT devices. In particular, the devices send an "alive" signal whenever the vehicle is *active*. An active vehicle is defined as a vehicle that is either in transit, or static for some operational purpose (e.g., unloading food or equipment). An example of a real-time snapshot of the vehicle locations is shown in Figure 6, where we distinguish between active vehicles (green) and non-active ones (purple).

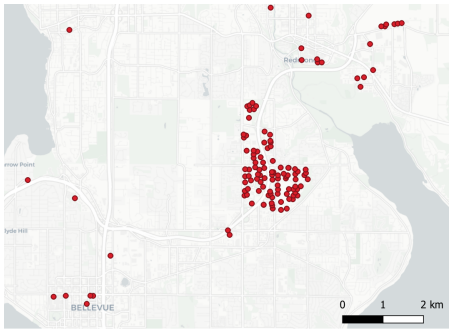


Figure 4: Microsoft campus locations

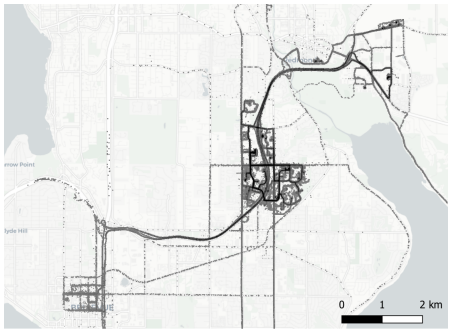


Figure 5: Vehicle trace heatmap

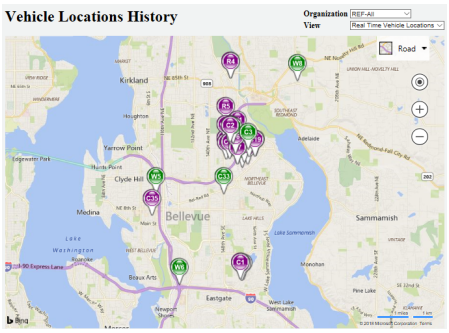


Figure 6: Real time vehicle locations

The overall activity of the entire fleet provides a good measure for the vehicle usage efficiency. Figure 7 shows the percent of active vehicles across the fleets of both catering events and water distribution. As can be seen from the figure, even at peak periods, the percentage of active vehicles is less than 30%. Our interviews with the human schedulers indicate that a large fraction of this inefficiency is due to the inability of humans to generate schedules where a single vehicle serves multiple buildings in a single trip. This is due to the complex requirements of the catering division, combined with lack of vehicle sharing among the various organizations. Thus, vehicles simply wait at various locations for several hours until specific events are over when they could be potentially used to serve other events. While these observations highlight the potential for reducing the fleet size, sophisticated optimization techniques are required to schedule the different tasks while ensuring that all requirements are met.

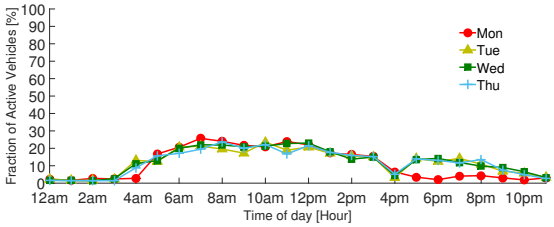


Figure 7: Active trucks as measured by our IoT devices.

2.3 The Optimization Problem

At a high level, an optimization problem involving the more complicated templates of Figure 1 can be described as a capacitated vehicle routing problem with time-windows, precedence, service duration, skills, workforce scheduling, driver assignment, and synchronization. For simplicity of exposition, we use the term "event" to refer to any type of operation (Templates 1-5). Given a set of events, the objective we are mainly focusing on is to maximize the number of events that are successfully served while minimizing the number of vehicles used; we also incorporate travel-time minimization, but give it less weight in the objective.

The formulation of such a problem must exhibit a high level of flexibility in choosing routes and must be able to easily express the extensive coordination requirements. We can formally define the problem using a MILP model on a time-expanded network. The nodes in our network correspond to each work-item in every event at every time point in the specified discretization T. For example, T might be defined by uniformly partitioning the time horizon, determined by the earliest and latest time-windows of the steps in the requested events. The resulting MILP formulation is fairly involved technically. As the problem description can be inferred from the template definitions (and accompanying descriptions), we defer the MILP model description to Appendix A. Unfortunately, this MILP model is intractable for anything but trivial problem instances. Hence, we develop UFSO – an algorithm that uses LP within a neighborhood-search meta-heuristic, allowing us to scale to practical problem sizes.

3 OPTIMIZING EVENT SCHEDULING

3.1 Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search (ALNS) is a meta-heuristic proposed by [30], originally designed for solving the Pickup and Delivery Problem with Time Windows. Starting from a feasible state, it iteratively *destroys* and *repairs* the current solution, searching for an improvement. The destroy-repair pair of operators is chosen from a customized set of procedures, and the probability of choosing a pair is based on its success in previous iterations (i.e., it adaptively updates the weights). The iterations are embedded in a simulated annealing framework. Termination is typically after either reaching a timeout or a pre-determined number of iterations.

A brief description of ALNS is provided below:

(1) Select a destroy operator D and a repair operator R as a function of their weights (details below).
(2) Using D, remove a number of requests from the schedules of the current solution.
(3) Using R, attempt inserting any unserved requests to the partial schedules obtained in Step 2 until obtaining a new feasible solution.
(4) If the new solution is better than the current one (the *incumbent*), accept the new solution. If the new solution is worse than the current one, accept it with certain probability. If the new solution is accepted, it becomes the incumbent.
(5) Update the selection weights for the operators.
(6) If the new solution is the best discovered so far, update the global best solution.

We next provide some additional details on the different selection mechanisms. Suppose the goal is to maximize an objective $f(\cdot)$. In each iteration, the solution s', having a worse objective value, may replace the incumbent s with probability $e^{(f(s')-f(s))/T}$. T is a temperature parameter, which is reduced after each iteration by a multiplicative factor $\alpha < 1$. A similar notion of gradual temperature cooling is used in other search-based heuristics, such as simulated annealing. The high-level idea here is to judiciously control the level of exploration, and to restrict attention to the best-valued solutions as time progresses. For the operator selection, we maintain weights w_{dr} for each pair of destroy-repair operators (d, r). In each iteration, each pair of operators is chosen with probability $p_{dr} = w_{dr} / \sum_{d' \in D} \sum_{r' \in R} w_{d'r'}$. Weights do not remain constant during the execution of the algorithm; they are updated based on the quality of the solutions that were generated through their use. The weight of the used pair of operators is updated via $w_{dr} := \gamma \cdot V + (1-\gamma) \cdot w_{dr}$, where $0 < \gamma < 1$ is a fixed constant, and V is a "score" that is determined by the iteration outcome (in decreasing quality order: new best solution found, better-than-current solution found, new solution was accepted, new solution was rejected); γ and the specific values used for V are parameters set prior to execution.

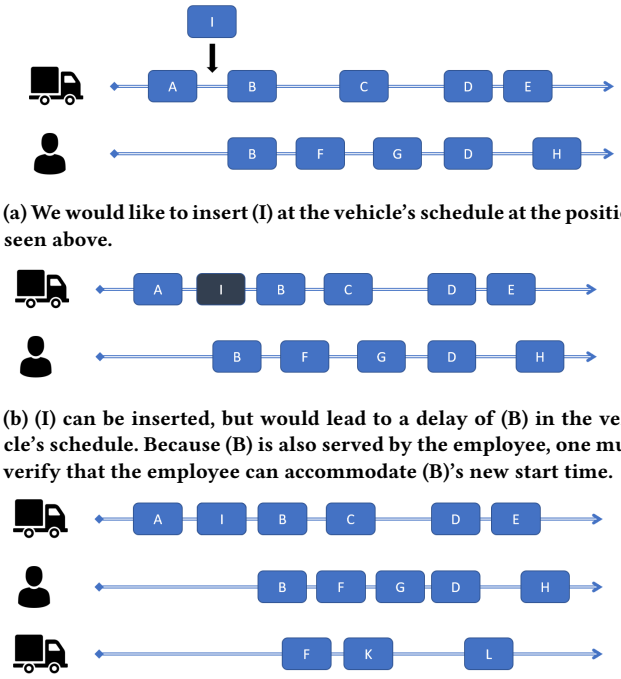
3.2 Coordination across Resources

3.2.1 The challenge. ALNS has been successfully applied to several variants of the Vehicle Routing Problem (VRP), however the event scheduling problem is inherently more complicated due to the coordination and synchronization across vehicles and employees. Consequently, the typical ALNS repair/destroy operators for VRP are not appropriate for our setting. In traditional VRPs, each customer request is uniquely served by a vehicle, i.e., vehicle schedules are independent. When a route is modified, no other schedule is affected, and so it is easy to check for validity.

However, in the catering domain, resources are coupled through the tasks they need to satisfy. Suppose the work-item W depends on two resources, R_1 and R_2 (simultaneously). If an additional task is assigned to R_1, which causes a feasible delay to W, it is entirely possible that a completely different task (assigned to R_2), becomes infeasible due to this delay. See Figure 8 for a concrete example. In this example, the insertion of one work-item creates a domino effect of changes on multiple resources. More generally, such domino effect might result in an infeasible schedule. Unfortunately, it is not straightforward to check whether an insertion of a work-item results in a feasible schedule. We propose an efficient method based on Linear Programming (LP), which we describe next.

3.2.2 The method: Global Schedule Coordinator (GSC). The idea behind our approach is to *decouple* the combinatorial decisions (e.g., which resource performs which work-item) from the timing considerations (e.g., time-window and coordination constraints). This separation enables us to design a tractable algorithm for checking the feasibility of an entire solution; we call this algorithm the *Global Schedule Coordinator (GSC)*.

Let K be the set of resources available in our system. Our combinatorial solution gives the schedule S^k for each resource $k \in K$. Each schedule is an ordered sequence of work-items, but can also be viewed as a list of pairs of consecutive work-items (i.e., $S^k \subset W \times W$).



(a) We would like to insert (I) at the vehicle's schedule at the position seen above.

(b) (I) can be inserted, but would lead to a delay of (B) in the vehicle's schedule. Because (B) is also served by the employee, one must verify that the employee can accommodate (B)'s new start time.

(c) Even if the employee can serve (B) at its new time, we now need to adjust the timing of an additional work-item (F).

Figure 8: Domino effect due to work-item insertion.

Given these schedules, GSC will output a suitable timing (e.g., start-time for each work-item) or otherwise indicate infeasibility, by solving the Linear Program (1)-(6). The variables $\hat{t}_w \geq 0$ correspond to the start-time of each work-item $w \in W$. Since the main goal of GSC is to verify feasibility, the objective function (1) can, in principle, be arbitrary. However, to find compact timings for schedules that reduce idle time, we minimize the difference (in time) between the first and last work-item of each event $e \in E$. That is, $\hat{t}_{w(e)}$ and $\hat{t}_{w'(e)}$ denote the start-time of the first and last work-item of e, respectively. We assume such ordering always exists and each event has at least two work-items, as is the case in all templates (Fig. 1). The constraints ensure that all time requirements are taken into account. In particular, constraints (2) and (3) enforce that the time windows (t_w, t'_w) for starting each work-item $w \in W$ are satisfied. Constraints (4) take into account precedence requirements among work-items, with l_w denoting the duration of $w \in W$, and P_w the set of work-items dependent on completing w. Finally, (5) ensures that each resource schedule S^k accounts for both work duration and travel time, $\tau_{(w_i, w_{i+1})}$, between consecutive work-items. Coordination of multiple resources is implicitly enforced via shared timing variables $\hat{t}_w$; if w requires multiple resources, (the same) $\hat{t}_w$ would appear in multiple constraints (5) for the different resources involved. We note that additional application-specific timing constraints can be easily incorporated into GSC, such as enforcing a minimum break between consecutive work items, or a maximum period that a vehicle can be away from the base location.

$$\min \sum_{e \in E} \left(\hat{t}_{w'(e)} - \hat{t}_{w(e)} \right) \tag{1}$$

$$\hat{t}_w \geq t_w \qquad \forall w \in W \tag{2}$$

$$\hat{t}_w \leq t'_w \qquad \forall w \in W \tag{3}$$

$$\hat{t}_{w_i} + l_{w_i} \leq \hat{t}_{w_j} \quad \forall w_i \in W, w_j \in P_{w_i} \tag{4}$$

$$\hat{t}_{w_i} + l_{w_i} + \tau_{(w_i, w_{i+1})} \leq \hat{t}_{w_{i+1}} \quad \forall k \in K, (w_i, w_{i+1}) \in S^k \tag{5}$$

$$\hat{t}_w \in \mathbb{R}_+ \qquad \forall w \in W \tag{6}$$

It is easy to see that feasible timings remain feasible when work-items are removed (i.e., by destroy operators). Thus, GSC is only required when inserting work-items (i.e., repair operators). As described above, if an insertion is feasible, GSC will output the timing of the work-items. However, if the LP is infeasible, the insertion is aborted. From a practical perspective, we note that each update of the schedule typically adds or removes only a small number of constraints. Consequently, one can use the previously obtained solution as warm start to an LP-solver, which results in very efficient running times, even in case of infeasibility. To be able to consistently leverage warm-start, we run GSC also for deletions (although it is not necessary for feasibility). See Section 3.4 for additional algorithmic enhancements.

3.3 UFSO Design

In this section, we focus on the combinatorial decisions, which are later verified by the GSC. We start with the design of the various repair and destroy operators of UFSO, and then describe how these operators are used in conjunction with GSC.

3.3.1 Operators. Operators must be simple to allow for many quick iterations, and they should include some form of randomness to efficiently explore the search space. We design the following operators based on these principles.

Destroy operators. RANDOM EVENT REMOVAL: chooses a random number, selects uniformly the corresponding amount of events, and then removes all work-items (in these events) from their associated schedule(s). VEHICLE SCHEDULE REMOVAL: instead of removing events at random, this operator selects a random number of vehicles, and clears their entire schedule. The vehicles are chosen in an increasing order of their schedule load; the idea here is to have a manageable number of work-items, which would likely be accommodated later by the remaining vehicles.

Repair operators. EARLIEST SCHEDULE INSERTION: for each unserved work-item, this operator iterates over all compatible vehicles and employees in a random order, and tries to insert the work-item in their schedule at the earliest valid position; accounting for time-windows helps to quickly skip over invalid insertions. BEST SCHEDULE INSERTION: iterates over all compatible vehicles in a random order, and tries to insert the work-item in their schedule at the position that minimizes its travel time. OPTIMAL RESOURCE REUSABILITY INSERTION - EARLY: it maximizes the load of already active resources, before assigning any work-items to resources with empty schedules. In particular, for each work-item, the active vehicles are examined in random order; for a given active vehicle, the operator attempts to insert the work-item at the earliest valid position. OPTIMAL RESOURCE REUSABILITY INSERTION - BEST: similar to the previous operator, it first tries to maximize the load of already active resources, but the work-item is inserted at the schedule position that minimizes the vehicle's travel-time. Note that we don't use these operators for employee assignment, as we often wish to have balanced workload among the employees.

3.3.2 Initialization of UFSO. We initialize the GSC by adding all variables $\hat{t}_w$ for each work-item $w \in W$. We also add the time window constraints (2) and (3), as well as the precedence constraints between work-items of the same event that have explicit ordering requirements (4). Note that these constraints are static, i.e., independent of the schedule. Finally, starting with an empty schedule, we apply the repair operators until a feasible solution is obtained; this would be the initial solution for UFSO.

3.3.3 UFSO iteration. Each iteration of the algorithm follows the steps outlined in Section 3.1. We next provide additional necessary details for Steps 2 (destroy) and 3 (repair); we start by describing the repair step, as the destroy step is simpler and would be better understood after reading about the repair step.

Step 3: Repair. Let S denote the current plan which consists of resource schedules that may only support a subset of the events. Starting with a random permutation of all the unsatisfied events, we try to add them to S, one work-item at a time, until all their requirements have been met. In particular, for each unsatisfied event e, Algorithm 1 InsertEvent(S, e) is used to determine if there are available vehicles, drivers, and employees (caterers, etc.) to cover the needs of all of its work-items. If at least one of the required resources is missing for any of the work-items, the insertion of e fails, and the entire event is removed from S.

Algorithm 1: InsertEvent(S, e)

```
for w in e.workitems
  if not InsertVehicle(S, w) or not InsertDriver(S, w) or
  not InsertEmployee(S, w)
    return false;
return true;
```

Algorithm 2 presents the vehicle search for work-item w when the selected repair operator inserts steps at the earliest possible schedule position (operators that insert at the best position are based on similar ideas). The first action is to retrieve the coupled (dependent) work-items of w that must be assigned to the same vehicle. An example includes loading items at the depot and unloading them at the event location. Then, SelectVehicles() selects and sorts the candidate vehicles in accordance with the chosen repair operator (random vehicle selection or optimal resource reusability).

For each vehicle, we iterate sequentially through its schedule and, for each position, TryInsert() checks if the current vehicle load and capacity suffice for the new items, as well as if the time window of the current work-item is compatible with the time windows of the preceding and subsequent work-items in the schedule. It also tries to find compatible positions for all coupled work-items under consideration. We then proceed to UpdateGSC() which updates the schedule precedence constraints (5). In particular, if a work-item w_i is added at the i-th position of a resource schedule, we add the constraint (5) between w_i and w_{i+1} (if w_i is not the last step), in order to ensure that there is enough time to complete w_i and travel to w_{i+1} on time. Similarly, we add constraint (5) between

Algorithm 2: InsertVehicleEarliest(S, w)

```
workitems ← GetCoupledWorkItems(w)
vehicles ← SelectVehicles()
for v in vehicles
  if NotVehicleShortage(workitems)
    break
  for index i in 0 to v.ScheduleLength
    canInsert ← v.TryInsert(w, i, workitems)
    if not canInsert
      continue
    feasible ← UpdateGSC(v, workitems)
    if not feasible
      RevertChanges(w, i, workitems)
      continue
if NotVehicleShortage(workitems)
  return true
return false
```

w_{i-1} and w_i, and remove the constraint (5) between w_{i-1} and w_{i+1}, if applicable. Removing redundant constraints is important in order to make it easier to insert and remove steps in future iterations, as well as to keep the model at the smallest possible size. If the GSC cannot find an optimal solution after the update, RevertChanges() brings the schedules and the LP model to their pre-insertion state.

Once the vehicle requirement has been completed successfully, we search similarly for a driver for the vehicle we just selected, as well as for employees that are needed for the work-item. To simplify the search, we assume that employees who are not driving a vehicle can be transported via passenger vans (as is the case in Compass), which are not explicitly scheduled in our model.

Step 2: Destroy. This step removes a subset of work-items in accordance with the chosen destroy operator. The work-items are removed from all resource schedules that they participate, and the GSC is updated accordingly in order to remain up-to-date with the current solution. The updates are analogous to the ones we described for Repair, e.g., if w_i is removed and it is not the first/last work-item, we add the precedence constraint for w_{i-1} and w_{i+1}.

3.4 Performance Enhancements

We conclude this section by briefly describing some algorithmic enhancements that we use for accelerating the run-time of UFSO.

Accelerated abort. For any insertion, we perform some quick checks that may lead to a quicker determination that the solution is infeasible. In particular, we examine whether the individual schedules are feasible. Because we examine each resource in isolation, we do not need to solve an LP, but rather go over the work-items in sequential order and check that time windows and precedence constraints are satisfied. If a single resource fails this check, insertion is aborted. Otherwise, GSC solves the LP as described in Sec. 3.2.

Batch insertions. The idea behind batch insertions is to use the GSC in order to confirm multiple insertions with a single invocation to the LP. In particular, we perform several accelerated insertions as described above, and if they all pass, we update the LP model with all changes, and run it once. If a feasible solution exists, then all pending insertions are added successfully to the solution, otherwise they are all rejected (since it is not known which insertions affected feasibility). The batch size naturally has an impact on the effectiveness of batch insertions. Intuitively, as the schedule becomes "busier" it is harder to insert large batches simultaneously. Based on this intuition, UFSO determines the batch size adaptively, using the following simple approach. UFSO initiates the batch size to a large value, and reduces it gradually in case GSC rejects the solution; the reduction rate is a tunable parameter.

4 EXPERIMENTAL RESULTS

In this section we demonstrate the efficacy of our methods by employing them on a week-worth of real event data obtained from Compass for two different divisions – *food catering* (FC), and *water distribution* (WD); see Section 2.1 for the operational characteristics for both divisions.

4.1 Experimental Setup

Problem instances. In our default setting, each problem instance consists of a one-day worth of request data. The only exception is §4.2.1, in which we construct small instances from existing data to compare against a lower bound.

Objective. Given the substantial vehicle costs, a key goal for Compass is to better utilize their vehicles which would lead to fleet size reductions. We choose to add the number of served events in the objective as well (and not in the constraints) as it allows UFSO to expand its search on a larger feasible solution set. Therefore, we maximize a hierarchical objective consisting of the number of served events, the number of vehicles, and their travel time, with coefficients set to 100, -1, and -0.0001, respectively.

UFSO parameters. The initial temperature parameter is set to 10% the objective value of the initial solution of the algorithm. The temperature is multiplied by $\alpha = 0.999$ in each iteration. The adaptive weights are all initialized as 1, and at each new solution we reward the generation of a new best solution with 5, a better solution than the current with 2, an accepted solution with 1, and a rejected with 0. The value of γ for the update of the weights is set to 0.2. In each destroy stage of the algorithm, we allow up to 20% of orders or vehicle schedules to be removed. Finally, the batch size in each iteration is initialized to 10 events, as experiments showed that larger values were not beneficial, with a reduction rate of 0.5. We impose a global timer of 10 minutes for each run. We obtain the travel times between building locations through Bing Maps queries.

4.2 Algorithm Evaluation

In this subsection, we evaluate the performance of UFSO on the catering request data. Ideally, we would like to compare the results of our algorithm against an optimal solution for the problem. Such a solution can be theoretically obtained using our MILP formulation (Appendix A). Unfortunately, the high complexity of this problem does not allow generating optimal solutions in reasonable running times, even for small instances. Hence, we will compare our results against a lower bound for the problem, which is an adaptation of the well-studied Pick-up and Delivery Problem. This problem is a well-known NP-hard problem, being a generalization of the Traveling Salesman Problem. Consequently, even the lower bound

Instances		1st Obj. (#Events)		2nd Obj. (#Vehicles)		3rd Obj. (Travel time)		Run times (s)	
Events	#Instances	LB	UFSO Gap	LB	UFSO Gap	LB	UFSO Gap	LB	UFSO
3	100	3	0.0% ± 0.0%	1.01	0.0% ± 0.0%	43.12	0.0% ± 0.0%	0.23	0.27
4	100	4	0.0% ± 0.0%	1.07	0.0% ± 0.0%	60.06	0.5% ± 2.8%	1.24	1.10
5	100	5	0.0% ± 0.0%	1.19	0.0% ± 0.0%	73.24	0.6% ± 1.9%	22.20	3.07
6	100	6	0.0% ± 0.0%	1.26	0.0% ± 0.0%	86.16	0.7% ± 1.7%	479.76	7.68

Table 1: Comparison of UFSO with the lower bound. The second column shows how many instances were generated for each size. The solution is evaluated upon three metrics (considered hierarchically): number of events served, number of vehicles used, travel time. For each of them, we present the average value of the lower bound (LB) and the average gap of UFSO followed by the standard deviation. The last two columns report the average running times.

cannot be computed exactly in reasonable time for large instances. We therefore carry out our evaluation in three parts.

(1) In Section 4.2.1, we show that UFSO is able to generate near optimal solutions by comparing it to the lower-bound solution.
(2) As mentioned earlier, the unique combination of characteristics of catering events precludes using as-is other solution approaches for similar problems. Nonetheless, in Section 4.2.2, we compare UFSO against a natural heuristic for the problem, which mimics the scheduling principles used by Compass dispatchers.
(3) In Section 4.2.3, we compare the fleet sizes used by UFSO with those used by Compass in practice. The latter are obtained by analyzing historical GPS traces.

4.2.1 Comparison with the lower bound. We first briefly describe how we construct the lower bound for our problem; see Appendix B for a detailed description. The main idea behind the lower bound is to omit certain constraints that appear in the original problem, which will make the mixed integer problem easier to solve. In particular, the lower bound formulation excludes the workforce and coordination requirements, while keeping only the vehicle routing aspect. The resulting problem is akin to the Pick-up and Delivery Problem; accordingly, we adopt ideas from related work [29] to derive an adequate formulation for the optimization problem.

Experiment setup. A day worth of events cannot be tackled by the lower-bound formulation. Accordingly, we synthetically generate smaller instances by taking random subsets of the original instances. Since the number of events we select is rather small, the problem might become trivial if the events are sufficiently spread out throughout the day. To avoid this, we make sure that the event start times in each instance are within a window of three hours. This makes the solution process more challenging as events are likely to overlap, hence compete for the same resources at similar times. Our resources for these instances consist of two vehicles and four employees (which suffice to complete all events). We set a time limit of one-hour for the lower bound solution, and exclude instances which are not optimally solved by that time limit. For UFSO, since the instances are small, we set a time limit of one minute per instance.

Results. The results are summarized in Table 1. We observe that UFSO is able to generate optimal solutions in terms of the two more important objectives (number of events served and number of vehicles used). In the third objective (travel time), we observe very small average gaps, which suggests that our algorithm generates near optimal solutions. Importantly, UFSO obtains the solution much faster than the lower bound. The difference in running times becomes larger with the instance size: With six events, the lower bound calculation has an average running time of 480 seconds, while UFSO produces its solution in less than 8 seconds on average. Unfortunately, instances with 7 events or more typically become intractable for the lower-bound, i.e., reach the time-out without converging (recall that each event consists of multiple steps that introduce a large number of variables in the formulation). Nonetheless, having a relatively steady low performance gap as we have increased the instance sizes from three to six events is a positive indication for the robustness of UFSO. Further, we note that the performance gap itself does not necessarily imply that UFSO obtains a sub-optimal solution; see Figures 9 and 10 for an example in which the gap is due to the additional complexity of coordinating resources and scheduling the workforce (aspects excluded from the lower-bound formulation).

4.2.2 Comparison against practically-inspired heuristic. Unfortunately, for our given request data, we could not obtain the corresponding schedule that was implemented by Compass. Nonetheless, to be able to compare our UFSO to Compass' current practice, we have interviewed several dispatchers to understand their thought process for producing schedules. We have concluded that current scheduling is based on grouping events in neighboring locations and typically pairing them with a fixed set of resources until their completion. Based on this intuition, we design a heuristic we term *Loop-Resource Decomposition (LRD)*; the full details can be found in Appendix C. In a nutshell, LRD first constructs *loops*, i.e., routes that start from the depot, "visit" a number of work-items, and return to the depot. As a second step, LRD assigns resources (vehicles and employees) to loops ensuring that overlapping loops cannot be served by the same resources. As a result, all resources that serve the same work-item stay together for the duration of the loop; we note that a similar principle of team formation has been used in field-service optimization [9, 20, 21]. The results we obtain for UFSO, including running times are summarized in Table 2, and the comparison with LRD appears in Fig. 11. We note that both methods served all events in all instances, hence the main comparison is with respect to vehicle count. We observe that UFSO requires substantially less vehicles to complete the same set of events. Interestingly, UFSO requires 18% more vehicle travel time on average compared to LRD (recall, however, that travel-time is the least important factor in the objective). Intuitively, a smaller fleet implies that each vehicle serves more steps, hence might require excess trips between locations.

4.2.3 Comparison with vehicle usage statistics. To provide additional evidence for the effectiveness of our approach, we have collected vehicle activity data for a period of one month, using our

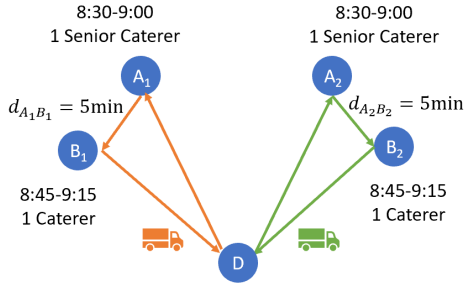


Figure 9: The time windows express when each step (A_1, A_2, B_1, B_2) of duration 15 minutes must start. This solution minimizes the vehicle travel time as obtained by the lower bound.

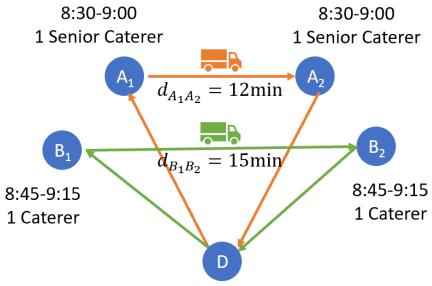


Figure 10: If there is a single Senior Caterer (SC), Fig. 9 is no longer feasible as it does not allow the SC time to go from A_1 to A_2. This is the optimal solution, requiring larger travel time.

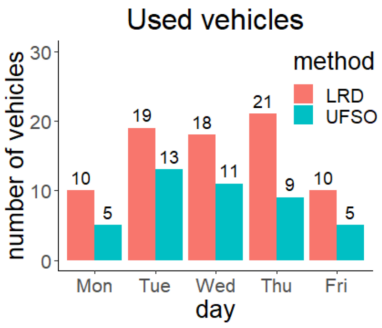


Figure 11: Used vehicles in UFSO compared to practically-inspired heuristic LRD.

Instances		UFSO		Run
Day	Events	Served	Vehicles	time (s)
Mon	107	107	5	174
Tue	167	167	13	237
Wed	156	156	11	113
Thu	162	162	9	193
Fri	81	81	5	566

Table 2: UFSO results on real catering instances.

IoT technology (see §2). We process the activity data to infer the distinct number of vehicles used per work day. Our analysis reveals that 21 vehicles have been used on average, a number much higher than our maximum usage (13). To make a concrete, yet conservative assessment on potential fleet size reductions, we also obtain from the data the 80-th percentile of vehicle count, which is 26 vehicles (that is, a fleet of 26 vehicles is sufficient for accommodating all events in 80% of the days). Thus, using UFSO has the potential to reduce fleet size by roughly 2x.

4.3 Towards Unified Operations

We conclude this section by examining the effect of using UFSO to *jointly* optimize the operations of multiple divisions. In particular, the input to UFSO is now the requests of both the catering and the water distribution divisions. The results are summarized in Table 3. We observe that we use at least two fewer vehicles than when we optimize for each division separately. To compare against current practice, we note that the water distribution currently utilizes 10 trucks daily. Thus, combined with the statistics from §4.2.3 the savings compared to a separate manual scheduling process for each division increase to 2.4× (15 vs. 36).

Day	Served	Vehicles used			
	requests	FC	WD	FC&WD	Savings
Mon	139	5	3	6	2
Tue	241	13	5	15	3
Wed	236	11	6	15	2
Thu	230	9	4	11	2
Fri	174	5	6	9	2

Table 3: Results for unified operations with UFSO.

5 RELATED WORK

Resource-constrained vehicle routing. The problem we consider in this paper can be modeled as a resource constrained routing and scheduling problem (see [25] and references therein). This is a generalization of the Vehicle Routing Problem (VRP) that combines routing and scheduling of resources (vehicles, employees) to serve customer requests with specific requirements and real-world needs. VRPs have been widely studied in the literature; see, e.g., [22, 23] for surveys on different variants and solution approaches. [6, 24] study VRP and scheduling models that include precedence and synchronization constraints but these studies provide less attention to the assignment of heterogeneous workforce, which highly complicates the underlying optimization problems. Other related work focuses on crew [11, 18] and workforce [8, 13] scheduling.

Related applications. The water distribution operations are closely related to the *Pick-up and Delivery* problem [4, 31]. The catering event scheduling is related to the *Technician Routing and Scheduling* problem, where skilled technicians travel to repair locations in order to perform a service within specific time windows (see, e.g., [9, 20, 33]). Typical technician scheduling problems can be modeled as Template 3 (see Figure 1). [9, 20, 21] consider team formation, however the assumption is that teams stay together for the entire day. In our models, employees are not bound as a team, e.g., different subsets serve different events. The *Home Health-Care Scheduling* problem focuses on generating shifts and schedules for nurses to visit client locations. The problem definition may involve time windows and synchronization, see [15] for a survey. However, an important distinction from our setting is that the transportation of items and equipment typically do not involve strict capacity constraints (e.g., as in capacitated vehicle routing problems). Similar types of problems appears in [16] in the context of airport operations, however this work does not consider item transportation. Food distribution has been considered in [2], however without the service aspect which requires skill-aware scheduling.

Techniques. Given the complexity of the above problems, especially the most complicated versions that are closer to ours, there are few exact approaches ([7] based on cutting planes, [28] based on branch and price). Most of the works in the literature focus on developing heuristics and meta-heuristics. [33] develops a greedy

heuristic, a local search heuristic, and a greedy randomized adaptive search procedure (GRASP) for the Technician Routing and Scheduling Problem. [9] solves the same problem using an ALNS approach, an approach that is also used in [10] to solve a Multi-Itinerary Optimization Problem. The same metaheuristic framework appears in our approach as well, but since our setting is much more complicated, more aspects need to be incorporated in order to obtain feasible and good quality solutions. [1] develops a Particle Swarm Optimization technique to schedule home health care personnel with homogeneous skill sets. Finally, [14] proposes a two-stage metaheuristic based on creating promising walking routes and optimizing the system using a tabu search framework.

Additional related domains. Our work is somewhat related to the Resource Constrained Project Scheduling Problem [19], although the geographical aspect herein is not explicit. In these problems, a project consists of a set of activities, each of which needs a certain amount of resources. Activities might have precedence constraints, and setup times might be used to represent traveling times among them. Finally, the Flexible Job Shop Scheduling Problem considers jobs as a sequence of operations, and each operation can be performed in one out of a set of compatible machines. This problem is generally solved using heuristics [5, 26].

6 CONCLUSION

In this paper, we introduce a unified optimization framework for onsite food-service operations. Our results with real workloads from Compass show substantial cost reductions for its catering division. Further, our framework allows different divisions to share resources and schedule their operations simultaneously, resulting in significant reductions in vehicle fleet size. We believe that the core ideas of our work apply beyond food-service, and can serve as basis for other complex field service operations that require routing and coordination across tasks and resources.

7 ACKNOWLEDGEMENTS

We are grateful to Tim O'Brien, the Director of Operations for the Compass Group's catering division (Eventions at Microsoft Campus, Redmond) and his team for tirelessly helping us in our partnership. We also thank Water World and Beverage divisions of Compass Group for their guidance.

REFERENCES

[1] Chananes Akjiratikarl, Pisal Yenradee, and Paul R Drake. 2007. PSO-based algorithm for home care worker scheduling in the UK. *Computers & Industrial Engineering* 53, 4 (2007), 559–583.

[2] Pedro Amorim, Sophie N Parragh, Fabrício Sperandio, and Bernardo Almada-Lobo. 2014. A rich vehicle routing problem dealing with perishable food: a case study. *Top* 22, 2 (2014), 489–508.

[3] Cynthia Barnhart, Amy M Cohn, Ellis L Johnson, Diego Klabjan, George L Nemhauser, and Pamela H Vance. 2003. Airline crew scheduling. In *Handbook of Transportation Science*. Springer, 517–560.

[4] Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. 2007. Static pickup and delivery problems: A classification scheme and survey. *Top* 15, 1 (2007), 1–31.

[5] Paolo Brandimarte. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research* 41, 3 (1993), 157–183.

[6] David Bredström and Mikael Rönnqvist. 2008. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research* 191, 1 (2008), 19–31.

[7] Paola Cappanera, Luis Gouveia, and Maria Grazia Scutellà. 2013. Models and valid inequalities to asymmetric skill-based routing problems. *EURO Journal on Transportation and Logistics* 2, 1-2 (2013), 29–55.

[8] J Arturo Castillo-Salazar, Dario Landa-Silva, and Rong Qu. 2016. Workforce scheduling and routing problems: Literature survey and computational study. *Annals of Operations Research* 239, 1 (2016), 39–67.

[9] Jean-François Cordeau, Gilbert Laporte, Federico Pasin, and Stefan Ropke. 2010. Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling* 13, 4 (2010), 393–409.

[10] Alexandru Cristian, Luke Marshall, Mihai Negrea, Flavius Stoichescu, Peiwei Cao, and Ishai Menache. 2019. Multi-Itinerary Optimization as Cloud Service (Industrial Paper). In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 279–288.

[11] Guy Desaulniers, Jacques Desrosiers, I loachim, Marius M Solomon, François Soumis, and Daniel Villeneuve. 1998. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In *Fleet Management and Logistics*. Springer, 57–93.

[12] Michael Drexl. 2012. Synchronization in vehicle routing – a survey of VRPs with multiple synchronization constraints. *Transportation Science* 46, 3 (2012), 297–316.

[13] Andreas T Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. 2004. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153, 1 (2004), 3–27.

[14] Christian Fikar and Patrick Hirsch. 2015. A matheuristic for routing real-world home service transport systems facilitating walking. *Journal of Cleaner Production* 105 (2015), 300–310.

[15] Christian Fikar and Patrick Hirsch. 2017. Home health care routing and scheduling: A review. *Computers & Operations Research* 77 (2017), 86–95.

[16] Martin Fink. 2016. *The Vehicle Routing Problem with Worker and Vehicle Synchronization: Metaheuristic and Branch-and-Price Approaches*. Ph.D. Dissertation. Technische Universität München.

[17] Chuhan Gao, Mehrdad Hessar, Krishna Chintalapudi, and Bodhi Priyantha. 2019. Blind Distributed MU-MIMO for IoT Networking over VHF Narrowband Spectrum. In *MobiCom*. 1–17.

[18] Asvin Goel and Stefan Irnich. 2016. An exact method for vehicle routing and truck driver scheduling problems. *Transportation Science* 51, 2 (2016), 737–754.

[19] Sönke Hartmann and Dirk Briskorn. 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207, 1 (2010), 1–14.

[20] Hideki Hashimoto, Sylvain Boussier, Michel Vasquez, and Christophe Wilbaut. 2011. A GRASP-based approach for technicians and interventions scheduling for telecommunications. *Annals of Operations Research* 183, 1 (2011), 143–161.

[21] Attila A Kovacs, Sophie N Parragh, Karl F Doerner, and Richard F Hartl. 2012. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling* 15, 5 (2012), 579–600.

[22] Rahma Lahyani, Mahdi Khemakhem, and Frédéric Semet. 2015. Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research* 241, 1 (2015), 1–14.

[23] Gilbert Laporte. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 3 (1992), 345–358.

[24] Ran Liu, Yangyi Tao, and Xiaolei Xie. 2019. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. *Computers & Operations Research* 101 (2019), 250–262.

[25] Dimitris C Paraskevopoulos, Gilbert Laporte, Panagiotis P Repoussis, and Christos D Tarantilis. 2017. Resource constrained routing and scheduling: Review and research prospects. *European Journal of Operational Research* 263, 3 (2017), 737–754.

[26] Ferdinando Pezzella, Gianluca Morganti, and Giampiero Ciaschetti. 2008. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research* 35, 10 (2008), 3202–3212.

[27] David Pisinger and Stefan Ropke. 2007. A general heuristic for vehicle routing problems. *Computers & Operations Research* 34, 8 (2007), 2403–2435.

[28] Matias Sevel Rasmussen, Tor Justesen, Anders Dohn, and Jesper Larsen. 2012. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research* 219, 3 (2012), 598–610.

[29] Stefan Ropke and Jean-François Cordeau. 2009. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* 43, 3 (2009), 267–286.

[30] Stefan Ropke and David Pisinger. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40, 4 (2006), 455–472.

[31] Martin WP Savelsbergh and Marc Sol. 1995. The general pickup and delivery problem. *Transportation Science* 29, 1 (1995), 17–29.

[32] Sapna Thottathil and Annelies Goger (Eds.). 2018. *Institutions as Conscious Food Consumers*. Academic Press.

[33] Jiyang Xu and Steve Y Chiu. 2001. Effective heuristic procedures for a field technician scheduling problem. *Journal of Heuristics* 7, 5 (2001), 495–509.

A FORMAL PROBLEM DESCRIPTION

In this section, we formally define the problem using a MILP model on a time-expanded network. The nodes in our network correspond to each work-item in every event at every time point in the specified discretization T. For example, T might be defined by uniformly partitioning the time horizon, determined by the earliest and latest time-windows of the steps in the requested events.

Let E be the set of requested events, and W_e be the set of work-items that comprise the event $e \in E$. We define $W = \cup_{e \in E} W_e$ as the set of all work-items. Work-item $w \in W$ has service length $l_w \in \mathbb{Z}_+$, earliest start-time $t_w \in \mathbb{Z}_+$, and latest start-time $t'_w \in \mathbb{Z}_+$ all measured in minutes. For notational convenience we consider each depot as a single work-item in its own event. Each transportable resource (required items, vehicles, crew members) will appear at their associated depot at the beginning of their time-window.

Let R_w be the set of items (e.g., food, tools) required to perform work-item $w \in W$. We define $R = \cup_{w \in W} R_w$ as the set of all required items. Items require transport via a vehicle, and multiple items can be consolidated together. We assume that an item cannot be split, however this can be loosely supported by defining several smaller items as input. Each item $r \in R$ requires μ^r units of vehicle capacity. Let V be the set of vehicles. Vehicles are non-homogeneous and have capacity μ_v for vehicle $v \in V$. Each vehicle requires at least one driver. Let C be the set of crew members, and let $C(v)$ be the set of crew members that can drive vehicle $v \in V$. We assume that crew members who are not driving a vehicle can be transported via a separate system (i.e. corporate shuttle, personal vehicle, ride-share), which is not explicitly scheduled in our model – however this could be easily supported as an extension.

We define $K = R \cup C \cup V$ as the set of transportable resources. Each transportable resource $k \in K$ requires its own subset copy of the time-expanded network, and is defined as $\mathcal{G}_T^k = \left(\mathcal{N}_T^k, \mathcal{A}_T^k\right)$. That is, redundant nodes and arcs specific to k may be removed to reduce problem size. There are three types of arcs connecting our nodes: dispatch, holding, and activation $\left(\mathcal{W}_T^w\right)$. Dispatch arcs connect work-items $w_1, w_2 \in W$ with $w_1 \neq w_2$ such that a dispatch at time t from w_1 will arrive at w_2 at time $t + \tau_{(w_1, w_2)}$. We define $\mathcal{D}_T$ as the set of all dispatch arcs. Holding arcs connect the same work-item forward through time, for example, the holding arc for work-item $w \in W$ at time t may be defined as $((w, t), (w, t+1))$. Every work-item $w \in W$ has a set of activation arcs, $\mathcal{W}_T^w$, these are a special kind of arc that indicate when a work-item is performed, and are explicitly used for synchronization. It has a similar definition to a holding arc, however it incorporates service time (which may be zero), i.e., $((w, t), (w, t + l_w))$ for $t \in T$. Each transportable resource $k \in K$ has an origin $o^k \in \mathcal{N}_T^k$, and destination $d^k \in \mathcal{N}_T^k$.

In addition to items (e.g., food, tools), a work-item may require a vehicle (e.g., loading/unloading) or multiple crew members with a certain set of skills. Let W_V be the set of work-items that require a vehicle for activation. Let S_w be the set of required skill sets (including level) for work-item $w \in W$. Each element $s \in S_w$ is a set of skills and associated level. The skills and level that crew member $c \in C$ can perform for work step $w \in W$ is defined by $S_w(c)$. Finally, each work-item may require other work-items to have already been completed before its execution may start. To allow these precedence constraints, let P_w be the set of work-items that can begin only after work-item w has been completed.

Decision variables. We define flow variables $x_a^k \in \{0, 1\}$ for each arc $a \in \mathcal{A}_T^k$ in the time-expanded network of resource $k \in K$. When $x_a^k = 1$ it is said that resource k will travel on arc a – this could be a dispatch, holding or activation. Auxiliary activation variables $y_a^w \in \{0, 1\}$ are defined for work-item $w \in W$ with $a \in \mathcal{W}_T^w$. Event indicator variables $z_e \in \{0, 1\}$ have $z_e = 1$ if event $e \in E$ is scheduled for service. Item packing variables $q_a^{v,r} \in \{0, 1\}$ assign the required item $r \in R$ to vehicle $v \in V$ along dispatch arc $a \in \mathcal{D}_T$. Driver variables $h_a^{v,c} \in \{0, 1\}$ assign driver $c \in C(v)$ to vehicle $v \in V$ across dispatch arc $a \in \mathcal{D}_T$. Skill matching variables $u_a^{c,s,w} \in \{0, 1\}$ indicate that crew member $c \in C$ services the work-item $w \in W$ by performing skill $s \in S_w(c)$ on activation arc $a \in \mathcal{W}_T^w$. Finally, the variables $g_v \in \{0, 1\}$ indicate whether vehicle $v \in V$ is used.

A.1 The MILP Model

$$\max \left\{ \sum_{e \in E} z_e, -\sum_{v \in V} g_v \right\} \tag{7}$$

Routing and timing

$$\sum_{a \in \delta_k^+(n)} x_a^k - \sum_{a \in \delta_k^-(n)} x_a^k = \begin{cases} +1 & n = o^k \\ -1 & n = d^k \\ 0 & \text{o/w} \end{cases} \quad \forall k \in K,\, n \in \mathcal{N}_T^k \tag{8}$$

Activation requirements

$$\sum_{a \in \mathcal{W}_T^w} y_a^w \leq 1 \qquad \forall w \in W \tag{9}$$

$$\sum_{w \in W_e} \sum_{a \in \mathcal{W}_T^w} y_a^w = |W_e|\, z_e \qquad \forall e \in E \tag{10}$$

$$\sum_{a \in \mathcal{W}_T^w} x_a^r \leq 1 \qquad \forall w \in W,\, r \in R_w \tag{11}$$

$$\sum_{r \in R_w} x_a^r = |R_w|\, y_a^w \quad \forall w \in W,\, a \in \mathcal{W}_T^w \tag{12}$$

$$\sum_{v \in V} x_a^v = y_a^w \quad \forall w \in W_V,\, a \in \mathcal{W}_T^w \tag{13}$$

$$\sum_{c \in C} \sum_{s \in S_w(c)} u_a^{c,s,w} = |S_w|\, y_a^w \quad \forall w \in W,\, a \in \mathcal{W}_T^w \tag{14}$$

$$\sum_{s \in S_w(c)} u_a^{c,s,w} = x_a^c \quad \forall w \in W,\, a \in \mathcal{W}_T^w,\, c \in C \tag{15}$$

$$\sum_{c \in C:\, s \in S_w(c)} u_a^{c,s,w} \leq 1 \quad \forall w \in W,\, s \in S_w,\, a \in \mathcal{W}_T^w \tag{16}$$

$$\sum_{a = ((w,t),(w,t+l_w)) \in \mathcal{W}_T^w} (t + l_w)\, y_a^w \leq \sum_{a' = \left((w',t'),(w',t'+l_{w'})\right) \in \mathcal{W}_T^{w'}} t'\, y_{a'}^{w'} \quad \forall w \in W,\, w' \in P_w \tag{17}$$

Vehicle packing and driver requirements

$$q_a^{v,r} \leq x_a^v \qquad \forall v \in V,\, r \in R,\, a \in \mathcal{D}_T \tag{18}$$

$$\sum_{v \in V} q_a^{v,r} = x_a^r \qquad \forall r \in R,\, a \in \mathcal{D}_T \tag{19}$$

$$\sum_{r \in R} \mu^r q_a^{v,r} \leq \mu_v \qquad \forall v \in V,\, a \in \mathcal{D}_T \tag{20}$$

$$\sum_{v \in V : c \in C(v)} h_a^{v,c} \leq x_a^c \qquad \forall c \in C,\, a \in \mathcal{D}_T \tag{21}$$

$$\sum_{c \in C(v)} h_a^{v,c} = x_a^v \qquad \forall v \in V,\, a \in \mathcal{D}_T \tag{22}$$

$$g_v \geq x_a^v \qquad \forall v \in V,\, a \in \mathcal{D}_T \tag{23}$$

The objective (7) maximizes the number of scheduled events, while minimizing the number of vehicles used. The flow constraints (8) ensures appropriate routes and timing for the resources. Equation (9) enforces at most one activation per work-item, and (10) specifies that an event can be scheduled at most once, and additionally requires all-or-none of the work-items to be performed. Equation (11) enforces at most a single activation per required item, while (12) requires all associated items for activation. For all work-items that require a vehicle, (13) ensures that a vehicle is available at activation. Workforce scheduling and support for skills are handled by (14), (15), and (16). Specifically, (14) ensures that all skills are fulfilled for activation; (15) enforces that a crew member can only fulfill a single skill; and, (16) a single skill can only be fulfilled by at most one crew member. Equation (17) defines work-item precedence. The remaining equations relate to vehicle packing and driving. Equation (18) requires a vehicle to transport each item. These items are packed into each vehicle by (19), and capacity is enforced by (20). Each vehicle requires capable driver, which is enforced by (21) and (22). Finally, (23) sets $g_v = 1$ if vehicle $v \in V$ is ever used.

B LOWER BOUND

This lower bound is based on the formulation that [29] develop for the Pickup and Delivery Problem. In order to transport items, the vehicles need to load them at the origin location and unload them at the destination. Assume the instance has n loading actions, denoted by indices $L = \{1, \dots, n\}$, and n unloading actions $U = \{n+1, \dots, 2n\}$, such that loading i corresponds to unloading $n+i$. We also consider an action 0 for the start of each vehicle route from the depot, and $2n+1$ for the return to the depot. Let $N = \{0, \dots 2n+1\}$. Binary variables x_{ij}^v are equal to 1 if vehicle v does action i and then action j, and 0 otherwise, and variables g^v are equal to 1 if vehicle v is being used, and 0 if it remains inactive for the whole instance. We also have continuous variables t_i^v for the start time of the visit of vehicle v to action i, and y_i^v for the load of the vehicle once the action at i is completed. For each action i, we use q_i for its required capacity (positive for pick-up, negative for drop-off), l_i for its duration, $[a_i, b_i]$ for its time window, τ_{ij}^v for the distance between the locations of i and j, and μ_v for the capacity of vehicle v. Finally, the parameter α is being used to set the hierarchy of the two objectives, and in the experiments we use $\alpha = 0.0001$.

$$\min \sum_{v \in V} g^v + \alpha \sum_{v \in V} \sum_{i \in N} \sum_{j \in N} \tau_{ij}^v x_{ij}^v \tag{24}$$

$$\text{s.t.} \sum_{v \in V} \sum_{j \in N} x_{ij}^v = 1 \qquad \forall i \in L \tag{25}$$

$$\sum_{j \in N} x_{ij}^v - \sum_{j \in N} x_{n+i,j}^v = 0 \qquad \forall v \in V, i \in L \tag{26}$$

$$\sum_{j \in N} x_{0j}^v = 1 \qquad \forall v \in V \tag{27}$$

$$\sum_{j \in N} x_{ji}^v - \sum_{j \in N} x_{ij}^v = 0 \qquad \forall v \in V, i \in L \cup U \tag{28}$$

$$\sum_{i \in N} x_{i,2n+1}^v = 1 \qquad \forall v \in V \tag{29}$$

$$x_{i0}^v = 0, x_{2n+1,i}^v = 0 \qquad \forall v \in v, i \in N \tag{30}$$

$$t_j^v \geq t_i^v + (\tau_{ij}^v + l_i) x_{ij}^v - b_i (1 - x_{ij}^v) \qquad \forall v \in V, i, j \in N \tag{31}$$

$$y_j^v \geq y_i^v + q_j x_{ij}^v - \mu_v (1 - x_{ij}^v) \qquad \forall v \in V, i, j \in N \tag{32}$$

$$y_j^v \leq y_i^v + q_j x_{ij}^v + \mu_v (1 - x_{ij}^v) \qquad \forall v \in V, i, j \in N \tag{33}$$

$$t_i^v + \tau_{i,n+i}^v \leq t_{n+i}^v \qquad \forall v \in V, i \in L \tag{34}$$

$$g^v \geq 1 - x_{0,2n+1}^v \qquad \forall v \in V \tag{35}$$

$$a_i \leq t_i^v \leq b_i \qquad \forall v \in V, i \in N \tag{36}$$

$$y_i^v \geq \max\{0, q_i\} \qquad \forall v \in V, i \in N \tag{37}$$

$$y_i^v \leq \min\{\mu_v, \mu_v + q_i\} \qquad \forall v \in V, i \in N \tag{38}$$

$$x_{ij}^v \in \{0, 1\} \qquad \forall v \in V, i, j \in N \tag{39}$$

$$g^v \in \{0, 1\} \qquad \forall v \in V \tag{40}$$

The objective minimizes the number of vehicles used and their travel time. Constraints (25) ensure that all pick-ups are done, and (26) that the same vehicle performs corresponding pairs of pick-ups and drop-offs. According to (27) all vehicles leave node 0 (schedule start), (28) enforces the flow conservation at the nodes, and (29) ensures that all vehicles go to node $2n+1$ (schedule end). No incoming flow exists for node 0 and no outgoing for node $2n+1$ (30). Constraints (31) allow enough time for the work duration and the travel time between consecutive work-items. (32) and (33) update the vehicle load, and (34) ensures each item pick-up precedes its drop-off. (35) turns g^v equal to 1 if a vehicle visits nodes besides 0 and $2n+1$. Finally, (36)-(38) are time window and capacity constraints.

C LOOP-RESOURCE DECOMPOSITION

The main steps are the following: (i) initialize with short loops, (ii) keep merging promising loops until no more merging is possible, (iii) assign resources to loops. The initialization is performed by generating one loop per event preparation (bring items and set up) and event completion (clean up and take items back). Each loop has an associated *cost* which we define as $V_\ell + 0.1E_\ell + 0.0001D_\ell$, where V_ℓ takes values 1, 2, 3 depending on the type of vehicle that is required (van, sprinter, truck), E_ℓ is the number of employees required for the loop, and D_ℓ is its duration. We then iteratively merge the pair of loops that leads to the best total cost, making sure that loops do not exceed a two hour duration since longer loops will restrict our flexibility during the assignment stage. The assignment is performed using the following Integer Program, here showing only vehicle assignments, as employee assignments are done in the same way. Variable z_e is equal to 1 if event e is served, g_v is 1 if vehicle v is used, w_ℓ is 1 if loop ℓ has the required resources, and $y_{\ell v}$ is 1 if vehicle v is assigned to loop ℓ (all are 0 otherwise). μ_v is the capacity of vehicle v, C_ℓ the capacity loop ℓ requires, and I the set of pairs of overlapping loops.

$$\max 100 \sum_e z_e - \sum_v g_v - 0.0001 \sum_{v,\ell} y_{\ell v} \tag{41}$$

$$\text{s.t. } g_v \geq y_{\ell v} \qquad \forall v, \ell \tag{42}$$

$$\sum_v \mu_v y_{\ell v} \geq C_\ell w_\ell \qquad \forall v, \ell \tag{43}$$

$$z_e \leq w_\ell \qquad \forall e, \ell, e \cap \ell \neq \emptyset \tag{44}$$

$$y_{\ell_1 v} + y_{\ell_2 v} \leq 1 \qquad \forall v, (\ell_1, \ell_2) \in I \tag{45}$$

$$g_v, z_e, w_\ell, y_{\ell v} \in \{0, 1\} \qquad \forall v, e, \ell \tag{46}$$