Partitioning Orders in Online Shopping Services

Sreenivas Gollapudi
Google
Mountain View, CA
sgollapu@google.com

Ravi Kumar
Google
Mountain View, CA
ravi.k53@gmail.com

Debmalya Panigrahi
Duke University
Durham, NC
debmalya@cs.duke.edu

Rina Panigrahy
Google
Mountain View, CA
rinap@google.com

ABSTRACT

The rapid growth of the Internet has led to the widespread use of newer and richer models of online shopping and delivery services. The race to efficient large scale on-demand delivery has transformed such services into complex networks of shoppers (typically working in the stores), stores, and consumers. The efficiency of processing orders in stores is critical to the profitability of the business model. Motivated by this setting, we consider the following problem: given a set of shopping orders each consisting of a few items, how to best partition the orders among a given number of shoppers working for an online shopping service? Formulating this as an optimization problem, we propose a family of simple and efficient algorithms that admit natural constraints such as number of items a shopper can process in this setting. In addition to showing provable guarantees for the algorithms, we also demonstrate their efficiency in practice on real-world data, outperforming strong baselines.

CCS CONCEPTS

•Information systems → Online shopping; •Theory of computation → Approximation algorithms analysis; •Software and its engineering → E-commerce infrastructure;

KEYWORDS

E-commerce; Vehicle routing; Order partitioning

1 INTRODUCTION

In an online shopping home delivery service (such as Amazon Fresh or Google Express), customers place online orders for items to be delivered at home, each order comprising one or more items from a certain store. These orders are sent in batches to the store where *pickers* pick the items in the orders physically from the store so as to be dispatched for shipping back to the customer. The job of picking the items in the collection of orders needs to be divided among a certain number of pickers. Since an order is directly sent for delivery after its items have been picked, it is important that an entire order is assigned to exactly one picker and is not split across multiple pickers. This ensures that the dispatching operation remains simple and efficient. Thus, the orders need to be partitioned among the pickers. Each picker is to be given a *pick-list* that contains a list of items s/he is supposed to pick by physically walking through the store. To keep the business model profitable, it is key to minimize the amount of walking required for picking items in a pick-list.

When a collection of orders at a store is partitioned among the pickers, it makes sense to partition these orders among the pickers so that 'similar' or 'nearby' orders get assigned to the same picker. Clearly, it would be best to partition orders in the way that two orders are together if the items in one order are physically near the items in the other order and can be picked easily without requiring too much of extra walking. E.g., it is better to assign an order of milk and cheese and another order of butter and yogurt to the same picker, since in most stores, these items are in the same or nearby aisles. Thus we want to partition the orders among the pickers so that the **sum** of tour lengths of all the pickers is minimized (the *sum objective*). In practice, one is also often interested in completing the picking of all the orders at the earliest possible time. If the completion time is to be optimized, then we get a variant of this problem where the **maximum** tour length among all the pickers is to be minimized (the *max objective*).

Our work. In this paper, we study order partitioning problems for both these natural objectives, and their variants. Our goal is to develop algorithms that are easy to implement in practice, have provable guarantees of performance, and are more effective than natural heuristics.

A first-cut approach to solving the order partitioning problem would be to replace all the items in one order by a single item that acts as a representative for the order (e.g., an item close to the geometric center of the actual items in the order) and solve the partitioning problem on the centers, i.e., partition the centers so as to minimize the sum of tour lengths over them. While visiting a center, one can extend the walk to pick all the items represented by it. However, it is easy to construct simple examples where this heuristic performs very poorly and does not have (constant-factor) approximation guarantees (Section 3).

Our main contribution is a simple 4-approximation algorithm for minimizing the sum objective over the tour lengths of all the pickers. This algorithm is based on the primal-dual LP-based approach for the Steiner forest problem [1, 18], and carefully combining it with a solution to the minimum spanning tree (MST) problem. For the max objective, we employ a heuristic on top of our primal-dual algorithm

CIKM'17, November 6–10, 2017, Singapore.
 ISBN 978-1-4503-4918-5/17/11.
DOI: http://dx.doi.org/10.1145/10.1145/3132847.3132903

so as to bound the size of each tour, and show that this additional optimization does not significantly affect the sum objective while significantly improving the max objective.

We also perform experiments on real-world data obtained from an online shopping service as well as from publicly available market basket datasets. Our experiments show that our algorithms perform much better than natural baseline algorithms for the partitioning problem.

2 RELATED WORK

Clustering of points into a small number of groups so that nearby points fall in the same group is a common task in many applications, and has been widely studied in the literature. A well-motivated variant is the problem of *clustering with constraints*, which has been studied in the data mining and machine learning communities [14, 15, 17]. In this setting, we have the usual clustering problem on a set of points, but certain constraints that force specific pairs of points to go to the same cluster, or to different clusters, must be satisfied (cf. [13]). While the constraints in our problem are of the first type—items in an order must not be split across shoppers—the clustering cost in our case is given not by the distances among the points but by the tour induced on the points. Clustering with constraints is typically solved by either modifying the clustering objective to penalize constraint violations or enforcing the constraints to be satisfied during the assignment step in the clustering. Our approach is also similar in spirit in that we enforce the constraints to be satisfied when we create the clusters. However, unlike the mostly heuristic work on clustering with constraints, we strive to obtain a principled approach for our problem.

Our problem can broadly be classified as a vehicle routing problem (VRP), an area that has been widely studied in the operations research and approximation algorithms communities (see [19, 28] for books on VRP in operations research; in approximation algorithms, some of the well-studied variants include orienteering [6, 7, 10], dial-a-ride [9, 20, 21], and capacitated VRP [8, 22, 26]). Technically, the closest problem to ours is that of Arkin, Hassin, and Levin [4] and others [3, 5, 16, 27] on the minimum and min-max vehicle routing problem (also called the multiple TSP or mTSP problem in the operations research literature). The problem setting resembles ours: given a set of points in a metric space and a certain number of vehicles with the requirement that each point has to be covered by a vehicle, minimize the average or maximum distance traveled by a vehicle. For these problems, the above papers obtain constant-factor approximations, where the constant is 3 or more. The crucial difference in our problem is that all points in an order must be visited by the same vehicle, i.e., an order cannot be split among multiple vehicles. This introduces heterogeneity in the connectivity requirement of the point set and renders the existing VRP algorithms unusable. We note that many other variants of VRP have also been studied, e.g., with service time windows, vehicle capacities, loading and unloading constraints, etc. However, to the best of our knowledge, ours is the first work to consider heterogeneous connectivity requirements of the point set, thereby requiring us to develop a new algorithm distinct from the existing VRP literature.

The traveling salesperson problem (TSP), a special case of VRP where there is only a single vehicle, is one of the most well-studied problems in combinatorial optimization [2]. There are natural versions of TSP where instead of points, we are given subsets of points. The goal is to obtain a tour with minimum cost and touch at least one point from each subset. This problem, called the *set TSP* or *generalized TSP*, has been studied in the operations research community [23, 25]. For the case of a single shopper, our problem simply asks for the shortest tour that picks all items across all orders, thereby resembling TSP. But, for multiple pickers, our problem deviates from the existing TSP literature because of the constraint that all items in the same order must be picked by the same picker.

Next, we comment on the relationship between our problem and the well-known Steiner forest problem. Recall that in the Steiner forest problem, one is given a collection of sets of points (terminals) and the objective is to find a Steiner forest of minimum cost so that all the terminals in one set are connected by a tree in the forest. The classical algorithm for this problem uses a linear program (LP) and its dual. The algorithm can be viewed as increasing dual variables and picking Steiner forest edges as the dual LP constraints become tight. Our problem differs from the Steiner forest problem in that not only are we required to connect all the items in one order but also have no more than a given number of trees (the number of pickers in the store). The second difference is that we are interested in computing a collection of tours as opposed to a forest.

Finally, while there has been a lot of work on market basket analysis [24], our work asks a completely new data processing question on market basket data.

3 PRELIMINARIES

3.1 Background and problem formulation

We now formalize the problem of designing tours for pickers in a store to collect the items in a set of orders so as to ensure that the total distance traversed by the pickers is minimized and all items in an order are assigned to the same picker.

Let $G = (V, E)$ be a weighted graph that represents the store map, where the vertices are the item locations and the weight $d_{u,v} \geq 0$ on an edge denotes the shortest walking distance between u and v. We are given $k \geq 1$, the number of pickers, and a set of orders each comprising a subset of store items. For ease of exposition, we identify each order by a distinct *color*. Let C denote the set of colors and therefore, each order is of the form $V_c \subseteq V$ for a unique $c \in C$. Our goal, then, is to design tours for the k pickers such that each order belongs to a single tour, with the sum objective or the max objective in mind.

In practice, it is desirable that the number of items be approximately balanced across the pickers. There are several possible ways to encode this requirement. Perhaps the most natural way would be to set an upper bound on the number of items that are picked by a picker. Unfortunately, adding such an upper bound constraint makes it difficult to obtain a provable guarantee on the approximation factor of the algorithm; instead, we give a heuristic solution for the problem of minimizing the maximum tour length among the pickers. In other scenarios, we would like each picker to pick *at least* a given number of items to recoup the overhead of engaging the picker. In this case, we encode balance requirements as a given lower bound $\ell > 0$ on the number of items that a picker has to pick. The interpretation is that no picker engaged by the algorithm

should have a sparse workload, which is consistent with a more balanced workload among the pickers. For this problem, we give an algorithm with a provable approximation factor. We describe this algorithm first, and then give the heuristic that minimizes the maximum tour length of the pickers. [1]

For technical ease, we would like each item to be unique to an order. In our graph formulation, this is easily enabled by replication: replace each item in each order by the pair (item, order) and connect these replicas with edges of length 0. It is easy to see that the overall cost remains unchanged by this transformation. Let V_c denote the vertices with color c (V_C is known as a *vertex color class*). After our transformation, the color classes become disjoint.

The problem we solve in this paper, dubbed the *balanced traveling k-salesperson (k-BTSP)* problem, is then to find at most k tours (cycles), each containing at least ℓ items, that cover all the vertices in the graph with the constraint that all vertices of a color class must belong to the same tour and with the objective of minimizing the sum (or the maximum) lengths of the tours. Our main technical contribution in this paper is to provide a simple, efficient, and provably good algorithm for the k-BTSP problem, where we optimize the sum objective, and a heuristic where we optimize the max objective.

3.2 A general partitioning framework

Before we delve into the technical details, we note that the k-BTSP problem is, in fact, a representative of a broader class of problems where a set of points with a distance metric is divided into color classes, and has to be partitioned into at most k subsets such that each vertex color class is contained in one of the subsets. Several problems can be defined in this framework, with different objectives. For instance, as we mentioned in Section 2, clustering problems naturally fit this framework—one can define k-means, k-median, or k-center problems with the constraint that every color class has to be contained in a single cluster.

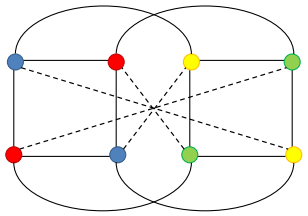

Figure 1: An instance with $k = 2$ on which the two-stage algorithm for k-BTSP produces an $\Omega(k)$ approximation. The solid edges are of unit length, while the dotted edges are of length $\alpha < 1$. For all other pairs of vertices that do not share an edge, their mutual distance is the length of the shortest path between them.

A natural algorithmic strategy for this category of problems is to replace each color class by a representative point, and use these representative points in solving the problem at hand. For instance, one can find the optimal representative point for each color class in the clustering problems, and run a standard clustering algorithm on the representative points. It is not hard to show that this strategy loses an approximation factor of at most 2 for the clustering objectives (in fact, for k-means there is no loss in the approximation factor). Similarly, for the k-BTSP problem, a natural strategy would be to construct optimal tours for individual color classes, contract the edges in these tours, and then find the optimal k tours on these contracted points. This clearly yields a feasible solution, but unfortunately, the length of the tours produced by this two-stage algorithm can be $\Omega(k)$ times the optimum.

To see, consider an instance (see Fig. 1) with k^2 colors, where each color class has k points. The color classes are partitioned into k group where in each group, the points of the color classes are placed on a cycle with unit length edges between adjacent points. The color classes of the point on any cycle are defined in a round-robin manner, i.e., a cycle can be further partitioned into k contiguous segments, each segment containing one point from every color class in that cycle. Furthermore, the color classes are grouped into k groups, with one color from each cycle in a group, and a complete graph of edge length $\alpha < 1$ is defined on the points belonging to different cycles if they are in the same group. All other distances are defined by the shortest paths of this graph. The optimal solution is to define each cycle as a tour, and this has a total length of k^3. However, the algorithm outlined above will first construct tours of length k^2 for each color class, contract these tours, and then construct k tours, one for each group. The total length of these tours is $\Omega(k^4)$, thereby yielding an approximation factor of $\Omega(k)$. Our algorithm (Section 4) will instead give a factor 4 approximate solution to this instance.

4 ALGORITHMS

In this section we present our main algorithm to solve the k-BTSP problem. The overall algorithm uses some closely related problems as building blocks. First, we define these problems.

Balanced Steiner k-forest (k-BSF). The input comprises a graph $G = (V, E)$ with edge costs where the vertices are partitioned into color classes $\{V_c : c \in C\}$. For given parameters k and ℓ, the goal is to find at most k trees, each containing at least ℓ vertices, that cover all the vertices in the graph. The constraint is that all vertices of a color class must belong to the same tree, and the objective is to minimize the sum of costs of edges in the trees.

Balanced Steiner forest (BSF). The input comprises a graph $G = (V, E)$ with edge costs where the vertices V are partitioned into color classes $\{V_c : c \in C\}$. The goal is to find a minimum cost spanning forest that connects all vertices in every color class (i.e., every vertex color class is contained in some tree of the forest). In addition, the forest must satisfy the balance constraint—every tree in the forest must contain at least ℓ vertices, for a given balance parameter $\ell > 0$. This is identical to the k-BSF problem without the constraint that the number of trees in the spanning forest cannot be more than k.

k-spanning forest (k-SF): The input comprises a graph $G = (V, E)$ with edge lengths. The goal is to find a minimum cost spanning forest with at most k trees, for a given parameter k. This is identical

[1] We remark that the algorithms in this paper can be easily adapted to a similar condition on the minimum number of orders (instead of items) that every picker must serve. To avoid confusion, however, we will only impose this condition on the number of items throughout the rest of the paper.

to k-BSF problem without the inseparability constraints on the vertex color classes and the balance constraints.

Fig. 2 gives the overall algorithm for the k-BTSP problem using an algorithm for the k-BSF problem (on the same input instance), which in turn, comprises two algorithms for the BSF and k-SF problems. The k-SF algorithm simply removes the most expensive $k-1$ edges from a minimum spanning tree of the graph (which can be computed using, e.g., Kruskal or Prim's algorithm [12]). This leaves us with the task of describing the algorithm for the BSF problem.

An algorithm for the BSF problem. The BSF problem combines two sets of constraints—that every vertex color class must belong to a single tree in the spanning forest (we call these Steiner forest constraints) and every tree contains at least ℓ vertices (we call these balance constraints). If these constraints are considered in isolation, then the corresponding problems have existing primal dual algorithms (Goemans and Williamson [18], see also Agrawal, Klein, and Ravi [1]), each with an approximation factor of 2. Using these algorithms separately would yield an approximation factor of 4 for the BSF algorithm; instead, we show that a refined combination of the algorithms yield an approximation factor of 2 for the BSF problem.

First, we define a linear program (LP) to represent the BSF problem. Let x_{uv} be the indicator variable for whether edge (u, v) is in the solution or not. To simplify notation, we denote by $\mathbf{S}$ the cuts that separate a vertex color class (we call these *separating cuts*) or contain less than ℓ vertices:

$$\mathbf{S} := \{S \subseteq V : \exists c \in C, 0 < |V_c \cap S| < |V_c|\} \cup \{S \subseteq V : |S| < \ell\}.$$

We call $\mathbf{S}$ the set of *demand cuts*. The LP is now given in Fig. 3(a). If we insist that $x_{uv} \in \{0, 1\}$, then the formulation is an integer linear program, which exactly captures the problem but is NP-hard to solve. Instead, we use the fractional relaxation given in Fig. 3(a), which can be solved in polynomial time, in our algorithm. Our algorithm is a primal dual algorithm and hence requires us to define the dual LP. To simplify notation, we define:

$$\mathbf{S}_{uv} = \{S \in \mathbf{S} : |\{u, v\} \cap S| = 1\}.$$

The algorithm has two phases, namely, a "forward add" phase followed by a "reverse delete" phase. Initially, the primal solution X is empty and the *active duals* are the singleton vertices $\{\{v\} : v \in V\}$. At any intermediate stage, the active duals correspond to the demand cuts among the connected components of the primal solution X. We also call the other (non-demand) connected components of X *inactive* cut duals. The active duals are raised uniformly until one of the dual constraints becomes tight. We view this as a time-indexed process where the active duals increase at unit rate. At any time t, we denote the current definition of X by X_t. When a dual constraint becomes tight, the edge corresponding to the tight constraint is added to the primal solution X, and the components and active cut duals are recomputed. (See Fig. 4(a) for a depiction of the graph during the forward add phase.) The forward add phase ends when there is no active dual among the components of X. Let T be the time index at this point. Clearly, X_T is a feasible solution for the k-BSF problem instance.

In the reverse delete phase, the algorithm defines the final solution F. To this end, F is initialized to X_T and edges in F are considered in reverse order of their addition to X (in the forward delete phase). In other words, the edge added to X at time T is considered first and that added at time 0 is considered last in the reverse delete phase. When an edge is considered, it is removed from F if such removal preserves the feasibility of F as a solution for the k-BSF problem instance. We view the reverse delete phase as a time-indexed process as well, but one where the time index decreases from T back to 0, and denote the current definition of F at time t by F_t. (See Fig. 4(b) for a depiction of the graph during the reverse delete phase.) The final solution is the set F after the reverse delete phase, which by our notation is F_0.

In Section 6, we give a theoretical analysis of the overall algorithm and prove the following.

THEOREM 4.1. *The k-BTSP problem can be approximated to within a factor of* 4.

4.1 A heuristic for max objective

While the k-BSF algorithm satisfies the balance constraint among all the pickers it selects, it does not guarantee the utilization of all pickers. As we described earlier, a more stringent balance constraint is one that imposes an upper bound on the number of items that in a picklist. Unfortunately, this problem is related to the k-densest subgraph problem, which is a notoriously hard problem to approximate. Indeed, the problem of finding the tour that maximizes the number of orders for a single picker, under an upper bound on the tour length, generalizes the k-densest subgraph problem. While this does not directly imply an approximation hardness for the k-BSF problem with an upper bound, it suggests that proving an approximation guarantee for this problem might be difficult. Therefore, we resort to heuristics to impose a stringent balancing requirement, while keeping the structure of the k-BSF algorithm. We keep one additional piece of information regarding the state of each order in a component during the course of the forward add phase in the k-BSF algorithm. Specifically, we keep the number of fulfilled orders, i.e., orders completely contained in a component, as the forward add phase of the algorithm progresses. Using this information, we run the forward add phase of the algorithm until either of the following happens.

(i) The number of items in fulfilled orders of a component in the solution reaches the maximum limit on the number of items that can be assigned to a picker (ideally n/k for n items and k pickers). In this case, the k-BSF tree for this component is added to final solution (set of trees), the corresponding orders removed from the input, and the k-BSF algorithm is restarted on the new input instance from the beginning.

(ii) The forward add phase of the k-BSF algorithm completes and outputs a forest. In such a case, we complete the algorithm by running the reverse delete phase on this forest.

Finally, edges of the resulting forest in the second case above are contracted and a truncated Kruskal's algorithm is run to produce the remaining $k - w$ trees, where w is the number of trees added to the solution in the first case above. Note that this results in a total of k trees, as desired.

Computational complexity. The algorithm in Fig. 2 can be implemented in near-linear time. More precisely, on a graph with m edges and n vertices, it takes $O((m + n)\log^2 n)$ time. This can

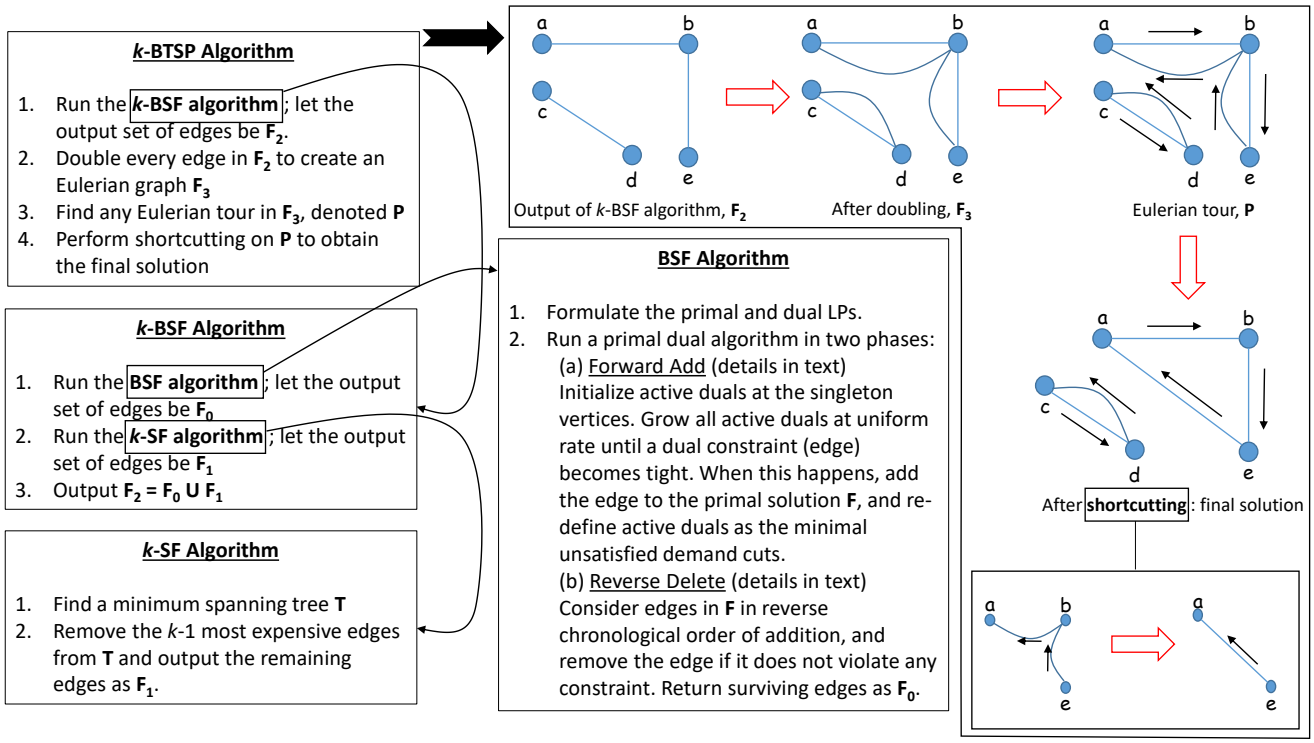

Figure 2: The k-BTSP algorithm: the high level schema and the individual component subroutines.

$$\begin{array}{l|l} \text{min.} \quad \sum_{(u,v)\in E} d_{uv}x_{uv} \text{ s.t.} & \text{max.} \quad \sum_{S\in \mathbb{S}} y_S \text{ s.t.} \\ \sum_{|\{u,v\}\cap S|=1} x_{uv} \geq 1, & \sum_{S\in \mathbb{S}_{uv}} y_S \leq d_{uv}, \\ \quad \forall S \in \mathbb{S}; & \quad \forall (u,v) \in E; \\ x_{uv} \geq 0, \quad \forall (u,v) \in E. & y_S \geq 0, \quad \forall S \in \mathbb{S}. \\ \text{(a)} & \text{(b)} \end{array}$$

Figure 3: The primal LP and the dual LP for the BSF problem.

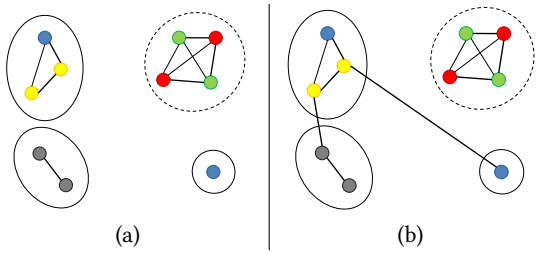

Figure 4: (a) A depiction of the forward add phase. The edges in bold are those in X_t. The components represented by solid circles correspond to active cut duals, while the ones that are dashed represent inactive cut duals. Edges between the cut duals are not shown for clarity—none of these edges is in X_t. The value of ℓ is 3. (b) A depiction of the reverse delete phase of the algorithm. The edges in bold are those in F_t. The cut duals—active (solid circles) and inactive (dashed circles)—corresponding to the vertices in G_t are shown. Other edges between the cut duals that are not in F_t are not shown for clarity. The value of ℓ is 3

be achieved by using a standard MST algorithm (such as Kruskal's algorithm) that runs in $O(m + n \log n)$ time to implement the k-SF subroutine, and the implementation of the primal-dual schema in $O((m + n) \log^2 n)$ time due to Cole et al. [11] to implement the BSF subroutine. For the heuristic that minimizes the max objective, the primal-dual BSF subroutine might be repeated k times, thereby giving an overall running time of $O(k(m + n) \log^2 n)$.

Starting location. The above algorithms assume that the k pickers can have arbitrary starting locations. In some systems, all pickers must start at the same location. For the sum objective, this reduces the problem to the $k = 1$ case, i.e., the traveling salesman problem. For the max objective, we simply add the starting location to each of trees, whether it be produced by the forward add phase or after the truncated Kruskal's algorithm.

5 EMPIRICAL ANALYSIS

We measured the efficiency of our algorithm on store data obtained from an online shopping service, by comparing its performance with three natural baselines that we describe in Section 5.1. Our algorithms consistently outperform the baselines both on sum

and max objectives. One might also be tempted to compare our approximation-based approach with an exact algorithm that employs a commercial LP-solver and might be practical for reasonably sized real world instances (notwithstanding the exponential worst-case running time). The difficulty in this approach is that our ILP formulation uses an exponential number of constraints, which even encoding it for the ILP solver takes exponential time. (Note that this exponential dependence is for *any* instance, not just worst-case instances.) While the connectivity constraints can be encoded into polynomial constraints using standard flow-cut duality, we do not know of a similar succinct encoding for the balance constraints.

5.1 Baselines

Clearly, there are two key components to the order partitioning problem, *viz.*, assignment of the orders to the pickers and the design of the tour for every picklist. We offer three baselines for comparing the different algorithms. The heuristics progress in the degree of sophistication used in the definition of the underlying allocation and tour design.

Baseline 1. The first baseline, RoundrobinGreedy, uses a näive allocation rule of ordering the orders in descending order of their sizes and then allocating them to the available pickers in a round robin manner. This was generally observed to produce reasonably balanced allocations (we will present these observations in more detail when we compare the algorithms). Next, we adopt a natural rule often used by pickers in stores, *viz.*, greedily select the item closest to the current position of the picker and proceed in this manner until the shopping list is exhausted. Note that we always allocate an *entire* order to the picker in the first step. Thus, at the end of the tour, the picker would have picked all the items in orders on her list independent of the sequence in which the items in each order are picked. Algorithm 1 illustrates the greedy selection procedure. The function $back(T)$ returns the last vertex in tour T.

Algorithm 1 ComputeGreedyTour

```
1: function ComputeGreedyTour(s, V)                    ▷ start at s
2:     S ← V
3:     T ← {s}
4:     while S ≠ ∅ do
5:         u ← arg min_{v∈V} d(back(T), v) + d(v, s) − d(back(T), s)
6:         S ← S \ u
7:         T ← T.u
8:     end while
9: end function
```

Baseline 2. The second baseline, RoundrobinMST, adopts a more complicated tour generation step. In fact, it mimics the second step of k-bsf algorithm in Section 4. Instead of incrementally building the tour, we compute the MST over the set of vertices V input to the procedure. Note that this can be converted to a TSP tour by losing at most a factor 2 in the performance of the heuristic by doubling the edges of the MST and obtaining a Eulerian tour on the resulting graph. We ignore this step as it is common to all the heuristics and report the MST cost as the cost of the heuristic.

merchant	store	id	order id	sku	category
M1	S1	1	O1	sku1	detergent
M1	S1	2	O1	sku3	soda
M1	S1	3	O2	sku1	detergent
M1	S1	4	O2	sku6	door mats

Table 1: Schema of the store order data

Baseline 3. The third baseline, GreedyMST, further improves on the allocation step. Instead of assigning orders in a round robin manner to the pickers, it allocates the next order to the picker whose set of unique items including the new order is the smallest. Consider the following example. Suppose there are two pickers A and B with picklists $P_A = \{(a, c), (a, b, c)\}$ and $P_B = \{(a)\}$ respectively. This results in item sets $I_A = \{a, b, c\}$ and $I_B = \{a\}$. When the next order $N = (a, c, d)$ needs to be allocated, we check $|I_i \cup N| - |I_i|$ and assign N to the picker i with the smallest value of this difference. Such an assignment ensures a balanced allocation even as it tries to keep orders with overlapping items together. This can greatly help the shopper minimize her tour cost. Indeed, we observe this in our experiments.

5.2 Store map generation

In order to get up to date information on the layout of items in each store, we also periodically created the underlying store graph representing the layout of the store in terms of access times between any two items (sections) in the store. Depending on the availability of data, we smoothed the item to its category and used a graph representing distances between categories of products in a store. We verified that there was no substantial loss in accuracy because of the smoothing step. The access times themselves were obtained from historically aggregated behavior of pickers in the stores. When there was no edge information available, we completed the graph using Djikstra's shortest path algorithm and made sure the metric property was always satisfied.

5.3 Datasets

Dataset. We ran our algorithms on order data collected from four stores served by the online shopping service. The data was collected over a fixed duration of time during the day in each store and resulted in between 300 and 900 items for processing in each store. All data was sufficiently scrubbed so as to not leak any sensitive information. Table 1 illustrates the schema of the order data used in this study. There was sufficient coverage of items across hundreds of categories in each store. In fact, a large fraction (over 40%) of items were unique in each store order list. Further, more than 50% of the orders had two or more items.

Ta-Feng dataset. We also considered a large publicly available dataset containing four months of shopping transactions of the Ta-Feng supermarket. This data was released by ACM RecSys[2]. In order to use this data in our context, we need one critical piece of information this data lacks, *viz.*, the store layout. To achieve this, we mapped the product categories in the data to our categories for which we have layout information in a grocery store. The

[2] http://recsyswiki.com/wiki/Grocery_shopping_datasets

mapping involved a simple step of associating the top categories in our catalog to a similarly ranked order of categories in the data set. This was performed under the assumption that item distributions in basket data across grocery stores tend to be similar. Another approach would have been to physically reconstruct the map of a grocery store by visiting a store.

5.4 Experimental results

We report the performance of all five methods—the k-BSF algorithm from Section 4, the heuristic from Section 4.1, and the three baselines—for both the sum and max objectives and differing the number of pickers. The experiments confirm our expectation that the k-BSF algorithm significantly outperforms the baselines for the sum objective, but does not achieve good balance and therefore is outperformed on the max objective. To remedy this shortcoming, we designed the k-BSF heuristic, which as expected performs well on the max objective. Crucially, the k-BSF heuristic does not sacrifice the gains of the k-BSF algorithm on the sum objective and achieves similar performance, significantly outperforming the baselines. With this evidence, we conclude that the k-BSF heuristic is a desirable practical algorithm, which performs well on both the sum and max objectives. We detail the comparisons for both objectives next.

Sum objective. We first compare the performance of the algorithms with respect to the sum objective, i.e., the sum of tour lengths of all the active pickers selected by the algorithm. In order to keep the exposition clear, the cost is normalized with respect to the least efficient run of the RoundrobinGreedy algorithm for each store. Table 2 illustrates the relative performance of the three baselines for all four stores and the Ta-Feng dataset. k-BSF outperforms the best of baseline heuristics (GreedyMST) by 7%–125%. Even as the relative performance of all the algorithms decreases as the number of pickers increase, k-BSF degrades much more slowly as the number of pickers increase. The decrease in relative performance is possibly due to decrease in order density, i.e., the number of similar or close by items assigned to a picker, as the number of pickers increase. The slower degradation in picker performance implies that k-BSF does a better job of assigning similar orders to the same picker. However, the metric where k-BSF suffers is the utilization of the pickers as it does not support a constraint on the maximum number of items a picker can process. Indeed, the number of pickers chosen by the algorithm never went up above 3 (except store 2) while all the baselines did well on the utilization of pickers. The k-BSF heuristic does not suffer on this metric, but achieves comparable performance to the k-BSF algorithm on the sum objective.

Table 2 also illustrates the performance of the algorithms on the TaFeng dataset for the sum objective. The overall trends are similar to the online shopping dataset. Specifically, one noticeable difference is the relatively good performance of the k-BSF heuristic compared to the k-BSF algorithm for the sum objective. Thus, the k-BSF heuristic is also a good alternative to the k-BSF algorithm for this metric.

Max objective.. We next compare the algorithms with respect to the max objective, i.e., the maximum of the tour lengths of all the active pickers selected by the algorithm. Again, we report the relative performance of the algorithms compared to the least performing baseline, viz., RoundRobinGreedy, i.e., we peg the value of RoundRobinGreedy to 1.0 and scale the other algorithms accordingly. As Table 2 shows, the k-BSF algorithm starts to outperform the baseline algorithms as the number of pickers increases and remains competitive with the baselines for smaller numbers of pickers. One interesting observation is that in the regime that k-BSF heuristic performs well, the k-BSF algorithm does poorly in terms of the max objective. This is because k-BSF algorithm is not able to partition the items into the required number of pickers while the k-BSF heuristic does not run into this problem and hence is able to reduce the cost of the MST. Thus, one possible implementation could be to use the k-BSF algorithm for small number of pickers and adopt the heuristic as the number of pickers grows larger.

Interestingly, on the TaFeng dataset, the relative difference in the performance between the k-BSF algorithm and the k-BSF heuristic is similar. This behavior was slightly different in the case of the sum objective where the heuristic out-performed the algorithm even for smaller number of pickers. However, the overall trends are similar to the online shopping dataset.

Finally, we observe that the k-BSF heuristic is also balanced in terms of the number of items assigned to each picker. Further, it always utilizes all the available pickers in the picklist assignment.

6 THEORETICAL ANALYSIS

In this section, we show that the approximation factor of the overall k-BTSP algorithm is 4 (Theorem 4.1). First, we analyze the approximation ratio of the BSF algorithm. Recall that F_0 represents the set of edges output by the BSF algorithm. Let

$$F_S := \{(u,v) \in F_0 : S \in \mathbf{S}_{uv}\}.$$

Recall that the primal objective is

$$\sum_{(u,v)\in F_0} d_{uv} = \sum_{(u,v)\in F_0} \sum_{S\in\mathbf{S}_{uv}} y_S = \sum_{S\in\mathbf{S}} |F_S| \cdot y_S$$

and the dual objective is $\sum_{S\in\mathbf{S}} y_S$. Let $\mathbf{S}_t$ be the set of active duals in the forward add phase at time t. By the above formulas, at time t, the dual changes at the rate of the number of active duals, i.e., at the rate $|\mathbf{S}_t|$, while the primal changes at the rate of $\sum_{S\in\mathbf{S}_t} |F_S|$. We will show that

$$\sum_{S\in\mathbf{S}_t} |F_S| \le 2 \cdot |\mathbf{S}_t|,$$

which will establish the ratio of primal and dual solutions.

Let us define G_t as a meta graph corresponding to time index t where the components in X_t are contracted into single vertices. Note that in the forward add phase at time t, the vertices in G_t exactly correspond to the active and inactive duals. The set of edges in G_t is denoted E_t. Note that these are exactly the edges in F_t that are not in X_t, since the connected components formed by edges in X_t are contracted in G_t. Therefore,

$$E_t := F_t \setminus X_t = \cup_{S\in\mathbf{S}_t} F_S.$$

The following are direct consequences of the algorithm.

Fact 6.1. *At any time index t of the forward add phase, X_t is a forest whose trees connect the vertex subsets represented by vertices of G_t. At any time index t in the reverse delete phase, F_t is acyclic*

		sum objective					max objective				
	Pickers	2	4	6	8	10	2	4	6	8	10
Store 1	RoundrobinGreedy	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RoundrobinMST	0.33	0.37	0.40	0.43	0.46	0.19	0.28	0.39	0.31	0.29
	GreedyMST	0.33	0.36	0.39	0.41	0.43	0.19	0.27	0.36	0.29	0.29
	k-BSF heuristic	0.32	0.34	0.36	0.37	0.40	0.18	0.25	**0.35**	**0.26**	**0.27**
	k-BSF	**0.32**	**0.32**	**0.34**	**0.35**	**0.37**	**0.12**	**0.18**	0.55	0.45	0.46
Store 2	RoundrobinGreedy	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RoundrobinMST	0.14	0.19	0.17	0.22	0.22	0.22	0.30	0.25	0.28	0.37
	GreedyMST	0.13	0.18	0.16	0.20	0.20	0.20	0.30	0.26	0.30	0.39
	k-BSF heuristic	0.13	0.16	0.15	0.12	0.14	0.21	0.30	0.23	**0.09**	**0.30**
	k-BSF	**0.06**	**0.08**	**0.06**	**0.08**	**0.08**	**0.13**	**0.24**	**0.21**	0.26	0.34
Store 3	RoundrobinGreedy	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RoundrobinMST	0.12	0.14	0.18	0.20	0.23	0.10	**0.09**	0.27	0.13	0.17
	GreedyMST	0.12	0.14	0.18	0.19	0.21	0.13	0.09	0.21	0.13	0.18
	k-BSF heuristic	0.11	0.13	0.16	0.17	0.18	0.12	0.14	0.20	**0.06**	**0.14**
	k-BSF	**0.11**	**0.12**	**0.15**	**0.15**	**0.17**	**0.09**	0.12	**0.19**	0.20	0.27
Store 4	RoundrobinGreedy	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RoundrobinMST	0.15	0.22	0.21	0.24	0.23	0.18	0.18	0.23	0.25	0.26
	GreedyMST	0.14	0.21	0.20	0.22	0.21	**0.17**	**0.16**	0.22	0.27	0.25
	k-BSF heuristic	0.13	0.18	0.18	0.20	0.20	0.22	0.17	**0.13**	**0.23**	**0.21**
	k-BSF	**0.11**	**0.17**	**0.15**	**0.15**	**0.14**	0.21	0.31	0.60	0.60	0.75
Ta-Feng	RoundrobinGreedy	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	RoundrobinMST	0.38	0.46	0.48	0.54	0.60	0.41	**0.41**	0.42	0.54	0.32
	GreedyMST	**0.38**	0.45	0.47	0.54	0.59	0.41	0.51	0.37	0.53	0.36
	k-BSF Heuristic	0.40	**0.40**	0.48	0.54	**0.53**	0.47	0.51	0.39	**0.46**	**0.29**
	k-BSF	0.39	0.42	**0.37**	**0.47**	0.56	**0.39**	0.49	**0.34**	0.50	0.41

Table 2: Relative performance with respect to the RoundrobinGreedy on the sum and max objectives.

and comprises X_t (which is as above) and E_t, which forms an acyclic subgraph on the vertices of G_t.

Proof. The acyclicity of X_t follows since if an edge is added to X in the forward add phase, it must be a crossing edge of the partition defined by the current connected components of X. The claims follow from the definitions of F_t, G_t, and E_t. □

We will show that

$$\sum_{S \in \mathbf{S}_t} |F_S| \leq 2 \cdot |\mathbf{S}_t|.$$

Recall that this immediately gives an approximation factor of 2 for the bsf algorithm. We show this bound in two steps. Note that $\cup_{S \in \mathbf{S}_t} F_S = E_t$. An edge in E_t is said to be a *special edge* if it is the only edge incident on an inactive dual leaf (i.e., degree 1) vertex in G_t. By Fact 6.1, the edges in E_t form a forest—we call a tree in this forest that contain at least one special edge as a *special tree.*

Lemma 6.2. *At any time t, if R is a special tree in G_t, then R contains exactly one special edge, and exactly one inactive dual vertex.*

Proof. Let e be a special edge in E_t incident on an inactive cut dual S at time t. Since S is inactive, it is not a demand cut, and therefore, in particular, not a separating cut (recall that separating cuts are those that separate some vertex color class). Hence, $F_t \setminus \{e\}$ does not separate any vertex color class. Consider time t^* when edge e was added to X in the forward add phase. Since $t^* \geq t$, we have $F_{t^*} \supseteq F_t$. Hence, $F_{t^*} \setminus \{e\}$ does not separate any vertex color class either. Therefore, the fact that e was not removed from F_{t^*} at time t^* of the reverse delete phase implies that such removal would have violated a balance constraint (recall that a balance constraint enforces that any tree in F must contain at least ℓ vertices).

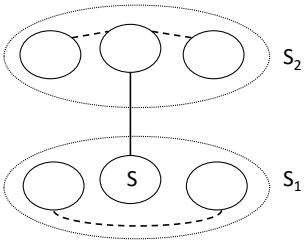

Figure 5: A depiction of special edges in the proof of the approximation factor. The solid edge is special and the dashed edges are the other edges in E_t. The solid circles represent cut duals at time t and the dotted circles represent cut duals at time t^*.

Let S_1 and S_2 be the components of X_{t^*} that e is incident on, where $S \subseteq S_1$. A we argued above, it must be the case that the removal of e from F in the reverse delete phase at time t^* would have made one of S_1 or S_2 to violate their respective balance constraints, $|S_1| < \ell$ or $|S_2| < \ell$. Since S is an inactive cut dual at time t, it is not a demand cut and hence $|S| \geq \ell$. It follows that $|S_1| \geq \ell$ since $S_1 \supseteq S$, and therefore, $|S_2| < \ell$. Moreover, since e is the only edge in F_{t^*} in the cut $(S_2, V \setminus S_2)$, it must also be the only edge in this cut in F_t (since $F_t \subseteq F_{t^*}$). (See Fig. 5 for a depiction.) Since S is a

leaf in G_t, it follows that the component containing edge e in F_t is a subset of $S_2 \cup S$. To prove the lemma, we then need to show that all components in X_t that are subsets of S_2 correspond to active duals, i.e., to demand cuts. This follows from the fact that $|S_2| < k$ and is a demand cut itself; therefore, all its subsets are also demand cuts. □

We are now ready to bound the cost of the BSF solution (i.e., the length of the edges in F_0) in terms of the dual objective.

LEMMA 6.3. *The length of edges in F_0 produced by the* BSF *algorithm is at most 2 times the objective of the feasible dual solution produced by the algorithm.*

PROOF. To show the lemmas, we will prove that at any time t,

$$\sum_{S \in \mathbf{S}_t} |F_S| \leq 2 \cdot |\mathbf{S}_t| - 1.$$

Let $\mathbf{S}_R$ denote the active duals in a tree R in G_t. First, suppose R is special. Then, by Lemma 6.2, the number of edges in R is exactly $\mathbf{S}_R$, since every component except one corresponds to an active dual. It follows that

$$\sum_{S \in \mathbf{S}_R} |F_S| = 2 \cdot |\mathbf{S}_R| - 1.$$

Now, consider a non-special tree R. By the definition of special trees, every inactive cut dual in R has at least two edges in E_t incident on it. If tree R has r vertices in G_t, then it has $r - 1$ edges in E_t. Therefore,

$$\sum_{S \in \mathbf{S}_R} |F_S| \leq 2(r-1) - 2(r - |\mathbf{S}_R|) = 2 \cdot |\mathbf{S}_R| - 2.$$

Summing over all trees R in G_t, we get the desired bound. □

Lemma 6.3 immediately yields an approximation factor of 2 for the BSF algorithm by weak duality, thereby proving Lemma 6.4.

LEMMA 6.4. *The approximation factor of the* BSF *algorithm is 2.*

Next, we show that the algorithm for the k-SF problem produces an optimal solution.

LEMMA 6.5. *For any k, removing the $k-1$ most expensive edges from a minimum spanning tree gives an optimal spanning k-forest.*

PROOF. Fix k. We will prove a slightly stronger property that removing the t most expensive edges from the minimum spanning tree gives a forest that contains an optimal spanning k-forest for any $t \leq k-1$. Setting $t = k-1$ yields the lemma.

We use induction on t. For the base case, we consider $t = 0$, i.e., need to show that an MST (call it T) contains an optimal k-spanning forest (call it F). Suppose not, and let $(u, v) \in F \setminus T$. Since F is acyclic, there is some edge, say (x, y), on the $u - v$ path in T that is not in F. Furthermore, the cost of (x, y) is no less than the cost of (u, v), else replacing (x, y) by (u, v) in T produces a spanning tree of lower cost. Hence, we can replace (u, v) by (x, y) in F without increasing its cost or the number of trees. These swaps can be repeatedly performed until $F \subseteq T$.

Now, we consider the inductive case. Assume that $F \subseteq T_{t-1}$, where T_{t-1} is the forest with the most expensive $t-1$ edges removed from T. Let (u, v) be the most expensive edge in T_{t-1}. If $(u, v) \notin F$, then the induction holds. So, we assume that $(u, v) \in F$. Then, there is some other edge $(x, y) \in T_{t-1}$ that is not in F. Since the cost of (x, y) is no more than the cost of (u, v), we can replace (u, v) by (x, y) in F without increasing its cost or the number of trees. At this point, we have $F \subseteq T_t = T_{t-1} \setminus \{(u, v)\}$, and the induction holds. □

Combining the optimality of the k-SF algorithm with Lemma 6.4 gives the overall approximation factor of the k-BSF algorithm.

THEOREM 6.6. *The approximation factor of the k-*BSF *algorithm is* 3.

The approximation factor of the k-BTSP algorithm is clearly upper bounded by 6 since every edge is doubled from the 3-approximate k-BSF solution. We will now give a more refined analysis (this idea has been used earlier, e.g., in Goemans and Williamson [18]) to obtain a tighter approximation factor of 4 for the k-BTSP algorithm.

First, let us write an LP relaxation for the k-BTSP problem. Let x_{uv} be the indicator variable for whether edge (u, v) is in the solution or not. Recall that $\mathbf{S}$ denotes the set of demand cuts, i.e., separating cuts and those with less than ℓ vertices. Let Π denote the set of partitions containing more than k subsets—we call these *separating partitions*. Here, we will crucially need the fact that the graph is a metric, i.e., has edges between all vertex pairs (note that this is ensured by taking the shortest path on the actual input graph and shortcutting). We also define a *demand partition* of the vertices as one that has more than k vertex subsets and let Π be the set of demand partitions: $\Pi := \{\pi = (\pi_1, \pi_2, \ldots, \pi_r) : r > k\}$.

For any edge (u, v), let Π_{uv} be the set of demand partitions that the edge crosses:

$$\Pi_{uv} = \{\pi = (\pi_1, \pi_2, \ldots, \pi_r) \in Pi : \exists i \neq j, u \in \pi_i, v \in \pi_j\}.$$

The LP relaxation is now given by:

$$\begin{aligned} \text{Minimize} \quad & \textstyle\sum_{(u,v) \in E} d_{uv} x_{uv} \text{ such that} \\ \sum_{|\{u,v\} \cap S| = 1} x_{uv} \quad & \geq \quad 2 \quad \text{for all } S \in \mathbf{S} \\ \sum_{(u,v): \Pi \in \Pi_{uv}} x_{uv} \quad & \geq \quad |\pi| - k \quad \text{for all } \pi \in \Pi \\ x_{uv} \quad & \geq \quad 0 \quad \text{for all } (u,v) \in E \end{aligned}$$

Note that we are not insisting that the selected edges form at most k tours for the separating partitions, rather that they form at most k connected components. This is a strictly weaker requirement but we will eventually produce a solution that comprises at most k tours.

We now define the dual LP. Recall that $\mathbf{S}_{uv}$ (resp., Π_{uv}) denotes the demand cuts (resp., demand partitions) that edge (u, v) is part of. The dual LP is given by:

$$\begin{aligned} \text{Maximize} \quad & 2 \cdot \textstyle\sum_{S \in \mathbf{S}} y_S + \sum_{\pi \in \Pi} (|\pi| - k) \cdot z_\pi \text{ such that} \\ \sum_{S \in \mathbf{S}_{uv}} y_S + \sum_{\pi \in \Pi_{uv}} z_\pi \quad & \leq \quad d_{uv} \quad \text{for all } (u,v) \in E \\ y_S, z_\pi \quad & \geq \quad 0 \quad \text{for all } S \in \mathbf{S}, \pi \in \Pi \end{aligned}$$

We compare the solution produced by the BSF algorithm with the optimal cost of the k-BTSP instance.

LEMMA 6.7. *The cost of the solution produced by the* BSF *algorithm is at most the optimal cost of the k-*BTSP *instance.*

Proof. Consider the following solution for the k-BTSP dual: for every cut dual, we set it exactly as in the BSF algorithm, and for every partition dual, we set it to 0. By feasibility of the BSF dual, this solution is also feasible for the k-BTSP dual. However, note that the k-BTSP dual objective is twice that of the BSF dual objective. Therefore, by Lemma 6.3, the cost of the BSF solution is no more than the objective of the k-BTSP dual, which in turn is upper bounded by the optimal cost of the k-BTSP instance by weak duality. □

By Lemma 6.5, the cost of the solution produced by the k-SF algorithm is equal to the optimal cost of the k-SF instance, which is at most the optimal cost of the k-BTSP instance. Combining Lemma 6.7 with this, we get the following bound on the cost of the solution produced by the k-BSF algorithm.

Lemma 6.8. *The cost of the solution produced by the k-BSF algorithm is at most 2 times the optimal cost of the k-BTSP instance.*

Recall that the k-BTSP algorithm doubles every edge in the k-BSF solution and uses shortcutting on any Eulerian tour of the resulting graph to produce the k tours comprising the final solution. Using the above lemmas, we can therefore conclude that the approximation factor of the k-BTSP algorithm is 4 (Theorem 4.1).

7 CONCLUSIONS

In this paper, we considered the problem of assigning store orders to a set of pickers and designing their routes for picking the items in their assigned orders. We formulated a general partition framework and defined this problem in the framework. We then used primal dual optimization techniques to give a provably good approximation algorithm. This is substantially better from an analytical perspective than the natural approach of replacing each order with a representative item and designing the tours based on these representatives. From an empirical perspective, we compared the performance of our algorithm to a set of baselines on various real world datasets. Our algorithm outperforms the baselines on the desired objective of minimizing the total length of the paths traversed by the pickers. We also considered the practically important goal of balancing the workload evenly among the pickers, and showed that our algorithm is comparable with or better than the baselines on this objective as well. We leave the design of a provable algorithm with the explicit goal of balancing the tour lengths of the pickers as future work.

Acknowledgements. The authors thank Aslam Mohammed and Ashish Kumar in the Google Express fulfillment team for helping out with access to data and useful discussions on online shopping services.

REFERENCES

[1] A. Agrawal, P. N. Klein, and R. Ravi. 1995. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.* 24, 3 (1995), 440–456.
[2] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. 2007. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press.
[3] D. L. Applegate, W. J. Cook, S. Dash, and A. Rohe. 2002. Solution of a min-max vehicle routing problem. *INFORMS Journal on Computing* 14(2) (2002), 132–143.
[4] E. M. Arkin, R. Hassin, and A. Levin. 2006. Approximations for minimum and min-max vehicle routing problems. *J. Algorithms* 59(1) (2006), 1–18.
[5] I. Averbakh and O. Berman. 1997. $(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with min-max objective. *Discrete Applied Mathematics* 75 (1997), 201–216.
[6] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. 2004. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *STOC*. 166–174.
[7] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. 2007. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM J. Comput.* 37, 2 (2007), 653–670.
[8] M. Charikar, S. Khuller, and B. Raghavachari. 2001. Algorithms for capacitated vehicle routing. *SIAM J. Comput.* 31, 3 (2001), 665–682.
[9] M. Charikar and B. Raghavachari. 1998. The finite capacity dial-a-ride problem. In *FOCS*. 458–467.
[10] C. Chekuri, N. Korula, and M. Pál. 2012. Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms* 8, 3 (2012), 23:1–23:27.
[11] R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. 2001. A faster implementation of the Goemans–Williamson clustering algorithm. In *SODA*. 17–25.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms (3. ed.).* MIT Press.
[13] I. Davidson and S. Basu. 2007. A survey of clustering with instance-level constraints. *ACM TKDD* 1 (2007), 1–41.
[14] I. Davidson and S. S. Ravi. 2005. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *PKDD*. 59–70.
[15] I. Davidson, S. S. Ravi, and M. Ester. 2007. Efficient incremental constrained clustering. In *KDD*. 240–249.
[16] G. N. Fredrickson, M. S. Hecht, and C. E. Kim. 1978. Approximation algorithms for some routing problems. *SIAM J. on Computing* 7 (1978), 178–193.
[17] R. Ge, M. Ester, W. Jin, and I. Davidson. 2007. Constraint-driven clustering. In *KDD*. 320–329.
[18] M. X. Goemans and D. P. Williamson. 1995. A general approximation technique for constrained forest problems. *SIAM J. Comput.* 24, 2 (1995), 296–317.
[19] B. L. Golden and A. A. Assad (Eds.). 1988. *Vehicle Routing: Methods and Studies.* Studies in Management Science and Systems, Vol. 16. North-Holland, Amsterdam.
[20] I. L. Gørtz, V. Nagarajan, and R. Ravi. 2015. Minimum makespan multi-vehicle dial-a-ride. *ACM Trans. Algorithms* 11, 3 (2015), 23:1–23:29.
[21] A. Gupta, M. T. Hajiaghayi, V. Nagarajan, and R. Ravi. 2010. Dial a ride from k-forest. *ACM Trans. Algorithms* 6, 2 (2010), 41:1–41:21.
[22] M. Haimovich and A. H. G. Rinnooy Kan. 1985. Bounds and heuristics for capacitated routing problems. *Math. Oper. Res.* 10, 4 (1985), 527–542.
[23] A. L. Henry-Labordere. 1969. The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem. *RAIRO Operations Research* B2 (1969), 736–743.
[24] J. Hipp, U. Guntzer, and G. Nakhaeizadeh. 2000. Algorithms for association rule mining—A general survey and comparison. *SIGKDD Explorations* 2, 1 (2000), 58–64.
[25] G. Laporte, Y. Nobert, and S. Taillefer. 1988. Solving a family of multi-depot vehicle routing and location-routing problems. *Transp. Sci.* 22 (1988), 161–172.
[26] C-L. Li and D. Simchi-Levi. 1990. Worst-case analysis of heuristics for multidepot capacitated vehicle routing problems. *INFORMS Journal on Computing* 2, 1 (1990), 64–73.
[27] C-L. Li, D. Simchi-Levi, and M. Desrochers. 1992. On the distance constrained vehicle routing problem. *Operations Research* 40(4) (1992), 790–799.
[28] P. Toth and D. Vigo (Eds.). 2001. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.