Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)
Special Track on AI for Computational Sustainability and Human Well-being

An Exact Single-Agent Task Selection Algorithm for the Crowdsourced Logistics

Chung-Kyun Han and **Shih-Fen Cheng***
School of Information Systems, Singapore Management University
ckhan.2015@phdcs.smu.edu.sg, sfcheng@smu.edu.sg

Abstract

The trend of moving online in the retail industry has created great pressure for the logistics industry to catch up both in terms of volume and response time. On one hand, volume is fluctuating at greater magnitude, making peaks higher; on the other hand, customers are also expecting shorter response time. As a result, logistics service providers are pressured to expand and keep up with the demands. Expanding fleet capacity, however, is not sustainable as capacity built for the peak seasons would be mostly vacant during ordinary days. One promising solution is to engage crowdsourced workers, who are not employed full-time but would be willing to help with the deliveries if their schedules permit. The challenge, however, is to choose appropriate sets of tasks that would not cause too much disruption from their intended routes, while satisfying each delivery task's delivery time window requirement. In this paper, we propose a decision-support algorithm to select delivery tasks for a single crowdsourced worker that best fit his/her upcoming route both in terms of additional travel time and the time window requirements at all stops along his/her route, while at the same time satisfies tasks' delivery time windows. Our major contributions are in the formulation of the problem and the design of an efficient exact algorithm based on the branch-and-cut approach. The major innovation we introduce is the efficient generation of promising valid inequalities via our separation heuristics. In all numerical instances we study, our approach manages to reach optimality yet with much fewer computational resource requirement than the plain integer linear programming formulation. The greedy heuristic, while efficient in time, only achieves around 40-60% of the optimum in all cases. To illustrate how our solver could help in advancing the sustainability objective, we also quantify the reduction in the carbon footprint.

*Contact Author

1 Introduction

In recent years, retail activities have increasingly moved online, and this trend has created substantial pressure for the logistics industry to catch up both in terms of volume and response time. Despite the best efforts from the logistics industry, the spiking nature of the demands (mostly caused by shopping holidays such as Black Fridays) has made capacity planning more challenging than ever. On the other hand, the pursuit for shorter response time means that most logistics service providers are facing the touch decision on how best to increase their service capacity. Increasing fleet sizes is a quick fix yet it would not be sustainable, as capacity built for the peak seasons would be mostly vacant during ordinary days. As a result, the increasing numbers of half-filled trucks plying the streets have posted serious challenges to the city management [Guo *et al.*, 2019].

A promising alternative that could address these challenges is the crowdsourced logistics paradigm, which centers around the idea of utilizing part-time workers driving their own vehicles to complete the delivery tasks. Utilizing crowdsourced workers has the benefit of not having to commit to capacities that are only required during demand peaks. Having crowdsourced workers also makes last-mile logistics more sustainable; as crowdsourced workers could integrate deliveries into their own schedule and make deliveries en route to their actual destinations. This concept is most famously demonstrated by Walmart, who asks selected employees to help with last-mile deliveries on their way home.

From the perspective of crowdsourced workers who only commit on an ad hoc basis, the major hurdle of performing crowdsourced deliveries lies in having to go through the list of tasks and find ones that are most compatible with his/her own itinerary in terms of time and location. A worker might have to follow a particular route and pass through a number of predetermined locations within designated time windows (we define such route as a worker's *routine route*), while only has limited amount of time for delivery tasks. Individual tasks might also have requirements on the delivery time windows. Taking into account all these requirements is non-trivial, and it is no wonder that most crowdsourced workers would just adopt very simple heuristics, such as choosing tasks that are around their final destinations.

In this paper, we aim to provide a single-agent planning algorithm that takes into account worker's routine route re-

quirements and tasks' delivery time windows. In pursuing this research, we make the following major contributions:

- First, we formulate the problem as an integer linear program (ILP). The major innovations are the inclusion of aggregated pickup points and a crowdsourced worker's routine route.
- Second, we propose a branch-and-cut algorithm which can solve the problem exactly. To make the algorithm efficient, we introduce several classes of valid inequalities that would tighten the formulation and speed up the algorithm.
- Finally, we demonstrate how our approach could reach optimality in all numerical instances with far fewer computational resources when compared against the ILP approach. We also quantify the large optimality gaps experienced by a real-world-inspired greedy heuristic. To conclude the experiments, we highlight how the crowdsourcing paradigm could help in reducing carbon emission if planned appropriately using our approach.

2 Related Literature

Optimization problems dealing with the last-mile logistics are categorized in two threads depended on whether or not details about a routing problem are considered in an optimization model. With some assumptions and approximations on the routing part, Winkenbach *et al.* (2016) and Huang *et al.* (2018) solve network design problems, and Wang *et al.* (2016) deal with an assignment problem. Some researchers target optimization problems considering a routing sequence for each vehicle or worker. Various heuristics are suggested for large-scale last-mile delivery problems. Gansterer *et al.* (2017) model a multi-vehicle profitable pickup and delivery problem and suggest a method based on the general variable neighborhood search. Behrend *et al.* (2019) address an item-sharing and crowd shipping problem and suggest an exact solution method incorporating a set packing formulation and a labeling algorithm. Stenger *et al.* (2013) and Arslan *et al.* (2019) consider a pickup and delivery problem utilizing ad hoc drivers or subcontractors for the last mile delivery. Stenger *et al.* (2013) solve the problem with an adaptive variable neighborhood search and Arslan *et al.* (2019) propose a rolling horizon framework. Holland *et al.* (2017) deal with a practical routing problem of United Parcel Service, a logistics company, and share the results of the ORION algorithm consisting of an adaptive large neighborhood search, Lagrangian relaxation, and simulated annealing.

In the crowdsourcing area, most researchers consider a task to happen only at a particular location, and the focuses are mostly on how optimization techniques can be utilized to solve different problem formulations [Kazemi and Shahabi, 2012; Chen *et al.*, 2015; Deng *et al.*, 2016; Cheng *et al.*, 2017; Tran *et al.*, 2018; Behrend *et al.*, 2019]. These models are not directly applicable to our problem, as delivery tasks contain both a pick-up point and a drop-off point, with the implicit assumption that a task must be picked up before it can be dropped off.

Wang *et al.* (2016) and Arslan *et al.* (2019) have formulated and studied the crowdsourced logistics planning problems and proposed scalable optimization approaches to solve these problems. There are also a few field trials on proving the concept; e.g., Kim (2015) has piloted a crowdsourcing delivery system and Sun *et al.* (2018) have developed a system for the online delivery route recommendation for a single agent. However, to the best of our knowledge, no past work explicitly considers both a worker's pre-existing routine route with time windows and also individual tasks' time windows.

3 The Problem Formulation and the Integer Linear Programming Model

Let K denote the set of all tasks. For task $k \in K$, let r_k, v_k, and w_k be the reward, volume, and weight respectively. The pickup and delivery nodes of task $k \in K$ are represented by h_k and n_k respectively. Let P be the set of all pickup nodes and D the set of all delivery nodes. We assume that each task is uniquely represented by its delivery node, even though some of tasks might share the same physical delivery location. For cases where delivery locations are identical, they are still treated as different nodes, but the traveling distance between these nodes will be set to zero. However, pickup nodes can be shared among multiple tasks. Let R represent the set of nodes associated with a routine route sequence including the origin node, o, the destination node, d, and other nodes where the crowdsourced worker has to visit in sequence. The visiting precedence among R is specified by $c_{i,j}$, where $c_{i,j} = 1$ indicates that i is to be visited before j. Note that by definition $c_{o,j} = 1, \forall j \in R \setminus \{o\}$ and $c_{i,d} = 1, \forall i \in R \setminus \{d\}$. There are resource limits on the volume, weight and travel time and they are denoted by **v**, **w** and **u** respectively.

Our problem is defined on a complete directed graph $G = (N, A)$, where $N = P \cup D \cup R$. Each node i is associated with a time window $[\alpha_i, \beta_i]$. We denote $t_{i,j}$ as the time it takes to travel from i to j.

There are two types of decision variables: $x_{i,j}$ indicates whether arc (i, j) should be included in the final solution (1 if yes, 0 if no); and μ_i denotes when the agent visits node i.

We aim to maximize the sum of collected rewards, which is formulated as follow:

$$\max \sum_{k \in K} r_k \sum_{j \in N} x_{j,n_k}, \tag{1}$$

where the reward of task k can be collected when there is an in-flow to the delivery node n_k.

We divide constraints into three groups. The first group of constraints deals with flows between nodes. Equations (2) and (3) ensure that all nodes in the routine route are visited, and the agent starts from o and ends in d. Equation (4) turns the final route into a Hamiltonian cycle, which helps to simplify the representation of valid inequalities for the branch-and-cut algorithm. Equation (5) eliminates self-loops. Equation (6) enforces that a delivery of task k at n_k is only possible if the pickup node h_k is visited. Finally, Equation (7) ensures

flow conservation at all pickup/delivery nodes.

$$\sum_{j\in N} x_{o,j} = \sum_{j\in N} x_{j,d} = 1, \tag{2}$$

$$\sum_{j\in N\setminus\{i\}} x_{i,j} = \sum_{j\in N\setminus\{i\}} x_{j,i} = 1, \forall i \in R \setminus \{o,d\}, \tag{3}$$

$$x_{d,o} = 1, \tag{4}$$

$$x_{i,i} = 0, \forall i \in N \tag{5}$$

$$\sum_{j\in N} x_{n_k,j} \leq \sum_{j\in N} x_{j,h_k}, \forall k \in K, \tag{6}$$

$$\sum_{j\in N} x_{i,j} = \sum_{j\in N} x_{j,i} \leq 1, \forall i \in P \cup D. \tag{7}$$

The second group of constraints deals with visiting precedence of nodes and time windows. Equation (8) initializes the arrival time at the origin node. In Equation (9), the constant M is large enough to ensure that this constraint is only active when $x_{i,j} = 1$, which implies that μ_j is greater than μ_i by at least $t_{i,j}$. Equation (10) enforces time windows for all nodes. Equation (11) enforces binary precedence relationship encoded in $c_{i,j}$. Finally, Equation (12) ensures that the agent always visits the pickup node before the delivery node. If the final route does not include task k, the second term of the right-hand-side will make the constraint inactive.

$$\mu_o = \alpha_o, \tag{8}$$

$$\mu_i + t_{i,j} \leq \mu_j + M(1 - x_{i,j}), \forall (i,j) \in A \setminus \{(d,o)\}, \tag{9}$$

$$\alpha_i \leq \mu_i \leq \beta_i, \forall i \in N, \tag{10}$$

$$c_{i,j} \cdot \mu_i \leq \mu_j, \forall i,j \in R, \tag{11}$$

$$\mu_{h_k} \leq \mu_{n_k} + M(1 - \sum_{j\in N} x_{j,n_k}), \forall k \in K. \tag{12}$$

The final group of constraints deals with agent's resource limits. Equations (13), (14), and (15) ensure that agent's volume, weight, and time limits are observed.

$$\sum_{k\in K} v_k \sum_{j\in N} x_{j,n_k} \leq \mathbf{v}, \tag{13}$$

$$\sum_{k\in K} w_k \sum_{j\in N} x_{j,n_k} \leq \mathbf{w}, \tag{14}$$

$$\sum_{(i,j)\in A\setminus\{(d,o)\}} t_{i,j} x_{i,j} \leq \mathbf{u}. \tag{15}$$

4 The Valid Inequalities

This section describes several problem-specific valid inequalities that may be able to tighten bounds of the linear relaxation of the original model introduced in the previous section. We demonstrate the impact of the inequalities through numerical experiments in another section.

There are additional notations to represent the inequalities concisely. Given a node set $S \subseteq N$, let $x(S)$ be the total flows between nodes associated with S, which is mathematically represented by $x(S) = \sum_{i,j\in S} x_{i,j}$ and, similarly, let $x(T) = \sum_{(i,j)\in T} x_{i,j}$. Additionally, let $\delta(S)$ is the set of edges connecting nodes in the set S and those of the complement of the set, mathematically $\delta(S) = \{(i,j) \in A \mid i \in S, j \notin S\}$ and $K(S)$ denotes the set of tasks whose delivery nodes are included in set S, $K(S) = \{k \mid n_k \in S, k \in K\}$

4.1 The Subtour-Elimination Constraints

The following inequality represents the simple subtour-elimination constraint and, generally, S is a proper subset of the whole nodes, $S \subset N$:

$$x(S) \leq |S| - 1 \text{ or } x(\delta(S)) \geq 2. \tag{16}$$

Whereas this inequality can be easily adopted to the general pickup and delivery problem, some modifications are needed for orienteering problems because it is not mandatory to visit all nodes. [Fischetti *et al.*, 1998] deals with this issue by introducing additional binary variables denoting whether or not a node is visited by the final tour. Whereas, we put our focus on defining the subsets more finely. In our problem, if a set S is the superset of the set of routine nodes, $S \supseteq R$, the subtour-elimination constraints may not be applicable. When a set $S \supseteq R$ does not include pickup nodes of $K(S)$, the suggested constraints are valid, which also means the set S is not a valid set if $S \setminus \{h_k \mid k \in K(S)\} = \emptyset$.

4.2 The Capacity Constraints

Both the volume and weight limits are dealt with by the following inequality simultaneously:

$$\begin{gathered} x(S) \leq |S| - \max\{1, \lceil v(S) \,/\, \mathbf{v} \rceil, \lceil w(S) \,/\, \mathbf{w} \rceil\}, \\ \text{or} \\ x(\delta(S)) \geq 2 \times \max\{1, \lceil v(S) \,/\, \mathbf{v} \rceil, \lceil w(S) \,/\, \mathbf{w} \rceil\}, \end{gathered} \tag{17}$$

where $S \subseteq N \setminus \{o,d\}$, $v(S) = \sum_{k\in K(S)} v_k$ and $w(S) = \sum_{k\in K(S)} w_k$. Intuitively, the second term of the right-hand side represents the number of times that flows must enter and leave set S to service all nodes in the set considering the capacity limits. Whereas other researchers generally deal with one-dimensional capacity inequalities, we consider the volume and weight constraints together and use the one tighter than another.

4.3 The Routine Sequence Constraints

Based on the common precedence inequality [Ruland and Rodin, 1997], we devise a problem-specific valid inequality. The following is the mathematical formulation of the new inequality:

$$\begin{gathered} x(S) \leq |S| - (1 + f(S)), \\ \text{or} \\ x(\delta(S)) \geq 2 \times (1 + f(S)), \end{gathered} \tag{18}$$

where a set S includes the origin node, $o \in S$, not the destination node, $d \notin S$, and $f(S)$ is the number of times that flows enter and leave the boundary of the set S to retain the original sequence of the routine route.

4.4 The Infeasible Path Constraints

Given a directed path $\mathcal{P} = \{i_1, i_2, ..., i_p\}$, where $|\mathcal{P}| = p$, and the path violates any constraints, the following inequality is valid:

$$\sum_{k=1}^{p-1} x_{i_k, i_{k+1}} \leq p - 1. \tag{19}$$

The purpose of this inequality is to forbid a (sub) path formed by a fractional solution of the linearized model, but infeasible in an integer solution in advance. In our branch-and-cut algorithms, which will be introduced in the next section, we use constraints such as time windows, the travel time limit and the retainment of the routine route to capture infeasible paths and produce valid inequalities.

4.5 The Enforced Delivery Constraints

The following inequalities can tighten bounds by manipulating a feature of aggregated pickup nodes:

$$\sum_{j \in N} x_{i,j} \leq \sum_{k \in K_i} \sum_{j \in N} x_{n_k, j}, \quad \forall i \in P, \tag{20}$$

where $K_i = \{k \mid h_k = i, k \in K\}$ is the set of tasks sharing the same pickup node i, given node $i \in P$. Following inequalities (20), any out-flow is enforced to at least one delivery node if the associated task's pickup node has any outflows. Whereas other inequalities introduced earlier suffer from finding effective subsets in advance, we can add all these valid inequalities to the model at the root node of a branch-and-bound tree.

5 The Branch-and-Cut Algorithm

The branch-and-cut (BnC) approach is a popular method for solving optimization problems, and most commercial solvers dealing with integer linear programs (ILPs) have adopted this approach. Basically, it consists of the branch-and-bound (BnB) process and the cutting-plane method, which tightens the bound of linearly relaxed ILPs.

A key design challenge for the BnC approach is to add only the promising cuts (valid inequalities). For each BnB node, the BnC approach first solves the linearly relaxed problem (LP). If this solution from the LP model is integral and has better objective value than the previous incumbent solution, it becomes the new incumbent solution.

If the solution is fractional, a cutting-plane (or separation) algorithm is applied to generate cuts (new constraints) which could make the current fractional solution infeasible without removing feasible solutions. The identified cuts are added to the LP model, and the algorithm repeats the above procedure. If we cannot find additional cuts, the algorithm executes the BnB process.

In our research, we reply on CPLEX to execute standard BnB process, but engage CPLEX via its APIs to generate and insert new cuts. Cuts are generated by solving the separation problem, yet an important design question is on how precisely we should solve it. Solving the separation problem exactly generates tighter bounds, but at the expense of longer execution time. On the other hand, we could also solve the separation problem heuristically, generating cuts more quickly, but with looser bounds and potentially leading to more time spent in the BnB procedure.

Except for the enforced delivery constraints (20), which can be added in advance since their quantity is small, we develop our own separation heuristics to generate all other types of cuts. The basic idea of our heuristics is to iteratively construct a route in a greedy manner based on the fractional flows provided by the LP model until the route forms a cycle or some constraints are violated.

The first heuristic is for the first three families of valid inequalities: the subtour elimination (SE) constraints, the capacity (CA) constraints, and the routine sequence (RS) constraints. In all three cases, we construct a path by starting from a routine node $i \in R \setminus \{d\}$ and adding a node j with the highest in-flow from node i, i.e., $j = \text{argmax}_{j \in N} x_{i,j}$. The path construction proceeds until it forms a cycle or reaches the destination. When the path forms a cycle, we consider it as a valid cut and add it to the original ILP model if the path satisfies one of the following sets of constraints ((21), (22), and (23) correspond to SE, CA, and RS cuts respectively).

$$x(S) > |S| - 1, \tag{21}$$

$$x(S) > |S| - \max\{1, \lceil v(S) \,/\, \mathbf{v} \rceil, \lceil w(S) \,/\, \mathbf{w} \rceil\}, \tag{22}$$

$$x(S) > |S| - (1 + f(S)). \tag{23}$$

Depending on the conditions on the definition of a set S, the heuristic runs the above path construction procedure $|R \setminus \{d\}|$ by setting each routine node as a starting node for the SE constraints. For the CA and the RS constraints, the heuristic executes the procedure $|R \setminus \{o, d\}|$ and $|\{o\}|$ times respectively.

The second separation heuristic is similar to the first one, but its purpose is to find an infeasible path. For each routine node $i \in R \setminus \{d\}$, it constructs a path until the path reaches the next routine node following the visiting sequence of the routine route. Whenever the heuristic adds the next node in the path construction, it checks the time window and the time limit constraints, and also whether or not the precedence between routine nodes is retained. If it finds any violation, we add an inequality (19) to the ILP model. To further boost efficiency, the heuristic cuts off the path construction if the in-flow of the next node is less than 0.5.

The standard BnC implementation requires separation problems to be solved at all BnB nodes, yet we observe that cuts generated at majority of the BnB nodes are not effective, as they do not contribute to the tightening of LP bound (upper bound). To eliminate time spent on solving unnecessary separation problems, we only solve separation problems when a particular BnB node updates the LP bound.

6 The Numerical Experiments

This section explains procedures for generating synthetic problem instances and compares our branch-and-cut algorithm against a greedy heuristic and the ILP model. All approaches are implemented in C++ and tested on identical hardware/software environment (a server with 4 Intel® Xeon® Gold 6154 CPUs (a total of 72 cores) and 512GB RAM, running Red Hat Linux v6.9). We use CPLEX 12.8, a commercial solver, to solve ILP models using default settings.

6.1 Problem Instance Generation

We generate random problem instances based on the following parameters:

- nr: The number of routine nodes, $nr = |R|$.
- np: The number of pickup nodes, $np = |P|$.
- nd: The number of delivery nodes, $nd = |D|$.
- ca: The agent's volume and weight capacity limits.
- dt: The additional detour time the agent is willing to spend on delivery tasks, expressed as the percentage of traveling time to complete the agent's routine route.

The x and y coordinates of all nodes are generated in the range of $[0, 1]$. Our problem instances are characterized by (nr, np, nd). For a particular set of (nr, np, nd), we first generate the agent's routine route, which stays fixed for all random instances in the set. The routine route always starts from $(0, 0)$, ends in $(1, 1)$, with remaining routine nodes evenly spread out along the x-axis but with randomly generated y coordinates. Each task in the instance is characterized by a pair of pickup and delivery nodes. As there are usually far fewer pickup nodes (representing warehouses), we generate them first. The delivery nodes are then generated one by one randomly. A random task is then generated by associating each delivery node with the closest pickup node.

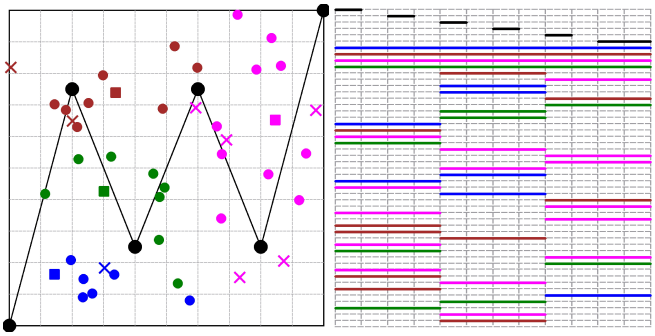
Figure 1a illustrates an example with parameters $(6, 4, 40)$. The routine route contains 6 black circles that are connected by lines. There are 4 pickup nodes, which are denoted as squares in different colors. All delivery nodes are denoted as either colored circles or crosses. The color of a delivery node depends on the pickup node it is associated to. A delivery node in cross means that it cannot be served feasibly due to agent's travel time limitation or time window constraints. A delivery node in circle means that it can be served feasibly.

For all nodes in an instance, we also need to generate their associated time windows, which are defined on the discretized planning horizon. The duration of the planning horizon is $T_{od} \cdot (1 + dt)$, where T_{od} is the time it takes to traverse the routine route, and we divide the planning horizon into 12 time periods of equal length. The time windows of routine nodes are determined based on their orders in the visit sequence and the travel time between routine nodes. For pickup nodes, the agent can visit them any time; for delivery nodes, they will be randomly assigned one of three time windows: periods 1 to 4, 5 to 8, or 9 to 12. A visualization of time windows for the 40-task instance is given in Figure 1b, in which the first 6 rows are for routine routes, the next 4 rows are for pickup nodes, and the rest are for delivery nodes.

Finally, the volume and weight of a task are generated uniformly between 0 and 1, while the reward is the larger value among volume and weight.

6.2 Performance Comparison

Besides the ILP model, the greedy heuristics, and our branch-and-cut (BnC) approach with all the valid inequalities (cuts), we also introduce variants of our BnC approach, where each variant contains only one class of the valid inequality. We include these variants to see the impacts of different classes of valid inequalities. In our result presentation, we use the following acronyms:



(a) Position of nodes. (b) Time window of nodes.

Figure 1: A 40-task problem instance with 4 pickup nodes.

- GH: the greedy heuristic that iteratively adds the feasible task with highest reward.
- ILP: the ILP model is solved by using CPLEX 12.8.
- ALL: the BnC approach with all the cuts.
- SE: the BnC approach with subtour elimination cuts.
- CA: the BnC approach with capacity cuts.
- RS: the BnC approach with routine sequence cuts.
- IP: the BnC approach with infeasible path cuts.
- ED: the BnC approach with enforced delivery cuts.

The GH and ILP approaches are the baselines we compare our BnC variants to. For all our numerical experiments, we fix $nr = 6$, $np = 4$, and $dt = 25\%$, and vary nd and ca. For each configuration (nd, ca), we generate 20 random numerical instances. As the ILP approach does not scale well, we explore only configurations where we can solve the ILP models exactly within 2 hours of wall clock time.

We summarize the performance comparisons of all configurations in Table 1 and Figure 2. As all our BnC variants manage to solve all problem instances exactly, we use Table 1 to summarize the performances of the ILP and the GH approaches, and use Figure 2 to visualize the efficiency comparisons between the ILP approach and all BnC variants.

In Table 1, the first two columns are for nd and ca. The third and fourth column denote the number of constraints (rows) and decision variables (columns) of the ILP models. The fifth column, avgG, shows the average optimality gaps between GH and ILP, normalized by the ILP objective value. In most cases, the optimality gaps between the GH and the ILP approaches are more than 50%.

In terms of the execution time, the GH approach solves all instances in less than a second while the solution time of the ILP approach grows exponentially. Columns avgT and stdT summarize the average and the standard deviation of the *CPU time* in seconds. Generally speaking, the ILP approach solves less restrictive problem instances (with higher ca) more efficiently. The large standard deviation implies that the computation time is highly dependent on the problem data.

To quantify the sustainability benefits of having crowdsourced workers, we compare the additional travel distance required to finish the chosen delivery tasks (not including the distance needed to traverse the worker's own routine route) against the required distance to serve the same set of selected tasks with a dedicated vehicle. The presented ratio, *saveC*

nd	ca	# rows	# cols	avgG	avgT	stdT	saveC
20	5	1103	930	0.43	77	76	69%
20	10	1103	930	0.44	46	25	70%
40	5	2823	2550	0.54	1325	2161	70%
40	10	2823	2550	0.59	309	340	71%
60	5	5343	4970	0.51	5738	4909	73%
60	10	5343	4970	0.62	3835	6118	74%
80	5	8663	8190	0.51	29991	30560	69%
80	10	8663	8190	0.66	29712	26408	73%

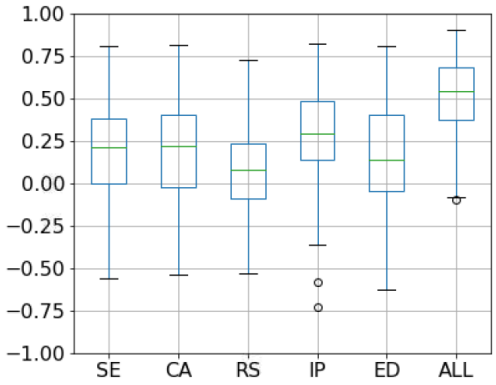
Table 1: Parameter settings, performance comparisons, and savings in carbon emissions of randomly generated problem instances.

in the table, is the saving in travel distance, normalized by the distance traveled by a dedicated vehicle. As shown in the table, by performing deliveries along crowdsourced worker's routine route, we can save around 70% of the travel distance.

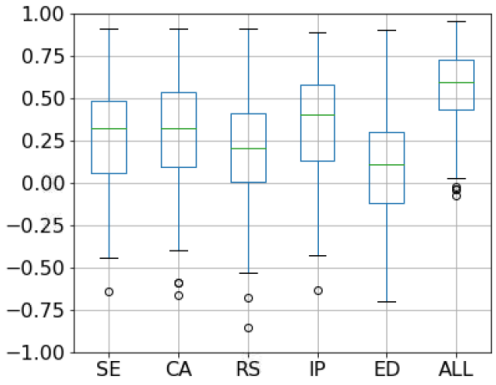
The computational performances of our BnC variants are visualized in Figure 2 using in the number of BnB nodes and the amount of CPU time consumed. In both sub-figures, the y values are computed as (ILP - BnC) / ILP. In other words, the plotted values are the improvement of the BnC variant over the ILP model. Improvements from all numerical instances are summarized using box plots, where the box covers the range from 25% (Q_1) to 75% (Q_3), with the line in the box representing the median. The whiskers attached to the box covers the range of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$. Any points that are not covered by the range of the whiskers are labeled as outliers.

On the number of processed nodes, all cut variants are useful in tightening bounds, which result in fewer explored BnB nodes than that of the ILP solver. While the variant combining all cuts (ALL) results in nearly 50% reduction in the explored BnB nodes.

On the CPU time, we note that although medians and interquartile ranges are in the positive realm for most cut variants (all except ED), their full range is quite variable, again demonstrating a strong dependency of the computation time on problem parameters. A positive note is that by combining all cut variants, the median and the interquartile range improve significantly, and almost all instances (except for a few outliers) end up in the positive domain. The median improvement of the BnC approach over the ILP solver is around 60% when all cut variants are implemented.



(a) Number of processed BnB nodes.



(b) CPU time.

Figure 2: Comparing the ILP formulation and our BnC variants with various types of cuts.

7 Conclusions

In this paper, we propose a decision-support algorithm for a single crowdsourced worker that suggests delivery tasks to perform in his/her upcoming route, considering this worker's routine route, associated time window at all locations along the route, and tasks' delivery time windows. The problem is a variant of the orienteering problem and the pickup and delivery problem, where no known efficient algorithm exists. We contribute to the understanding of this problem class by first formulating this planning problem using an integer linear programming (ILP) model. Since the ILP formulation is not scalable, we come up with a new branch-and-cut (BnC) algorithm to solve the same problem more efficiently. Our major innovations are in the generation of a wide variety of valid inequalities and the modifications of the classical BnC process so the computationally expensive separation problems (for cuts generation) are only solved beneficial.

In our numerical experiments, we first demonstrate how the ILP formulation can lead to significantly better solution quality than a real-world-inspired greedy heuristics. We also quantify the potential sustainability improvement by adopting the crowdsourcing scheme in the last-mile logistics. We then further drill down into our various BnC variants, and quantify the pros and cons of including different classes of valid inequalities. Finally, we show that combining all cuts while using our separation heuristics lead to the best performance gains computationally.

Our immediate next step is to generalize our approach so that it could handle the planning of multiple agents simultaneously.

Acknowledgments

This research is supported by the Agency for Science, Technology and Research (A*STAR), Fujitsu Limited, and the National Research Foundation Singapore as part of the A*STAR-Fujitsu-SMU Urban Computing and Engineering Centre of Excellence

References

[Arslan *et al.*, 2019] Alp M Arslan, Niels Agatz, Leo Kroon, and Rob Zuidwijk. Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers. *Transportation Science*, 53(1):222–235, 2019.

[Behrend *et al.*, 2019] Moritz Behrend, Frank Meisel, Kjetil Fagerholt, and Henrik Andersson. An exact solution method for the capacitated item-sharing and crowdshipping problem. *European Journal of Operational Research*, 279(2):589–604, 2019.

[Chen *et al.*, 2015] Cen Chen, Shih-Fen Cheng, Hoong Chuin Lau, and Archan Misra. Towards city-scale mobile crowdsourcing: Task recommendations under trajectory uncertainties. *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1113–1119, 2015.

[Cheng *et al.*, 2017] Shih-Fen Cheng, Cen Chen, Thivya Kandappu, Hoong Chuin Lau, Archan Misra, Nikita Jaiman, Randy Tandriansyah, and Desmond Koh. Scalable urban mobile crowdsourcing: Handling uncertainty in worker movement. *ACM Transactions on Intelligent Systems and Technology*, 9(3):1–24, 2017.

[Deng *et al.*, 2016] Dingxiong Deng, Cyrus Shahabi, Ugur Demiryurek, and Linhong Zhu. Task selection in spatial crowdsourcing from worker's perspective. *Geoinformatica*, 20(3):529–568, 2016.

[Fischetti *et al.*, 1998] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10:133–148, 1998.

[Gansterer *et al.*, 2017] Margaretha Gansterer, Murat Küçüktepe, and Richard F. Hartl. The multi-vehicle profitable pickup and delivery problem. *OR Spectrum*, 39:303–319, 2017.

[Guo *et al.*, 2019] Xuezhen Guo, Yngrid Jaqueline Lujan Jaramillo, Jacqueline Bloemhof-Ruwaard, and G.D.H. Claassen. On integrating crowdsourced delivery in last-mile logistics: A simulation study to quantify its feasibility. *Journal of Cleaner Production*, 241:118365, 2019.

[Holland *et al.*, 2017] Chuck Holland, Jack Levis, Ranganath Nuggehalli, Bob Santilli, and Jeff Winters. UPS optimizes delivery routes. *INFORMS Journal on Applied Analytics*, 47(1):8–23, 2017.

[Huang *et al.*, 2018] Yixiao Huang, Martin Savelsbergh, and Lei Zhao. Designing logistics systems for home delivery in densely populated urban areas. *Transportation Research Part B*, 115:95–125, 2018.

[Kazemi and Shahabi, 2012] Leyla Kazemi and Cyrus Shahabi. Geocrowd: Enabling query answering with spatial crowdsourcing. *Proceedings of the 20th International Conference on Advances in Geographic Information*, pages 189–198, 2012.

[Kim, 2015] Yongsung Kim. Libero: On-the-go crowdsourcing for package delivery. *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 121–126, 2015.

[Ruland and Rodin, 1997] K.S. Ruland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1–13, 1997.

[Stenger *et al.*, 2013] Andreas Stenger, Daniele Vigo, Steffen Enz, and Michael Schwind. An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping. *Transportation Science*, 47(1):64–80, 2013.

[Sun *et al.*, 2018] Dezhi Sun, Ke Xu, Hao Cheng, Yuanyuan Zhang, Tianshu Song, Rui Liu, and Yi Xu. Online delivery route recommendation in spatial crowdsourcing. *World Wide Web*, 2018.

[Tran *et al.*, 2018] Luan Tran, Hien To, Liyue Fan, and Cyrus Shahabi. A real-time framework for task assignment in hyperlocal spatial crowdsourcing. *ACM Transactions on Intelligent Systems and Technology*, 9(3):37:1–37:26, 2018.

[Wang *et al.*, 2016] Yuan Wang, Dongxiang Zhang, Qing Liu, Fumin Shen, and Loo Hay Lee. Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions. *Transportation Research Part E*, 93:279–293, 2016.

[Winkenbach *et al.*, 2016] Matthias Winkenbach, Paul R. Kleindorfer, and Stefan Spinler. Enabling urban logistics services at la poste through multi-echelon location-routing. *Transportation Science*, 50(2):520–540, 2016.