Multicapacity Facility Selection in Networks

Alvis Logins
Aarhus University

Panagiotis Karras
Aarhus University

Christian S. Jensen
Aalborg University

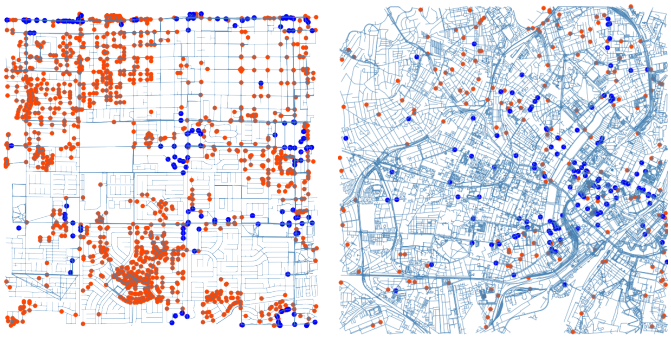
***Abstract*—Consider the task of selecting a set of facilities, e.g., hotspots, shops, or utility stations, each with a *capacity* to serve a certain number of customers. Given a set of customer locations, we have to minimize a cumulative *distance* between each customer and the facility earmarked to serve this customer within its capacity. This problem is known as the Capacitated k-Median (CKM) problem. In a data-intensive variant, distances are calculated over a network, while a data set associates each candidate facility location with a different capacity. In other words, going beyond positioning facilities in a metric space, the problem is to select a small subset out of a large data set of candidate network-based facilities with capacity constraints. We call this variant the Multicapacity Facility Selection (MCFS) problem. Linear Programming solutions are unable to contend with the network sizes and supplies of candidate facilities encountered in real-world applications; yet the problem may need to be solved scalably and repeatedly, as in applications requiring the dynamic reallocation of customers to facilities. We present the first, to our knowledge, solution to the MCFS problem that achieves both scalability and high quality, the Wide Matching Algorithm (WMA). WMA iteratively assigns customers to *candidate* facilities and leverages a data-driven heuristic for the SET COVER problem inherent to the MCFS problem. An extensive experimental study with real-world and synthetic networks demonstrates that WMA scales gracefully to million-node networks and large facility and customer data sets; further, WMA provides a solution quality superior to scalable baselines (also proposed in the paper) and competitive vis-á-vis the optimal solution, returned by an off-the-shelf solver that runs only on small facility databases.**

I. INTRODUCTION

A type of problem arising in smart city applications calls for selecting an attractive subset out of a set of candidate *facility locations* (e.g., telecommunications hotspots, meeting points, bike stations, recycling stations, electric vehicle charging stations, or waste disposal sites) to provide a service in an urban network. The fitness of the selected set is measured by means of total convenience or utility with respect to a set of geographically located *customers*. This problem may need to be solved repeatedly; for example, one may need to periodically decide on a set of service locations, depending on which customers declare interest for a certain offering.

The input data typically includes a weighted graph, representing a road network and associated candidate facility locations and customer locations. Figure 1 provides two examples, where we need to select a subset out of a set of eligible facilities (in *blue*) so as to serve a predefined set of customers (in *red*) in Las Vegas or Copenhagen.

In the *Multicapacity Facility Selection* (MCFS) problem, each candidate facility has a *capacity*, and we need to choose k facilities and assign each customer to one of them while observing the capacity constraints; as the number of served customers is bounded by capacity constraints, an objective of maximizing that number does not arise; the objective is to optimize a notion of *customer convenience*, defined in terms of the distances between customer and the facilities they are assigned to. The MCFS problem amounts to the *hard* and *nonuniform* case of the *capacitated k-median* (CKM) problem [1] over a *network*. Here, *hard* indicates that only one facility can be placed at a certain location; the *soft* version allows for multiple facilities at the same location. Next, *nonuniform* indicates that facility capacities differ; in the *uniform* version, all capacities are equal. Last, the *network* setting need not yield a metric distance notion.



(a) Las Vegas (b) Copenhagen

Fig. 1: Customers (red), and cafés (blue).

Unfortunately, the problem is already NP-hard in the *soft* and *uniform* case over a *metric space* [2]. Small instances can be solved exactly by Linear Programming (LP) and Mixed Integer Programming (MIP) solvers [3]. However, such solvers do not scale beyond networks with a few thousand nodes. Past research has proposed LP *relaxation* [4] methods that provide approximation guarantees while violating constraints on facility *capacity* or *cardinality*. The most recent works in the area introduce an LP formulation, called *rectangle LP*, tailored to the uniform [5] and soft nonuniform [6] capacitated k-median problems, and develop rounding algorithms that achieve constant approximation guarantees while violating the cardinality constraint k. Such solutions remain impracticable in real-world applications due to their high-polynomial time complexities, while an approximation algorithm for the nonuniform hard-capacitated case has yet to be developed [7].

Local search techniques exist for the CKM problem and related *facility location* problems [2], [8], known as *group nearest group queries* in the database community [9]; however, such solutions solve only the *uncapacitated* and *uniform soft-*

Work done primarily while all authors were with Aalborg University.

capacitated problem cases; they accommodate neither nonuniform nor hard capacity constraints. Thus, to our knowledge, no existing solution achieves both high quality and scalability to large networks and customer sets in the MCFS problem.

We present an effective and scalable MCFS solution, the Wide Matching Algorithm (WMA). WMA progressively assigns customers to strategically chosen candidate facilities, translating a bipartite assignment under capacity constraints to a network setting, and decides on its termination by means of a SET COVER heuristic. We contribute the following:

- We attempt the first, to our knowledge, solution for the MCFS problem that achieves high quality and is applicable to large real-world networks.
- We develop an algorithm, WMA, that combines a data-driven heuristic for set cover with a principled spatial assignment subroutine.
- We introduce a reasonable baseline MCFS heuristic that clusters customers in groups satisfying capacity constrains, following a Hilbert space-filling curve.
- We conduct an experimental study with synthetic and real data, demonstrating that WMA scales to million-node and million-edge networks with large customer and facility sets and achieves near-optimal solution quality, as seen in cases where the exact solution can be computed.

II. PROBLEM STATEMENT

Consider a network represented as a weighted (directed or undirected) graph $G = (V, E, W)$, where V is a set of nodes that model urban locations such as intersections and road ends, E is a set of edges that model road segments, and W is a mapping from edges to positive integer weights that model road segment lengths. Further, we are given a set of m customers $S = \{s_i\}_{i=1}^{m} \subseteq V$, and a set of candidate facility locations $F_p = \{f_j\}_{j=1}^{l} \subseteq V$; each $f_j \in F_p$ comes with a capacity constraint c_j. Given a cardinality value k, the problem is to select k candidate facilities $F \subseteq F_p, |F| = k$ and assign each customer to exactly one facility in F, so that each selected facility $f_j \in F$ is at most c_j assigned customers and the sum of network distances between customers and their allocated facilities is minimized.

We use two binary variables x_j and y_{ij}; x_j indicates whether the candidate facility at node v_j is selected, $j \in \{1, .., \ell\}$, while y_{ij} indicates whether the customer at node v_i is assigned to the facility at node v_j. Also, let d_{ij} be the shortest-path distance between v_i and v_j. Note that d_{ij} values need not define a metric matrix and need not be given as input; instead, they may be computed on the fly over the input network, a feature distinguishing our problem setting. Then, our minimization objective over x_j and y_{ij} is:

$$\min_{x_j, y_{ij}} \sum_i \sum_j d_{ij} y_{ij} \quad (1)$$

subject to:

$$y_{ij} \leq x_j, \quad x_j, y_{ij} \in \{0, 1\} \quad (2)$$

$$\sum_j y_{ij} = 1, \quad \sum_i y_{ij} \leq c_j, \quad \sum_j x_j \leq k \quad (3)$$

Constraint (2) implies that a customer can be assigned to a node v_j where a selected facility is located. The other constraints stipulate that each customer is assigned exactly to one facility, a facility v_j is matched with at most c_j customers, and k facilities are selected. Table I outlines our notations.

Notation	Description
G	A weighted graph (network)
E, V	Sets of edges and nodes of G
$v.dist$	Distance from considered customer to node v
$v.p$	Potential of node v
$dist(v_1, v_2)$	The shortest path distance between nodes v_1 and v_2 in G
$S \subseteq V$	Locations of customers
n	Number of nodes in G, $n = \lvert V \rvert$
m	Number of customers
k	Number of selected facilities
ℓ	Number of candidate facilities
c_j	Capacity of facility j
$F_p \subseteq V$	Set of candidate facility locations
$F \subseteq F_p$	Selected facilities, $\lvert F \rvert = k$
G_b	Bipartite directed graph between S and F_p
E'	Set of edges in G_b
d_{ij}	Distance between i-th customer and j-th candidate facility
$x_j \in \{0, 1\}$	Indicator of whether f_j is in F
$y_{ij} \in \{0, 1\}$	Indicator of whether s_i is allocated to f_j
d_i	Demand of a customer s_i in bipartite graph G_b
	Assignment of customers to facilities in G_b
$N_j(G_b)$	Set of customers assigned to facility j in G_b

TABLE I: Notations.

III. RELATED WORK

An array of facility location problem variants have attracted attention for a long time. Farahani and Hekmatfar [4] provide a comprehensive overview of state-of-the-art algorithms for several of those variants. These algorithms are mostly based on linear programming (LP), using LP-rounding and Lagrangian relaxation as approximation tools. The problem we study is a network-based version of the nonuniform hard-capacitated k-median problem [1], [5], [10].

A. Scalable Facility Location

Some recent works consider a special facility location problem variant, called Optimal Location Query (OLQ) [8], [11]–[13], which calls to place a single new facility that attracts the highest amount of customers (the MaxSum objective), or minimizes the maximum distance between a customer and its nearest facility (the MinMax objective). OLQ solutions are based on a Bichromatic Reverse Nearest Neighbor (BRNN) technique, where each customer is associated with a Nearest Location Region (NLR), such that each point therein is closer than the nearest existing facility. To optimize for MaxSum, we place a new facility in the region with the highest amount of overlapping NLRs [12], [13]. To optimize for MinMax, we sort customers by distance to nearest facility, obtain a set of top-k customers whose NLRs' intersection is nonempty, and find an optimal region therein [8].

As the OLQ bears some resemblance to to the MCFS problem, we could apply it iteratively, as a heuristic, to obtain a solution to MCFS. Figure 2 illustrates the result of facility selection by such an approach, employing the intuitively reasonable MaxSum objective. We start with no facility placed and select node 1 for the first facility, as it is the one that minimizes the aggregate distance to customers, $d_1 = 5$. Dashed curves in the figure indicate the resulting NLRs for each

customer. Node 2 has the highest number of intersecting NLRs (i.e., attracted customers), so we select it. Yet the optimal MCFS solution is to select nodes 4 and 5. Thus, unfortunately, placing facilities by an iterative BRNN-based approach does not fare well with our optimization objective.

We implemented a BRNN-based approach that sequentially selects k nodes as facilities, recalculating a set of NLRs at each step and breaking ties arbitrarily. We include this approach in our experimental comparison; as we will see, its results are significantly worse than those of other approaches.

Fig. 2: BRNN application.

Fig. 3: Example network G.

B. Bipartite Matching

The MCFS problem implies a bipartite matching of customers with facilities. To address this need, we adapt the Simplified Incremental Algorithm (SIA) [14], [15] from the case of Euclidean distances to that of network distances. SIA adapts the Successive Shortest Path Algorithm (SSPA) [16] to a bipartite graph, enhancing it with an edge pruning capability, which allows finding a provably optimal matching after accessing only a few edges adjacent to each node, using an edge weight threshold derived from node potentials. In Section V, we enhance this pruning threshold.

IV. The Wide Matching Algorithm

In a nutshell, the Wide Matching Algorithm progressively enriches candidate facilities with potential serviced customers until it finds a set of k facilities that can service the full customer set within their capacities.

A. Algorithm Overview

Throughout the operation of the algorithm, each customer s_i maintains an increasing demand value d_i, reflecting the number of candidate facilities in F_p it has to be assigned to. In each iteration, we increase the demand of a chosen subset of customers and assign each customer with increased demand to exactly one new facility; while doing so, we may rewire previous choices, i.e., reallocate previous customer-to-facility allocations, if beneficial, while observing capacity constraints. Thereby, customers explore candidate assigned facilities, though eventually they are allocated to exactly one of those. We then select a subset $F \subseteq F_p$, $|F| = k$, such that the elements of F collectively cover (i.e., are allocated to within their capacities) as many customers as possible, by means of a GST COVER heuristic; this heuristic iteratively picks a facility that brings the biggest marginal gain to the number of covered customers. We resolve ties by selecting the facility f chosen least recently in previous iterations. This diversification strategy avoids getting trapped in non-optimal local minima. An exploration vector specifies the increase of d_i values per iteration: d_i is set to 1 if and only if s_i has been left uncovered by the set F selected in the previous iteration and $d_i < \ell$; this choice lets all customers grow their demand values evenly. The main phase WMA terminates when it detects a subset $F \subseteq F_p$ that covers all customers in S, or all demands reach ℓ; the latter case invokes special measures, which we discuss in Section IV-C.

B. Example

We illustrate the operation of WMA with an example. Figure 3 shows a network of nodes, a_i for customers and b_j for candidate facilities. For visualization's sake, we do not place facilities on the same nodes as customers.

Assume we have to place $k = 2$ facilities, with uniform capacity $c = 2$. Figure 4 shows the bipartite graph G_b from customers to candidate facilities across iterations. Each edge is weighted by the distance between its adjacent nodes. Table II depicts part of the adjacency list of G_b with each node's three nearest adjacent nodes in ascending order.

a_1	$b_4(1)$	$b_2(4)$	$b_5(9)$
a_2	$b_5(1)$	$b_6(2)$	$b_3(9)$
a_3	$b_1(1)$	$b_2(4)$	$b_4(9)$
a_4	$b_3(1)$	$b_2(5)$	$b_6(6)$

TABLE II: Sample adjacency list for G_b; weights in brackets.

(a) Iteration 1 (b) Iteration 2 (c) Iteration 3 (d) Iteration 4

Fig. 4: Bipartite graph G_b through WMA iterations.

First, each customer is matched to its nearest facility in Figure 4a. Now each of the four facilities covers one customer. We resolve ties arbitrarily, selecting two facilities out of four, b_4 and b_5, and set the exploration vector to $d = \{0, 0, 1, 1\}$.

In effect, a_3 and a_4 need to explore the network further. They do so and both acquire a new match, facility b_2, obeying the capacity constraint $c = 2$. Thus, by the end of the second iteration a_1 and a_2 have been matched to one facility each, while each of a_3 and a_4 has been matched to two facilities. Now b_2 is the most popular facility, in the sense that it is matched to more customers than any other facility, namely customers a_3 and a_4. After discounting these covered customers, the second most popular facility in terms of marginal gain is either b_4 or b_5, bringing a gain of one customer each, i.e., a_1 and a_2, respectively. We arbitrarily select one of the two, b_5. At this point a_1 is the only uncovered customer. Hence, we set $d_1 = 1$ and $d_i = 0$ for the other three customers, as Figure 4b illustrates.

The third iteration (Figure 4c) demonstrates the benefit of using an assignment algorithm. Now a_1 has a demand to

be matched with two facilities, yet its next nearest facility, b_2, has reached its capacity; a greedy approach would then match a_1 to b_5, the next nearest available facility. Rather than doing so, our matching algorithm rewires previous choices, i.e., reconsiders previous allocations and substitutes them with new ones, if beneficial: in particular, it reassigns a_4 to b_6 so that it can assign a_1 to b_2. The newly used b_6 along with b_2 collectively cover a_1, a_3, and a_4. Now a_2 is uncovered, and hence $d_2 = 1$. Eventually, the fourth iteration matches a_2 to b_6. Now two facilities, namely b_2 and b_6, cover all customers, as Figure 4d shows, with objective value 16.

C. Algorithm Outline

WMA operates on a complete directed bipartite graph between customers and candidate facilities, and progressively satisfies demand and capacity constraints by bipartite matching. This operation can be time consuming on a complete graph, while previous work has not considered bipartite matchings among nodes anchored in a network. Still, we effectively transfer a pruning technique for bipartite matching with Euclidean distances [14], [15] to a network setting.

The core idea is this: if we can ascertain that there is no possible beneficial reassignment that would match node a_i to b_j, we can eschew b_j from consideration. To ascertain that, we do not need to know the exact weight of edge (a_i, b_j); it suffices to know that b_j is farther than another possible match b_k. We can expand knowledge of such weights incrementally, on demand, running an instance of Dijkstra's algorithm on G per customer in each iteration.

Algorithm 1 outlines WMA. In each iteration, we first try matching with current customer demands (Lines 4–5); then we check whether we can select a set of facilities F that cover all customers (Line 6); if we cannot, we raise demands appropriately (Lines 7–8). Lines 10–11 cover the special case where there exists a set F such that $|F| < k$ and F already covers all customers. In that case, we locate the remaining $k - |F|$ facilities in the vicinity of customers with the most unsuccessful assignments; this measure retains coverage and improves the cost objective. Algorithm 4 in Section IV-G illustrates this process. In case the selected facilities fail to cover some customers even after their demands reach k, Lines 12–13 revise F ensuring it suffices to cover all customers, i.e., all disconnected network components. Algorithm 5 in Section IV-G provides the details. Eventually, Lines 14–15 call the same process recursively, setting the demand of each customer to 1, so as to build a single optimal-cost assignment, (G_b), of customers to the k selected facilities in F; the edges in (G_b) outgoing from a selected facility f_j define the set of customers $_j(G_b)$ matched to f_j.

WMA maintains two graphs throughout its operation: first, the input network G that contains locations of customers and candidate facilities; second, the bipartite graph G_b, used for extracting assignments among those entities. Edge weights in G_b reflect shortest-path distances between customers and facilities in G. We assume that a single facility can be located on any network node; the algorithm can be straightforwardly extended to any restrictions on such placements by tuning the candidate facility nodes in G_b.

Algorithm 1 Wide Matching Algorithm

```
1:  function LOCATEFACILITIES(G; S; F_p; k)
2:      G_b ← Bipartite empty graph based on G
3:      d_i = 1  ∀i
4:      repeat
5:          for all s_i ∈ S : d_i > |f_j | s_i ∈ _j(G_b)g| do
6:              G_b ← FINDPAIR(G_b; s_i)
7:          {F; d; covered} ← CHECKCOVER(G_b; k)
8:          d ← d + d
9:      until ∀i  d_i = 0
10:     if |F| < k then
11:         SELECTGREEDY(F; G)
12:     if not covered then
13:         F ← COVERCOMPONENTS(S; F; G)
14:     if |F_p| > k then
15:         return LOCATEFACILITIES(G; S; F; k)
16:     else
17:         return F; (G_p)
```

D. Matching Function

Let us discuss the matching function that iteratively assigns new customers to facilities in G_b and reassigns previously matched pairs. The complete bipartite graph has $n \cdot m$ edges, where each edge requires an execution of Dijkstra's algorithm for its weight calculation. For large problem instances, that would cause excessive computation. Therefore, we add edges to G_b only on demand.

We initialize G_b with two sets of nodes: customers and facilities, without edges. We add edges progressively, running a variant of the Successive Shortest Path Algorithm (SSPA) [16] with node potentials; such potentials encapsulate the goodness of the current arrangement for a node in question, so that we can calculate the benefit of updates involving that node. The process terminates when we can guarantee that the running matching is optimal in the complete G_b. The SSPA solves the Minimum-Cost Flow problem (to which bipartite matching is reduced) using iterative Dijkstra executions from a source to a sink, and flow augmentation. In our problem, the source is a customer, the sink is the closest non-fully occupied facility in G_b; flow augmentation amounts to substituting an edge with one of opposite weight (given that a customer can be matched to each facility only once). SSPA guarantees optimality by adding new edges in an order sorted by weight: it maintains the running weight of the next candidate edge to be taken into consideration, and derives a threshold indicating whether that weight can affect the current solution. We discuss this threshold in Section V. We achieve this order by one Dijkstra execution per customer, yielding distances to candidate facilities in non-decreasing order; such distance values give the weights of new edges in G_b.

Algorithm 2 presents the pseudocode for matching a customer in G_b and updating the running assignment by wiring

as necessary. The loop of Lines 4–12 adds edges to G_b until it can accept a new match for the given customer. In each iteration, we run a Dijkstra instance on G_b (Line 8), to find a shortest path in G_b from the given customer s to the nearest usable (i.e., not fully occupied) facility; this Dijkstra instance works with weights reduced by potential values $v.p$, and it returns the found path and the set of visited nodes. We add each visited node v to a heap with a threshold value that we justify in Theorem 1 (Section V). This threshold depends on the distance from v to its next-nearest neighbor in the network graph G ($nnDist$), the distance from customer to v in G_b ($v.dist$), and a potential value $v.p$ (Lines 9–11). When the condition in Line 12 is satisfied, we can proceed to update the running bipartite assignment.

Algorithm 2 Matching Function

```
1:  function FINDPAIR(G_b, s)
2:      heap ← empty heap
3:      heap.add(⟨s, 0⟩)
4:      repeat
5:          x ← heap.topKey
6:          nn ← node in G_b for next NN of x in G
7:          add edge (x, nn) to G_b
8:          {path, visited} ← DIJKSTRA(s)
9:          for all v ∈ visited \ S do
10:             nnDist ← distance to next NN of v in G
11:             heap.add(⟨v, v.dist + nnDist − v.p⟩)
12:     until path.length < heap.topValue
13:     for all e ∈ path do
14:         e ← e                          ▷ Reverse edge
15:         w(e) ← −w(e)                   ▷ Reverse edge weight
16:     for all v ∈ visited do
17:         v.p ← v.p + path.length − v.dist
18:     return G_b
```

WMA runs two independent Dijsktra instances: one on the bipartite graph G_b (Line 8) for the sake of updating its running assignment and another on the network graph G (Line 10) for the sake of incrementally calculating edge weights on G_b. As both operate over graphs, they require no spatial index. The path found in Line 8 contains a new match, while observing capacity constraints. Then, the loop of Lines 13–15 performs flow augmentation: it increases the flow value by 1 along this path and performs necessary assignment and reassignment actions. Line 17 adjusts potential values.

One execution of the Matching Function assigns exactly one facility to one customer. The flow augmentation in SSPA is constrained only by the target's capacity and edge capacities; therefore, it is possible to augment flow by more than one in some cases. However, we do not need to do so, as we need never match the same customer with the same facility again. As we want many customers to be assigned to each facility, we set the capacities of edges in G_b to 1; thus, whenever time a customer is assigned to a candidate facility by FINDPAIR, the flow is increased by 1.

E. Set Cover Routine

The need check whether we can select a subset F that covers all customers raises a SETCOVER problem. As this problem is NP-hard, we employ a heuristic solution. After each iteration, we rank all candidate facilities by their (dynamically updated) marginal gains and greedily select the top-k of our selection covers all customers, WMA terminates. Algorithm 3 illustrates this approach. We place all candidate facilities in a heap, organized on the number of customers they cover, and extract facilities from the heap one by one, checking whether all customers served by the last extracted facility remain uncovered. If so, we include the facility in our selection. Otherwise, we recalculate that facility's marginal gain and put it back in the heap. If we reach k facilities without achieving full coverage, we have not yet reached termination.

Algorithm 3 Checking top-k facilities

```
1:  function CHECKCOVER(G_b, k)
2:      heap ← empty heap
3:      for all f_j ∈ F_p do
4:          f_j.m ← |_j(G_b)|
5:          heap.add(⟨f_j, f_j.m⟩)
6:      F ← ∅, ℓ ← 1, ∀i d_i ← 1
7:      for ℓ ∈ 1::k do
8:          f_j ← heap.top
9:          m' ← |_j(G_b)|
10:         if f_j.m ≠ m' then
11:             f_j.m ← m'
12:             heap.add(⟨f_j, f_j.m⟩)
13:         else
14:             F ← F ∪ {f_j}
15:             for all {s_i | s_i ∈ _j(G_b) ∧ d_i = ℓ} do
16:                 d_i ← 0
17:             if ∀i d_i = 0 then
18:                 return ⟨F, ℓ, true⟩
19:     return ⟨F, ℓ, false⟩
```

F. Updating Demands

A crucial operation in WMA is the update of customer demands. A simple approach would increase the demand of all customers by 1 in each iteration. We have found that it is much more effective to increase the demand by 1 only for those customers that were not covered in the last iteration. This selective increase introduces those uncovered customers to more facilities, increasing the chances that they get covered sooner rather than later. Further, we keep track of how recently a facility has been used in a previous iteration to break ties between facilities that incur equal marginal gains.

G. Special Provisions

We have noted that Algorithm 1 (Section IV-C) makes provisions for two special cases: the case in which fewer than k facilities already cover all customers, and the one in which facilities fail to cover some customers even after their demands reach ℓ. Here we describe these provisions.

Algorithm 4, called in Line 11 of Algorithm 1, provides the former special provision: it selects additional facilities until $|F| = k$. Each iteration of the main loop adds to F a new facility $f \in F_p \setminus F$ that is nearest to the customer s having the highest current distance to the nearest facility in F. Thereafter, Lines 14–15 in Algorithm 1 build an assignment using the enlarged F, yielding improved cost.

Algorithm 4 Greedy addition of facilities

```
1: function SELECTGREEDY(F, G_b)
2:     while |F| < k do
3:         s ← arg max_s { min_{f∈F} dist(s, f) | s ∈ S }
4:         f ← arg min_f { dist(s, f) | f ∈ F_p \ F }
5:         F ← F ∪ {f}
```

Algorithm 5 provides the latter special provision: it receives a set of selected facilities F as input and replaces facilities therein to ensure that each connected component G_g of G is allocated sufficient capacity to cover all its customers. Line 3 calculates the difference $g.p$ between the collective capacity of selected facilities that are within connected component g, which we denoted as the set F_g, and the number of customers in g, $|S_g|$. A positive value of $g.p$ indicates that the facilities allocated to g by F suffice to cover the customers therein, with some possible reallocation. A negative $g.p$ means that component g should be offered more facilities or facilities with higher capacities. The loop in Lines 4–9 runs as long as a component with negative $g.p$ exists, substituting the lowest-capacity selected facility f in the highest $g.p$ component g_M with the highest-capacity unselected facility in the lowest-$g.p$ component g_m. Theorem 3 proves that, if a solution exists, this loop terminates.

Algorithm 5 Selecting facilities that cover all customers

```
1: function COVERCOMPONENTS(S, F, G)
2:     for all g ← connected components of G do
3:         g.p ← Σ_{f_j ∈ F_g} c_j − |S_g|
4:     while ∃g : g.p < 0 do
5:         g_m ← arg min_g {g.p}
6:         g_M ← arg max_g {g.p}
7:         f ← arg min_{f_j} { c_j | f_j ∈ g_M }
8:         F ← (F \ {f}) ∪ { arg max_{f_j} { c_j | f_j ∈ g_m, f_j ∉ F } }
9:         Update g.p; g_M.p
10:    return F
```

V. MATCHING OPTIMALITY

Here, we prove that the FINDPAIR routine of Section IV-D yields an optimal assignment, even while using a simpler pruning criterion than the one in [15].

The sets of customers $S = \{s_i\}$ and facilities $F_p = \{f_j\}$ form the two sets of nodes in bipartite graph G_b. E_f^0 is the complete set of all possible edges of G_b, while E' is the set of edges that we are choosing to add to G_b. Also, $dist(s_i, f_j)$ is the weight of edge $(s_i, f_j) \in E_f^0$; by the definition of G_b, $dist(s_i, f_j)$ is the shortest-path distance between customer s_i and facility f_j in graph G; $v.dist$ denotes the length of the shortest path sp from customer s to node v found by Dijkstra on G_b, and $v.p$ the potential of v; there is one Dijkstra execution for each FINDPAIR call.

An assignment is optimal if $\{(s_i, f_j) \in E_f^0 \cap E'\}$, such that adding (s_i, f_j) to E' would yield a better assignment. Notably, each call of FINDPAIR(G_b, s) updates the running assignment as soon as it finds in E' a shortest path sp from customer s to a non-fully occupied facility. Then the assignment is optimal if E_f^0 contains no other path sp', from s to a non-fully occupied facility, such that $sp'.length < sp.length$ [15].

Line 12 of Algorithm 2 verifies this optimality condition. Once the condition is satisfied and the loop is over, the assignment is defined for a current E', and the flow augmentation phase follows.

Theorem 1: Let sp be the shortest path from customer s to a non-fully occupied facility in E' and

$$sp.length \leq \min_{i,j} \{ s_i.dist + dist(s_i, f_j) - s_i.p \}. \quad (4)$$

Then sp is the shortest path from customer s to a non-fully occupied facility in E_f^0.

Proof: The Dijkstra's algorithm call in Line 8 of Algorithm 2 adjusts edge weights by node potentials to remove any negative cycles created by flow augmentation. The original weight of an edge (v_1, v_2) is $w(v_1, v_2) = dist(v_1, v_2)$, while its reduced weight is

$$w_r(v_1, v_2) = dist(v_1, v_2) - v_1.p + v_2.p, \quad (5)$$

where $dist(v_1, v_2)$ is the distance between v_1 and v_2 on G. The length of any path in G_b found by Dijkstra is calculated as the sum of reduced weights. Since $\forall v : v.p \geq 0$, Equation (4) implies that

$$sp.length \leq \min_{i,j} \{ s_i.dist + dist(s_i, f_j) - s_i.p + f_j.p \}. \quad (6)$$

Due to Equation (5), Equation (6) means that

$$sp.length \leq \min_{i,j} \{ s_i.dist + w_r(s_i, f_j) \} \quad (7)$$

Now, assume another path sp' from customer to a non-fully occupied facility exists that is shorter than sp and includes at least one edge $(s', f') \in E_f^0 \setminus E'$. Then the length of sp' includes the reduced weight of the edge (s', f'):

$$sp'.length \geq s'.dist + w_r(s', f') \quad (8)$$

Yet after edge (s', f') is included in G_b, tautologically,

$$s'.dist + w_r(s', f') \leq \min_{i,j} \{ s_i.dist + w_r(s_i, f_j) \} \quad (9)$$

By Equations (8), (9), and (7), we get a contradiction:

$$sp'.length \leq \min_{i,j} \{ s_i.dist + w_r(s_i, f_j) \} \leq sp.length \quad (10)$$

■

In contrast, the threshold used by U et al. [15] is:

$$sp.length \leq \min_{i,j} \{ s_i.dist + dist(s_i, f_j) \} + \theta_{max}^0 \quad (11)$$

$$\theta_{max}^0 = \max \{ s.p \mid f : dist \leq \min_{i,j} \{ s_i.dist + dist(s_i, f_j) \} \} \quad (12)$$

The bound we employ is tighter in case the minimizing s in Equation (4) has $s.dist > \min_{i,j} \{ s_i.dist + dist(s_i, f_j) \}$ and $s.p > \theta_{max}^0$. Besides, this θ_{max}^0-based threshold burdens Algorithm 2 with the overhead of maintaining θ_{max}^0.

VI. Analysis of WMA

Theorem 2: The worst-case complexity of WMA is:

$$O(m|E|\log n + m^2\ell^2(\log(\ell + m) + k\log\ell)) \quad (13)$$

Proof: The matching function of WMA finds a usable facility by iteratively adding new edges to G_b. To that end, it maintains a heap of at most $m\ell$ candidate edges. In the worst case, the heap has to be fully rebuilt at each iteration. If the candidate facility reached by the Dijkstra call on G_b does not satisfy the optimality criterion in Line 12 of Algorithm 2, the loop reiterates. This condition can be violated only if there exists an edge that should be added to E'. As G_b has at most $m\ell$ edges, the Dijkstra result can be invalidated at most $m\ell$ times. Thus, Dijkstra's algorithm is called at most $m\ell$ times. With a heap-based implementation of Dijkstra applied on a sparse connected graph, the complexity is $O(|E'|\log(m+\ell))$, where $|E'|$ grows iteratively from 0 to $m\ell$. While Dijkstra's algorithm runs on G_b with every FindPair() call, we also run another Dijkstra instance on the graph G for each customer. New edges in E' result from successful executions of that instance, while the heaps for these executions per customer persist across FindPair() calls. This gives an additional $O(m|E|\log n)$ complexity. The combined complexity is:

$$O(m|E|\log n + m^2\ell^2\log(m+\ell))$$

CheckPopular() builds a heap of all reached candidate facilities in $O(\ell\log\ell)$. At each greedy iteration, we check the top value of the heap and update it if needed. In the worst case, we may update the whole heap. In total, we do k greedy steps and return false if no set cover is found within the top-k facilities. Then the complexity of the set cover routine per iteration is $O(k\ell(\log\ell + m))$, where m stands for checking whether all customers are covered. The total number of iterations is $m\ell$, since, in the worst case, we increase the demand of only one customer by 1 in each iteration. Putting it all together, the total worst-case time complexity is:

$$O(m|E|\log n + m^2\ell^2(\log(\ell + m) + k\log\ell))$$

As our experiments document, WMA performs far below this worst-case complexity thanks to its pruning ability. ■

Theorem 3: WMA provides a correct solution if one exists.

Proof: The main loop in Algorithm 1 terminates when no d_i is increased, i.e., when a set cover is found or all uncovered customers reach demand $d_i = \ell$. In both cases, it selects a set of facilities F with cardinality $|F| \le k$; if $|F| < k$, Algorithm 4 amends it so that $|F| = k$. Algorithm 5 revises F to ensure that all disconnected components of G are allocated sufficient capacity. Let k_g be the minimum number of facilities required to cover all customers within component g, $k_g = \min_{F' \subseteq F}\{|F'| : \sum_{f_j \in F'} c_j \ge |S_g|; F' \in g\}$. A solution to MCFS is feasible if and only if the budget k suffices to allocate to each component at least k_g facilities, i.e., iff $\sum_g k_g \le k$. Algorithm 5 proceeds towards a state where each component t is allocated a set of $\ge k_g$ facilities in terms of capacity values. Therefore, if a solution is feasible, it eventually terminates. Last, the recursive call in Algorithm 1 produces an optimal bipartite assignment from customers to facilities that does not violate any capacity constraint. ■

Fig. 5: Randomly scattered points used to generate networks.

VII. Experiments

Given the impracticality of approximation algorithms [5], we compare WMA vs. an optimization solver, the Gurobi Optimizer[3], and three simple baselines. Our implementations are in C++. We run all experiments on a 2.2 GHz AMD Opteron 6376 machine with 512GB RAM running Ubuntu 14.04.

A. Baselines

The first baseline follows an approach as in [17]: it divides the input customer set into k buckets and assigns each bucket to the candidate facility node closest to the bucket's centroid. We form buckets containing $\lfloor m/k \rfloor$ consecutive customers using the spatial order defined by a Hilbert space-filling curve [18]. We denote this baseline as Hilbert. The second baseline is a BRNN-based method that iteratively selects k nodes, calculating NLRs at each step; it then runs SIA to produce a final assignment from customers to selected facilities and obtain the objective value. The third baseline is a simplified version of WMA, WMA Naïve. Instead of using an exact bipartite matching, WMA Naïve uses a greedy procedure to satisfy customer demands: in each iteration, it processes customers in a randomly generated order and assigns each customer to its closest d_i candidate facilities that have not yet reached their capacities.

B. Datasets

We use synthetic and real-world networks. Our real-world data are road networks in Aalborg, Riga, Copenhagen, and Las Vegas, obtained from OpenStreetMap[1]. Table III provides statistics. We report objective values and distances in meters.

	Aalborg	Riga New	Copenhagen	Las Vegas
Nodes	50,961	287,927	282,826	425,759
Edges	55,748	322,109	322,349	508,522
Avg degree	2.2	2.2	2.2	2.4
Max degree	7	29	10	21
Avg edge length	30.2	28.7	32.6	50.4

TABLE III: Real-world data sets.

We create synthetic graphs by placing points on a $10^3 \times 10^3$ square. We use two distributions, uniform and clustered. In the clustered case, we place cluster centers uniformly at random. We then assign an equal number of points to each cluster, and form a Gaussian distribution for each cluster with the center as mean and $\sigma^2 = \frac{1}{\text{number of clusters}}$. We connect pairs of points with an edge if they are closer than $\alpha\frac{1}{\sqrt{n}}$, where α is a

[1] https://www.openstreetmap.org/

Fig. 6: Results on uniform distribution, variable graph size.

tunabledensityparameter and n is the network size in nodes. We connect cluster centers to each other in a clique and assign edge weights equal to Euclidean distances. Figure 5 presents examples of such distributions for 10^4 points given $=0, 20$, and 5 clusters, and a uniform distribution. On synthetic networks, we select customer locations uniformly at random. A solution is feasible only if there is enough total capacity to serve all customers, i.e., $\sum_{j=1}^{k} c_j \quad m$; in the uniform case, $c \quad d \frac{m}{k} e$, while an occupancy value, de ned as $o = \frac{m}{c\,k} \quad 1$, indicates how close we are to full capacity.

C. Experiments with Uniform Synthetic Data

We rst evaluate performance on uniform data when varying the graph size. We set $F_p = V$, meaning that a facility can be placed on any node in a graph. We present results for Gurobi for instances where it completed within 24 hours. When it does not complete in 24 hours, we say that it fails.

In Figure 6a, we use density $=2$, which corresponds to an average of two adjacent edges per node. We randomly assign customers to 10% of all nodes and set $k = 0:1m$; hence, we need to place facilities at 1% of all nodes; we set capacities to $c=20$, yielding $o=0:5$, i.e., capacities are twice the minimum required size. BRNN performs signi cantly worse than others, so we eliminate it from further consideration. The objective values attained by Hilbert, WMA, and Gurobi do not differ signi cantly, with WMA performing almost as well as Gurobi. This is because this dataset has a simple uniform structure; Hilbert handles it well, even without taking network distances in consideration. However, Hilbert deviates from WMA as data size grows. WMA exhibits a far more scalable runtime trend than Gurobi, which failed on network sizes beyond 8,192 nodes; WMA scales no less gracefully than Hilbert as the data size grows. WMA Naïve has similar runtime to WMA, yet its objective value is more than double that of WMA across the parameter range.

Figure 6b shows results for a similar con guration, but with higher customer and facility density. Here, we set capacities to $c = 4$ and again obtain an occupancy of $o = 0:5$. Results are similar to the previous ones, though the divergence in objectives between Hilbert, WMA, and WMA Naïve is more pronounced. Further, the achieved objective values are smaller for all algorithms due to higher density (the y-axis range has changed). Gurobi's runtime overhead has increased, as the runtime of LP is highly dependent on the number of variables and constraints; the other algorithms are less sensitive to those parameters. Now the runtime of WMA eventually matches that of Hilbert, even while delivering signi cantly better quality. WMA Naïve is faster than Hilbert on larger networks with higher customer and facilitiy densities, as it eliminates the time-consuming bipartite matching step of WMA.

Figure 6c presents a case with a sparser and less connected network, with $=1:2$, more similar to real road networks. Customer and facility densities lie between those of the previous two cases, with customers as in Figure 6a and facilities as in Figure 6b. We set $c=10$, resulting in an occupancy of $o = 0:2$; this makes the problem relatively easier, balancing out the effect of network sparsity. Even so, the disconnected network structure makes an optimal solution hard to nd. Thus, Gurobi's runtime is signi cantly higher than in the previous case, although the number of decision variables is smaller and the occupancy is looser. WMA also has a higher runtime, and its objective value is closer to that of Hilbert, and similar to that in Figure 6a, where we have half the customers with half the facilities, meaning that the cumulative distances remain relatively stable. Hilbert also has almost the same objective as before, as it considers each component separately, calculating required facilities per component proportionally to the number of customers in the component. On a graph with many small components, this approach quickly leads to good results. As in previous experiments, as the scale increases, WMA Naïve becomes faster than Hilbert.

We also experiment with nonuniform capacities. Figure 6d shows the results with settings like those for Figure 6c, except that now each node is assigned a uniformly random capacity in the range 1 to 10. Hilbert selects locations rst, as if capacities were uniform, and then assigns customers to facilities according to nonuniform capacities using bipartite matching. We observe a similar trend: WMA steadily outperforms Hilbert and WMA Naïve, while Gurobi struggles in terms of runtime.

(a) $m = 0.05n$, $k = 0.1m$, $c = 20$ (b) $m = 0.01n$, $k = 0.8m$, $c = 5$ (c) $m = 0.05n$, $k = 0.2m$, $c = 20$ (d) $m = 0.1n$, $k = 0.1m$, $c = 20$

Fig. 7: Results on Clustered Distribution vs. size, $= 2$, 20 clusters in (a,b,c), 5 in (d)

(a) $n = 10^4$, $m = 10^3$, $k = 10^3$, $c = 15$, $= 2$ (b) $c = 10$, $n = 10^4$, $k = 200$, $= 1.5$ (c) $n = 10^4$, $k = 128$, $o = 0.1$, $= 1.5$ (d) $n = 10^4$, $m = 10^3$, $= 1.5$

Fig. 8: Results on Clustered Distribution, 20 clusters. Variable n, m, and k.

As the problem becomes harder, the gap between the optimal solution provided by Gurobi, and that provided by our heuristics slightly increases in comparison to Figure 6c. The runtime of WMA Naïve is now higher than those of Hilbert and WMA, as it becomes harder for its greedy heuristic to find a set cover when facilities have irregular tight capacities.

D. Experiments with Clustered Synthetic Data

We now turn to clustered synthetic data. Here, the parameter no longer corresponds to the average number of adjacent edges per node, as distances between nodes depend on the standard deviations of Gaussian distributions. We tune this deviation so that clusters cover the plane.

Figure 7 shows results for variable network size settings. These results highlight the advantage of WMA further, as the differences between network and geometric distances become more pronounced with clustered data. Hilbert fails to spot good facility locations, as those depend on the network structure. WMA Naïve stands as an outlier with significantly worse results. In terms of runtime, WMA exhibits similar trends as with uniform distributions.

Figures 7a, 7b, and 7c present experiments with highly clustered points. Figure 7a has more customers and relaxed capacity constraints. WMA provides a good tradeoff between effectiveness and efficiency, with both objective and runtime in-between Hilbert and Gurobi. In this experiment we include BBNN, observing that it also underperforms with clustered data; thus, we again omit it from subsequent figures. Figure 7b depicts results for a smaller occupancy and a smaller capacity. WMA performs more similar to Hilbert, though still outperforming it. Figure 7c shows a different low-occupancy setting. WMA and Hilbert yield smoother curves, showing a clear trend. Yet, the problem becomes more challenging for WMA as size grows.

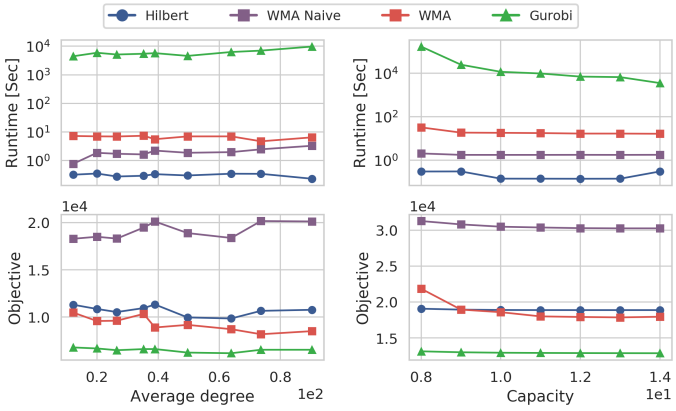
Figure 7d shows results for a case with 5 clusters, coming closer to a uniform distribution, and occupancy $= 0.5$. Here the clustering-based approaches perform well, with Hilbert becoming almost as good as WMA.

Now we consider the effects of varying the major problem parameters other than network size with clustered data.

1) Variable number of candidate facility locations: On a clustered graph of size $n = 10^4$, we randomly pick F_p, varying its size from 40% to 100% of all nodes. Figure 8a presents our results, using dense customer distribution and high capacity. Gurobi failed for F_p sizes above 60% of all nodes. Hilbert is sensitive to the size of F_p due to its clustering nature. In contrast, both WMA variants show stable runtime and objective, with the regular WMA achieving objective values very close to those of Gurobi. This indicates that WMA finds good alternatives in case some nodes are not candidate facilities, while Hilbert falters.

2) Tuning Customers and Facilities: Figures 8b and 8d present our results when varying the numbers of customers and facilities, respectively. The objective increases as the number of customers grows, but drops as the number of facilities grows, other parameters being equal. Remarkably, the runtimes of the WMA variants drop with increasing facilities as well, as they perform fewer iterations. Figure 8c scales up the amount of customers, also allowing for multiple customers per node, with occupancy of $o = 0.1$. WMA slightly outperforms Hilbert, and both are very close to Gurobi in terms of objective. WMA Naïve shows worse results. Gurobi fails for large numbers of customers.

3) Effect of Graph Density α: We now study the effect of graph density α with 5-cluster data. Figure 9a shows the results. As α affects the average degree, the x-axis shows the measured average degree instead of α, resulting in non-equal parameter gaps. The objective improves for WMA with larger degree, coming closer to the optimal solution by Gurobi and outperforming Hilbert and WMA Naïve. WMA finds better locations as optimal facilities become available within fewer hops, thereby simplifying the set cover sub-problem. Gurobi is surprisingly stable, showing that a network with no throughput constraints on edges is resistant to intermediate density increase.



(a) $n = 10^4$, $m = 1000$, $k = 300$ (b) $n = 10^4$, $m = 1000$, $k = 100$

Fig. 9: Effect of Density ($c = 10$), Capacity c ($\alpha = 1.5$).

4) Effect of Capacity c: Last, we vary capacity values as Figure 9b shows. The capacity has little effect on the result quality, except in the challenging case of very small capacity, where the occupancy is high. This is reasonable: once a good matching is achieved for some capacity, letting capacity grow further does not improve the solution. Remarkably, While other algorithm have stable runtime, Gurobi gains in efficiency as capacity grows, rendering the optimization easier.

E. Experiments with Real Data, Uniform Capacities

Now we turn our attention to the performance of WMA on real-world data, using four urban road network data sets of different size. We first examine the uniform capacity case with $F_p = V$. We distributed 512 customers randomly in each city network, and tasked the algorithms with placing 51 facilities. We could only obtain results for WMA and Hilbert, as Gurobi did not terminate on such data within one week due to the large number of candidate facility locations.

Table IV presents quality and runtime results. WMA achieves a solution that is around 30% better than the most competitive Hilbert basline on all cities except Las Vegas. Las Vegas has a regular grid-like road network structure (see Figure 1a), rendering clustering approaches more effective; thus, we obtain only a 9% improvement.

	BRNN	Hilbert	WMA Naïve	WMA
Aalborg	3.51 / 1 min	0.59 / 10 s	0.83 / 5 min	0.41 / 5 min
Riga	6.02 / 6 min	1.30 / 5 min	1.86 / 3.0 h	0.90 / 3.3 h
Copenhagen	4.20 / 6 min	0.93 / 5 min	1.29 / 3.7 h	0.66 / 5.9 h
Las Vegas	3.67 / 6 min	1.16 / 13 min	1.63 / 12.4 h	1.06 / 7.5 h

TABLE IV: Objective $[\cdot 10^6]$ / Runtime, $m = 512$, $k = 51$, $c = 20$, $l = n$

Further, we test the scalability of WMA on the Aalborg network, for growing number of both customers and facilities, with fixed occupancy $o = 0.5$, $c = 20$, and setting $k = 0.1m$. Figure 10 shows that the advantage of WMA manifests itself as the numbers of facilities and customers grow: its runtime is aligned with that of Hilbert, and it scales well with the problem size, while the quality improves continuously over that of Hilbert. WMA Naïve achieves a worse objective than WMA, although it is competitive in terms of runtime. Interestingly, as both WMA variants struggle to find a feasible set cover with sparse customers and facilities, their runtimes are at their lowest in middle problem sizes. Further, we ran BRNN on this real-world data set in order to reexamine the conclusions reached on synthetic data. The objective of BRNN grows rapidly, indicating its instability on real-world tasks. In addition, BRNN presents the worst runtime behavior, as it has to repeatedly calculate NLR intersections. Last, Gurobi failed in these experiments.

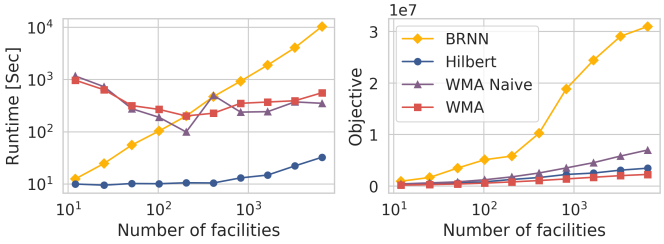


Fig. 10: Aalborg experiment, $o = 0.5$, $\ell = n = 50961$

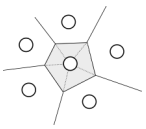
F. Experiments with Real Data, Nonuniform Capacities

We now consider real-world data with nonuniform capacities and $\ell < n$, which corresponds to the most general case of the MCFS problem. The problem is to select a set of facilities among diverse options, each associated with a capacity derived from real-world constraints. We study two applications: (i) the selection of meeting places for coworkers, and (ii) the selection of bike docking stations.

1) Coworking: this trend allows independent professionals to share a working environment [19], saving expenses for office rental while enjoying the advantages of the structure

and community of working with others [20]. In addition, coworking spaces enable group meet-ups and other temporary activities. Cafés and restaurants provide affordable coworking options, offering part of their spaces during non-rush hours. We let city amenities serve as facilities, while their daily operational hours define their nonuniform capacities. Assuming uniform utilization during these working hours, a number of coworkers need to select coworking facilities out of potential options. We consider this problem on data from two cities: Las Vegas and Copenhagen.

a) Las Vegas case: We use Yelp[2] data to generate a distribution of customers from known facility occupancy, using an existing technique [13]; we divide space to Voronoi cells, and each cell to triangles, as illustrated on the figure to the right. The number of customers in a triangle is:



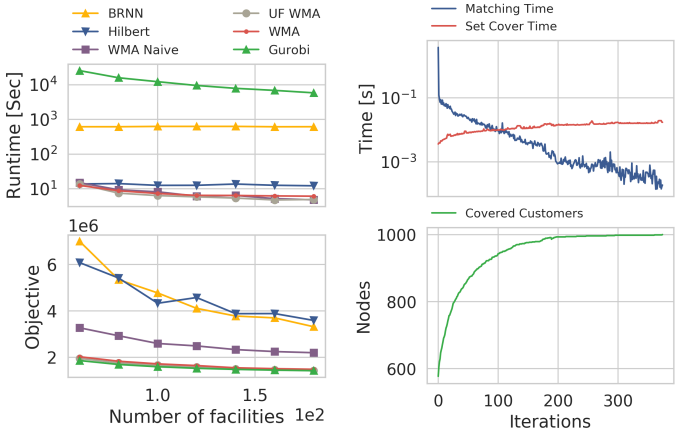
$$m = O_i \quad \omega \quad \frac{O_j}{\sum_j O_j} + (1 \quad \omega) \quad \frac{Area}{Area_l}$$

where O_i is the occupancy of the central node, O_j is the occupancy of a neighbor node, $Area$ is the area of a triangle, $Area_l$ is the area of the Voronoi cell, and ω is a parameter set to 0.5 by default [13]. We use user check-ins available from Yelp, considering all restaurants as candidate facility locations, and derive a customer distribution. Instead of using Euclidean Voronoi cells, we adapt the approach to road networks via network distance calculations. We then generate customer numbers proportional to derived values. We place *1,000 customers* at appropriate road network nodes using this method. We downloaded the road map data from OpenStreetMap, and we identified 4089 venues with available operational hours in the Yelp dataset. Figure 1a shows the distribution of customers and facilities in the city center.

b) Copenhagen case: We use data from the "Open Data København" portal[3]. We generate a customer distribution proportional to that of district populations in Copenhagen, and randomly place *200 customers* at road network nodes. We obtained information about cafés and restaurants from OpenStreetMap; 164 venues have operational hours available (the average is 9 hours in both cities), which we use as a proxy for a venue's capacity. Figure 1b shows the distribution of customers and facilities in the city center.

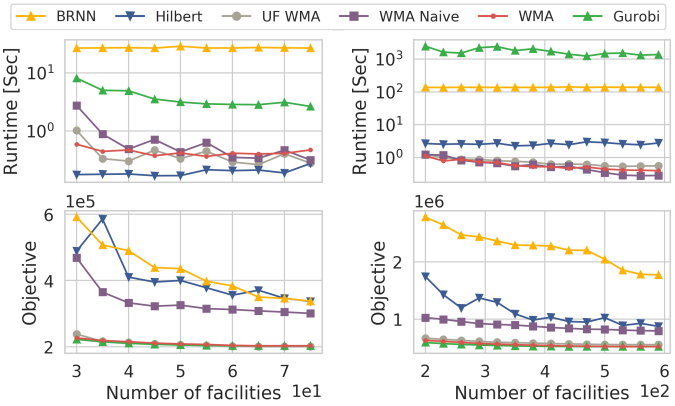
We solve the problem in two ways: (i) the *Direct* solution, whereby WMA accommodates the given nonuniform capacities and proceeds as usual; and (ii) the *Uniform First* (UF) solution, where we first solve the problem as if capacities were uniform using the average capacity, and then reassign customers to facilities using the real nonuniform capacities in a single bipartite matching step. This alternative might represent a better heuristic, in case it detects better locations under uniform capacities, before specializing to the nonuniform ones; this is a conjecture worth investigating.

Figures 12a and 13a show our results on the Direct and UF versions of WMA, the optimal solution provided by Gurobi, and the three baselines — Hilbert, BRNN, and WMA Naïve. Since WMA Naïve yields poor quality vs. WMA, we do not include results for its UF variant for the sake of readability. As more facilities can be used to satisfy the given demand, the problem becomes easier. Since we use a small F_p, Gurobi solves the problem in reasonable runtime; that would not be so if we had more candidate facilities or a country-scale network. For both cities, WMA outperforms Gurobi's runtime by several order of magnitude and matches its quality. UF WMA meets the optimal solution as well in most cases. The accuracy of Hilbert improves with increasing number of facilities, replicating the trend observed with synthetic data (Figure 8d). WMA Naïve shows a better objective than Hilbert, as also witnessed in Figure 8a: Hilbert cannot adapt to a small F_p, leading to objectives as bad as BRNN; BRNN has even worse runtime than Gurobi in the Copenhagen case.



(a) Selection of meeting places (b) WMA Profiling

Fig. 12: Las Vegas experiments.



(a) Selection of meeting places (b) Bike docking case study

Fig. 13: Copenhagen experiments.

We also report statistics on the operation of WMA for selection of meeting places in the Las Vegas network with $k = 600$. Figure 12b shows 3 quantities: covered customers at the end of each iteration, time for matching, and time for the set-cover operation. The set-cover time is lower than the matching time except for later iterations where reassignment

[2] https://www.yelp.com/dataset/ [3] http://data.kk.dk/