Multi-Vehicle Refill Scheduling with Queueing

Giovanni D'Urso[a,*], Stephen L. Smith[b], Ramgopal Mettu[c,1], Timo Oksanen[d], Robert Fitch[e,2]

[a] *Australian Centre for Field Robotics, The University of Sydney*
[b] *Department of Electrical and Computer Engineering, University of Waterloo*
[c] *Department of Computer Science, Tulane University*
[d] *Department of Electrical Engineering and Automation, Aalto University*
[e] *Centre for Autonomous Systems, University of Technology Sydney*

Abstract

We consider the problem of refill scheduling for a team of vehicles or robots that must contend for access to a single physical location for refilling. The objective is to minimise time spent in travelling to/from the refill station, and also time lost to queuing (waiting for access). In this paper, we present principled results for this problem in the context of agricultural operations. We first establish that the problem is NP-hard and prove that the maximum number of vehicles that can usefully work together is bounded. We then focus on the design of practical algorithms and present two solutions. The first is an exact algorithm based on dynamic programming that is suitable for small problem instances. The second is an approximate anytime algorithm based on the branch and bound approach that is suitable for large problem instances with many robots. We present simulated results of our algorithms for three classes of agricultural work that cover a range of operations: spot spraying, broadcast spraying and slurry application. We show that the algorithm is reasonably robust to inaccurate prediction of resource utilisation rate, which is difficult to estimate in cases such as spot application of herbicide for weed control, and validate its performance in simulation using realistic scenarios with up to 30 robots.

Keywords: Agricultural robotics, Multi-robot systems, Multi-robot scheduling, Multi-vehicle scheduling, Refill scheduling, Queuing, Spot spraying, broadcast spraying, slurry application

---

1. Introduction

In agricultural operations, timing is crucial; if operations are completed too early, or specifically too late, profitability is reduced due to decreases in crop yield or quality. Timing of operations can be negatively impacted by issues with the required components, such as: agricultural vehicle(s), the

* Corresponding author

Email addresses: `g.durso@acfr.usyd.edu.au` (Giovanni D'Urso), `stephen.smith@uwaterloo.ca` (Stephen L. Smith), `rmettu@tulane.edu` (Ramgopal Mettu), `timo.oksanen@aalto.fi` (Timo Oksanen), `rfitch@uts.edu.au` (Robert Fitch)

[1] This work was done while the author was supported by an IRCA fellowship at The University of Sydney.

[2] This work was done in part when the author was with The University of Sydney.

input material (seeds, fertilizer, herbicide, etc.), and the driver(s). Late completion of the operation can be caused by too few machines, problems in logistics of input material, and availability of driver/operator. By using robotics, the issue of driver/operator availability can be solved through *autonomous* operation. However, core questions remain concerning the proper number of machines to use and the parameters of these machines, such as operational width. In the case of multiple machines (autonomous or human driven) the logistics of input material also plays an important role with respect to operational efficiency.

Operational efficiency, or more specifically *field efficiency*, is defined in the ASAE D497.7 (2011) standard as the real operational performance of a vehicle compared to its theoretical maximum with the given speed and width, without turns. Field efficiency is less than 100% due to turning, irregularly shaped field plots, and refilling, among other factors. Derived from collected data, the ASAE D497.7 (2011) standard defines 70% (+/- 10%) field efficiency for fertiliser spreaders and 65% (+/-15%) for boom sprayers. These numbers are typically used when selecting the proper size of machine for a specific farm.

In the case of multiple robots or vehicles, an important factor in maintaining high field efficiency is to determine the proper refill timing for each unit. Refilling the container of the vehicle with seeds, fertilisers, herbicide, fungicide, pesticide, manure, slurry, lime or fuel is usually done at the edge of the field area. Refilling, or replenishing, the supply of input materials must be done semi-regularly at refill stations and the refill procedure can require a substantial amount of time. Due to varying shaped fields and the distances that vehicles must travel to the refill station, the order in which vehicles are refilled cannot always be the same. Otherwise, the quickest vehicle with the shortest routes has to wait until the others have refilled. Harvesting operations where tanks are emptied at the edge of the field or at a central storage location are analogous to refilling, but for simplicity, in this work we focus our discussion on refilling. If multiple vehicles work simultaneously, a given vehicle may need to wait its turn, or *queue*, at the refill station.

We are interested in understanding the optimisation problem that arises in these scenarios: at what points in time should a vehicle pause its work and travel to a refill station such that total refill time (travel, queuing, and refilling) is minimised? We refer to this optimisation problem as *refill scheduling with queuing.*

The refill scheduling problem is relevant to both traditional and robotic agricultural operations. In traditional broadacre agriculture, for example, broadcast spray rig operators typically employ a *greedy* decision strategy where they wait until the spray tank is empty and then drive to the refill station. This strategy, unfortunately, can lead to surprisingly large time losses. Agricultural robots are subject to similar, or worse, time losses (Richards et al., 2015). These losses are exacerbated in small, relatively slow-moving robot systems operating in large areas; a single round-trip to a refill station can require several hours of travel time, as we show through experiments in Sec. 6. It is critically important to develop a principled theoretical understanding of this problem in order to design efficient algorithms that will support current and future applications of agricultural robots, and increase efficiency in traditional operations.

Interestingly, there has been surprisingly little work that addresses refill scheduling with queuing. Oksanen and Visala (2009) proposed an efficient greedy algorithm that addresses travel time, but not queuing. Bochtis and Sorensen (2009) formulated a variety of related problems under the umbrella of the vehicle routing problem, which is NP-hard, but did not provide a rigorous complexity analysis. The existence of polynomial-time algorithms for certain variants (Oksanen and Visala, 2009; Patten et al., 2016) contradicts the assumption that all variants that can be formulated as vehicle routing problems are NP-hard, and thus motivates the need for a more rigorous approach.

In this paper, we present analysis and algorithms for refill scheduling with queuing. We show that, although polynomial-time algorithms exist for the case of instantaneous refill time, the general problem with non-zero refill time is NP-hard. We also show that the ratio of working time to refill time imposes a limit on the number of vehicles that can work together productively given a single refill station. Based on this analysis, we present two algorithms. The first is an exact algorithms that computes an optimal refill schedule, but is infeasible in practice for all but the smallest problem instances. The second algorithm computes an approximately optimal solution and is effective in practice. The algorithm maintains upper and lower bounds on the optimal solution, and tightens these bounds iteratively. Thus, the algorithm produces higher quality solutions given more computation time, but can produce a valid solution at any time. An algorithm with this property is known as an *anytime* algorithm.

We report simulation results, using examples of spot spraying, broadcast spraying and slurry spreading robots, that characterise the practical performance of our solution in comparison to the greedy approach. Our results show that the performance gap between methods, measured in terms of total time attributed to refilling, can be wide. Importantly, we also analyse the sensitivity of our solution to variations in the actual rate of resource consumption versus the estimated rate. This analysis shows that our algorithm exhibits reasonable performance, particularly in the case where the usage rate is overestimated, and motivates further work in developing methods that directly consider uncertainty in the consumption rate estimate.

The contributions of this work are to provide the first complexity analysis of the refill scheduling with queuing problem, and to present exact and approximate solutions. Our algorithms support the design of software tools that apply to any agricultural robot system that consumes and refills physical resources, and similarly to manually operated agricultural vehicles.

Throughout the paper, we use the term *robot* to loosely imply either an autonomous or human-operated vehicle. We use the term *field plot* to mean an agricultural area where crops are grown.

2. Related Work

The work most closely related to ours is by Oksanen and Visala (2009), who propose a greedy algorithm for refill scheduling to reduce time lost in travelling to and from a refill station. The robot monitors its resource level and greedily chooses when to refill. In our previous work, we give an optimal polynomial-time algorithm for this case (Patten et al., 2016). Neither paper, however, considers queuing.

A series of papers has explored the idea of modelling a wide range of optimisation problems in agricultural field operations as instances of the general vehicle routing problem (VRP) (Bochtis and Sorensen, 2009, 2010; Jensen et al., 2015a,b). This work is important for multiple reasons; it focuses attention on the benefits of addressing the computational problems inherent in field operations, and provides a pathway to the convenient use of off-the-shelf solvers. However, there are two severe limitations of this approach. First, the VRP cannot express all possible computational problems of interest to field operations. The problem we study in this paper is one such instance. Second, formulating a problem as an instance of a VRP does not theoretically imply that the problem is as computationally difficult as the VRP. Our previous work (Patten et al., 2016) provides a concrete example of a variant that can be solved in polynomial-time, but also can be (undesirably) formulated as a VRP.

The branch and bound approach is one method that can be used to solve VRPs (Toth and Vigo, 2002) and a wide range of other problems such as information gathering (Best and Fitch, 2016; Binney and Sukhatme, 2012). In adopting this approach, it is necessary to compute upper and lower bounds on the cost of the (unknown) optimal solution. We develop specific algorithmic procedures for calculating bounds that both minimise total refill time (including queuing) and also exhibit reasonable run-time performance. Our work also allows for replanning to account for uncertainty in usage rate estimation, as in Edwards et al. (2015), but we show that replanning is not always necessary.

The problem of computing a plan that visits the entire area of a field plot is an instance of *coverage planning*, a well-studied problem in robotics. A recent survey can be found in Galceran and Carreras (2013). Both single- and multi-robot coverage are NP-hard problems (Rekleitis et al., 2008), but reasonable solutions can be computed using simple methods such as the *boustrophedon decomposition* (Choset et al., 2005). Recent work specific to agricultural applications focuses on choosing an optimal track orientation (Oksanen and Visala, 2009; Jin and Tang, 2011; Hameed, 2014). Here, we assume that track orientation is given, and that the output of a coverage planner is also given. These are reasonable assumptions because track orientation is often fixed ahead of time (as in controlled traffic farming), and coverage planning solutions for this case are readily available in the literature.

Our formulation of refill scheduling is related to the problems of fixed-route vehicle refuelling (Suzuki, 2014; Lin et al., 2007) and electric vehicle recharging (Schneider et al., 2014; Bruglieri et al., 2015; Keskin and Catay, 2016). This work does not address queuing, however. Nam and Shell (2015) address resource contention, but in the context of multi-robot task allocation which does not directly apply to refill scheduling.

In our work we assume that a given field plot has been segmented and that the area to be covered by each robot is thus known. Another view of the refilling problem is then as a scheduling problem in which refilling each robot is a task that must be scheduled periodically (i.e., the span of time in which a robot does not refill has a hard upper bound). At any point on the path of a robot, there exists a fixed cost to schedule the task that is simply the travel distance to the refill station. Then, the goal is to schedule k tasks in a periodic fashion so as to minimise total time spent due to queuing and scheduling costs. This problem bears closest resemblance to *group interval scheduling* (Keil, 1992), in which a set of n independent tasks of possibly differing execution times must be scheduled for execution. This problem is considerably simpler than ours and, due to the queueing costs, even a simple problem has an exponentially sized number of jobs. Our problem can also be formulated as other scheduling problem variants (Leung, 2004). However, these formulations are not practical and are hard to solve (Chen et al., 1998) due to the number of intervals involved.

3. Problem Statement and Characterisation

In this section we explain our formulation of the problem, prove that the problem is NP-hard, and provide bounds on the amount of work that multiple robots can perform concurrently.

Intuitively, we define the refill scheduling problem as the question of how to modify robots' paths by judiciously splicing in trips to a refill station. In other words, the problem is how to take an initial path in which a robot prematurely exhausts its resource (herbicide, for example), and create a new path by choosing points at which the robot stops and refills. This new path is constructed such that the robot can complete its work without running empty. We would like to minimise the additional time spent in refilling, which includes travel, queuing, and the refill operation itself. We assume that we are given: the number of robots, a path that completes the task without refilling, and the performance characteristics of the robots (e.g., travel speed, resource usage rate, refill rate). We further assume that the robots are identical, or *homogeneous* and that resource usage rate is constant within a field plot. Because we are motivated by agricultural applications, we assume that a robot must traverse a road network to reach the refill station, as opposed to taking the shortest obstacle free path (which likely would involve the undesirable arbitrary traversal of a field plot). Two potential solutions for an example problem are shown in Fig. 1.

Formally, we state the refilling problem as follows. We are given a graph $G = (V, E)$ whose vertices

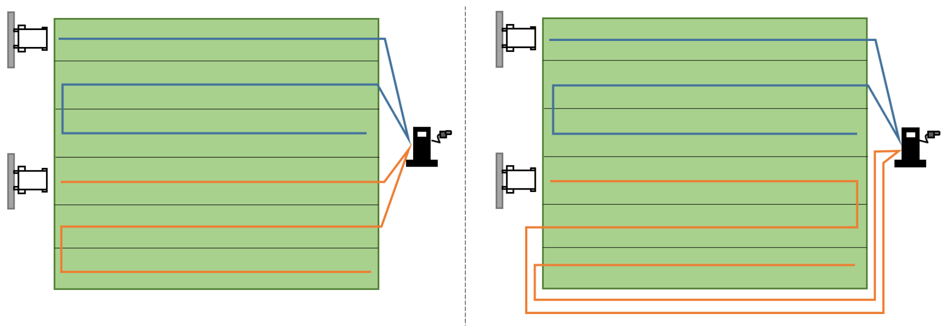


Figure 1: An example multi-robot refill scheduling problem where each robot needs to perform at least one refill. The left schedule may appear optimal, but if the refill time is lengthy, the right could be less costly. By forcing one of the robots to take a longer path to the refill station and incurring a longer travel cost, queuing can be avoided.

represent waypoints in the field plot, and whose edges represent travel between waypoints (i.e., straight line travel, turns etc.). There are k robots, and each robot is assigned a portion of the field plot *a priori*. The vertices are partitioned accordingly into sets $V^1, V^2, \ldots, V^k$. Robot j must cover all vertices in V^j to complete its task and can decide at each vertex whether to refill its resource. Naturally, we assume that the capacity of each robot is insufficient to fully complete its assigned task without refilling; otherwise we would not have to make any refilling decisions along the path.

For notational simplicity, we assume that the field plot is equally partitioned and that each set V^j has n vertices. We also assume without loss of generality that robot j visits vertices V^j in sequence (i.e., $v_1^j, v_2^j, \ldots, v_n^j$). We denote the edge (v_i^j, v_{i+1}^j) as e_i^j for convenience. We consider the refilling problem with a single refill station at a fixed location in the field plot. Let T_d denote the capacity of the resource held by the robot (e.g., charge, fertiliser, herbicide, etc.) which decreases at a known rate R_f during operation, and let T_w be the amount of time the robot can work before refilling (equivalent to $T_w = R_f/T_d$). Let T_r denote the amount of time needed to refill from empty. We assume that a robot can also refill from a non-empty state in proportionately less time (i.e., $T_r/2$ time to refill from $T_d/2$ capacity). We assume that the robot starts with full capacity and must complete its task with full capacity. At vertex v_i^j, robot j is also assigned a travel time cost r_i^j for travelling to and from the refill station. We define the time spent working and travelling between subsequent vertices v_i^j and v_{i+1}^j without refilling as $d(v_i^j, v_{i+1}^j)$. Thus, the time cost for working and travelling without refilling between two given vertices: v_i^j and v_m^j is $d(v_i^j, v_m^j) = \sum_{c=i}^{m-1} d(v_c^j, v_{c+1}^j)$. This is equivalent to the sum of the time taken to traverse each edge in the path between the two vertices.

Our goal is to select, for each robot j, a set of waypoints $W^j \subseteq V^j$ that defines the points where robot j will stop working and perform a refill operation. In order to capture traversal from the start to the first chosen waypoint, W^j must contain v_1^j, the first vertex in a robot's path. We now formally define the objective function we are interested in optimising. We need to select subsets W^j,

$j \in \{1, \ldots, k\}$ that minimise the total refill and waiting times:

$$\sum_{j=1}^{k} \sum_{v_i \in W^j} \left(r_i^j + T_r \cdot (1 - f(v_i, v_{i+1})) + Q(v_i, W^1, \ldots, W^k) \right) \qquad (1)$$

$$\textit{Subject to} \qquad f(v_i, v_{i+1}) \leq T_d \qquad \forall v_i \in W^j$$

Where $Q(\cdot)$ measures the waiting time for robot j after travelling to the refill station from vertex v_i^j. Function $f(v_i, v_m)$ is the amount of resource used to traverse from vertex v_i to another vertex v_m without refilling.

3.1. Complexity Analysis

In this section we prove the complexity class of the refilling problem by reducing the group interval scheduling problem to it. The reduction uses the fact that the group interval scheduling problem has been proven to be NP-complete to prove the complexity of the refilling problem.

The group interval scheduling problem is defined as follows.

Problem 3.1 (Group interval scheduling problem)**.** We are given m sets (groups), each containing several nonempty intervals of $\mathbb{R}_{\geq 0}$. We write set T_j, $j \in \{1, \ldots, m\}$ as

$$T_j = \{[s_1^j, e_1^j], \ldots, [s_{n_j}^j, e_{n_j}^j]\}.$$

Does there exist a selection of one interval from each set T_j such that all intervals are pairwise disjoint?

This problem is NP-complete, even when each interval has identical width and each group has the same number of intervals (Keil, 1992). We use this problem to establish the following result.

Theorem 3.2. *The multi-robot refilling problem is NP-hard.*

Proof. Consider an instance of group interval scheduling in which each group contains n intervals of equal width. To establish the result, we give a reduction from this instance of group interval scheduling to multi-robot refilling (that is, we show how an optimal algorithm for multi-robot refilling could be used to solve group interval scheduling). Consider a group T_j for some $j \in \{1, \ldots, m\}$, which consists of intervals $[s_i^j, e_i^j]$, for $i \in \{1, \ldots, n\}$. We begin by sorting the intervals such that $s_1^j \leq s_2^j \leq \cdots \leq s_n^j$. Since each interval has equal width, there is a constant $\Delta t > 0$ such that for each j,

$$e_j - s_j = \Delta t.$$

To encode this group of intervals as a multi-robot refilling problem, we introduce a robot j with a path $v_s^j, v_1^j, v_2^j, \ldots, v_n^j, v_f^j$, where v_s^j and v_f^j are the start and finish vertices of the robot path, and $v_1^j, v_2^j, \ldots, v_n^j$ are n intermediate vertices.

We define the resource consumed to move between vertices on this path as

$$f(v_s^j, v_1^j) = \frac{1}{2}$$
$$f(v_i^j, v_{i+1}^j) \geq 0 \quad \text{for all } i \in \{1, \dots, n-1\}$$
$$f(v_n^j, v_f^j) = \frac{1}{2},$$

where the terms $f(v_i^j, v_{i+1}^j)$ are any positive numbers satisfying

$$\sum_{i=1}^{n-1} f(v_i^j, v_{i+1}^j) = \frac{1}{2}.$$

By this construction, one-half of the resource tank is required to travel from the start v_s^j to v_1^j. Another one-half of the tank is needed to travel from v_1^j to v_n^j. Finally, one-half of the tank is needed to travel from v_n^j to the finish v_f^j. Since the total resource needed from start to finish is one and one-half tanks, the robot must refill at least once.

We define the tank to be full when the robot starts at v_s^j, and allow the tank to reach empty at the moment when v_f^j is reached. Notice that by this construction, the robot can reach v_f^j with an empty tank by refilling exactly one-half of its tank at any of the vertices $v_1^j, \dots, v_n^j$.

Next, we define the time T_r to refill as

$$\frac{1}{2}T_r = \Delta t.$$

We fix a constant $c > 0$, and define the time to travel from vertex i to and from the refill station as $r_i^j = c$ for each vertex i. The time from vertex i to the refill station is $c/2$, as is the time from the refill station back to vertex i.

Finally, the times to travel between vertices $d(\cdot, \cdot)$ are defined such that if the robot travels directly from v_s^j to v_i^j without refilling, then it arrives at vertex v_i^j at the time $s_i^j - c/2$. Thus, in this case the robot arrives at the refill station at time s_i^j, and if there is no wait to refill, it finishes refilling at time e_i^j.

From this construction, the minimum amount of time for robot j to complete its path is achieved by refilling exactly once at a vertex v_i^j without any wait at the station. In this case the total time is

$$d(v_s^j, v_i^j) + \frac{c}{2} + \Delta t + \frac{c}{2} + d(v_i^j, v_f^j) = d(v_s^j, v_f^j) + c + \Delta t.$$

Any solution in which the robot refills more than once will incur the cost c twice, and thus will require strictly more time. Moreover, any solution in which the robot must wait to refill will result in the robot departing the refill station at a time after e_i^j and thus a strictly larger time.

Therefore, after performing this construction for each group of intervals $j \in \{1, \dots, m\}$, we have a multi-robot refilling problem with m robots. If the optimal refilling schedule for this problem has each robot refill exactly once, and with no waiting, then the corresponding refill vertices v_i^j for each robot j yield an interval from each set T_j that are pairwise disjoint, and thus show that there exists a solution to the group interval scheduling problem. If the optimal refilling schedule requires multiple refills for a robot, or requires a robot to wait, then there is no solution to the group interval scheduling problem. Since the construction of the multi-robot refilling instance can be performed in polynomial time, the reduction is complete, and the result established. □

In our proof we assume each robot has the initial condition of a full tank and the termination condition of an empty tank. We conjecture that a similar proof can be formulated for the more general cases of arbitrary initial and termination conditions for problem variants with those properties.

3.2. A Bound on Concurrency

In this section we give a bound on the maximum number of robots that can work concurrently without queuing. This bound is a function of the refill operation length (T_w/T_r).

Let k_{max} be the maximum number of robots that can effectively work together such that all robots are either working or refilling (i.e., no robots are queuing at the refill station). For the purposes of this proof, we also assume that there is no travel cost involved in travelling to the refill station.

Given the definition of k_{max}, we can prove that this number is essentially limited by the ratio of a robot's *work time* (how long a robot can work before needing to refill) to *refill time* (how long it takes for a robot to refill itself to maximum capacity).

Theorem 3.3. *The maximum number of effective working robots k_{max} satisfies $k_{max} \leq \frac{T_w}{T_r} + 1$.*

Proof. Let t_1 and t_2 be points in time during the schedule. We define $F^j(t)$ as the remaining amount of resource for robot j at time t. We then define a steady-state time interval T as

$$\begin{aligned} T \quad &= \quad \min\,(t_2 - t_1) \qquad (2) \\ \textit{subject to} \quad & \quad t_2 > t_1 \\ & \quad F^j(t_1) = F^j(t_2) \forall j \in (1, \ldots k) \end{aligned}$$

such that T is a sufficiently long minimal time window. This time window is a "snapshot" of the system in operation. At the start of this time interval, each robot has a certain capacity remaining before needing to refill. By *steady-state*, we mean that each robot has the same capacity at the end of the time interval as it had at the start.

In order to maintain capacity according to our definition of steady-state, any work time in T must be balanced by an equivalent amount of refill time, which we denote as R. The refill time can be split across multiple refill events. Time R is simply the refill time required *in total* to maintain capacity within the given time interval.

We define R in terms of depletion and refill time: $R = T(\frac{T_r}{T_r+T_w})$. Now, in order for no robot to queue, the sum of refill times of all robots must not exceed the total time available, which is T. Therefore, we have $kR \leq T$.

Now we solve for k in order to determine k_{max}. Rearranging terms, we have $k \leq \frac{T}{R}$. Substituting for R, we get $k \leq \frac{T}{T(\frac{T_r}{T_r+T_w})}$. Eliminating T, we have $k \leq \frac{1}{\frac{T_r}{T_r+T_w}} = \frac{T_w}{T_r} + 1$, as claimed. □

This theorem implies that it is not always advantageous to construct very large robot teams; excess robots will simply queue for the refill station indefinitely. Conversely, any increase in productivity of a system through robots working in parallel necessarily involves an appropriate increase in the number or capacity of refill stations. This notion is intuitive but the value of our formalism is to provide simple analytical methods for designing systems.

4. Exact Solution

In this section we present our first solution approach, which solves the refill scheduling problem optimally. We then briefly discuss the applicability of this solution in practice.

The algorithm we present is based on an algorithmic technique known as *dynamic programming (DP)* (Cormen et al., 2001). DP is a method of finding an exactly *optimal* or "best" solution to a problem with respect to some metric or *cost function*. DP works by breaking the initially large and hard-to-solve problem into smaller subproblems that can be solved more easily. A problem that can be solved by combining optimal solutions to subproblems is said to have the property of *optimal substructure*. Utilising this decomposition allows DP algorithms to recursively compute the optimal solution by reusing optimal solutions to subproblems, thereby avoiding an exhaustive search over the solution space. Avoiding exhaustive searching allows dynamic programming to calculate a far smaller number of possible solutions and thus reduce the amount of computation required to compute an optimal solution.

4.1. Formulation for a Single Robot

For exposition, we first we give the optimal substructure for the case in which we have a single robot (i.e., $k = 1$) that must choose waypoints for refilling. We use the terminology given in Sec. 3 except we drop the superscript since we only have a single robot. For $1 \leq i \leq n$, let t_i be the optimal cost that is achievable at vertex v_i. Then, we seek to compute t_n, along with refilling decisions at every vertex. We have that

$$t_n = \min_{i|f(i,n)\leq 1} \left[t_i + d(v_i, v_n) + r_n + T_r \cdot (1 - f(v_i, v_n)) \right], \quad (3)$$

where $f(i,n)$ is the resource cost of travelling from v_i to v_n without any refilling. This characterisation of t_n admits a dynamic programming approach because we can compute t_i, for $1 \leq i \leq n$ successively; we note that f_i can be computed simultaneously. This algorithm runs in $\mathcal{O}(n^2)$ time.

4.2. Formulation for k Robots

We can generalise the above formulation by including a term that captures queuing at the refill station that corresponds to Q in the objective function. Let T_n be the cost of an optimal joint schedule for all k robots to complete their tasks. We then have that

$$T_n = \min_{A_{1,n},\ldots,A_{k,n}} \left[\sum_{j=1}^{n} \left(t_n^j + d(v_i, v_n) + r_n^j + \sum_{l | A_{l,n} < A_{j,n}} r_n^l/2 + T_r \cdot (1 - f(v_i^l, v_n^l)) \right) \right],$$

where $A_{j,n}$ is the arrival time of robot j at the refill station from v_n^j. We note that these arrival times can be computed along with $f(v_i^l, v_n^l)$. For any robots that do not refill, $A_{j,n}$ can be set to a sentinel value that excludes it from the summation.

To give an upper bound on running time we examine the worst case scenario where each robot has enough resource to reach the penultimate vertex in its schedule without refilling. For each robot, there are $(n-1)!$ possible refill schedules, because at each vertex the robot can reach any of the remaining vertices without refilling. All combinations for the k robots must be considered. Thus, the worst case running time is $\mathcal{O}([(n-1)!]^k)$.

4.3. Feasibility in Practice

Because the refill scheduling problem is NP-hard (proved earlier in Sec. 3.1), it is not feasible to find an exact solution for large problem instances. However, the exact approach may be useful for small problem instances and therefore it is interesting to consider the limitations of the exact approach in practice. The computational cost of considering additional robots is exponential in k due to the large number of possible schedules that must be evaluated. This combinatorial effect dominates both the computation and memory cost; solving multi-robot problems exactly, in a timely manner, and with limited memory resources becomes infeasible. This issue is discussed in more detail in Sec. 6.2. If the number of robots and rows are limited (such as in smaller scale agriculture operations), the DP approach is feasible and can be utilised to calculate an exact solution. As we discuss next, we can address the issues with solving multi-robot problems by considering an approximate solution (with a provably bounded cost).

5. Branch and Bound Solution

We mitigate the memory and computation time requirements of our DP approach by designing a *branch and bound (BnB)* algorithm. BnB algorithms search the solution space (all possible solutions) by building a tree that partitions the solution space iteratively. An example is shown in Fig. 2. In this tree the root corresponds to the entire solution space and its children correspond to a partition of the solution space. For each of these children we calculate *bounds* on the cost of the partition the child represents. If a given child is identified to have potential, then a *branching* step is performed which further partitions the search space. Alternatively if a child has no potential it can be safely eliminated (*pruned*). This process continues until the algorithm is terminated by a user or completes its exploration of the search space. A beneficial feature is that the algorithm is anytime: it can be stopped at any time to output the current best solution. The algorithm also provides the cost for a given solution along with bounds on the cost of the optimal solution. Hence the algorithm is capable of giving a quantitative statement of solution quality (nearness to optimal).

In this section we formally define our BnB algorithm (Algorithm 1), including algorithms for computing both upper and lower bounds (Algorithm 2), and branching. We also provide complexity analysis for the bounds computations, and give branching heuristics that improve the algorithm's rate of convergence to an optimal solution.

5.1. Branch and Bound Formulation

We formulate the branch and bound tree $T = (N, E)$, where nodes (or equivalently, vertices) N represent refilling stop decisions for k robots, and edges E represent the working area covered between refill stops. Leaf-to-root paths in the tree encode a complete refill schedule, and paths from interior nodes to the root likewise represent a partial schedule. We define a node b such that $N_b = \{v_n^1, \ldots, v_n^j, \ldots, v_n^k\}$ where v_n^j is the nth vertex in V^j for robot j, and N_b^j corresponds to the vertex in N_b for robot j. Let P_b^j be the path from the root to a given node b for a given robot j.

To maintain the resource budget constraint, the BnB tree is constructed only using valid edges. An edge is valid if no robot exceeds its resource budget $f(N_p, N_c) \leq T_d$, where N_p is a parent node, N_c is a child node, and $f(l, m)$ is the resource cost of travelling from v_l^j to v_m^j without refilling. Consequently, we say that a schedule is valid if it consists entirely of valid edges.

The cost assigned to a tree node C_b is the sum of the cost to follow a refill schedule P_b and the cost of performing a refilling operation (including the queuing time cost):

$$C_b \;=\; C_p + \left(\sum_{j=1}^{k} d(N_p^j, N_b^j) + r_p^j + T_r \cdot (1 - f(N_p^j, N_b^j)) + Q(v_b^j, P_b^1, \ldots, P_b^k) \right),$$

where $d(l, m)$ is the cost of covering the work area between vertices v_l^j and v_m^j, and C_p is the cost of the parent of node b.

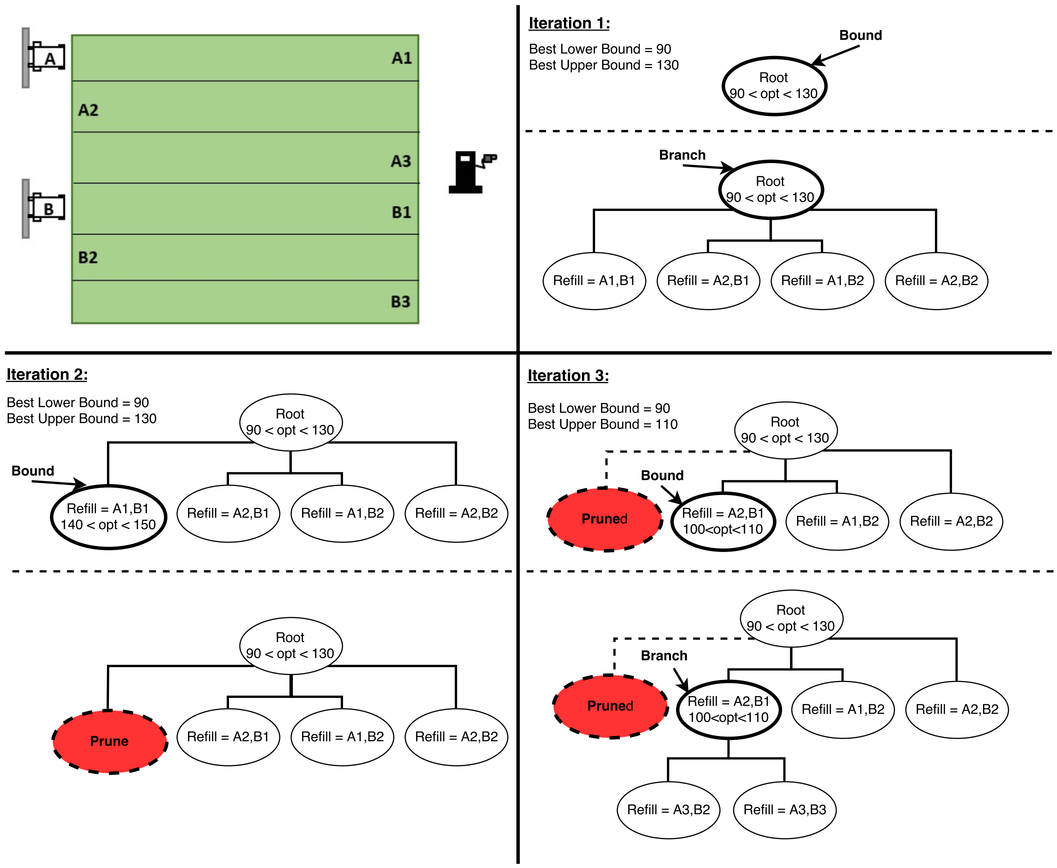


Figure 2: Diagram of BnB tree construction for an example problem with two robots (robot A and robot B). Each robot needs to perform at least one refill before reaching the end of its path, and the letter-number pairs indicate candidate refill locations. At each iteration the BnB algorithm uses its computed bounds to determine which nodes to branch and which to prune. The node examined in Iteration 1 is a branch node because its subtree has potential to contain a better (lower cost) solution than has been found so far. In Iteration 2 the examined node does not have potential to improve the best solution, so it can be safely pruned. This process continues until the entire search space has been examined or the algorithm is terminated by the user.

5.2. Computing Upper and Lower Bounds

The BnB approach is based on the idea of computing *bounds* on the cost of an (unknown) optimal solution. Bounds are computed such that the optimal solution cost for a given partition of search space is known to fall within these bounds, even though the optimal solution's actual cost is unknown. There are two types of bounds: upper and lower. The actual cost must be at least as large as the lower bound, but no more than the upper bound. An important computational challenge is to develop an efficient method of finding bounds that is faster than finding the optimal solution for a given partition. Otherwise, the benefit of computing bounds is diminished. Pseudocode for computing bounds is detailed in Algorithm 2.

We first address the case of computing lower bounds (LB). The tightest possible lower bound would, of course, be equal to the cost of an optimal solution. However, computing the optimal solution here is infeasible due to the exponential size of the joint action space induced by the interaction of multiple robots. One of the reasons that we need to consider the joint action space is to compute the cost of queuing. It is possible to consider a lower bound that ignores queuing cost, but unfortunately this bound would still be difficult to compute because the problem space remains large. Relaxing the queuing constraint does have a benefit; it also removes interactions between robots and decouples their costs. The lower bound computation we propose makes use of this insight. Rather than reasoning about joint actions, we instead compute a lower bound by considering each robot independently. Recall that each node in the BnB tree encodes a partial schedule. To bound the cost of a complete schedule that passes through a given node, we use the single-robot DP algorithm (Sec. 4.1) to complete the partial schedule optimally. The sum of the single-robot costs underestimates the true cost because it ignores queuing, and therefore represents a lower bound. The computational benefit of this approach is that the bound can be computed in polynomial time.

The lower bound for node b is formulated mathematically as:

$$LB_b \quad = \quad C_b + \left(\sum_{j=1}^{k} \sum_{v_i \in (OPT_b^j \setminus P_b)} d(v_i^j, v_{i+1}^j) + T_r \cdot (1 - f(v_i^j, v_{i+1}^j)) \right)$$

where OPT_b^j is an optimal single robot schedule that passes through node b for a given robot j.

Computation time can be further reduced by taking advantage of *memoisation* (caching the results of computation). Memoisation is effective here because many of the independent schedules appear in multiple joint schedules. Thus, the algorithm avoids the computational cost of evaluating bounds for any given independent schedule multiple times.

The upper bound (UB) is based on the schedules produced by the lower bounds; the single-robot schedules are combined into a multi-robot joint schedule (a schedule for the entire team). The key point is that, because an upper bound must be greater than or equal to the cost of the optimal solution, we must now now consider the cost of queuing. Our approach is to reuse the lower bound solution, but incorporate an estimate of the queuing cost. We sum the costs of the schedules for each robot (as with the lower bound) and add the queuing cost given that joint schedule. The cost of this solution cannot underestimate the optimal (because it is the cost of a complete, valid schedule) and therefore represents a valid upper bound.

Formally the upper bound on the cost of node b is defined as:

$$UB_b \quad = \quad \sum_{j=1}^{k} \sum_{v_i \in OPT_b^j} d(v_i^j, v_{i+1}^j) + T_r \cdot (1 - f(v_i^j, v_{i+1}^j)) + Q(v_i^j, OPT_b^1, \ldots, OPT_b^k)$$

The upper bound defined in this way can be computed in polynomial time. Its efficiency is due to the polynomial time complexity of evaluating the cost of (as opposed to finding) a joint schedule; evaluating a given schedule does not involve a combinatorial decision, because the refill stops have already been selected.

Another benefit of this formulation is that it naturally allows the BnB solution to act as an anytime algorithm. A valid solution is available at any time because the upper bound computation provides a complete valid refill schedule and associated cost. We maintain a copy of the current best upper bound and corresponding schedule, which can be output when the algorithm is terminated. The algorithm can be terminated while it is still searching (after a fixed period of time, for example) for an approximately optimal solution, or it can be allowed to run to completion and will yield an exact optimal solution. The quality of the solutions produced by the upper bound computation are discussed later in Sec. 6.1.

A major benefit of our lower and upper bound definitions, in combination, is to provide an indication of the quality of the approximate solution. The global optimal solution falls between the lower bound of the root and the upper bound of the candidate solution, and the cost of our approximate solution also falls between these bounds. Thus, the error between the optimal solution and our approximation can be no more than the difference between these upper and lower bounds. In practice, the benefit is that this difference can be used to determine by how much a solution could potentially improve if the BnB algorithm were allowed to continue its computation.

5.3. Branching

The other main component of the BnB algorithm is the *branching* step. Branching is the process by which a partition of the search space represented by a node is further partitioned. Branching considers two cases: 1) a partition can contain a solution that has potential to reduce cost if expanded, and 2) there is no possible way to reduce the cost further. A region of the search space can contain a solution with lower final cost when the lower bound is strictly less than the current best upper bound found so far. Functionally this means that there may exist a full schedule, based on this partial schedule, that has a lower cost. Alternatively if the lower bound is strictly greater than or equal to the current best upper bound, the solution can be safely deleted or *pruned*, as there is no possible schedule in that partition that will reduce the cost further.

The branching step is outlined in pseudocode as Algorithm 3. Branching expands partial solutions by creating child or branch nodes that represent all the next possible valid refill stops. The effect of branching is to incrementally extend the given partial schedule. We compute the set of child nodes as follows. We first step along each robot's path and build a list of all stops that are reachable without requiring a refill. Secondly we create a child node for each combination of reachable stops from the resulting lists.

The rate at which the cost converges to optimal can be improved by using additional pruning rules,

Algorithm 1 Branch and Bound (BnB) for fill scheduling with queuing

Precondition: k: number of robots, G: graph of field plot, T_d: resource capacity, T_r: resource refill time, $compTime$: computation time budget, n: cardinality of robot graph segments

function BnB(k, G, T_d, T_r)
    root $\leftarrow InitialiseTree()$
    optCost $\leftarrow CalculateLowerBound(root, 0, G, k)$ ▷ Lower bound of root is global optimum
    bestUpperBound, candidatePath $\leftarrow CalculateUpperBound(root)$
    unexplored $\leftarrow AddChildren(root, T_d, T_r, k, n)$ ▷ unexplored is a stack
    startTime, currTime $\leftarrow GetSystemTime()$
    while $Length(unexplored) > 0$ and $compTime - (startTime - currTime) > 0$ **do**
        currTime $\leftarrow GetSystemTime()$
        node $\leftarrow unexplored.pop()$
        nodeCost $\leftarrow \sum_{j=1}^{k} (coverageCost(j) + refillCost(j)) + queuing(node)$
        lowerBound, lowerPaths $\leftarrow CalculateLowerBound(node, nodeCost, G, k)$ ▷ Lower bound
        if $lowerBound \leq bestUpperBound$ **then** ▷ Branch
            upperBound, upperPath $\leftarrow CalculateUpperBound(node, lowerPaths)$ ▷ Upper bound
            if $upperBound \leq bestUpperBound$ **then**
                BestUpperBound,candidatePath $\leftarrow$ UpperBound, UpperPath
            end if
            children $\leftarrow addChildren(node, T_d, T_r, k, n)$
            unexplored $\leftarrow$ children
        else
            $Prune(node)$ ▷ prune sub tree
        end if
    end while
    approxRatio $\leftarrow \frac{bestUpperBound - optCost}{optCost}$
    return bestUpperBound, candidatePath, approxRatio
end function

Algorithm 2 Calculating bounds for a BnB node

function CALCULATELOWERBOUND($node, nodeCost, G, k$)
    lowerPaths ← []
    costs ← []
    for $j \leftarrow 1$ to k **do**
        lowerPaths[j],costs[j] ← $SingleRobotDP(node, G, j)$
    end for
    lowerBound ← $nodeCost + \sum_{j=1}^{k} costs[j]$
    return lowerBound, lowerPaths
end function
function CALCULATEUPPERBOUND($nodeCost, lowerBoundPaths$)
    upperPath ← $makeJointPath(lowerBoundPaths)$
    lowerBound ← $nodeCost + CalculateCostOfPath(upperPath)$
    return upperBound, UpperPath
end function

Algorithm 3 Adding childBnB node

function ADDCHILDREN($node, T_d, T_r, k, n$)
    children ← []
    reachable ← []
    for $j \leftarrow 1$ to k **do**
        for $i \leftarrow node_i^j$ to n **do**
            if $T_d \geq T_r \cdot (1 - resourceUse(node_i^j, v_i^j))$ **then** ▷ reachable refill stops
                reachable[j] ← v_i^j
            end if
        end for
    end for
    for $b \in C(reachable, k)$ **do** ▷ combinations of reachable vertices for k robots
        children ← b
        node.children ← b
    end for
    return children
end function

which are possible due our formulation of the bounds. Additional pruning can be performed if the upper and lower bounds of a node are equal; there is no further benefit that can be found by branching on this solution. If the bounds are not equal, there still may be an opportunity to prune. For a given node b, we can use the upper and lower bounds to infer if queuing occurs in the unexpanded section of the partial schedule. In other words, if UB_b minus the cost of P_b is equal to OPT_b, then no queuing occurs in the optimal expansion of the schedule and it can be pruned safely.

The convergence rate can also be increased heuristically by exploring high-quality candidate branches first. High-quality candidates are schedules that do not deviate far from the lower bound schedule, measured by the path distance between the next chosen refill stop and the lower bound schedule. Recall that the cost function is a sum of refilling and queuing costs. Deviating from the lower bound schedule will incur an increase in refilling path cost, and the total cost will be reduced only if there is an equal or greater reduction in queuing cost. Due to this property our algorithm first evaluates branch nodes that lie within some deviation from the next optimal stop given by the lower bound schedule computed from the parent node. Otherwise, the algorithm evaluates potential improvements ordered by greatest amount of resource usage.

6. Experiments and Results

In this section we report experimental results in simulation that validate the behaviour of our algorithms and evaluate their performance. We explain our experimental setup, present results for both algorithms, and conclude with a sensitivity analysis that shows that our algorithm is reasonably robust to errors in estimation of spray rate.

6.1. Experimental Setup

To validate the performance of our algorithms, we perform extensive experiments in simulation. Here, simulation requires *modelling* (creating digital versions of) both the environment and the robots. Our simulated environments consist of five field plots, based on real-world field plot geometry and given refill station locations, from a farm in Queensland, Australia. The five field plots span an area of roughly 1000 hectares in total; these field plots are shown in Fig. 3.

Coverage paths input to our algorithm can be generated by any coverage algorithm. Here we simply generated a *boustrophedon* or "lawnmower"-style path in the tradition of Choset et al. (2005). This path (G) was partitioned into approximately equi-distance segments (V^j) and allocated to the robots. Example coverage paths for three of the field plots are shown in Fig. 4. The coverage paths were generated for ten robots at eight-meter row spacing.

For the simulated robot models, we assume that robots are performing weed control using sprayed herbicide or slurry spreading and that fuel is not a limiting factor. We thus ignore fuel and consider

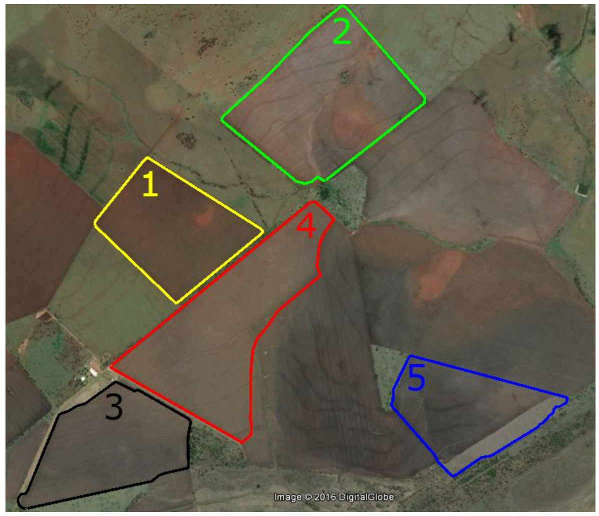
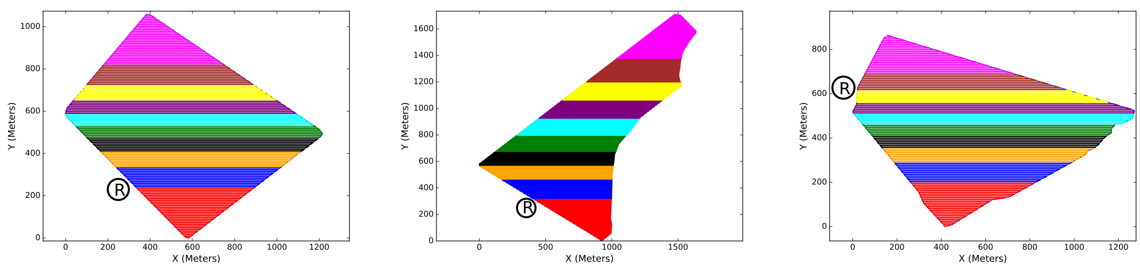


Figure 3: Overhead view of the field plots, in which the geometry and their relative sizes can be seen. Each field plot is assigned a number and colour: 1) yellow, 2) green, 3) black, 4) red, 5) blue.



(a) Field plot 1 (132 rows) (b) Field plot 4 (213 rows) (c) Field plot 5 (107 rows)

Figure 4: Example coverage paths for 10 spot spraying robots at eight-meter row spacing. The refill station is shown as the R inside a circle, and each robot path is shown as a different colour. Allocating equal length paths results in varying numbers of rows for each robot, which can be seen as the varying sizes of the path segments in the y dimension.

three operational cases: 1) spot spraying 2) broadcast spraying and 3) slurry application. Spot spraying is typified by small resource tank capacity and low usage rate. Broadcast spraying requires large tank capacity and uses resources at a higher rate. Slurry application is a bulk process and requires very large tank capacities and uses resources at a rapid rate. For the spot spraying robot model we base the parameters on a multi-robot system that is operating commercially in agriculture (Swarmfarm Robotics). To model the broadcast case we assume a commercially available boom sprayer, such as the Pegasus 6000 (Pegasus Boomsprays), with high-flow spray nozzles and a high-flow refill pump. The slurry spreading robot model is based on a commercial slurry spreader such as the Vredo VT 4556 (Vredo VT 4556). The parameters of the robot models are given in Tab. 1. To model spraying we chose to calculate a constant usage rate per linear meter traveled based on the area covered per

Parameter	Spot spray	Broacast	Slurry application	units
Resource Tank capacity T_d	400	6000	16000	Litres
Area covered per tank	20	75	1	Hectares
Travel speed	8	15	20	Km/h
Tank refill time T_r	6	10	10	Minutes
Row Spacing	8	36	20	Meters
Maximum effective team size k_{max}	33	10	2	Robots

Table 1: Robot model parameters

tank and the row spacing (and hence robot width). Spraying material is assumed to be used at this rate for both spraying cases regardless of the number or presence of weeds. The slurry application rate is also assumed to be constant.

Further, we assume the following: the robots are homogeneous, there are no collisions during travel, resources are used at a constant rate, the resource application rate is known accurately, and robots travel at a constant velocity. The strongest of these assumptions is that the resource application rate for spot spraying is known accurately. We investigate the practical effects of this assumption later in Sec. 6.4, and establish that algorithm performance is acceptable given reasonable inaccuracy in the resource application rate estimate.

In spot spraying and broadcast spraying, the graph vertices (potential stopping points) are located at the ends of each row of the field plot. Travel to/from the refill station is restricted to the road network to avoid unnecessary row traversal and limit soil compaction.

The slurry application required denser placement of potential stopping points to allow for the high rate of resource usage; it may not be possible to traverse a single row without refilling. Therefore, vertices were added along the rows at 250-meter intervals.

The hardware used to perform the experimental evaluation is a desktop computer with an i7-6700 CPU and 32 Gb system memory, running a 64-bit Ubuntu 15.10 operating system. The software was written in 64-bit Python 2.7.

6.2. Experiments with the Exact Algorithm

To analyse the dynamic programming formulation given in Sec. 4, we compare its results with those of a naive distance-based greedy heuristic approach. A greedy algorithm was selected because it produces valid schedules and is an intuitive, simple solution to the problem. The greedy algorithm represents the scenario where each robot drives until its resource tank is empty and does not consider refilling costs or the effects of queuing. This is the typical approach used in current practice.

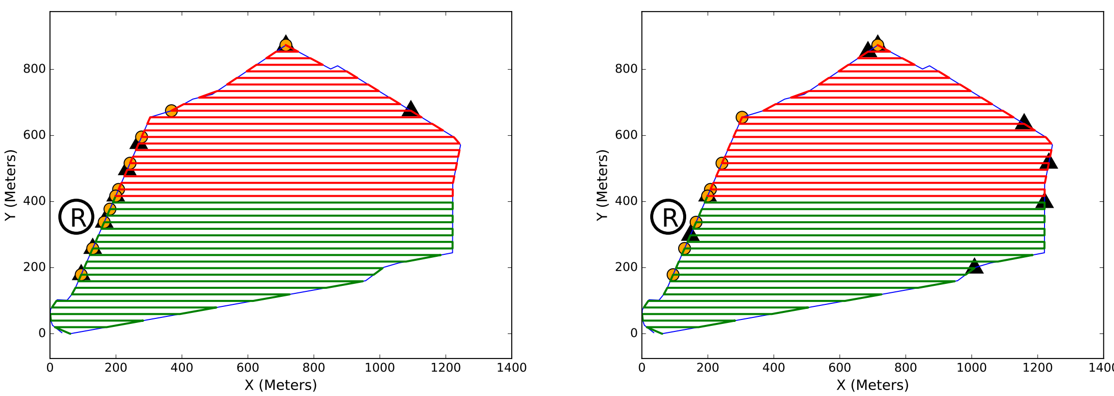


Figure 5: Example sub-optimal and unstable refilling choices of the greedy algorithm. A slight perturbation in the problem instance (in this case robot tank capacity) can negatively impact the algorithm's performance. This impact can be arbitrarily bad as it depends on the refill travel distance, queuing costs and the number of robots. The refill station is shown as the R inside a circle, and each robot path is shown as a different colour. The DP solution is shown as circles and greedy as triangles.

The greedy algorithm was found to produce sub-optimal results with highly variable quality. This variability is highlighted in Fig. 5, where a small perturbation in spray rate affected the greedy solution cost by over 10%. This figure also demonstrates a cause of the sub-optimality; the greedy algorithm can choose a number of refills far away from the refill station and incurs a high cost for those refills in contrast to the DP algorithm, which tends to refill closer to the refill station. For some problems the optimal solution can require a long travel distance to avoid queuing, but since the greedy solution does not consider these effects, its results can be arbitrarily poor.

The DP formulation was capable of solving the problem optimally for small problem instances. DP was found to be impractical for problems with more than 10-20 rows and 4 robots due to excessive system memory and computation time requirements. The exponential growth in computation time limits the practicality of the DP approach to small instances such as those with 2-3 robots and under 50 rows. Average computation times and solution quality is given in Tab. 2. Intuitively, it would seem that the DP algorithm should be capable of dealing with larger problem instances, because not all stops are reachable from each other. This sparse reachability means that not every possible permutation of refill stops needs to be computed. A smaller search space would imply that the running time would be favourable, compared to the worst case complexity bounds. In practice this effect is dominated by the combinatorial explosion of the search space due to the number of robots. This domination can be seen by observing in Tab. 2 that the running time is more strongly affected by an increase in the number of robots compared to an increase in the number of rows.

	20 rows		30 rows	
Number of robots	**Greedy**	**DP**	**Greedy**	**DP**
1	0.000053	0.00087	0.000045	0.0092
2	0.0003	0.034	0.00079	0.15
3	0.00081	383.95	0.0031	962.1
4	0.0023	1043.2	0.004	4278.3

Table 2: The computation time (in seconds) of the greedy and dynamic programming algorithms on field plot 3. The greedy algorithm completes in polynomial time, compared to the DP algorithm which runs in exponential time. The addition of robots has a stronger effect on computation time than the addition of rows.

6.3. Experiments with the Branch and Bound Algorithm

For the following set of experiments our BnB algorithm was given a computation time budget of one second. The approximation quality may be improved with longer computation time, but these results still demonstrate that even with short computation time our approach is effective. Later in the section we investigate the effect of computation time on BnB approximation quality.

The colours used in the figures in this section correspond to the field plots as shown in Fig. 3. The approximation factor is calculated as the ratio of the slack between the upper and lower bounds scaled by the lower bound. Formally, the approximation factor is $(UB - LB)/LB$.

For spot spraying robots our BnB algorithm achieves a near-optimal result. The quality of these results is shown in Fig. 6a. For all experiments the solution cost is within 35 percent of optimal (6 percent on average), and achieves a 4-40 percent reduction in cost compared to greedy (13 percent on average), even for the worst performing field plot. Interestingly, the algorithm continued to produce high quality results for a fixed computation budget, despite the growth in number of robots.

For the broadcast spraying case, our algorithm achieves performance increases over greedy, as shown in Fig. 6b. In all experiments the solution cost is within 80 percent of optimal (29 percent on average), and achieves a 6-50 percent reduction in cost compared to greedy (22 percent on average).

Results for the slurry application are shown in Fig. 7a. The effect of queueing is severe because the system requires a large number of refills. The performance of our algorithm approaches that of the greedy approach because the frequent refills severely restrict the number of potential waypoint selections. Multiple refill stations may be necessary to reduce queuing in this class of problem instances.

To understand how system throughput scales with additional robots, we measured work time (the maximum single-robot cost) for teams of varying size. Increasing the number of robots led to

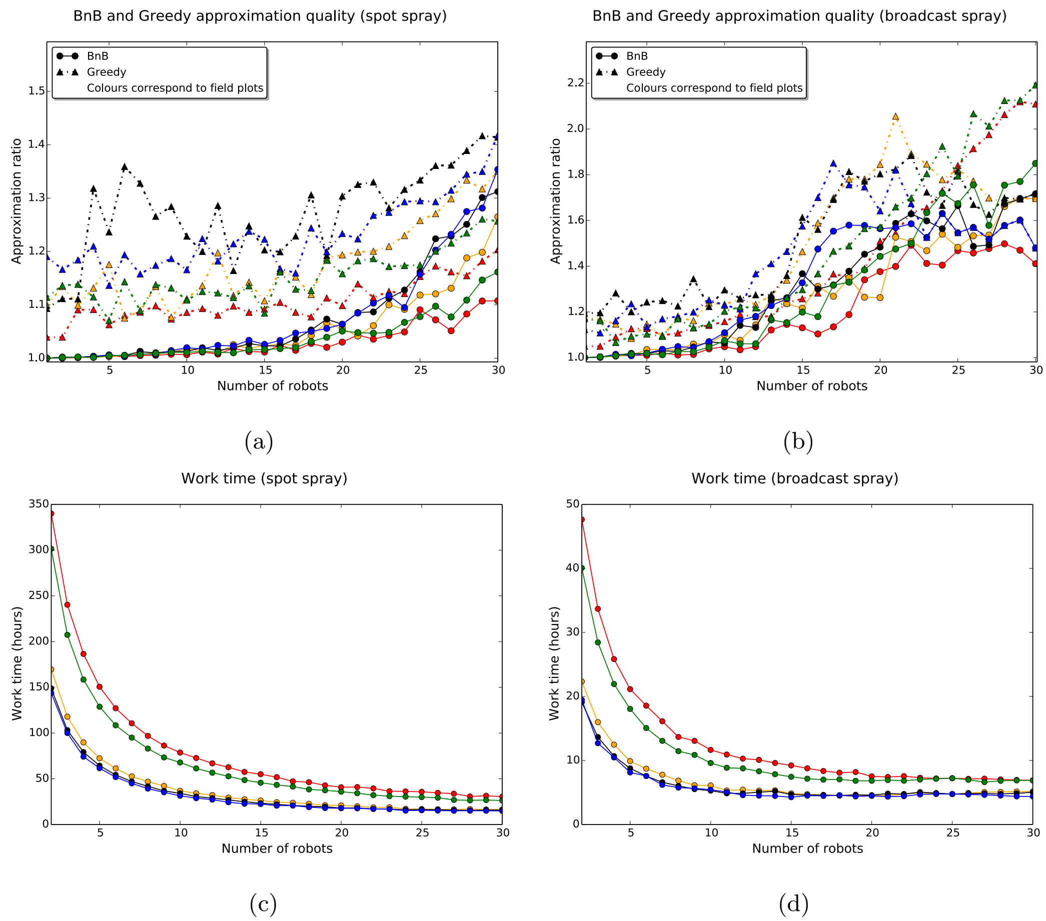
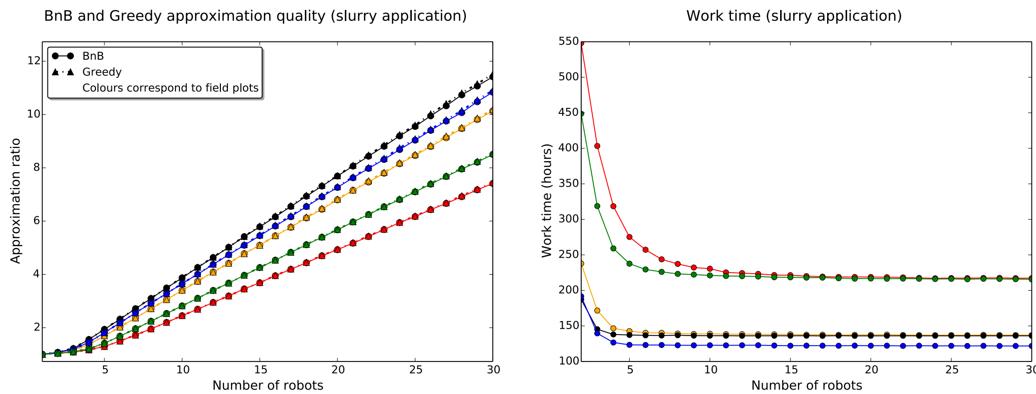


Figure 6: (a) shows that the BnB algorithm produces an average approximation ratio of 1.06 for a team of spot spraying robots. (b) shows that the BnB algorithm produces an average approximation ratio of 1.28 for a team of broadcast spraying robots. (c) shows how the number of robots affects the work time for a team of spot spraying robots. (d) shows how the number of robots affects the work time for a team of broadcast spraying robots. Work time is defined as the maximum single-robot cost, which indicates the completion time of the entire robot team.

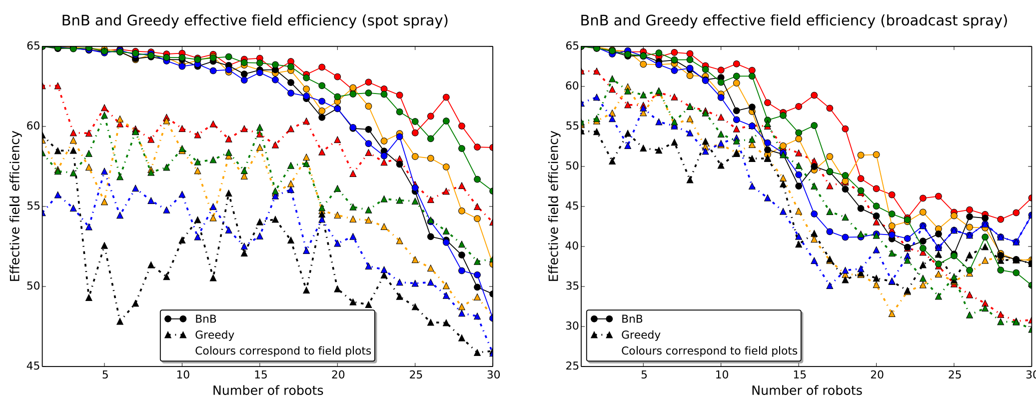
diminishing benefits for all three system types: spot spraying shown in Fig. 6c, broadcast spraying shown in Fig. 6d, and slurry application shown in Fig. 7b. This effect is due to the increase in time spent queuing and is expected; in Sec. 3.2 we proved that there is a limit on the maximum number of robots that can work together effectively, and therefore the benefit of adding robots eventually reaches zero.

To understand how the algorithms can affect the performance of real world systems we use the measure of effective field efficiency. We calculate the effective field efficiency as the mean field efficiency for boom spraying systems given in ASAE D497.7 (2011) divided by the average approximation ratio. An approximation ratio of 1 means the system performs optimally and would result in the ideal effective efficiency. Larger approximation ratios mean the system result in lower efficiency due to more than



(a) (b)

Figure 7: Effects of number of robots on slurry application scenarios. Both the BnB and greedy algorithms perform poorly in this scenario. The approximation ratio is large due to the large amount of queuing.



(a) (b)

Figure 8: Effects of number of robots on effective field efficiency. Out of a best case of 65% field efficiency, our BnB algorithm achieves an average effective field efficiency of 61.6% for spot spraying and 50.4% for broadcast spraying.

the ideal amount of time spent refilling or queuing. For spot spraying systems, shown in Fig. 8a, our BnB algorithm has an average effective field efficiency of 61.6%, compared to 54.7% for the greedy algorithm, resulting in an improvement of 6.9% in effective field efficiency over the greedy approach. Similarly, for broadcast spraying systems, shown in Fig. 8b, the average effective efficiency is 50.4% for the BnB algorithm compared to 44.2% for greedy, an improvement of 6.2% in effective field efficiency over the greedy approach. For slurry applications our algorithm performed the same as the greedy approach hence there is no change in field efficiency between the two algorithms.

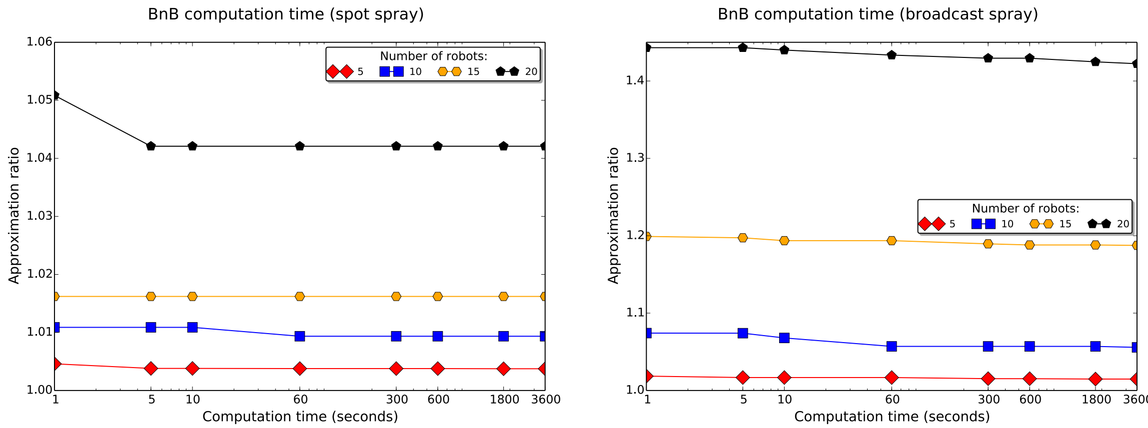


Figure 9: Effect of increasing computation time on BnB approximation. Additional computation time non-linearly improves the approximation ratio. Improvement is slow because the search space is large and the algorithm has to find solutions that reduce queuing costs without increasing travel costs.

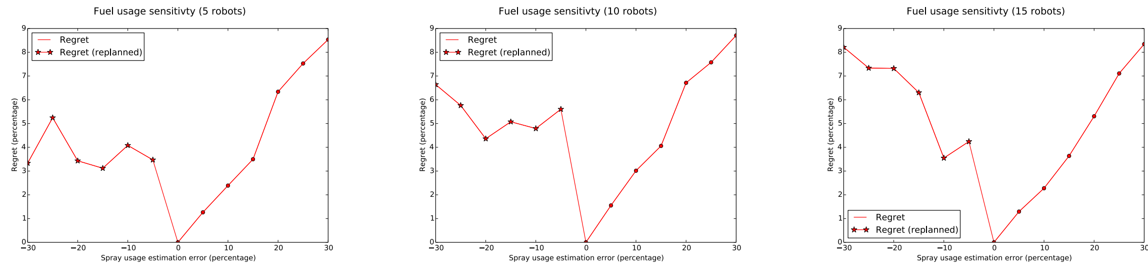
6.3.1. Effect of Computation Time

To examine the effect of the BnB computation time on solution quality, we allowed the algorithm to run for increasing lengths of time and observed the approximation quality. These computation time experiments are run on field plot 2. Slurry application systems exhibited a lack of flexibility in schedule selection, hence we focus our analysis on spot spraying and broadcast systems where there is more opportunity for further schedule optimisation.

Additional computation time improves the approximation in a non-linear fashion. It can be seen in Fig. 9 that for most scenarios, with additional computation, the BnB algorithm improves the approximation ratio and produces lower cost schedules. The improvements generally increase in magnitude as the number of robots increases, because the effect of queuing becomes more exaggerated, and there is more potential for small changes to have a cascade effect (one robot forces another to queue, which causes more queuing, and so on). However, a larger number of robots also results in an exponentially larger search space so improvements can take significantly longer to find.

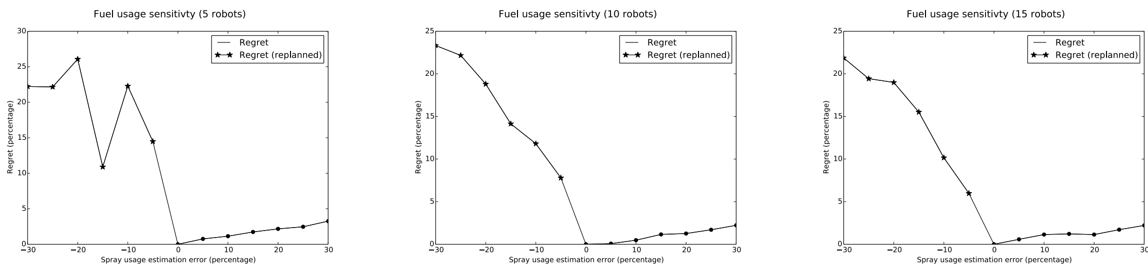
6.4. Sensitivity Analysis

One potential limitation of our work is that it can be difficult to predict spray rate, and thus it is important to understand how our algorithms break down when faced with errors in spray rate estimation. Broadcast spraying typically involves a constant chemical application rate target, but the spot spraying case is more challenging. Unlike broadcast, it is difficult to accurately estimate the amount of liquid applied per unit area because it is variable. One such example would be field plots with higher weed density than expected (underestimating the usage rate), or lower weed density (overestimating the usage rate).



(a) 5 robot sensitivity (b) 10 robot sensitivity (c) 15 robot sensitivity

Figure 10: Regret caused by incorrectly estimating resource usage rate for 5, 10, 15 robots (field plot 4).



(a) 5 robot sensitivity (b) 10 robot sensitivity (c) 15 robot sensitivity

Figure 11: Regret caused by incorrectly estimating resource usage rate for 5, 10, 15 robots (field plot 3). Overestimation has a lower regret than underestimation which provides insight about how to estimate usage rates when considering uncertainty.

To analyse the potential operational impact of this assumption, we investigate the sensitivity of the schedule to the estimation error of the spray usage rate. We measure this sensitivity using *regret*. In this case regret is defined as the extra cost incurred due to error in estimating the spray rate, either by: 1) overestimation causing a sub-optimal schedule, or 2) underestimation invalidating schedules and requiring replanning to fix. Formally, to calculate regret, let the spray rate estimation error Δ be the difference between the estimated spray rate and the actual spray rate. Let E_0 be the cost of a schedule using the estimated (expected) spray rate and let I_Δ be the ideal cost of a schedule using the real spray rate calculated as if there is no estimation error. These ideal and estimated costs are given by the lower bound of the BnB root, which allows us to bound the regret and account for variability in feasible solution costs (due to approximation).

Let A_Δ be the cost of a schedule that was computed using an estimated spray rate, but evaluated using the actual spray rate. If the schedule remains valid, this cost is equivalent to the BnB solution's upper bound. If the schedule is no longer valid, due to exhausting the spray resource earlier than estimated, it needs to be modified. Schedules are modified using a reactive greedy strategy. This strategy chooses to refill at the last possible stop before the spray resource is exhausted, resulting in a valid schedule. Let the offset from 0 be $\delta = \frac{A_0 - I_0}{E_s} \times 100$. Regret for a given estimation error Δ as a percentage of the estimated schedule cost is then calculated as:

$$\left(\frac{A_\Delta - I_\Delta}{E_0} \times 100\right) - \delta.$$

It can be seen from Fig. 11 and Fig. 10 that the solution is far more sensitive to underestimation than to overestimation. The reactive greedy strategy can lead to large regret, because the new schedules have higher refill counts and sub-optimal refill stop selections. Overestimation can quickly invalidate schedules, because the schedules tend to minimise the number of refills and hence maximise the amount of resource used. Alternatively the system seems fairly robust to overestimation, because the schedules are still valid. Schedules with overestimation still have some associated regret because they refill more frequently than necessary.

This sensitivity analysis suggests guidelines to be used by practitioners. Any inaccuracy in estimation sacrifices optimality, but overestimation is preferable to underestimation and the algorithm is not particularly sensitive to the magnitude of overestimation. Results for underestimation are unpredictable and may lead to large variance in run time (biased towards larger run times), because the algorithm behaviour is forced to be reactive, which devolves into a greedy replanning strategy. This analysis supports the use of our algorithm in practice because, with reasonably accurate rate estimation (up to 30% error underestimation and 5% overestimation error), even in the unfavorable scenarios our algorithm outperforms (or devolves to) a reactive greedy approach.

7. Discussion

In this section we discuss the practical implications of our work. We discuss the effect of reducing travel and queuing time for practical systems, capability to inform choice of robot team size, performance in spot spraying versus broadcast spraying, alternative spray tank finishing capacity constraints, the potential for using our algorithm to position the refill station, and future work.

The problem analysis and solutions presented here have strong potential to be useful in practice. The refill scheduling problem affects both teams of traditional vehicles and multi-robot systems in a breadth of circumstances including spraying, cargo delivery, and other tasks that involve replenishing resources at a shared location. Improving their refilling efficiency has appreciable positive benefits. The spraying operations (in this work) improve the field efficiency, by 6.9% for spot spraying and 6.2% for broadcast spraying, compared to the typical approach. Reducing the refilling costs allows the robots to spend less time transiting to/from the refill station and queuing, thus improving system throughput and the amount of fuel used. Both benefits result in lower operational costs and improve the environmental impact of spraying systems.

Another practical outcome of this work is that it informs the choice of *how many* robots to use for a given scenario. Our results show that, in practice, as the size of the robot team grows the queuing cost starts to the dominate the benefit of increasing the robot team size, and the throughput is limited. Hence, there is an ideal number of robots that should be deployed for a given problem, due to a growth in associated operational overhead and a reduction in marginal utility (due to queuing costs). Our algorithm makes it possible to easily explore the expected performance of teams of various sizes, and potentially can be used as an operational design tool.

A useful benefit of our approach is that it can be used to understand the scalability of multi-robot broadcast and spot spraying systems. The results show that the marginal benefit of adding more robots to a broadcast spraying system is less than that of spot spraying system. The reason is that broadcast systems have a lower work time/refill time ratio. A low ratio leads to sizable queuing periods, and this effect is worsened by the addition of more robots. The implication for broadcast systems is that adding more robots requires either a higher work/refill time ratio or additional refill stations.

Another area for consideration when applying our work is the amount of spraying material a robot should have left in the tank at the end of an operation. This requirement is dependent on the particular operation required. For our problem formulation we defined for the robot to finish with a full tank, such as would occur in a contract spraying scenario. Contract operators may often want the system to end with a full tank, in order to know exactly how much chemical was used (which is billed to the client). The robots start and end full, and then the amounts added at each refill (measured at the pump) is summed to compute the total chemical bill. Alternatively, there can be operations that would prefer the system to finish empty or with an arbitrary capacity. A scenario that motivates finishing with an arbitrary tank capacity is an operation where the spray tanks need to be flushed and the chemicals changed between field plots. To handle the arbitrary finishing tank capacity our problem formulation needs to be modified slightly. The problem needs to be constructed such that $r_n^j = 0$ and $f(i, n) = 0$ this means the final refill will have no time cost and is effectively ignored, thus allowing the system to finish with an arbitrary resource amount. Subsequently, there needs to be a separate check to ensure that the resource used to reach the final vertex does not exceed the tank capacity constraint, since the cost is set to 0. These modifications would allow the system to handle different operational realities.

Lastly, our approach can be used to inform the choice of where to place the refill station. It is sufficient to run the algorithm iteratively for various placements and choose the best result. This process is feasible because the algorithm is anytime and offers reasonable approximation quality. More interestingly, with a straightforward extension to the algorithm this process can also be used to consider placement of more than one refill station. A simple scenario is one fixed station plus one mobile station that can be towed into a chosen position. Adding further refill stations can reduce travel and queuing time, but also increases the complexity of the decision problem exponentially.

Multi-robot refill scheduling is a rich problem area with many important avenues of future work. Problem variants that could lead to reduced work time include those that consider multiple refill stations, mobile refill stations, and partial refilling. It would also be interesting to understand the effects of different coverage patterns (such as row interleaving) on travel/queuing time. Promising approaches to improve stability include probabilistic methods that deal with uncertainty in spray rate estimation, and replanning.

Areas for future improvement that instead focus on improving practicality include parallelising the algorithm, and producing a cloud based service. Our work focused on the characterisation of the problem and design of the approach, not the optimisation of the implementation. The design of our BnB algorithm lends itself to a parallelised approach, which would lead to speed increases proportional to the amount of hardware used. Lastly, establishing a cloud based service would allow agricultural system operators to more easily access the benefits of our results and promote wider adoption.

8. Conclusion

In this work, we characterised and provided solutions for the multi-robot refill scheduling problem with queuing. We defined the problem by constructing a subset selection problem with a non-linear cost metric. Also we proved that the problem is NP-hard and that there is a bound on the number of robots that can work effectively. We designed an algorithm based on dynamic programming that is capable of solving the problem optimally, but due to exponential complexity its practicality is limited to small problem instances. Realistic instances can be solved quickly and approximately using our anytime branch and bound algorithm. This anytime property allows BnB algorithm to be terminated at any time and to provided a feasible and valid solution for the problem, which is important for real-time systems. Another benefit is the bounds of the BnB algorithm provide information about the approximation quality and were designed to be computed in polynomial time. We tested our BnB algorithm on a range of simulated of real world agricultural applications, from small dose rate spot spraying to high dose rate slurry application and the results show functionality and applicability of the algorithm for the full range of agricultural operations. We also show empirically that our BnB algorithm produces quality approximately optimal solutions and out performs the typical greedy approach used in practice. The strongest assumption made by our algorithm is that the usage rate is known. It is difficult to accurately estimate the spray rate for spot spraying, but our sensitivity analysis showed that our algorithm's performance is reasonably robust to inaccurate estimates.

ASAE D497.7, 2011. Agricultural Machinery Management Data. ASABE American Society of Agricultural and Biological Engineers. St. Joseph, MI 49085-9659 USA.

Best, G., Fitch, R., 2016. Probabilistic maximum set cover with path constraints for informative path planning, in: Proc. of ARAA Australasian Conference on Robotics and Automation.

Binney, J., Sukhatme, G.S., 2012. Branch and bound for informative path planning, in: Proc. of IEEE International Conference on Robotics and Automation, pp. 2147–2154.

Bochtis, D.D., Sorensen, C.G., 2009. The vehicle routing problem in field logistics part I. Biosystems Engineering 104, 447–457.

Bochtis, D.D., Sorensen, C.G., 2010. The vehicle routing problem in field logistics: Part II. Biosystems Engineering 105, 180 – 188.

Bruglieri, M., Pezzella, F., Pisacane, O., Suraci, S., 2015. A variable neighborhood search branching for the electric vehicle routing problem with time windows. Electronic Notes in Discrete Mathematics 47, 221–228.

Chen, B., Potts, C.N., Woeginger, G.J., 1998. A review of machine scheduling: Complexity, algorithms and approximability, in: Handbook of combinatorial optimization. Springer, pp. 1493–1641.

Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W., Kavraki, L.E., Thrun, S., 2005. Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, Cambridge, MA.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. Introduction to algorithms. MIT press Cambridge.

Edwards, G., Jensen, M.A.F., Bochtis, D.D., 2015. Coverage planning for capacitated field operations under spatial variability. International Journal of Sustainable Agricultural Management and Informatics 1, 120–129.

Galceran, E., Carreras, M., 2013. A survey on coverage path planning for robotics. Robotics and Autonomous Systems 61, 1258–1276.

Hameed, I.A., 2014. Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain. Journal of Intelligent & Robotic Systems 74, 965–983.

Jensen, M.F., Bochtis, D., Sorensen, C.G., 2015a. Coverage planning for capacitated field operations, part II: Optimisation. Biosystems Engineering 139, 149 – 164.

Jensen, M.F., Norremark, M., Busato, P., Sorensen, C.G., Bochtis, D., 2015b. Coverage planning for capacitated field operations, part I: Task decomposition. Biosystems Engineering 139, 136 – 148.

Jin, J., Tang, L., 2011. Coverage path planning on three-dimensional terrain for arable farming. Journal of Field Robotics 28, 424–440.

Keil, J.M., 1992. On the complexity of scheduling tasks with discrete starting times. Operations Research Letters 12, 293–295.

Keskin, M., Catay, B., 2016. Partial recharge strategies for the electric vehicle routing problem with time windows. Transportation Research Part C: Emerging Technologies 65, 111–127.

Leung, J.Y., 2004. Handbook of scheduling: algorithms, models, and performance analysis. CRC Press.

Lin, S.H., Gertsch, N., Russell, J.R., 2007. A linear-time algorithm for finding optimal vehicle refueling policies. Operations Research Letters 35, 290 – 296.

Nam, C., Shell, D.A., 2015. Assignment algorithms for modeling resource contention in multirobot task allocation. IEEE Transactions on Automation Science and Engineering 12, 889–900.

Oksanen, T., Visala, A., 2009. Coverage path planning algorithms for agricultural field machines. Journal of Field Robotics 26, 651–668.

Patten, T., Fitch, R., Sukkarieh, S., 2016. Multi-robot coverage planning with resource constraints for horticulture applications, in: Acta Horticulturae, pp. 655–662.

Pegasus Boomsprays, . `http://croplands.com.au/Products/Trailed-Boomsprays/Pegasus`. Accessed: 2017-04-05.

Rekleitis, I., New, A.P., Rankin, E.S., Choset, H., 2008. Efficient boustrophedon multi-robot coverage: an algorithmic approach. Annals of Mathematics and Artificial Intelligence 52, 109–142.

Richards, D., Patten, T., Fitch, R., Ball, D., Sukkarieh, S., 2015. User interface and coverage planner for agricultural robotics, in: Proc. of ARAA Australasian Conference on Robotics and Automation.

Schneider, M., Stenger, A., Goeke, D., 2014. The electric vehicle-routing problem with time windows and recharging stations. Transportation Science 48, 500–520.

Suzuki, Y., 2014. A variable-reduction technique for the fixed-route vehicle-refueling problem. Computers & Industrial Engineering 67, 204–215.

Swarmfarm Robotics, . `http://www.swarmfarm.com/`. Accessed: 2017-04-05.

Toth, P., Vigo, D., 2002. Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics 123, 487 – 512.

Vredo VT 4556, . https://www.vredo.com/en/products/self-propelled-tracs/vt4556/. Accessed: 2017-22-05.