Online Pickup and Delivery Planning with Transfers for Mobile Robots

Brian Coltin and Manuela Veloso
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
{bcoltin, veloso}@cs.cmu.edu

Abstract

We have deployed a fleet of robots that pickup and deliver items requested by users in an office building. Users specify time windows in which the items should be picked up and delivered, and send in requests online. Our goal is to form a schedule which picks up and delivers the items as quickly as possible at the lowest cost. We introduce an auction-based scheduling algorithm which plans to transfer items between robots to make deliveries more efficiently. The algorithm can obey either hard or soft time constraints. We discuss how to replan in response to newly requested items, cancelled requests, delayed robots, and robot failures. We demonstrate the effectiveness of our approach through execution on robots, and examine the effect of transfers on large simulated problems.

Figure 1: CoBot-1, CoBot-2 and CoBot-4 navigate through the building to deliver items requested by users.

Introduction

We have deployed a set of robots, called CoBots, in an office building to satisfy user requests. Users visit a website and ask the CoBots to pickup and deliver objects, such as food, drinks, printouts, or mail, to deliver a message, or to escort visitors between rooms. A centralized server assigns tasks and a task ordering to each of the robots such that the tasks are completed within user-specified time constraints. The CoBots then navigate autonomously through the building to complete their assigned tasks (see Figure 1).

Our goal is to form schedules which complete as many tasks as possible within the requested time windows at the lowest cost in energy. In particular, we are interested in exploiting the presence of multiple robots to *transfer* items and reduce the cost of delivery even further. Since requests come in an online fashion and must be accepted or rejected immediately so that the requester can be informed, the scheduling algorithm must add new tasks to the schedule quickly.

Previously, the scheduler assigned tasks by optimally solving a mixed integer program (MIP). Users made requests online, and a new MIP was solved every time a request arrived. This is effective for small numbers of tasks, on the order of two robots and fifteen tasks without transfers. CoBot was deployed extensively using this approach (Veloso et al. 2012). However, solving an MIP is infeasible for large numbers of tasks and robots, and even more challenging with transfers.

In this paper, we introduce an auction mechanism to plan online, time-constrained pickup and delivery schedules with transfers. We replan in response to robot delays and failures. We evaluate our approach experimentally on the CoBots and on large simulated problem instances.

The main novel contributions of this work are:

- Planning for online pickup and delivery tasks with transfers, under both hard and soft time constraints.
- Replanning with transfers in response to failures.
- Executing the planned schedules on physical robots.

We previously developed a two-approximate heuristic for collecting and delivering a set of items to the same location with transfers (Coltin and Veloso 2012). We also developed three algorithms to plan for ridesharing problems with transfers, in which riders offer passengers rides along their way (Coltin and Veloso 2013b). In this work, we extend the auction algorithm we developed for ridesharing to obey time constraints and to run online by revising existing schedules.

We first discuss related work, then define the online pickup and delivery problem with time windows and transfers more formally. Finally, we present an algorithm to plan online with transfers, and share experimental results.

Related Work

We focus on the Pickup and Delivery Problem (PDP) in which a set of vehicles pickup and deliver a set of items. Offline PDPs are commonly solved optimally with branch and bound methods, or approximately using various heuristics and metaheuristics such as Tabu search, simulated annealing, and genetic algorithms (Parragh, Doerner, and Hartl 2008). PDP requests may include time constraints, specified as windows of time the request must be completed in.

In online pickup and delivery problems, such as scheduling the CoBots, requests come in over time and are not known beforehand. Existing static solutions can be applied to form new schedules from scratch as new information arrives. Alternatively, heuristics (Popken 2006; Rubinstein, Smith, and Barbulescu 2012) or metaheuristics such as Tabu search and simulated annealing (Gutenschwager, Niklaus, and Voß 2004) can be applied to extend and adjust schedules to incorporate new tasks. Some work has been done on responding to dynamic events such as cancellations, traffic delays, and accidents (Haghani and Jung 2005; Xiang, Chu, and Chen 2008). Our approach is unique in that we plan to transfer items, an idea that has not been explored in online settings.

A few researchers have explored *offline* PDPs with transfers, developing heuristics (Waisanen, Shah, and Dahleh 2007; Thangiah, Fergany, and Awan 2007) to solve the PDP with transfers. Our work differs in that we plan online, consider transfers that are not limited to a small, fixed number of exchange points, and execute the schedules on robots.

Problem Definition

We are given a set of robots R. The robots navigate on a map between locations L with a shortest path function sp, a distance function d, and a travel time estimate tt. Each robot $r \in R$ has a starting location $r_L \in L$ and a maximum item capacity r_C that cannot be exceeded. Robots may optionally have an ending point $r_E \in L$ if they must end at a final destination such as a charging station.

The robots must deliver a set of items M that are not necessarily known beforehand. Each item $m \in M$ has:

- An initial pickup location m_P;
- A final dropoff location m_D;
- An earliest possible pickup time m_S; and
- A latest possible delivery time m_E.

For simplification, we limit our discussion to pickup and delivery tasks. However, our algorithm can be extended to schedule tasks that take place at only a single location (such as delivering a spoken message) simply by introducing a pickup and delivery task with the same starting and ending location.

The goal is to form a schedule that retrieves and delivers each item m within the time window (m_S, m_E) at the lowest possible cost in energy, corresponding to the lowest total distance traveled by all of the robots. The time windows can either be *soft* or *hard*. If the time windows are hard, then they cannot be violated, and if all items canot be delivered within the time windows they are rejected. With soft time windows, items can be delivered after their time window has ended, but at a cost. The objective then is to minimize an objective function that is the sum of a late delivery fee and the energy cost. We use a linear late delivery fee, a product of a constant K_{late} and the time past the delivery window, but other costs are viable with our planning approach.

Planned schedules include the following actions:

- *Retrieve*(m): Retrieve item m at pickup location m_P.
- *Deliver*(m): Deliver item m to dropoff location m_D.
- *TransferSend*(m, r, loc): Transfer m to robot r at loc.
- *TransferReceive*(m, r, loc): Receive m from r at loc.
- *Start*(r): This is the first action performed by a robot, and always occurs at r_L.
- *End*(r): This is the last action performed by each robot, and occurs at r_E. If the robot has no ending point, the location is a special "wildcard location" which is zero units of distance and time away from every other point.

Every action a has a location a_L, a duration a_D, and a scheduled starting execution time a_T. Each robot r forms a plan r_{plan}, a sequence of these actions. A time cost for transferring items is represented by the duration of the *TransferSend* and *TransferReceive* tasks. We assume without loss of generality that each plan always begins execution at time 0, and adjust the time window constraints accordingly.

A valid schedule must have the following properties:

- ***Start* and *End* actions** begin and end every plan.
- A valid route **delivers every item**, consisting of a *Retrieve* action, a possible sequence of *TransferSend* and *TransferReceive* actions, and a *Deliver* action.
- **Scheduled times are feasible** such that for sequential actions a and b, $b_T \geq a_T + a_D + tt(a_L, b_L)$.
- **Transfer actions match** such that each action $a =$ *TransferSend*(m, r_2, loc) by robot r_1 matches a $b =$*TransferReceive*(m, r_1, loc) action in r_2's plan, and vice-versa, where $a_T = b_T$, $a_D = b_D$, and $a_L = b_L$.
- Obey **capacity constraints**.
- Obey **time windows**. For $a =$ *Retrieve*(m), $a_T \geq m_S$. With hard windows, for $a =$ *Deliver*(m), $a_T \leq m_E$.
- Robots **deliver items they are already carrying** in an online, modified schedule. If robot r is already carrying m, then r plans $a =$*Retrieve*(m) with $a_T = 0$, $a_D = 0$, and $a_L = r_L$, so the plan begins with r carrying m.

To execute a schedule, robots take the shortest path between action locations, and execute each action at the scheduled time. We can anticipate that the robots will *not* execute every action at the scheduled time. Delays may occur due to unexpected obstacles, crowded hallways, or robot failures. In each scenarios the schedule should be revised.

Revising Schedules with Transfers

At a high level, our scheduling approach is to revise a schedule with an auction. Robots place bids based on the additional cost that robot would incur to pick up and deliver an item. This cost is determined by an insertion heuristic which inserts the item pickup and delivery actions into the robot's schedule. Once an item is part of an existing schedule, robots place bids to split that item's route, inserting transfer points to make the delivery at lower cost. Time constraints are maintained with a Simple Temporal Network (STN). We provide a top-down explanation, first discussing the high level auction, then the insertion heuristic, and finally how time constraints are maintained.

Auctioning Pickups, Deliveries and Transfers

Algorithm 1 shows the auction algorithm for scheduling transfers with time constraints.

Algorithm 1 `auction`(R, M): Run an auction to form a plan for the robots R to deliver items M. The robots begin with partial schedules (which may consist of solely a *Start* and *End* action).

```
1: ∀r ∈ R bids_r ← ∞
2: ∀m ∈ M assigned_m ← True iff m is in any plan
3: for m ∈ M do
4:     if not assigned_m then
5:         ∀r ∈ R bid(m, r, insert(r_plan, m))
6:     else
7:         ∀r_1 ∈ R s.t. r_1 transports m, r_1 ≠ r_2 tbid(m,
8:             r_1, r_2, insert_transfer(m, r_1, r_2))
9:     end if
10: end for
11: done ← True
12: for r ∈ R do
13:     if r has a valid bid then
14:         Robot r wins bid of lowest cost, update the plan(s)
15:         Cancel conflicting bids
16:         done ← False
17:     end if
18: end for
19: if not done then
20:     Repeat auction
21: end if
```

When the auction algorithm is first called, we begin with an existing partial schedule. This partial schedule may include delivering other items which were scheduled earlier from online scheduling. Even if no items are delivered, the partial plans alway includes *Start* and *End* actions.

First, we check if each item is delivered in the existing partial schedule (line 2). If not, each robot places a bid to pick up and deliver that item by inserting a *Retrieve* and a *Deliver* action into its plan without changing the rest of the plan's ordering (line 5). The value of the bid is the additional cost incurred by the vehicle, including both additional distance travelled and the penalty for soft time window violations. If the item already is part of some robot's plan, then for each such robot, we attempt to insert transfer actions to split its route with another vehicle (line 7). The cost is again the additional cost in distance and time window violations incurred by all of the robots. We explain the `insert` and `insert_transfer` procedures in detail in the next section.

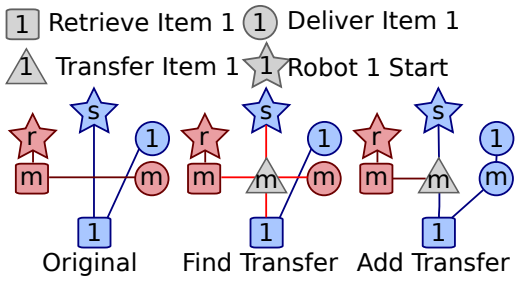


Figure 2: Robot r's pickup and delivery of item m is split with robot s using `insert_transfer`. First, a transfer point is chosen between two subsequent tasks on each robot's plan. Then the delivery point is removed from r's plan and inserted into s's, lowering the delivery cost.

Each robot is allowed to make one bid per round, the bid of lowest cost. Once all the bids are placed, the winning bids are evaluated (lines 11 - 17). Each winning bid is applied, and the schedule is updated accordingly to insert either a new item or a new transfer. Winning bids may conflict with later bids, for example, if two robots bid on the same item, or if two robots bid to transfer an item to or from the same robot. Due to time constraints, more subtle conflicts may occur if a new introduced transfer causes an action on an entirely different robot to be delayed. We detect these conflicts using temporal networks as discussed later.

To further optimize the auction algorithm, we use caching when possible so we do not need to reevaluate bids if the relevant section of the schedule has not changed.

Insertion Heuristic

The `insert`(r, m) subroutine plans to deliver item m by inserting the $Retrieve(m)$ and $Deliver(m)$ actions into r_{plan} without changing the ordering of the other actions in r_{plan}. It does this by iterating over every possible insertion point of the two actions that does not violate the capacity or time constraints, returning the plan of lowest total cost, with the cost including both total distance travelled and penalties for violating soft time constraints.

Similarly, the `insert_transfer`(m, r, s) subroutine inserts a transfer of an item m transported by robot r to or from an additional robot s. Intuitively, this routine splits robot r's transport of m in half with robot s: robot s takes responsibility for either m's retrieval or delivery (either a *Retrieve* or *Deliver* action, or a *TransferReceive* or *TransferSend* action) and the item is exchanged midway. See Figure 2 for an example of the expected output of the `insert_transfer` algorithm.

First, the `insert_transfer` algorithm searches over all subsequent pairs of actions a, b in r_{plan}, and subsequent actions c and d in s_{plan}. A *TransferReceive* action will be inserted between actions a and b, and a *TransferSend* ac-

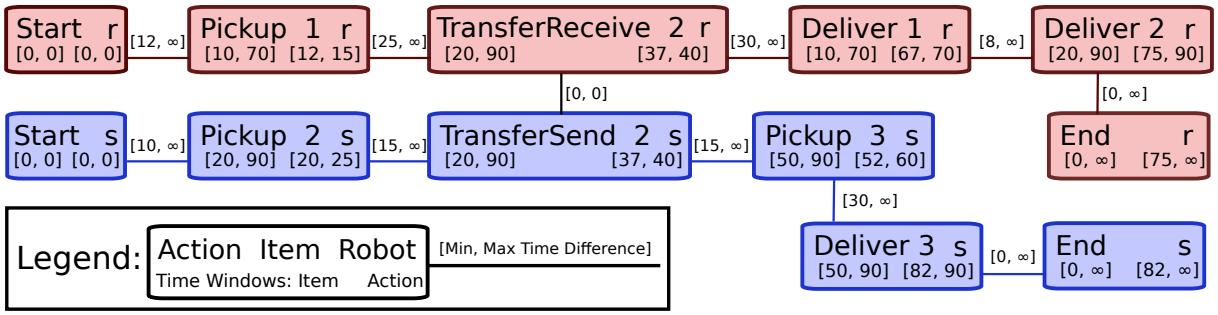


Figure 3: An example temporal network with two robots, three items and a single transfer. The feasible time windows for each action are computed based on the item time windows and the edge durations.

tion will be inserted between actions c and d if robot s will make the delivery, or vice versa if robot s will pick up item m originally in place of robot r. The inserted transfer point must not violate the capacity and must be reachable without violating any hard time constraints. The algorithm computes a proposed exchange point from a_L, b_L, c_L, and d_L. For CoBot's map, we simply find the first intersection point between the shortest path from a_L to b_L and the shortest path from c_L to d_L. If no such point exists then no transfer is made between these actions. More sophisticated methods of choosing transfer points can be used for other maps.

Once a transfer point is found, the algorithm attempts to have robot s pick up the item in place of robot r, then transfer it to r for r's original delivery. Next, it attempts to have r pick up the item in the original location, then transfer it to s, and have s deliver the item to r's original delivery location. We iterate through every possible insertion point for the new pickup or delivery point in s_{plan}, and choose the plan of lowest cost.

During this search, we check that the newly introduced transfer does not induce a cycle of robots waiting for each other by performing breadth first search on the graph formed by the robot's plans. In this graph, subsequent actions are connected, and *TransferReceive* / *TransferSend* actions are additionally connected to each other's subsequent actions. If one of the initial transfer actions is reached a second time in the graph search a cycle exists and the schedule is rejected.

Although the `insert_transfer` routine runs in polynomial time, it is still expensive for large problem instances. To speed things up and reduce the number of considered transfer points, we add a budget r_B for each vehicle. If the starting and ending points of item m's portion of r's route are both further than r_B units of distance from s's planned path, we disregard the potential transfer point. This limits the consideration of transfer points that are likely not to be cost-effective.

Maintaining Time Constraints

To maintain time constraints, we form a Simple Temporal Network (Dechter, Meiri, and Pearl 1991). Every action in the robots' plans is a node in the network, associated with a time window within which that action must occur. Each edge is associated with a time window which bounds the difference in time between two nodes. Every *Start* action node has the time window $[0, 0]$, and every *End* action has the time window $[0, \infty)$. The nodes for actions that transport item m have the time window $[m_S, m_E]$.

Every pair of subsequent actions a and b in a robot's plan have nodes linked with an edge with duration $[a_D + d(a_L, b_L), \infty)$, the minimum time in which a robot can complete action a and then travel to the location of action b. *TransferSend* actions are connected to the corresponding *TransferReceive* actions with edges of duration $[0, 0]$, ensuring that both actions take place at the same time. Whenever the schedule is modified, we solve the constraints in the temporal network to find valid windows of time in which each action could be executed without violating any constraints. Figure 3 shows an example temporal network and solution.

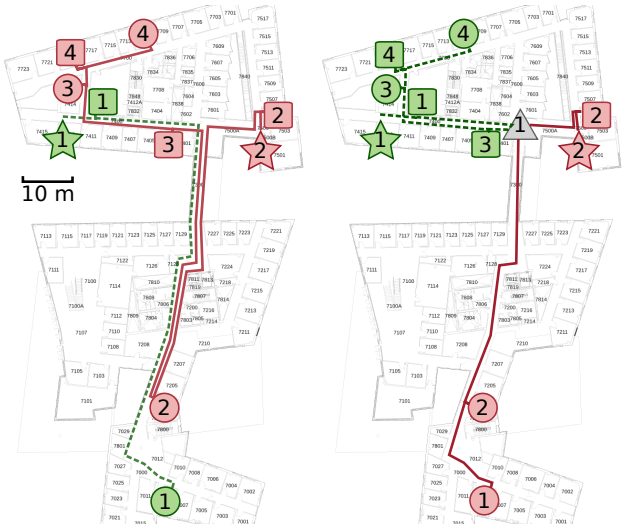
With hard time constraints, when a new action or set of actions is inserted into the schedule, we attempt to insert the new actions into the temporal network to determine whether or not the schedule remains feasible with the new actions. This does not require reconstruction or recomputation of the entire temporal network from scratch; the changes can be propagated from the insertion points. With soft time constraints, the temporal network is used to compute the earliest feasible execution time of each action by setting all delivery deadlines to infinity. The action execution times are then used to determine the cost of violated soft time constraints.

Online Planning and Replanning

We presented an algorithm to revise a schedule to transport new items. To replan online with this scheduler, the existing schedule must first be updated. First, completed tasks are removed from the schedule, and all times are updated to be relative to the current time, which is always time 0. Robots currently carrying items have *Retrieve* actions added to their schedule at the current location, with a time window of $[0, 0]$ and a duration of 0. These actions are not executed, but ensure the algorithm maintains its invariants.

We replan in four cases:

- **New Item Requested**: The auction algorithm inserts the new items into the existing schedule.
- **Request Cancelled**: Every action involving the removed item is removed from the schedule.
- **Delayed Robot**: If a robot is late to complete a task by a fixed amount of time, all tasks involving items the late



(a) (b)

Figure 4: Planned schedules to deliver four items, scheduled with the *(a)* MIP without transfers, and *(b)* auction with transfers. Items 1 and 2 are requested at time 0, Item 3 is requested after 150 s, and Item 4 is requested after 200 s. See Fig. 2 for symbol meanings.

robot is scheduled to carry and does not currently hold are removed from the schedule and re-inserted.

- **Dead Robot**: If a robot does not communicate with the server for a fixed time, it is marked as dead. Its tasks are re-added to the schedule, except for items it is holding.

The robots can also replan for other reasons, such as based on shared information about blocked hallways or closed doors shared from other robots (Coltin and Veloso 2013a).

Experiments

We first present several illustrative problems run on the CoBot robots, demonstrating the scheduler's ability to revise schedules with transfers. We then share extensive results from larger problem instances solved in simulation, demonstrating the scheduler's scalability and effectiveness.

Illustrative Revised Schedules on CoBot

Since the CoBots do not have arms, pickups, deliveries and transfers are made with human help. We assume in these examples that humans are readily available to retrieve, deliver and transfer items, although the time taken to ask for help is still included in the experiments. Other robots are capable of transferring items autonomously (Coltin and Veloso 2012).

For the first experiment, two robots placed four pickup and delivery requests (see Figure 4). Items 1 and 2 were requested at time 0, Item 3 at 150 s, and Item 4 at 200 s. Solving the MIP optimally without transfers in an online manner (our original approach) results in each robot performing one of the initial tasks. When Tasks 3 and 4 arrive, one robot travels all the way back to the starting point. Using the auction algorithm with transfers, only one robot travels to the opposite end of the building initially. Then the other robot is free to deliver items 3 and 4. The MIP approach took approximately 5 minutes 45 seconds to execute and traversed 280.7 m. The auction algorithm with transfers took approximately 4 minutes to execute and traversed 162.1 m.

For the second example, we used three robots to perform five tasks placed at the same time. Two robots were scheduled to transfer an item to a third robot for delivery. However, we immediately turned off the robot scheduled to deliver the three items. The server detected this, and a new plan was formed which the robots then executed (see Figure 5).

Large Simulated Problems

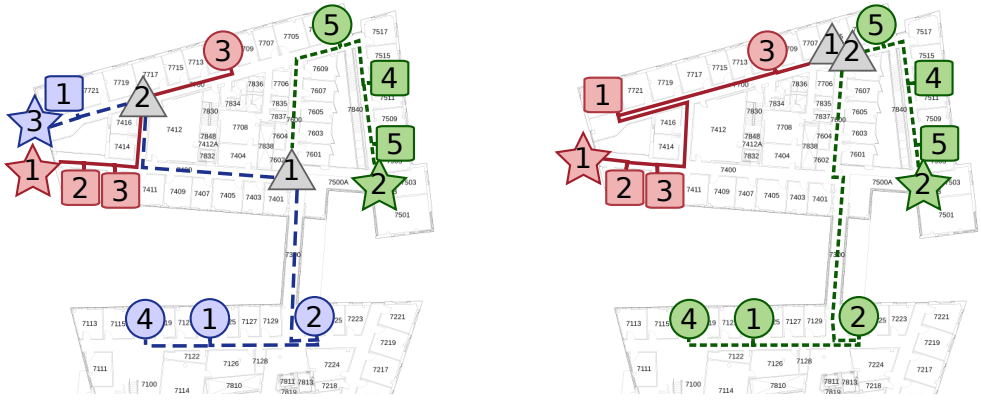
The final problem set was run on large simulated problem instances to test the scalability of the algorithms. The world is a 30x30 grid of city blocks, each one unit long. Item pickup and dropoff locations are chosen from the block intersection at random. Unlike in the CoBot domain, robots have an assigned end location, a station where they return to charge. Corresponding start and end points for both robots and items are constrained to be at least five blocks apart.

We ran tests in this domain with $|R| = 80$ robots and with the number of items $|M|$ varying from 20 to 240. We formed schedules for fifty different trials for each value of $|M|$. For every robot r, $r_C = 3$ and $r_B = 5$. Each vehicle travels one block per minute. Soft time windows are used, and $K_{late} = 50$, meaning every minute a delivery is late adds 50 units to the cost function. Every block traveled adds one unit to the cost function, and each transfer contributes four units (but takes a negligible time to execute). Each item is given a small time window of ten minutes for delivery. This is a very tight window: for some items it is not even physically possible to deliver the item in time. Our goal here is to make the delivery as quickly as possible. The start of this time window falls at a random point in the interval $[0, \frac{5}{4}|M|]$. The schedule is executed online, with new items scheduled one at a time. The scheduler is informed of each request half an hour before the time window begins (or immediately if the start of the window falls within the first half hour).

These problems are too large for us to solve optimally. Instead, we compare the auction algorithm with transfers to the auction algorithm without transfers. The costs of the solutions found with both algorithms are shown in Figure 6. With 240 items, the cost is reduced by almost 33% by using transfers. Most of this savings comes from an improved capability to deliver items within their time windows due to the availability of additional scheduling options. With 240 items, an average of 26.6 transfers were made. The auction without transfers took an average of 6.03 s to execute in total, while the auction with transfers took an average of 13.98 s. This amounts to an average of 0.058 s per request.

Conclusions

We have introduced an auction-based algorithm to schedule pickup and delivery problems with transfers and time windows. The algorithm runs online and replans in response to new requests, dead vehicles, and shared information. We have demonstrated the schedules formed on robots and in large simulated problem instances.



(a) (b)

Figure 5: *(a)* Deliveries are scheduled with three robots (including two transfers). *(b)* When one of the robots dies and fails to respond, the tasks are rescheduled. See Fig. 2 for symbol meanings.

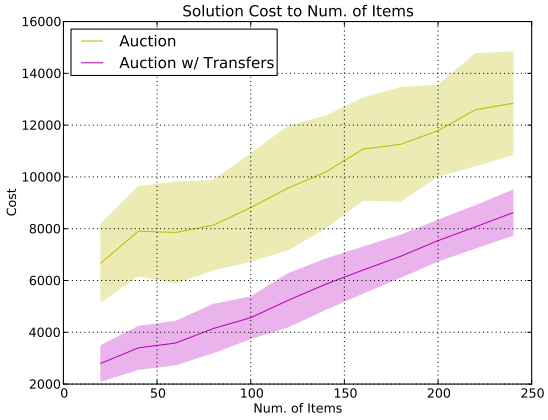


Figure 6: The mean cost of the generated schedules to the number of items. The shaded regions depict the standard deviation across the fifty trials.

Acknowledgements

This research was partially sponsored by the Office of Naval Research under grant number N00014-09-1-1031, and by the National Science Foundation under award number NSF IIS-1012733 . The views and conclusions expressed are those of the authors only.

References

Coltin, B., and Veloso, M. 2012. Optimizing for transfers in a multi-vehicle collection and delivery problem. In *Proc. of DARS*.

Coltin, B., and Veloso, M. 2013a. Towards replanning for mobile service robots with shared information. In *To Appear, Proc. of ARMS Workshop, AAMAS*.

Coltin, B., and Veloso, M. 2013b. Towards ridesharing with passenger transfers (extended abstract). In *Proc. of AAMAS*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence* 49(1):61–95.

Gutenschwager, K.; Niklaus, C.; and Voß, S. 2004. Dispatching of an electric monorail system: Applying metaheuristics to an online pickup and delivery problem. *Transportation science* 38(4):434–446.

Haghani, A., and Jung, S. 2005. A dynamic vehicle routing problem with time-dependent travel times. *Computers & operations research* 32(11):2959–2986.

Parragh, S.; Doerner, K.; and Hartl, R. 2008. A survey on pickup and delivery problems: Part ii, transportation between pickup and delivery locations. *Journal für Betriebswirtschaft* 58(2):81–117.

Popken, D. 2006. Controlling order circuity in pickup and delivery problems. *Transportation Research Part E: Logistics and Transportation Review* 42(5):431–443.

Rubinstein, Z.; Smith, S.; and Barbulescu, L. 2012. Incremental management of oversubscribed vehicle schedules in dynamic dial-a-ride problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Thangiah, S.; Fergany, A.; and Awan, S. 2007. Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research* 15(4):329–349.

Veloso, M.; Biswas, J.; Coltin, B.; Rosenthal, S.; Brandao, S.; Merili, T.; and Ventura, R. 2012. Symbiotic-autonomous service robots for user-requested tasks in a multi-floor building. In *Proc. of Assisstive Systems Workshop, IROS 2012*.

Waisanen, H.; Shah, D.; and Dahleh, M. 2007. Fundamental performance limits for multi-stage vehicle routing problems. *Operations Research*.

Xiang, Z.; Chu, C.; and Chen, H. 2008. The study of a dynamic dial-a-ride problem under time-dependent and stochastic environments. *European Journal of Operational Research* 185(2):534–551.