Pick, Pack, & Survive: Charging Robots in a Modern Warehouse based on Online Connected Dominating Sets

Heiko Hamann
Institute of Computer Engineering, University of Lübeck, Germany
https://www.iti.uni-luebeck.de
hamann@iti.uni-luebeck.de

Christine Markarian
Heinz Nixdorf Institute, Paderborn University, Germany
https://www.uni-paderborn.de
christine.markarian@upb.de

Friedhelm Meyer auf der Heide
Heinz Nixdorf Institute, Paderborn University , Germany
https://www.uni-paderborn.de
fmadh@upb.de

Mostafa Wahby
Institute of Computer Engineering, University of Lübeck, Germany
https://www.iti.uni-luebeck.de
mostafa.wahby@uni-luebeck.de

Abstract

The modern warehouse is partially automated by robots. Instead of letting human workers walk into shelfs and pick up the required stock, big groups of autonomous mobile robots transport the inventory to the workers. Typically, these robots have an electric drive and need to recharge frequently during the day. When we scale this approach up, it is essential to place recharging stations strategically and as soon as needed so that all robots can survive. In this work, we represent a warehouse topology by a graph and address this challenge with the *Online Connected Dominating Set* problem (OCDS), an online variant of the classical *Connected Dominating Set* problem [10]. We are given an undirected connected graph $G = (V, E)$ and a sequence of subsets of V arriving over time. The goal is to grow a connected subgraph that dominates all arriving nodes and contains as few nodes as possible. We propose an $\mathcal{O}(\log^2 n)$-competitive randomized algorithm for OCDS in general graphs, where n is the number of nodes in the input graph. This is the best one can achieve due to Korman's randomized lower bound of $\Omega(\log n \log m)$ [14] for the related *Online Set Cover* problem [2], where n is the number of elements and m is the number of subsets. We also run extensive simulations to show that our algorithm performs well in a simulated warehouse, where the topology of a warehouse is modeled as a randomly generated geometric graph.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases connected dominating set, online algorithm, competitive analysis, geometric graph, robot warehouse, recharging stations

Digital Object Identifier 10.4230/LIPIcs.FUN.2018.22

Acknowledgements This work was partially supported by the Federal Ministry of Education and Research (BMBF) as part of the project ‘Resilience by Spontaneous Volunteers Networks for

Coping with Emergencies and Disaster' (RESIBES) - grant number 13N13955 to 13N13957 and the German Research Foundation (DFG) within the Collaborative Research Center 'On-The-Fly Computing' (SFB 901).

1 Introduction

Ever wondered what happens in a warehouse when Amazon orders are placed? Modern warehouses (so-called pick-pack-and-ship warehouses [20]) are nowadays run by hundreds of mobile robots (cf. swarm robotics [4]). A popular example is the Kiva system [6, 11] that is used by the Amazon.com corporation. When orders are placed, robots bring inventory to warehouse workers while workers stay stationary. Since robots need to act autonomously, the system is considered as a multi-agent system that requires rather complex path planning and resource allocation. In this paper, we focus on the specific issue of recharging robots. The robots have rechargeable lead-acid batteries that need to be recharged frequently throughout the day. In the case of Kiva robots, for example, they recharge every 55 minutes for five minutes. Once a robot detects a low battery level, it drives autonomously to the next charging station. Such warehouses can have several 10,000 square meters and do 200,000 picks per day [11]. When one wants to scale up, it will get important to position the recharging stations strategically, otherwise the system's efficiency will decrease or robots will even run out of energy before they reach a charging station. In addition, extreme differences in the distribution shipping volumes per day are typical for Internet retailers, for example, before Christmas. Hence, it is important to be able to scale up on demand.

In a hypothetical scenario, one can imagine a huge warehouse where certain areas are more busy during the week than others. Hence, it would be useful to have movable charging stations and to position them *on demand* in the respective busy areas. In addition, one can assume that any two charging stations should not be too far apart from each other, because robots may be required to move from any of these busy areas to any other (i.e., there may be a massive task allocation going on, which is however beyond this paper's scope). As robots may not always be fully charged when they start their travel and because we want to be scalable to huge warehouses, robots may even require to recharge on their move. So besides positioning charging stations at busy areas we also may position charging stations in between busy areas as kind of relay stations to guarantee reachability of all busy areas. We represent the warehouse topology as a graph whose nodes represent potential locations for recharging stations and busy areas. Recharging stations are placed sequentially on the nodes depending on current demands. The edges in the graph represent distances that can safely be covered by robots with a battery of average charge. A sequence of requests arrives over time such that a request consists of a subset of nodes representing busy areas that require a charging station nearby. These areas cannot be known in advance and must be provided with a charging station as soon as revealed. Charging stations need to be placed on a small subset of nodes such that all busy areas have a nearby station. Moreover, to guarantee reachability of all busy areas, we require that the subset of nodes induces a connected subgraph (see Fig. 1). The goal is to serve all requests upon their arrival by placing *as few* recharging stations as possible and without knowing future busy areas. At the core of this scenario we have a complex online optimization problem, the *Online Connected Dominating Set* problem (OCDS), defined as follows.

▶ **Definition 1.** (Online Connected Dominating Set problem - OCDS) Given an undirected connected graph $G = (V, E)$ and a sequence of subsets of V arriving over time. In each step, nodes from V are revealed and need to be dominated by a connected subgraph of G - a

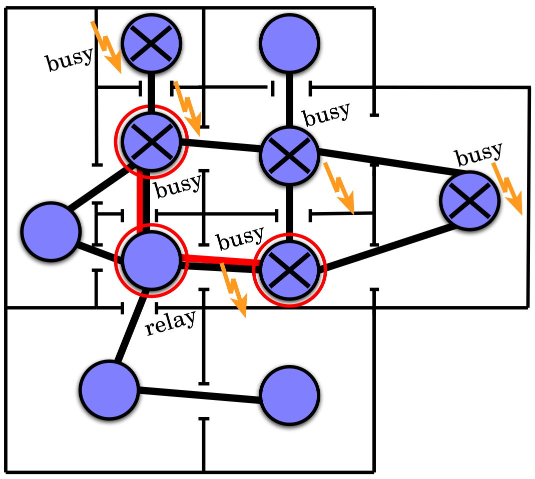


Figure 1 Warehouse topology modeled as a graph. Nodes represent busy areas and potential locations for recharging stations. Three recharging stations are placed (in red) one of which acts as a relay station.

subset is dominated if each node in it is either in the subgraph or adjacent to a node in the subgraph. OCDS asks to grow a connected subgraph containing as few nodes as possible without knowing future nodes in advance.

OCDS is an online variant of the classical *Connected Dominating Set* problem (CDS). Given a graph $G = (V, E)$, CDS asks to construct a minimum subset S of V such that each node in V is dominated and the subgraph induced by S is connected. CDS has been widely studied in both theory and practice, with various applications in wireless networks [21]. It is $\mathcal{NP}$-complete even in planar graphs [9]. For general graphs, it admits an $\mathcal{O}(\ln \Delta)$-approximation, where Δ is the maximum node degree of the input graph [10]. This is the best possible unless $\mathcal{NP} \subset DTIME(n^{\log n \log n})$ [10]. We propose a randomized online algorithm for OCDS and evaluate it using the classical notion of *competitive analysis*. Given an input sequence σ, let $\mathcal{C}_A(\sigma)$ and $\mathcal{C}_{OPT}(\sigma)$ denote the cost incurred by an algorithm A and an optimal offline algorithm OPT, respectively. Algorithm A is said to be *c-competitive* (or has competitive ratio c) if there exists a constant α such that $\mathcal{C}_A(\sigma) \leq c \cdot \mathcal{C}_{OPT}(\sigma) + \alpha$ for all input sequences σ. We show that our proposed algorithm is $\mathcal{O}(\log^2 n)$-competitive against an oblivious adversary, where n is the number of nodes in the input graph. This is the best one can achieve due to Korman's randomized lower bound for the related *Online Set Cover* problem (OSC) [2], an online variant of the classical *Set Cover* problem. OSC is defined as follows. Given a universe $\mathcal{U}$ of elements and a collection $\mathcal{S}$ of subsets of $\mathcal{U}$. In each step, elements from $\mathcal{U}$ are revealed and need to be covered by subsets from $\mathcal{S}$. OSC asks to cover all given elements while minimizing the total number of chosen subsets. Korman [14] has shown a randomized lower bound of $\Omega(\log n \log m)$ for OSC, where n is the number of elements and m is the number of subsets. It has been shown that the *Set Cover* problem can be reduced to CDS [8, 10, 15]. By a similar argumentation, a reduction from OSC to OCDS can be made, implying a randomized lower bound of $\Omega(\log^2 n)$ for OCDS. We also run extensive simulations to show that our algorithm performs well in a simulated warehouse, where the topology of a warehouse is modeled as a randomly generated geometric graph.

2 Related Work

In this section, we give an overview of literature related to online connected dominating sets and robot warehouses, respectively.

2.1 Online Connected Dominating Sets

While there are many works that address CDS and its variants in the offline setting [10, 21], only few consider the online setting. Eidenbenz [7] has studied an online variant of CDS, in which the input graph is restricted to a class (tree, unit disk graph, bounded degree graph) and is not known in advance. It is revealed over time such that in each step a node is either inserted or deleted. The goal is to maintain a connected subgraph that contains as few nodes as possible and dominates all present nodes. Eidenbenz has shown that a simple greedy approach attains a $(1 + \frac{1}{\mathcal{OPT}})$-competitive ratio in trees - where $\mathcal{OPT}$ is the cost of the optimal offline solution, an $(8 + \epsilon)$-competitive ratio in unit disk graphs - for arbitrary small $\epsilon > 0$, and b-competitive ratio in b-bounded degree graphs. Note that in OCDS, the input graph is given in advance (offline) and only the nodes that need to be dominated are revealed over time (online). For the *Online Set Cover* problem defined earlier, Alon *et al.* [2] have proposed an $\mathcal{O}(\log n \log m)$-competitive deterministic algorithm, where m is the number of sets and n is the number of elements. They have also shown a nearly matching deterministic lower bound of $\Omega(\frac{\log n \log m}{\log \log n + \log \log m})$ for interesting values of m and n. Naor *et al.* [17] have studied an online variant of the classical *Steiner Tree* problem with weighted nodes and edges, the *Online Node-Weighted Steiner Tree* problem (ONWST), defined as follows.

▶ **Definition 2.** (Online Node-Weighted Steiner Tree problem - ONWST) Given a graph $G = (V, E)$ with weighted nodes and edges. In each step, nodes from V (called terminals) are revealed and need to be connected to each other. ONWST asks to grow a subgraph S (called a Steiner tree) that connects all terminals while minimizing the total cost of edges and nodes in S.

Naor *et al.* [17] have given an $\mathcal{O}(\log n \log^2 k)$-competitive randomized algorithm for ONWST, where k is the number of terminals and n is the number of nodes in the input graph. A special case of ONWST in which all edges have cost 0 and all nodes have cost 1 can be reduced to OCDS by setting the nodes to be dominated as terminals. This variant will appear in the analysis of our algorithm in Section 4.

2.2 Robot Warehouses

Even though it may seem a rather specialized problem of positioning charging stations for multi-robot and swarm systems, there is a rich literature about this problem. Kannan *et al.* define the *Autonomous Recharging Problem* (ARP) as the problem of planning and coordinating when, where, and how to recharge robots [13]. They consider both static and mobile charging stations and how to find recharging schedules to maximize efficiency of the system. Couture-Beil and Vaughan argue that suboptimal positioning of the charging station may cause spatial interference. Therefore, they study an adaptive mobile charging station [5] where the charging station itself is a mobile robot. They argue that in a dynamic task the correct and adaptive placement of the charging station is even more important. A similar approach is studied by Arvin *et al.* for recharging a robot swarm with a mobile charging station [3]. Kamagaew *et al.* discuss the problem of how to switch from a central approach to a multiplicity of small self-organizing transport units in the warehouse [12]. They study a decentralized control system for autonomous vehicle swarms and how methods of swarm intelligence can help. In summary, the *Autonomous Recharging Problem* and in particular the optimal positioning of charging stations in multi-robot systems is relevant and currently investigated intensively.

3 Online Algorithm

In this section, we propose an online algorithm for OCDS. Given an undirected connected graph $G = (V, E)$. In each step t, a subset $D_t \subseteq V$ of nodes is given. Let S_t denote the set of nodes selected by the algorithm in step t. Let $S := \sum_t S_t$, $\forall t$ be the solution set and $D := \sum_t D_t$, $\forall t$ be the demand set. We fix a cost $c_j = 1$ to each $i \in V$. The algorithm runs in two phases. In the first phase, it dominates each node in D_t with a subset $S'_t \subseteq V$. In the second phase, it connects the nodes in S'_t with an additional subset $S''_t \subseteq V$, forming the set $S_t = S'_t \cup S''_t$. We call the set of nodes that can dominate i the *candidates* of i and denote it as Q_i. The algorithm uses a *randomized rounding* approach commonly used in designing online algorithms [17], in both of its phases. In Phase 1, to dominate a given node i, the algorithm buys fractions of i's candidates until they sum up to 1. These fractions are then rounded using a randomized process in attempt to add at least one candidate into the solution. If the latter does not happen, the algorithm arbitrary adds one of the candidates into the solution. In Phase 2, for each node j selected by the algorithm in Phase 1, the algorithm chooses a representative node k from D that is dominated by j and connects k to the current solution S, as follows. The *minimum cut value* (or *maximum flow*) between a node k and a set S is the smallest total weight of edges which if removed would disconnect k from S. These edges form a *minimum cut*. The algorithm transforms the weights of the graph from the nodes to the edges such that the weight of an edge (u, v) is set to $\min\{w_u, w_v\}$, where w_u and w_v are the weights of u and v, respectively. If either u or v is in S, the weight of edge (u, v) is set to the weight of the one not in S. It then constructs a minimum cut C between k and S by running the algorithm by Schroeder *et al.* for undirected connected edge-weighted graphs [19]. For an edge $(u, v) \in C$ and $w_u < w_v$, u is called a *minimum cut node*. As long as the maximum flow between k and S is less than 1, the algorithm constructs a minimum cut and increases the weights of the corresponding minimum cut nodes. Then, it rounds these weights using a randomized process in attempt to add at least one path connecting k to S into the solution. If the latter does not happen, the algorithm adds a cheapest path that connects k to S into the solution. For **Phase 1**, we maintain a fraction f_i to each node $i \in V$, initially set to zero and non-decreasing throughout the algorithm. We define a random variable μ as $\mu := \min\{X_{(q)}\}$ such that $2\lceil\log(n+1)\rceil$ independent random variables $X_{(q)}$ are distributed uniformly in the interval $[0, 1]$ and $1 \leq q \leq 2\lceil\log(n+1)\rceil$. As for **Phase 2**, we maintain a weight w_i to each $i \in V$, initially set to zero and non-decreasing throughout the algorithm. We define a random variable μ' as $\mu' := \min\{X_{(q)}\}$ such that $2\lceil\log(n+1)\rceil$ independent random variables $X_{(q)}$ are distributed uniformly in the interval $[0, 1]$ and $1 \leq q \leq 2\lceil\log(n+1)\rceil$. The two phases of the algorithm are depicted in **Algorithm 1** below. Fig. **??** shows the result of a two-step run on a randomly generated connected graph of 10 nodes.

4 Competitive Analysis

We dedicate this section to showing that the algorithm above is $\mathcal{O}(\log^2 n)$-competitive for OCDS, where n is the number of nodes in the input graph. Recall that $\Omega(\log^2 n)$ is a lower bound for OCDS [14].

Algorithm 1

```
Phase 1. For each $i \in D_t$ not dominated by some node in $S \cup S'_t$,
    Step 1: while $\sum_{j \in Q_i} f_j < 1$,
                for each $j \in Q_i$: $f_j = f_j \cdot (1 + 1/c_j) + \frac{1}{|Q_i| \cdot c_j}$
    Step 2: add $j \in Q_i$ to $S'_t$ if $f_j > \mu$
    Step 3: if $i$ is not dominated by some node in $S'_t$, add an arbitrary $j \in Q_i$ to $S'_t$
Phase 2. For each $j \in S'_t$, if it is adjacent to some node in $S$, add $j$ to $S$, else,
    Step 1: choose a node $k$ from the set $D_t$ dominated by $j$ and add it to $S''_t$
    Step 2: if $k$ is not connected to $S$,
      While the maximum flow between $k$ and $S$ is not 1,
       - Construct a minimum cut and select the minimum cut nodes $K \subseteq V \setminus S$
       - For each $i \in K$, set $w_i = w_i \cdot (1 + 1/c_i) + \frac{1}{|K| \cdot c_i}$
      Add $v \in V$ to $S''_t$ if $w_v > \mu'$
      If $k$ is still not connected via nodes in $S''_t$, choose a shortest path connecting $k$ to $S$
and add its nodes to $S''_t$
    Step 3: Add $j$ and the nodes in $S''_t$ that connect $j$ to $S$, to $S$
```

▶ **Lemma 3.** *The expected cost C_1 of the algorithm in Phase 1 is at most $\mathcal{O}(\log^2 n) \cdot \mathcal{OPT}$, where $\mathcal{OPT}$ is the cost of the optimal offline solution.*

Proof. Let C_1 be the expected cost of the algorithm in Phase 1 and let $\mathcal{OPT}$ be the cost of the optimal offline solution. For any $j \in V$, the probability that the algorithm adds j into the solution is the probability that $f_j > \mu$, which is at most $2\log(n+1) \cdot f_j$. Adding up over all $j \in V$, the expected cost C_1 of the algorithm will be at most:

$$\sum_{j \in V} 2\log(n+1) \cdot f_j = 2\log(n+1) \cdot \sum_{j \in V} f_j \tag{1}$$

Next we bound $\sum_{j \in V} f_j$. Whenever we want to dominate a node $i \in V$ not yet dominated, we increase the fraction corresponding to each of its $|Q_i|$ candidates. The fraction f_j of each candidate $j \in Q_i$ is increased by $\left(\frac{f_j}{c_j} + \frac{1}{|Q_i| \cdot c_j}\right)$. Summing up over all i's candidates, we get an overall fractional increase of:

$$\sum_{j \in Q_i} \left(\frac{f_j}{c_j} + \frac{1}{|Q_i| \cdot c_j}\right) \leq 2 \tag{2}$$

The above inequality holds since $\sum_{j \in Q_i} f_j \leq 1$ before any fractional increase and $c_j = 1$: $\forall j \in V$. An optimal solution must contain at least one node. Let us fix any such node $p \in Q_i$ ($\mathcal{OPT} \geq c_p$). The fraction f_p corresponding to p becomes at least 1 after at most $\mathcal{OPT} \cdot \log |Q_i|$ fractional increases and hence no further fractional increases can be made. With this observation together with inequality 2, we conclude that:

$$\sum_{j \in V} f_j \leq \mathcal{OPT} \cdot 2 \cdot \log(\Delta + 1) \tag{3}$$

The above inequality holds since $|Q_i| \leq \Delta + 1$, where Δ is the maximum number of nodes adjacent to any node in V. So far we have measured the cost of the algorithm during the first two steps. Equations 1 and 3 yield a cost of $\mathcal{O}(\log^2 n) \cdot \mathcal{OPT}$. It remains to measure the additional cost incurred by Step 3 of Phase 1, which is necessary to guarantee a feasible

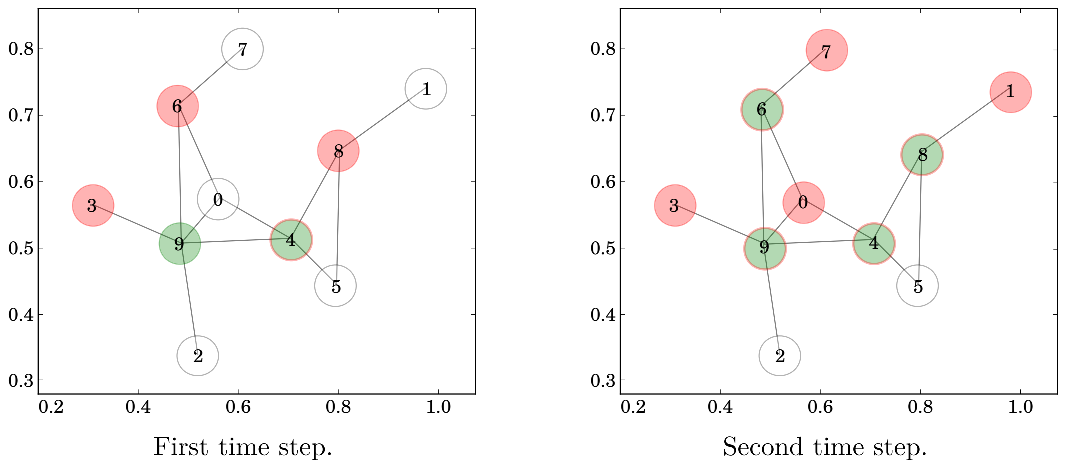


Figure 2 Two-step run of **Algorithm 1** on randomly generated connected graph with $|V| = 10$. Nodes in red or with red border represent demand nodes and nodes in green represent solution nodes.

solution. For a single $1 \le q \le 2\lceil \log(n+1) \rceil$, the probability that a node i is not covered is at most:

$$\prod_{j \in Q_i} (1 - f_j) \le e^{-\sum_{j \in Q_i} f_j} \le 1/e$$

The last inequality holds because the algorithm guarantees in the first step that $\sum_{j \in Q_i} f_j \ge 1$. Hence, the probability that i is not covered, for all $1 \le q \le 2\lceil \log(n+1) \rceil$, is at most $1/n^2$. The additional expected cost for each of the at most n nodes is then upper bounded by $n \cdot 1/n^2 \cdot \mathcal{OPT}$, since the cost of adding one additional node is clearly less than $\mathcal{OPT}$. Therefore, we conclude that $C_1 \le \mathcal{O}(\log^2 n) \cdot \mathcal{OPT}$. ◀

▶ **Lemma 4.** *The expected cost C_2 of the algorithm in Phase 2 is at most $C_1 + \mathcal{O}(\log^2 n) \cdot \mathcal{OPT}$, where $\mathcal{OPT}$ is the cost of the optimal offline solution and C_1 is the expected cost of the algorithm in Phase 1.*

Proof. in Appendix A. ◀

Adding C_1 and C_2 from Lemma 3 and Lemma 4, respectively, ultimately leads to the theorem below.

▶ **Theorem 5.** *There is an optimal $\mathcal{O}(\log^2 n)$-competitive randomized algorithm for the Online Connected Dominating Set problem (OCDS).*

5 Simulation Study

In this section, we show that our proposed algorithm for OCDS performs well in a simulated warehouse, where the topology of a warehouse is modeled as a randomly generated connected geometric graph.

Since no other algorithm has been proposed for OCDS in the literature, the only algorithm we could compare to is offline, that is, an algorithm for CDS. Since the latter is $\mathcal{NP}$-complete, the comparison is made against an optimal $\mathcal{O}(\ln \Delta)$-approximation algorithm for CDS, based on a greedy approach, following Guha *et al.* [10]. Recall that an input to CDS is a graph $G = (V, E)$ in which all nodes need to be dominated and to which the algorithm reacts once.

In OCDS, we are given a graph $G = (V, E)$ and the nodes to be dominated are given in steps. The algorithm needs to react to each step without knowing about future steps. The comparison is made at the final step, after which the algorithm had accumulated its solution over the steps. Without loss of generality, we assume all nodes at this step have been asked to be dominated in at least one of the steps. We perform our simulation study in five different settings, each with different number of nodes (i.e., potential locations for recharging stations): $|V| \in \{50, 100, 150, 200, 250\}$. For each setting, we perform 100 runs. Since the algorithm is randomized, we run it 10 times for each instance and observe its mean, best, and worst case performance. In each simulation run, we generate a connected geometric graph $G = (V, E)$ whose nodes are placed uniformly at random in a unit square Euclidean plane (see Fig. 2). The connectivity threshold r is set to a small value of 0.17 to provide graphs with minimal number of edges. A value below 0.17 decreases the chance to generate geometric graphs that are connected. The input to the approximation algorithm is $G = (V, E)$ whereas the input to the online algorithm is composed of offline and online parts. The offline part is $G = (V, E)$ and the online part is the sequence of subset of nodes revealed over time. In each step t, a subset D_t of nodes is revealed (i.e., the current demand of recharging stations at step t). The cardinality of D_t is uniformly sampled from the interval $\left[0, \dfrac{|V| - \sum_{i=1}^{i=t-1} |D_i|}{2}\right]$. The subset D_t excludes previously given nodes and is sampled randomly.

Results. The boxplots in Fig. 3 show the performance of the online algorithm and the offline approximation algorithm for $|V|$ = 50, 100, 150, 200, and 250. For each $|V|$, 100 instance graphs are generated.

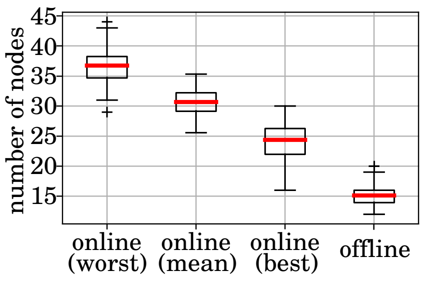
- The online algorithm is run 10 times for each instance graph: its *mean*, *best*, and *worst* case performance for each of the 100 instances are recorded.
- The offline approximation algorithm is run for each instance graph: its performance for each of the 100 instances is recorded.
- The datasets represented by the boxplots shown in Fig. 3 are all pairwise statistically significantly different (i.e., all p-values ≤ 0.05 based on Wilcoxon signed-rank test).

We define two performance measures:

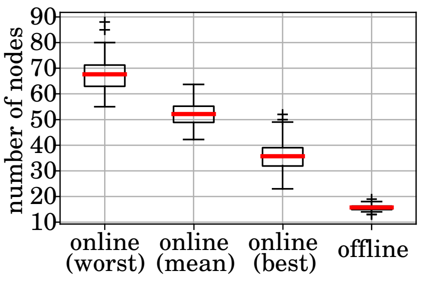
1. the *percentage difference*, which is the number of nodes the online algorithm outputs *more*, in comparison to the offline algorithm, in percentage.
2. the *average competitive ratio*, which is the ratio of the number of nodes outputted by the online algorithm to that by the offline approximation algorithm.

Table 1 shows the two performance measures for each $|V|$ in the mean, best, and worst cases, evaluated by taking the average over all 100 instance graphs. Notice that the percentage difference never exceeds 51.08% in the worst case. For $|V| = 50$, it is as small as 11.10% in the best case. Moreover, as $|V|$ grows, the percentage difference does not increase significantly. Instead, it sometimes gets smaller - for instance, in the mean case as $|V|$ grows from 150 to 250 (from 34.25% to 32.15% to 30.84%). One reasoning for the latter might be the following. Recall that the connectivity threshold r is set to 0.17 for all $|V|$ and the nodes are placed uniformly at random in a bounded region. Thus the generated graphs become denser as $|V|$ grows, resulting in less complex solutions.

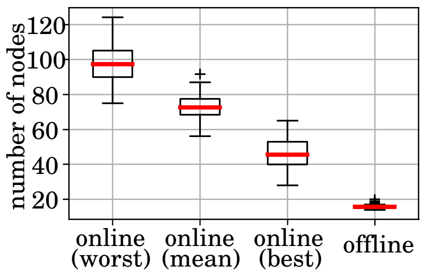
In terms of average competitive ratio, notice that the online algorithm's worst output is at most 6.83 times the output of the offline algorithm. Moreover, for $|V| = 50$, the online algorithm is nearly optimal with an average competitive ratio of 1.63 and 1.29 in the mean and best cases, respectively.



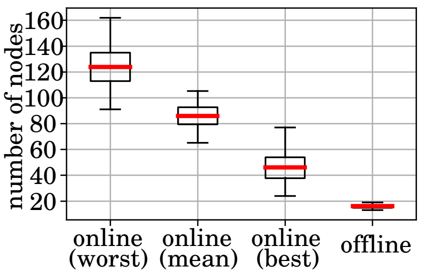
(a) $|V| = 50$



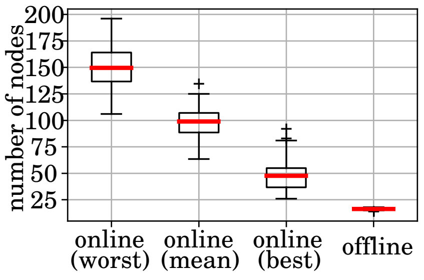
(b) $|V| = 100$



(c) $|V| = 150$



(d) $|V| = 200$



(e) $|V| = 250$

Figure 3 Boxplots showing the performance of the online algorithm (mean, best, and worst cases) and the offline algorithm for $|V| = 50$, 100, 150, 200, and 250. All datasets are pairwise statistically significantly different ($p \leq 0.05$, Wilcoxon signed-rank test).

6 Open problems

We have presented a provably optimal online approach to positioning recharging stations in a robot warehouse. Partially automated robot warehouses are state-of-the-art and in the future they will need to scale up. The problem of correctly placing charging stations is highly relevant and intensively studied. Efficient online algorithms for placing charging stations are essential to ensure scalability and efficiency. This work has been a small attempt towards this goal. There is certainly much more to do. As a first next step, it would be interesting to target better competitive ratios for OCDS in restricted graph classes, e.g., by employing properties of geometric graphs. Another important direction is to consider stations that do not serve forever but are renewed whenever needed. A related model is Meyerson's *leasing model* [16], that has been studied in the context of many optimization problems such as the *Online Set Cover* problem [1], in which sets can cover elements for limited duration and costs are incurred accordingly. Furthermore, our simulation results show that, even without knowing the demands of the day, an efficient placement of charging stations can be done using our algorithm. Moreover, our random-geometric-graph based simulated

Table 1 Percentage difference and average competitive ratio for $|V| = 50$, 100, 150, 200, and 250 in the mean, best, and worst cases.

Mean Performance	$\|V\|$	Percentage Difference	Average Competitive Ratio
	50	23.74	1.63
	100	31.37	2.51
	150	34.25	3.43
	200	32.15	3.96
	250	30.84	4.52
Best Performance			
	50	11.10	1.29
	100	14.96	1.72
	150	16.32	2.16
	200	12.21	2.12
	250	10.30	2.18
Worst Performance			
	50	35.78	1.95
	100	46.91	3.26
	150	50.79	4.61
	200	51.09	5.71
	250	51.08	6.83

warehouse arguably generalizes topologies of warehouses and the modeled temporal evolution of incoming demands is based on rather rough assumptions. Still, it would be interesting to extend our simulation study to include data acquired from actual Internet retailers. Therefore, the model would be refined so that it resembles actual evolutions of demands during the day or week in automated warehouses. Also an application to the domain of swarm robotics for any scenario, such as collective transport, collective construction, etc., may be possible and advantageous either with manually placed or autonomous mobile charging stations [3, 4]. We have also skipped the required task allocation for the robots in this study. An interesting extension hence can be a combined analysis of task allocation algorithms together with the challenge of positioning charging stations. Besides central task allocation algorithms, there are decentralized approaches to task allocation that don't require global information and scale up well [18].

References

1 Sebastian Abshoff, Peter Kling, Christine Markarian, Friedhelm Meyer auf der Heide, and Peter Pietrzyk. Towards the price of leasing online. *J. Comb. Optim.*, 32(4):1197–1216, 2016. `doi:10.1007/s10878-015-9915-5`.

2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 100–105. ACM, 2003. `doi:10.1145/780542.780558`.

3 Farshad Arvin, Khairulmizam Samsudin, and Abdul Rahman Ramli. Swarm robots long term autonomy using moveable charger. In *Future Computer and Communication, 2009. ICFCC 2009. International Conference on Future Computer and Communication*, pages 127–130. IEEE, 2009.

4 Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013. doi:10.1007/s11721-012-0075-2.

5 Alex Couture-Beil and Richard T. Vaughan. Adaptive mobile charging stations for multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 1363–1368. IEEE, 2009.

6 Raffaello D'Andrea. Guest editorial: A revolution in the warehouse: A retrospective on Kiva systems and the grand challenges ahead. *IEEE Transactions on Automation Science and Engineering*, 9(4):638–639, 2012.

7 Stephan Eidenbenz. Online Dominating Set and Variations on Restricted Graph Classes. Technical report, Department of Computer Science, ETH Zürich, 2002.

8 Uriel Feige. A threshold of ln n for approximating set cover (preliminary version). In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 314–318. ACM, 1996. doi:10.1145/237814.237977.

9 Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

10 Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998. doi:10.1007/PL00009201.

11 Eric Guizzo. Three engineers, hundreds of robots, one warehouse. *IEEE spectrum*, 45(7):26–34, 2008.

12 Andreas Kamagaew, Jonas Stenzel, Andreas Nettsträter, and Michael ten Hompel. Concept of cellular transport systems in facility logistics. In *5th International Conference on Automation, Robotics and Applications (ICARA)*, pages 40–45. IEEE, 2011.

13 Balajee Kannan, Victor Marmol, Jaime Bourne, and M. Bernardine Dias. The autonomous recharging problem: Formulation and a market-based solution. In *IEEE International Conference on Robotics and Automation (ICRA 2013)*, pages 3503–3510. IEEE, 2013.

14 Simon Korman. On the use of randomization in the online set cover problem. In *M.S. thesis, Weizmann Institute of Science*, 2005.

15 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994. doi:10.1145/185675.306789.

16 Adam Meyerson. The parking permit problem. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 274–284. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.72.

17 Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 210–219. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.65.

18 Giovanni Pini, Arne Brutschy, Gianpiero Francesca, Marco Dorigo, and Mauro Birattari. Multi-armed bandit formulation of the task partitioning problem in swarm robotics. In *8th Int. Conf. on Swarm Intelligence (ANTS)*, pages 109–120. Springer, 2012.

19 Jonatan Schroeder, André Guedes, and Elias P. Duarte Jr. Computing the minimum cut and maximum flow of undirected graphs. Technical report, Federal University of Paraná, Department of Informatics, 2004.

20 Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9, 2008.

21 Jiguo Yu, Nannan Wang, Guanghui Wang, and Dongxiao Yu. Connected dominating sets in wireless ad hoc and sensor networks - A comprehensive survey. *Computer Communications*, 36(2):121–134, 2013. doi:10.1016/j.comcom.2012.10.005.

A Proof of Lemma 4

Proof. Let C_2 be the expected cost of the algorithm in Phase 2 and let $\mathcal{OPT}$ be the cost of the optimal offline solution. Phase 2 connects the nodes in $S't : \forall t$ that were not connected to S in Phase 1. For each such node u, Step 1 chooses a representative node k from D that is dominated by u. These representative nodes are then connected in Step 2 and form the terminals of a special case instance of the *Online Node-weighted Steiner Tree* problem (ONWST) (**Definition 2**), in which all nodes have cost 1 and all edges have cost 0. Let R be the set of these representative nodes. By similar arguments as in Phase 1, we show that Step 2 of Phase 2 admits an $\mathcal{O}(\log^2 n)$-competitive algorithm for this special case of ONWST. Let C_{St} be the expected cost of the Steiner tree constructed in Step 2 of Phase 2 and let $\mathcal{OPT}_{St}$ be the cost of an optimal Steiner tree. For any $i \in V$, the probability that the algorithm adds i into the solution is the probability that $w_i > \mu'$, which is at most $2\log(n+1) \cdot w_i$. Adding up over all $i \in V$, the expected cost C_{St} will be at most:

$$\sum_{i \in V} 2\log(n+1) \cdot w_i = 2\log(n+1) \cdot \sum_{i \in V} w_i \tag{4}$$

Next we bound $\sum_{i \in V} w_i$. To connect a node $k \in V$ not yet connected, the algorithm constructs a minimum cut. Let P_k denote the corresponding minimum cut nodes of such a cut. We increase the weight corresponding to each node in P_k. The weight w_i of each $i \in P_k$ is increased by $\left(\frac{w_i}{c_i} + \frac{1}{|P_k| \cdot c_i}\right)$. Summing up over all the nodes in P_k, we get an overall weight increase of:

$$\sum_{i \in P_k} \left(\frac{w_i}{c_i} + \frac{1}{|P_k| \cdot c_i}\right) \le 2 \tag{5}$$

The above inequality holds since $\sum_{i \in P_k} w_i \le 1$ before any weight increase and $c_i = 1$: $\forall i \in V$. An optimal Steiner tree must contain at least one node $p \in P_k$, in order to connect k. The weight w_p corresponding to p becomes at least 1 after at most $\mathcal{OPT}_{St} \cdot \log |P_k|$ weight increases and hence no further weight increases can be made. The weight of w_p becomes 1 and so it cannot belong to any other minimum cut chosen afterwards, since the algorithm constructs a minimum cut only if the maximum flow is less than one. The same argument holds for all minimum cuts chosen afterwards such that each can contain a distinct node in the optimal solution. With this observation together with inequality 5, we conclude that:

$$\sum_{i \in V} w_i \le \mathcal{OPT}_{St} \cdot 2 \cdot \log n \tag{6}$$

The above inequality holds since $|P_k| \le n$. Equations 4 and 6 yield a cost of $\mathcal{O}(\log^2 n) \cdot \mathcal{OPT}_{St}$. Let $i \in P_k$ be a node with weight $w_i > \mu'$. Now, we need to measure the additional cost incurred in the last part of Step 2, which is necessary to guarantee a feasible solution. Note that, we have that the weight of each node in a path connecting k to S must be at least the flow going through the path. For a single $1 \le q \le 2\lceil \log(n+1) \rceil$, the probability that a node k is not connected is at most:

$$\prod_{i \in P_k} (1 - w_i) \le e^{-\sum_{i \in P_k} w_i} \le 1/e$$

The last inequality holds since the algorithm guarantees that $\sum_{i \in P_k} w_i \ge 1$. Hence, the probability that k is not connected, for all $1 \le q \le 2\lceil \log(n+1) \rceil$, is at most $1/n^2$.

The additional expected cost for each of the at most n nodes is then upper bounded by $n \cdot 1/n^2 \cdot \mathcal{OPT}_{St}$, since the cost of adding one additional node is clearly less than $\mathcal{OPT}_{St}$. Therefore, we conclude that:

$$C_{St} \leq \mathcal{O}(\log^2 n) \cdot \mathcal{OPT}_{St} \tag{7}$$

Thus, we have $C_2 \leq C_1 + C_{St} \leq C_1 + \mathcal{O}(\log^2 n) \cdot \mathcal{OPT}_{St}$, where C_1 results from adding at most one node for each node in $S't$. Moreover, since an optimal offline solution for OCDS dominates all the given nodes and is connected, it forms a Steiner tree over the demand set D and consequently over the set R of representative nodes. Hence, $\mathcal{OPT}_{St} \leq \mathcal{OPT}$ and therefore:

$$C_2 \leq C_1 + \mathcal{O}(\log^2 n) \cdot \mathcal{OPT} \tag{8}$$

◀