Multi-Robot Routing Algorithms for Robots Operating in Vineyards

Thomas C. Thayer *Student Member, IEEE,* Stavros Vougioukas *Senior Member, IEEE,*
Ken Goldberg *Fellow, IEEE,* Stefano Carpin *Senior Member, IEEE*

***Abstract*—We consider the problem of multi-robot routing in vineyards, a task motivated by our ongoing project aiming at creating a co-robotic system to implement precision irrigation on large scale commercial vineyards. The problem is related to a combinatorial optimization problem on graphs called "team orienteering". Team orienteering is known to be NP-hard, thus motivating the development of heuristic solutions that can scale to large problem instances. We propose three different parameter-free approaches informed by the domain we consider, and compare them against a general purpose heuristic formerly developed. In numerous benchmarks derived from data gathered in a commercial vineyard, we demonstrate that our solutions outperform the general purpose heuristic and are scalable, thus allowing us to solve instances with tens of thousands of vertices in the graphs.**

***Note to Practitioners:* Routing problems with budget and motion constraints are pervasive to many applications. In particular, the structural constraints considered in this problem are found not only in agricultural environments, but also in warehouse logistics and other domains where goods are arranged along regular linear structures. This paper proposes and analyzes algorithms that can be applied when multiple agents must be coordinated in these environments. In particular, by utilizing domain specific knowledge, the solutions proposed in this work outperform general purpose approaches that poorly scale with the size of the environment. The algorithms we present also ensure that no collisions occur between robots – an aspect normally neglected in algorithms formerly proposed to solve the team orienteering problem.**

***Index Terms*—Precision agriculture; multi-agent coordination; team orienteering.**

I. Introduction

Wine grapes are ideally grown following a *stress irrigation* regime, i.e., each vine receives a limited amount of water to bolster sugar content and emphasize flavonoids. Most irrigation systems in use today lack the ability to adjust water delivery at a fine grain level, e.g., on a per vine basis or based on small zones. While water stress is desirable for vines, over-stressing vines may lead to inferior yield and even to vine death. Therefore growers tend to over-irrigate avoiding potential losses. Delivering "the right amount of water" is the main objective of precision irrigation and remains an open challenge, especially in large-scale commercial vineyards. This problem is particularly acute in California, where wine grape production has a significant economic impact, but freshwater availability is limited. Recent multiyear droughts have made the situation even worse, especially when considering competition for water use in other high-value crops. The US - where freshwater usage in agriculture is estimated to be 85% - is not facing this problem alone; worldwide usage of managed freshwater in agriculture is estimated to be around 70% [1], making agricultural water conservation a major global challenge. If we look at the irrigation infrastructure from a control standpoint, we find an abundance of data gathered from vineyards through remote or in-site sensing, but the ability to modify the inputs to the system (i.e., water) is very limited. Equipping each vine (or small groups of vines) with an electrically actuated variable-rate emitter is prohibitively expensive, and unsuited for extended operations in adversarial environmental conditions. To mitigate this problem, passive, variable-rate emitters could be used, but these must be adjusted manually by workers. Due to the sheer size of typical vineyards and the ever increasing labor shortage in the agriculture industry, this approach does not scale and is economically infeasible.

In development by the University of California, RAPID (Robot Assisted Precision Irrigation Delivery) is a scalable irrigation management solution that aims to assist vine growers with water conservation efforts while preserving yield and quality. The objective of RAPID is to create a co-robot system with fleets of robots navigating through vineyards to adjust passive emitters delivering the appropriate amount of water to each vine. In [2] we presented PEAD (Portable Emitter Actuation Device), i.e., an actuator that can be used to latch and adjust a variable rate emitter, and in [3] we showed how this concept can be extended for mounting on a robotic arm. Ultimately, the robotic arm and actuator will be mounted on a mobile platform moving through a vineyard to perform the required adjustments to each emitters' setting.

The motion of the mobile robot through a vineyard is subject to various constraints. In particular, the robot will not be able to follow a straight line when moving between two arbitrarily chosen emitters, because vines and irrigation lines prevent changing vine rows unless the robot is at either end of the vineyard. This is illustrated in figure 1. Moreover, due to the limited onboard power supply, the robot needs to periodically return to a designated site for a recharge or swap of its batteries. In [4] we showed that this problem is related to the an optimization problem known as *orienteering*, and we proved that it remains NP-hard even when considering the

T.C. Thayer and S. Carpin are with the University of California, Merced, CA, USA. S. Vougioukas is with the University of California, Davis, CA, USA. K. Goldberg is with the University of California, Berkeley, CA, USA.

This material is based upon work that is partially supported by the USDA-NIFA under award number 2017-67021-25925 (NSF National Robotics Initiative). Thomas Thayer was also partially supported by the NSF under grant DGE-1633722. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the USDA and NSF.

Fig. 1: A test platform for identifying challenges in navigation within a vineyard. Irrigation lines and vine stocks block travel between rows, thus creating motion constraints.

special structure of the graph induced by the environment the robot operates in. Then, we proposed two heuristics informed by the domain that outperform standard heuristic methods proposed in the past. In [4] we however considered only the single robot case, whereas a more realistic scenario will have a fleet of robots deployed to expedite the process. In this paper, we therefore extend our former findings considering the case where multiple robots concurrently operate in the same vineyard. This problem is related to a problem known as *team orienteering* and is obviously as hard as the single agent case. From a practical perspective, there is an additional challenge. Because vines are densely grown, each row in the vineyard is narrow, and with multiple robots operating in the same block, one should avoid having two robots traveling at the same time along the same row in opposite directions, as there may not be enough space.[1]

The rest of this paper is organized as follows. Related work is discussed in section II and is followed by the formalization of the problem we consider in section III. Motivated by the intrinsic computational complexity, three different heuristics are presented in section IV, where we also shortly discuss a formerly developed method for this class of problems. Section V compares the different solutions on various problem instances based on data we collected from a commercial vineyard, and shows that our proposed solutions favorably scale with the size of the problem. Finally, conclusions and anticipated future work are presented in section VI.

[1]While at this stage of the project we use a small research robot platform, in a field deployment we anticipate that vehicles with a larger footprint will be used and this aspect will therefore become relevant.

II. Related Work

The reader unfamiliar with the challenges associated with precision irrigation is referred to [5], [6] for general introductions to the topic. In the remainder of this section we focus on computational issues related to the associated orienteering problem and provide selected pointers to the growing sector of robotics and automation in agriculture.

A. The Orienteering Problem

The problem considered in this paper is related to the classic *orienteering* problem (OP) whereby one agent needs to traverse a graph where each vertex has an associated reward and each edge has a defined cost. The objective is to compute a path maximizing the sum of rewards for visited vertices while ensuring that the sum of costs for traversed edges does not exceed a preassigned travel budget. If a vertex is visited multiple times, the associated reward is counted only once, whereas costs on edges traversed multiple times are counted in full. In the rooted version of the problem, starting and ending vertices are defined for the path, while the unrooted version allows the path to start and end anywhere in the graph. The OP was originally introduced in [7] and proven to be NP-hard in [8]. In [9], the authors showed that orienteering belongs in the APX-hard class. There exists numerous variants of the OP, and we point the reader to [10] for a recent survey.

Two main approaches are followed to tackle this problem. The first utilizes exact methods using branch-and-bound or branch-and-cut techniques, but applicability is limited to problem instances with a small number of vertices in the graph, i.e., less than 1,000 [11], [12]. To put this number into context, the problem instances we consider may have 50,000 vertices or more. The second approach aims at developing heuristics informed by domain specific knowledge, which may work well if used in the correct context. These often rely on assumptions made about the metric space to inform the path creation mechanism. Examples include the Center-of-Gravity heuristic [8] and our own Greedy Row and Greedy Partial-Row algorithms [4]. Others rely on iterative and Monte Carlo methods to discover solutions and improve them over time. Many popular heuristics combine multiple techniques, such as the Four-Phase Heuristic [13], in the hopes that the different techniques will help avoid local maxima and steer closer to the global maximum.

Approximation schemes do exist for the OP, but they are cumbersome to implement. For the unrooted version of orienteering, a polylogarithmic run time 2-approximation algorithm was given by [14]. For the rooted OP, [15] proposed the best known approximation algorithm giving a $(2+\epsilon)$-approximation solution with a run time of $n^{\mathcal{O}(1/\epsilon^2)}$, where n is the number of vertices. A $(1+\epsilon)$-approximation scheme for the case of fully connected planar graphs was given in [16], but the specific

requirements limit its usefulness and makes it inapplicable for our purposes.

B. The Team Orienteering Problem

The *team orienteering* problem (TOP) is a variant of the OP where multiple agents cooperate to collect the rewards, while all agents are subject to some individually independent budget constraint [17]. Evidently, this problem is as hard as orienteering, and therefore subject to the same computational challenges, and the cooperative nature of the problem calls for the development of specialized heuristics. Some exact methods have been created for the TOP, such as Branch and Price [18], but they have limited scalability (up to around 500 vertices [10]) which makes them impractical for our use. A literature review did not reveal any approximation techniques for the TOP, and therefore the only known way to achieve guaranteed performance is by using exact algorithms.

Like the OP, many different types of heuristic methods have been created to find solutions for the TOP. Most of these heuristics were developed for the general case and are therefore useful in many circumstances. Some are based on randomization techniques that have seen extensive use in other routing problems, such as Ant Colony Optimization [19] and Simulated Annealing [20], which work by modifying the tours with stochastic changes until improvements can no longer be made. Multi-phase meta-heuristics are ever popular as well, since they combine multiple techniques to build efficient paths. These include Guided Local Search (GLS) [21] and Skewed Variable Neighborhood Search [22], which show adequate performance for small problems but do not scale well to larger instances.

The problem we consider is also related to the multi-robot motion coordination problem, particularly because of our requirement to avoid having two or more robots traversing the same row in opposite direction at the same time. Our approach relies on considering the space/time composition to resolve conflicts, and is related to methods that determine how to schedule multiple robots along a preassigned set of routes [23]. Multi-robot path planning remains intractable even when restricted to problem instances defined on planar graphs, and heuristics are used in this domain as well [24], [25], [26].

C. Robots in Agriculture

Robot use in agriculture is quickly expanding and has a very promising future, with a multitude of diverse applications gaining in popularity [27]. Remote sensing for information gathering is a typical utilization, where Unmanned Aerial Vehicles gather images from the sky of grow sites providing unique insights not visible from the ground [28], [29]. Another growing trend is the deployment of robots (usually on the ground) to capture images of fruit on plants for use in yield estimation [30], [31]. Fruit harvesting [32] and plant pruning [33] are other utilizations of robots that interact directly with the plants themselves. Still, some other robots are specifically built to help with logistic problems, such as moving fruit bins to and from human harvesters within fruit orchards to promote optimal labor time usage [34]. Other robotic applications are built on top of the already mechanized farm processes, such as spraying of pesticides and fertilizer from tractors in minimal travel distance and working time [35]. Regarding irrigation optimization, literature review did not uncover any preceding work related to robotic tools in this domain, except for our own previous work [4].

III. Problem definition

We start with a formal definition of the team orienteering problem. Let $G = (V, E)$ be a complete, undirected graph, let $r : V \to \mathbb{R}_{\geq 0}$ be a reward function defined over the vertices, and let $c : E \to \mathbb{R}_{\geq 0}$ be a cost function defined over the edges. Starting from these two functions, a path in the graph can be associated with a total cost and a total reward. The total cost of a path is the sum of the costs of all edges along the path, while the total reward is the sum of the rewards of all vertices visited by the path. If a path visits the same vertex multiple times, the reward is added just once, whereas the cost of an edge is incurred every time the edge is traversed. For a given integer M (number of team members) and given positive real number T_{MAX} (budget) we want to determine M paths starting and ending at a preassigned vertex $v \in V$ that maximize the sum of the rewards of the paths subject to the two following constraints: 1) each path has cost at most T_{MAX}; 2) if a vertex is visited more than once by different agents, the associated reward is collected only once.

In our recent work [4] we introduced a special class of graphs called *irrigation graphs* that we indicated as $IG(m, n)$, where m and n are the number of rows and columns, respectively. Irrigation graphs are planar graphs of degree at most three that capture the motion constraints defined when a robot navigates in a vineyard. Each vertex in the irrigation graph represents the location of a water emitter (placed in close proximity to a vine), and edges show possible motions between emitters. The structure of the edges models the motion constraints for a robot operating in a vineyard, i.e., row swapping is possible at either end of the vineyard, but not in the middle. Figure 2 shows the structure of these graphs (see also Figure 1 and 3 for its relationship to vineyards structure).

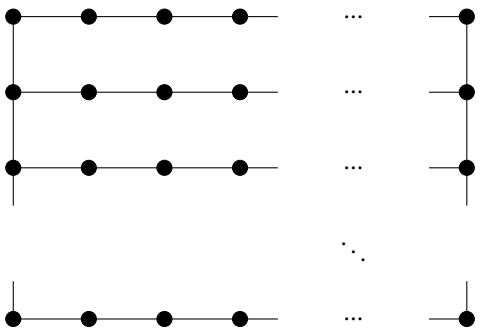

Fig. 2: Structure of an irrigation graph $IG(m, n)$. m is the number of rows, while n is the number of vertices in each row.

An irrigation graph can be formally defined as an undirected graph $IG(m, n) = (V, E)$, with each vertex $v(i, j) \in V$ where

i is an integer $1 \leq i \leq m$, j is an integer $1 \leq j \leq n$, and a set of edges E follows with the following properties:

- Each vertex of type $v(i,j)$ with $1 < j < n$ is connected to vertices $v(i,j-1)$ and $v(i,j+1)$.
- Each vertex of type $v(i,1)$ with $1 < i < m$ is connected to vertices $v(i-1,1)$, $v(i+1,1)$, and $v(i,2)$.
- Each vertex of type $v(i,n)$ with $1 < i < m$ is connected to vertices $v(i-1,n)$, $v(i+1,n)$, and $v(i,n-1)$.
- One edge connects $v(1,1)$ and $v(2,1)$.
- One edge connects $v(m,1)$ and $v(m-1,1)$.
- One edge connects $v(1,n)$ and $v(2,n)$.
- One edge connects $v(m,n)$ and $v(m-1,n)$.

Without loss of generality, we assume $m,n \geq 3$ [2].

An irrigation graph can be augmented to include a cost function $c : E \rightarrow \mathbb{R}_{\geq 0}$ that gives each of its edges a cost $c(e)$ and a reward function $r : V \rightarrow \mathbb{R}_{\geq 0}$ that gives each of its vertices a reward $r(i,j)$. With this augmentation, the OP and TOP can be applied. Heuristics and algorithms for solving orienteering often require complete graphs, and an irrigation graph can obviously be extended into a complete graph by adding the missing edges with costs set equal to the shortest path in the original graph. The basic version of the problem we consider in this paper is the following:

> **Irrigation Graph Team Orienteering Problem (IGTOP)**: Let G be an irrigation graph $IG(m,n)$, $v_1, v_n \in V$ be two of its vertices and c, r be cost and reward functions on G. For a given constant T_{MAX}, find M routes of maximum collective reward starting at v_1 and ending at v_n of cost no greater than T_{MAX}, and such that the reward for visiting a vertex is collected only once if the vertex is visited multiple times.

In [4] we proved that the special case of the IGTOP where $M = 1$ (single agent) and $c(e)$ is equal for all edges is NP-hard. Consequently, the IGTOP is NP-hard as well. This proof is evident by observing that irrigation graphs are $\mathcal{BP}3$, that is they are planar, bipartite (if one partitions the vertices between those for which $i+j$ is even or odd), and have vertices with degree of at most 3. In [36], it was shown that the Hamilton circuit problem for graphs in $\mathcal{BP}3$ is NP-Complete. Using the classic reduction from the Hamilton circuit problem, the decision Traveling Salesman Problem (TSP) on $\mathcal{BP}3$ is also NP-Complete. This is true in the case where all edge costs are 1 and we set the total cost to $T = V$, as well as in the general case. Building an instance of the single agent orienteering problem on a $\mathcal{BP}3$ with reward function $r(v) = 1 \ \forall v \in V$ and constant cost function $c(e) = k \ \forall e \in E$, a solution having total reward $R = |V|$ and total cost $C \leq T_{MAX}$ exists if and only if the answer to the decision TSP on the same graph with total cost $T \leq T_{MAX}$ is yes. Therefore, the constant cost orienteering problem on $\mathcal{BP}3$ is NP-hard. Finally, this is a special case of the version of this problem with non-constant costs and rewards, defined earlier as the IGTOP, and the IGTOP is therefore also NP-hard. Note that the assumption of constant edge costs is motivated by the domain we consider, because vines and water emitters are uniformly spaced, and terrain is flat. Additionally, we make the assumption that all robots are homogeneous; that is, they perform equally well in the same conditions.

[2]The problem we present later on becomes trivial if this is not the case

As mentioned in the introduction, when solving the IGTOP problem there exists the possibility that a solution may cause two or more robots to collide when it is implemented. This is because the spacing of trellises and vines may be too narrow to allow two robots to traverse the same row in opposite directions. Traveling in opposite directions along the vertical columns at either end of the graph is allowed, however, because those sections are typically much larger. The heuristic we will propose, therefore, will not only solve the IGTOP problem defined above, but also ensure that no collisions occur along the horizontal rows. These additional constraints, however, are problem specific and not enforced by the other general purpose heuristics formerly proposed in literature.

IV. Heuristics for the IGTOP Problem

The following contains descriptions of our proposed heuristic algorithms. Note that each of them is parameter-free - they take as input only the original IGTOP problem containing the irrigation graph, cost function, reward function, start vertex, end vertex, and budgets for each agent - and therefore do not require any tuning to produce good results. The algorithm we use to benchmark results against is also described, but unlike our proposed algorithms, it requires tuning parameters.

As previously mentioned. exact solution methods for solving the generic TOP do exist, and these are usually formulated similarly to the linear-integer program given in [10]. In these solvers, the decision variables are a vector of binary values corresponding to edges in the graph, where a value of 1 determines that the edge is included in the route. Our proposed methods, on the other hand, output a sequence of vertices as a route for each agent, where each successive vertex is immediately adjacent to its predecessor in the route.

A. Single Agent Greedy Partial Row Heuristic

The Greedy Partial Row Heuristic algorithm (GPR) we proposed in [4] only solves the single agent case, but is used as a building block for the various domain-specific heuristics we propose to solve the IGTOP problem. We therefore shortly summarize the GPR algorithm in the following, and refer the reader to [4] for a full discussion. Algorithm 1 shows the pseudocode.

GPR precomputes total reward values for completely traversing each row, and cumulative reward values for partially traversing rows from either side. Partial rewards are obtained entering a row, visiting a certain number of consecutive vertices into the row, then turning around and exiting from whence it came, leaving some vertices in the row unvisited. A *feasible* vertex is a vertex such that there is enough budget to visit the vertex and then return to the ending location v_n. Initially, all vertices are marked as feasible. Then, the main loop is performed where vertices, rows, or partial rows are added to the tour until there are no longer any feasible vertices for the remaining budget, at which point the tour concludes by going to v_n. When choosing the next row or partial row to

```
Compute cumulative rewards from left and right
Mark all vertices as feasible
while feasible vertices exist do
    Compute full-row heuristics
    Find best full-row
    Compute partial-row heuristics for current side
    Find best partial-row
    if best full-row is better then
        if feasible then
            Append to tour
        Mark as unfeasible
    if best partial-row is better or best full-row is unfeasible then
        if feasible then
            Append to tour
        Mark as unfeasible
    if both unfeasible then
        Mark both as unfeasible
Append path from current vertex to ending vertex to tour
return tour
```

Algorithm 1: Greedy Partial-Row Heuristic (GPR)

visit, the heuristic scores for all feasible paths are computed by dividing the potential rewards with the potential costs. In the case of full rows, the full row reward is divided by the cost to enter the row plus the cost to traverse the row. In the case of partial rows, the cumulative reward at each vertex within rows is divided by the cost to enter the row plus the cost to partially traverse the row up to the vertex plus the cost to turn around and exit the row on the same side it entered. After heuristics are computed, the best full row and partial row are chosen, then compared to each other. The better of the two is then added to the tour if it is feasible, and marked as unfeasible so its vertices are no longer considered. If the better one is not feasible, it is marked as such and (if feasible) the other is added to the tour and marked as unfeasible. If both are not feasible, then they are both marked as such. Each loop iteration will mark more vertices as unfeasible until no more are left, thus ensuring the algorithm eventually terminates.

The complexity of this algorithm is $\mathcal{O}(m^2 n)$ where m is the number of rows and n is number of vines per row. This complexity is derived from the fact that the algorithm is one large while-loop, where the loop iterates at most mn times (once for each vertex) and each iteration makes m heuristic calculations. In [4] we experimentally demonstrated that this heuristic is the best among those we introduced for irrigation graphs, and we also showed that it outperforms general purpose methods proposed in the past. Our various multi-agent extensions to solve IGTOP instances are therefore built on top of the GPR method.

B. Vineyard Sectioning

The simplest method to solve an instance of the IGTOP problem consists in splitting the graph into a set of M disjoint sections and assign one agent to each section. Each section is defined as a set of contiguous rows. This is extremely simple to implement and it can be very effective in some circumstances. For the case of IGTOP, each of the M agents is assigned its own section and solves the single-agent orienteering problem using the GPR algorithm. To preserve spatial cost information about navigation from the start location and to the end location, each instance of GPR receives a copy of the original irrigation graph, however rewards for all vertices not to be visited are zeroed out (marked unfeasible), and thus the agent only builds tours spanning its assigned block of rows. While it is possible to split the vineyard into M equal sections so that each agent has the same amount of area to service, this is obviously sub-optimal when the budget is too small for all vertices to be visited, as some sections will have more rewards to collect than others. To prevent agents from spending budget on low reward regions, blocks of rows are split by percentage of overall reward. Each agent will have a certain percentage of overall budget to expend, so it is assigned to a contiguous block of rows with approximately the same percentage of overall reward. Dividing the vineyard in this fashion normalizes the agents potential reward with its budget to utilize it more effectively. Because only one agent is assigned to each section, and each section contains only complete rows that are not assigned to any other section, collisions are avoided by construction.

This algorithm, called Vineyard Sectioning, is sketched in algorithm 2. The overall complexity of solving IGTOP using this technique is $\mathcal{O}(M \cdot m^2 n)$, due to the fact that GPR is run once for every agent.

```
Compute reward_total of vineyard
j = 1
for M agents do
    i = j
    sum = 0
    percent_M = budget_M / budget_total
    while sum/reward_total < percent_M do
        j = j + 1
        sum = sum + reward_j
    temprewardmap = rewardmap
    for all rows not between i and j do
        temprewardmap_row = 0
    Run GPR(temprewardmap) for current agent
```

Algorithm 2: Vineyard Sectioning

C. Series GPR

Instead of preliminarily sectioning the vineyard in M zones, an alternative strategy is to sequentially solve the single-agent orienteering multiple times on the whole irrigation graph, zeroing out the rewards collected by each agent, so that they are no longer considered by the following ones. Algorithm 3 shows how this is implemented. Perhaps more primitive than the vineyard sectioning approach, this method forgoes preplanning the area of visitation for each agent and instead allows them to freely roam collecting the highest available rewards. GPR is run M consecutive times, and after each run visited vertices and rows have their rewards set to zero so that every run afterwards ignores these areas in its search. To properly avoid collisions within rows, GPR is modified (see algorithm 4) to track where and when previous robots have visited a vertex by taking as input a conflict map containing

every vertex with times visited and passing this map as output filled with the old tours' and new tour's information.

Similarly to vineyard sectioning, running GPR in series will result in an overall complexity of $\mathcal{O}(M \cdot m^2 n)$, giving this algorithm a linear complexity with respect to the number of agents.

```
initialize conflictmap
for k agents do
    Run GPRwithAvoidance(rewardmap, conflictmap)
    for all visited vertices do
        rewardmap_vertex = 0
```

Algorithm 3: Series GPR

```
Compute cumulative rewards from left and right
while tour not concluded do
    Reset conflictmap and vertex feasibility to input
    tour = ∅
    while feasible vertices exist do
        Compute full-row and partial-row heuristics for current side
        Compute time for visiting full-rows and partial-rows
        Find best full-row without time conflicts
        Find best partial-row without time conflicts
        if best full-row or best partial-row not empty then
            if either is unfeasible then
                Mark as unfeasible where appropriate
            if best full-row is better and feasible then
                Append to tour
                Add vertices and times to conflictmap
                Mark as unfeasible
            else if best partial-row is feasible then
                Append to tour
                Add vertices and times to conflictmap
                Mark as unfeasible
        else
            Tell tour to wait 1 unit
    if exists path to end vertex without time confilct then
        Append path from current to end vertex to tour
        Add vertices and times to conflictmap
        return tour, conflictmap
    else
        Tell tour to save more time for the end
```

Algorithm 4: GPRwithAvoidance

D. Parallel GPR

Rather than plan the route for each agent independently, planning each route in parallel allows agents to take advantage of their current location when choosing who will cover the next best row or partial row. To apply this idea, an internal loop is added to GPR, such that heuristics are computed to reveal the best row and best partial row for each agent, which are kept track of in a list of candidates. The path with the greatest heuristic value is tested for feasibility as well as time conflicts, and if feasible and conflict free it is added to the corresponding agent's tour then marked as unfeasible for future iterations. However, if it is unfeasible then the next best candidate is tested and the unfeasible row/partial row and agent combination is added to a blacklist where it will be passed over for future iterations. Once a row or partial row is blacklisted for all agents, then it is marked unfeasible and will no longer be considered. Like GPR with avoidance, the algorithm will continue until all points of interest are marked unfeasible, and then each tour will be concluded at the ending vertex if there are no time conflicts.

The complexity of this algorithm is $\mathcal{O}(M \cdot m^2 n)$, again linear with respect to the number of agents. The complexity is derived from the fact that a new internal loop is added to the original GPR algorithm that computes heuristics for each agent at every iteration. Thus, it ends up with the same complexity as Series GPR. Pseudocode is shown in Algorithm 5.

```
Compute cumulative rewards from left and right
while all tours not concluded do
    Reset conflictmap, blacklist, and vertex feasibility
    All tours = ∅
    while feasible vertices exist do
        Clear candidates
        for all agent_M do
            Compute heuristics of full-row and partial-row for current side
            Compute visiting time of full-rows and partial-rows
            for all full-rows without time conflicts do
                Find best full-row not in blacklist for agent_M
                Add to candidates
            for all partial-rows without time conflicts do
                Find best partial-row not in blacklist for agent_M
                Add to candidates
            if nothing added to candidates for agent_M then
                Tell tour_M to wait 1 unit
        while Candidates is not empty do
            Find candidate with greatest heuristic
            if best candidate is feasible then
                Append to tour_M
                Add vertices and times to conflictmap
                Mark as unfeasible
            else
                Add to blacklist
        Mark vertices blacklisted by all agents as unfeasible
    for all agent_M do
        if exists path to end vertex without time conflict then
            Append path to tour_M
            Add vertices and times to conflictmap
        else
            Tell tour_M to save more time for the end
    if all tours at ending vertex then
        return tours, conflictmap
```

Algorithm 5: Parallel GPR

E. Guided Local Search

To benchmark our algorithms, we use the Guided Local Search (GLS) Metaheuristic [21], which builds an algorithm for solving TOP in the general case using a composition of several local search heuristics. GLS was chosen because it is easily extendable from the general TOP case with fully connected graphs to the case we study here with irrigation graphs. Initial construction is performed by creating M tours from the start location to the furthest possible vertices from the start and end, such that each tour visits only one vertex other than the start and finish. Next, cheapest insertion is performed on each tour until no longer possible, and then the algorithm enters a series of loops. These loops are iterated until the

solution can no longer be improved (for defined parameters), and within them the following local search heuristics are performed: *swap* which trades vertices between tours, *TSP* which performs a 2-opt cost improvement on individual tours, *move* which moves vertices between tours to group together budget left, *insert* which adds unvisited vertices to tours, *replace* which swaps a visited vertex for an unvisited one, and *disturb* which removes some number of vertices from the beginning or end of each tour. The guided local search metaheuristics are used in the *replace* and *TSP* heuristics to improve results. One important aspect to notice is that GLS was not designed around IGTOP and therefore does not account for possible collisions within rows. Additionally, unlike our proposed algorithms, GLS has numerous parameters that should be tuned for good performance. The complexity of this algorithm and others similar to it is $\mathcal{O}(T_{MAX} \cdot k^2 log^2(M \cdot k))$ where k is the number of vertices in the graph, i.e., $k = mn$ in our case. The complexity is then significantly higher than our heuristics. Note also the linear dependency on T_{MAX}.

V. Experimental Comparison

The algorithms presented in the previous section were tested on a series of simulated routing problems derived from a vineyard block at a commercial operation in central California. The block was rectangular in shape with $m = 275$ rows of $n = 214$ columns (vines) per row (see Figure 3). This generates problem instances with 58,850 vertices in the graph.

The reward $r(v)$ for each vertex v was defined as $r(v) = |T - m(v)|$, where T is a constant indicating the desired soil moisture in the vineyard (provided by a human expert), and $m(v)$ is the soil moisture at vertex v. This reward is the difference between desired moisture and actual moisture, thus revealing how under-watered or over-watered a vine is. Due to the large size of the ranch, soil moisture data was sampled at discrete locations within the vineyard using a GPS equipped manual probe (Campbell Scientific Hydrosense HS2P). From the finite set of samples, a soil moisture map for the whole block was obtained using Kriging [37] for interpolation. Figure 4 shows one example of a reward map created from collected data.

Over the course of summer 2018, data was collected every two weeks from this vineyard and used to produce ten soil moisture maps. These moisture maps were then used to test the algorithms described in section IV and the results were averaged across each reward map. Where GLS is used, the values for tuning parameters used were those suggested in [21].

The first test compared the three proposed algorithms (Vineyard Sectioning, Series GPR, Parallel GPR) with the GLS metaheuristic discussed earlier. Here, the irrigation graph tested was scaled-down to 300 vertices (12×25) because of computational constraints using GLS. For a thorough comparison, the number of agents and total budget (budget for all agents combined) was varied. In cases where more than one agent was considered, all agents had an equal budget. For the three proposed algorithms, no collisions occurred. This result is expected, as they were designed to avoid building tours with

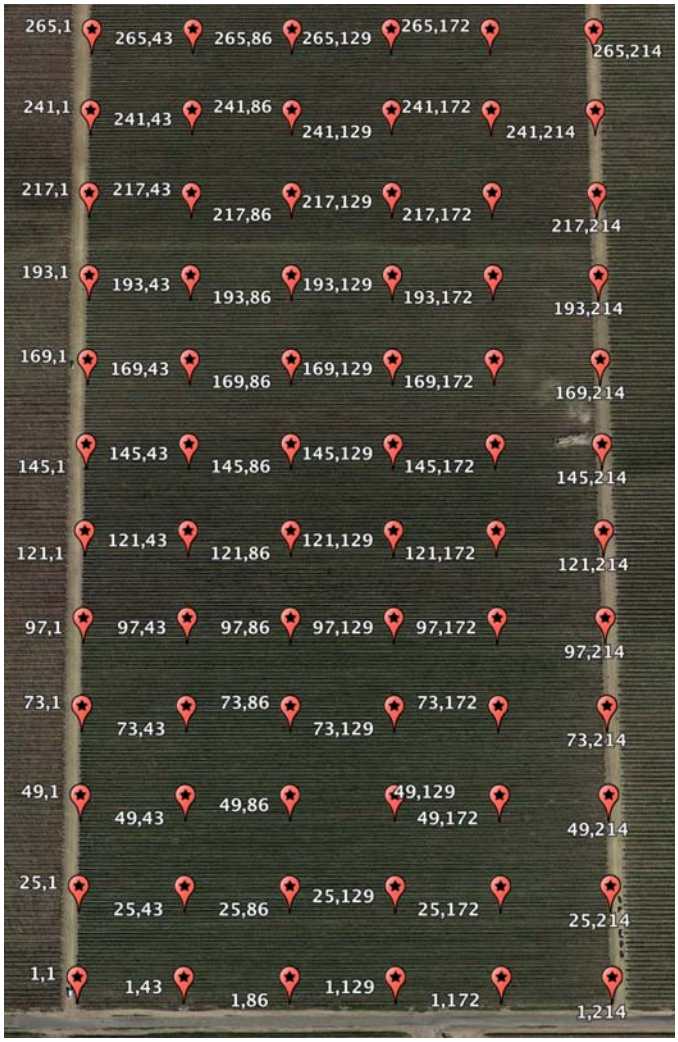

Fig. 3: An aerial view of the vineyard block used to collect data for our experiments and the locations where soil moisture data was collected (red pins).

collisions. GLS, on the other hand, did produce tours with some collisions, but these were very uncommon, occurring in 0.92% of cases. While collisions were expected using GLS, the small number detected is somewhat surprising since GLS has no built-in mechanism for avoiding such circumstances. Nonetheless, our heuristics are guaranteed to produce collision free solutions irrespective of the graph size and reward map, and the same is not true for GLS. Cases where GLS produced collisions were omitted when calculating the average results shown.

Figure 5 shows how the number of agents used to solve a IGTOP effects the overall outcome. Each data point corresponds to ten runs (one for each moisture map, except for GLS when a collision occurs) of the associated algorithm with the specified total budget and number of agents, where the fraction of total available reward collected is averaged to a single value. In each instance run, all agents shared the same total budget equal to T_{max}/M. For example, for a total budget of 100, two agents have 50 each, 4 have 25 each, etc. These results show that, for irrigation graphs, one agent with a larger

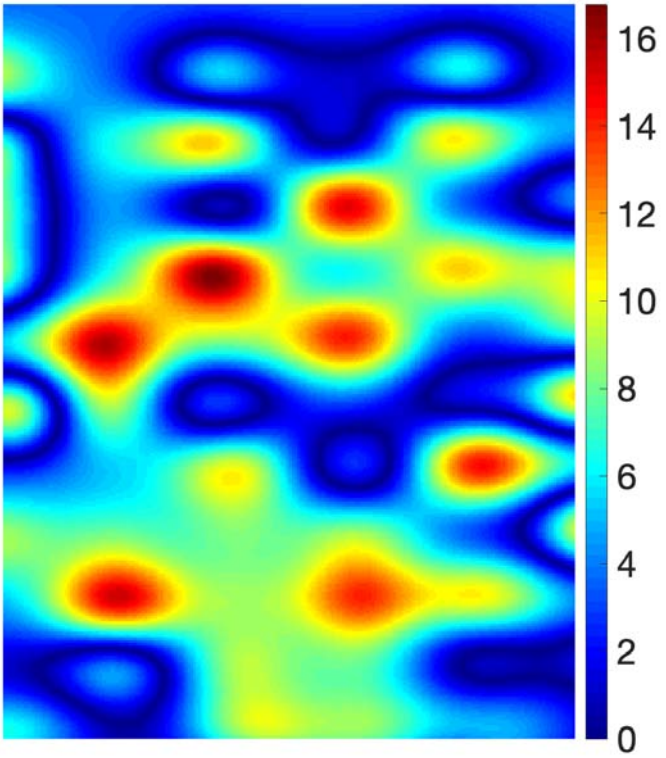

Fig. 4: A reward map compiled using data from the vineyard in Fig. 3.

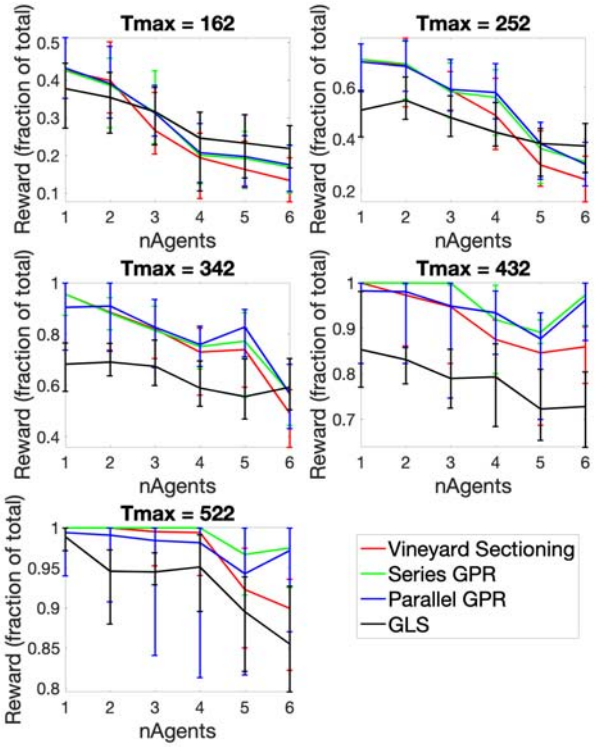

Fig. 5: Number of agents vs average fraction of total reward collected for fixed budgets on a 12×25 graph. Included are error bars showing the maximum and minimum of reward at each tested budget. Note that some lines and error bars are overlapping in some intervals.

budget would be more impactful than multiple agents with smaller budgets. This is expected because with a single robot no coordination is necessary. However, this is unfeasible in practice; it is technically not viable to deploy a single robot with autonomy equal to the sum of the autonomies of all M robots. Additionally, all three proposed algorithms perform better than GLS in most cases, with the exception of relatively low budgets, where GLS performs as well as the others.

Figure 6 shows the case where 5 agents are used and the combined budget is varied. Each data point is again the average of the results on ten runs with different moisture maps. GLS is able to collect the most reward at smaller budgets, beating out all three proposed algorithms, but as the budget increases it eventually becomes the worst performing algorithm. The performance of the three proposed algorithms shows similarity, with Series GPR and Parallel GPR trading off for the highest collected reward while Vineyard Sectioning lags behind. There is a maximum difference of 8.83% in collected reward between the three algorithms when $T_{MAX} = 342$. When GLS is considered, the gap between the top performer and GLS maximizes at 32.78% when $T_{MAX} = 342$.

The plot shown in figure 7 highlights inefficiencies in budget usage for each algorithm. Residual budget is the unused budget after the execution of the algorithm, meaning the algorithm cannot find a way utilize the rest of the budget to visit new vertices and increase the total reward. Unused budget emerges because the robots need to return to the deployment vertex before they can spend all their budget. An efficient algorithm should have a low residual as the fraction of total reward collected increases, with minimal spikes (signaling where graph structure prevents efficient use of budget), and a high reward ceiling at 1, where the residual begins to increase toward infinity. Series GPR and parallel GPR both have low residuals but high rewards ceilings, showing that they are efficient in path planning for irrigation graphs. Vineyard sectioning has high value residuals across the spectrum, which is the result of splitting the vineyard into blocks because some agents will be caged and unable to use their entire budgets, but also shows the ability to reach high fractions of reward collected. GLS tends to have a low residual budget, however it also does not do well collecting reward at high budgets, so it abruptly stops at its reward ceiling and is never able to collect all available rewards.

Having assessed the performance of our proposed heuristics against the known GLS used as a baseline, the next set of tests compares the three proposed algorithms to each other on the full sized irrigation graph with 58,850 vertices (275×214). Again, both the number of agents and total budget were variable parameters and collected rewards were averaged across all ten moisture maps so that the overall effectiveness of each algorithm could be explored. Moreover, no collisions were detected between robots, because the three heuristics ensure that no collisions will occur.

Figure 8 shows Series GPR and Parallel GPR are very

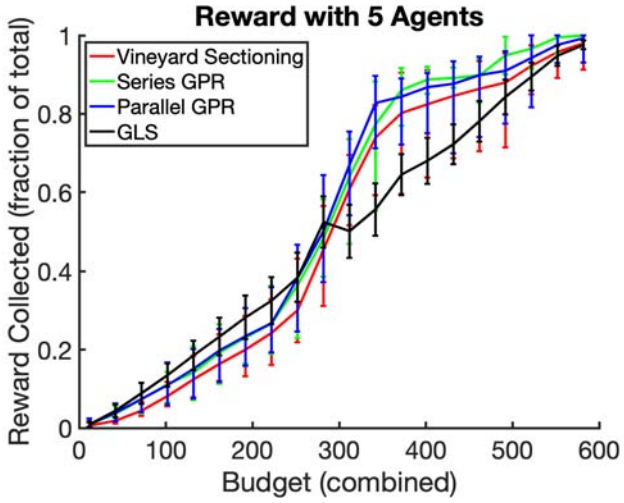

Fig. 6: Budget vs average fraction of total reward collected for 5 agents on a 12×25 graph. Included are error bars showing the maximum and minimum of reward at each tested budget. Note that some lines and error bars are overlapping in some intervals.

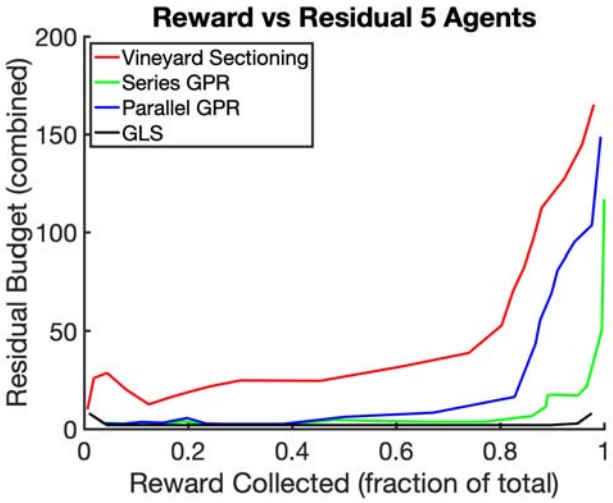

Fig. 7: Average fraction of total reward collected vs average leftover budget after paths are built for 5 agents on a 12×25 graph. Note that some lines are overlapping in some intervals.

close for much of the tested number of agents, while Vineyard Sectioning is considerably less efficient. The exception to this is when the budget $T_{max} = 29700$ and there are between 10 and 30 agents. Interestingly, there is a small but noticeable gap between the Series and Parallel algorithms, with Series GPR edging out the other for many of the experiments.

Figure 9 reveals that with the full sized graph and 50 agents, Series GPR and Parallel GPR perform very similarly. However, unlike in figure 6, Series GPR is the top performer more often than Parallel GPR. Vineyard sectioning also performed very well, however it never equals the reward collection performance of the other two algorithms. Specifically, with a budget of 88,550, both Series GPR and Parallel GPR can collect 100% of the rewards for all 10 moisture maps, but

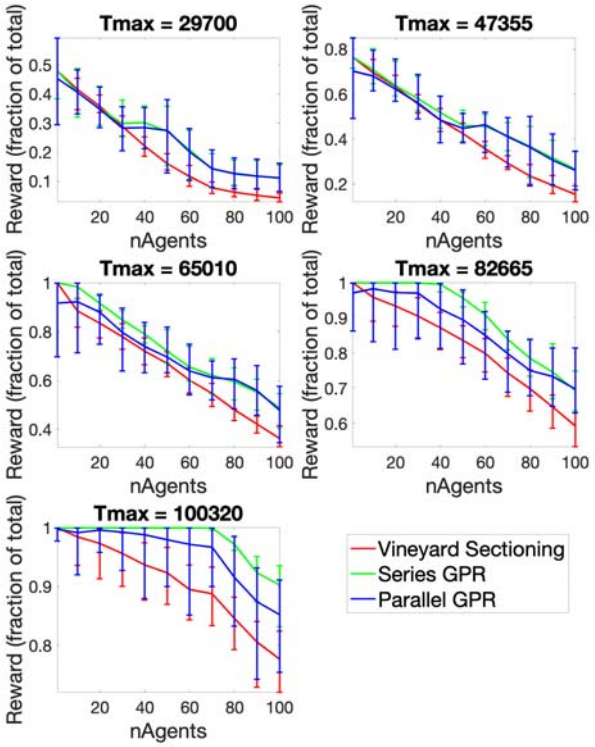

Fig. 8: Number of agents vs average fraction of total reward collected for fixed budgets on a 275×214 graph. Included are error bars showing the maximum and minimum of reward at each tested budget. Note that some lines and error bars are overlapping in some intervals.

Vineyard Sectioning is only able to collect on average 90.4% of the rewards. The results are similar in cases with any number of agents (1 through 100 agents tested), with reward collected equal for all algorithms when only one agent is considered but diverging as more agents are added.

The residuals in figure 10 show an interesting pattern emerge for the Vineyard Sectioning algorithm. As the combined budget increases, each agent has a larger budget to expend, however after a certain point much of the increased budget is wasted. Again, this is likely due to the compartmentalization of each agent, i.e. when they collect all the rewards in their area there is nothing left for them to do so they end their tour. Because some sections are larger than others and all the agents have the same budget, larger sections will have portions unexplored unless the budgets are increased dramatically beyond what is needed for smaller sections, so agents servicing smaller sections will have lots of excess. Additionally, when Series GPR and Parallel GPR collect all of their rewards, the residuals shoot up because there is nothing left for the agents to do, so extra budget goes unused. Again, results look similar for all numbers of agents tested.

VI. Conclusions and Future Work

In this paper we studied the problem of routing multiple robots within a vineyard - where movement is limited when a

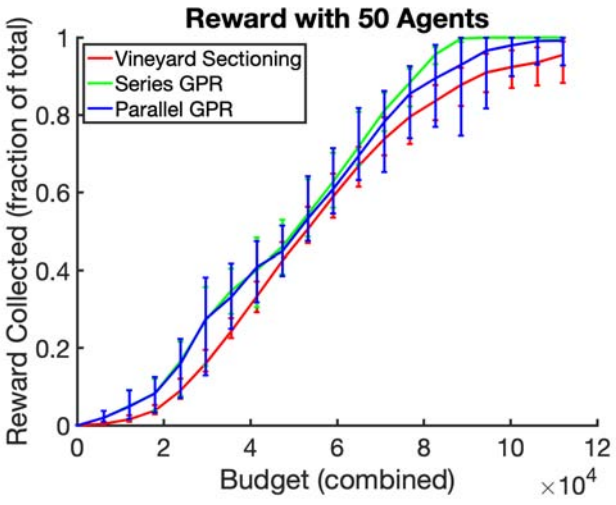

Fig. 9: Budget vs average fraction of total reward collected for 50 agents on a 275×214 graph. Included are error bars showing the maximum and minimum of reward at each tested budget. Note that some lines and error bars are overlapping in some intervals.

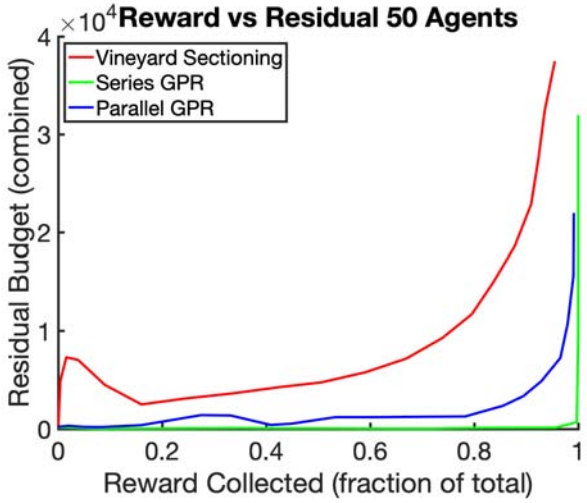

Fig. 10: Average fraction of total reward collected vs average leftover budget after paths are built for 50 agents on a 275×214 graph.

row is entered - for the application of precision irrigation. Recognizing this problem as NP-hard, we presented three parameter-free evolutions of the single robot routing algorithm we recently developed and extend its capabilities for teams of robots. These algorithms were compared with the GLS heuristic, a method formerly proposed. All three of the heuristics significantly outperformed GLS in computation time, performed as well as or better than GLS in reward collected for most cases. The proposed Series GPR was quantitatively shown to provide the best performance on large scale graphs, followed by Parallel GPR and the Vineyard Sectioning.

Future work in this domain will consider navigational and emitter adjustment uncertainties, heterogeneous agents (i.e. humans and robots), simultaneous reward collection and sampling, and possible improvements to the proposed algorithms. Another interesting problem is in studying how to possibly place a set of different start and ending points (i.e. deployment locations and recharging stations). Moreover, we will study the effect of different techniques to construct global moisture maps from a finite set of samples. Efforts to deploy a fully working prototype in the field are also ongoing.

ACKNOWLEDGMENTS

This paper extends a preliminary version appeared in [38]. We gratefully acknowledge Luis Sanchez and Nick Dokoozlian from E&J Gallo Winery for having granted access to their vineyards for data collection, and for the valuable and information provided during this project. We thank Carlos Diaz Alvarenga, Jose Manuel Gonzalez, Christine Breckenridge, Jonathan Garache, and Andres Torres Garcia for assisting with data collection in the field.

REFERENCES

[1] G. Schaible and M. Aillery, *Challenges for US Irrigated Agriculture in the Face of Emerging Demands and Climate Change*. Elsevier, 2017, ch. Competition for Water Resources 2.1.1, pp. 44–79.

[2] D. V. Gealy, S. McKinley, M. Gou, L. Miller, S. Vougioukas, J. Viers, S. Carpin, and K. Goldberg, "Co-robotic device for automated tuning of emitters to enable precision irrigation," in *Proceedings of the IEEE Conference on Automation Science and Engineering*, 2016, pp. 922–927.

[3] R. Berenstein, R. Fox, S. McKinley, S. Carpin, and K. Goldberg, "Robustly adjusting indoor drip irrigation emitters with the toyota hsr robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2018, pp. 2236–2243.

[4] T. C. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin, "Routing algorithms for robot assisted precision irrigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2018, pp. 2221–2228.

[5] R. Smith and J. Baillie, "Defining precision irrigation: a new approachj to irrigation management," in *Proceedings of the Irrigation and Drainage Conference*. Irrigation Australia Ltd, 2009.

[6] R. González Perea, A. Daccache, J. A. Rodríguez Díaz, E. Camacho Poyato, and J. W. Knox, "Modelling impacts of precision irrigation on crop yield and in-field water management," *Precision Agriculture*, vol. 19, no. 3, pp. 497–512, Jun 2018.

[7] T. Tsiligirides, "Heuristic methods applied to orienteering," *Journal of Operational Research Society*, vol. 35, no. 9, pp. 797–809, 1984.

[8] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics*, vol. 34, pp. 307–318, 1987.

[9] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, "Approximation algorithms for orienteering and discounted-reward tsp," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 653–670, 2007.

[10] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches, and applications," *European Journal of Operational Research*, vol. 255, no. 2, pp. 315–332, 2016.

[11] M. Fischetti, J. J. S. González, and P. Toth, "Solving the orienteering problem through branch-and-cut," *Journal on Computing*, vol. 10, no. 8, pp. 133–148, 1998.

[12] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, pp. 1–10, 2011.

[13] R. Ramesh and K. M. Brown, "An efficient four-phase heuristic for the generalized orienteering problem," *Computers and Operations Research*, vol. 18, no. 2, pp. 151–165, 1991.

[14] N. Garg, "Saving an epsilon: A 2-approximation for the k-mst problem in graphs," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, 2005, pp. 396–402.

[15] C. Chekuri, N. Korula, and M. Pál, "Improved algorithms for orienteering and related problems," *ACM Transactions on Algorithms*, vol. 8, no. 3, 2012.

[16] K. Chen and S. Har-Peled, "The orienteering problem in the plane revisited," in *Proceedings of the twenty-second annual symposium on Computational geometry*, 2006, pp. 247–254.
[17] I.-M. Chao, B. L. Golden, and E. A. Wasil, "The team orienteering problem," *European Journal of Operational Research*, vol. 88, pp. 464–474, 1996.
[18] S. Boussier, D. Feillet, and M. Gendreau, "An exact algorithm for team orienteering problems," *4OR*, vol. 5, no. 3, pp. 211–230, 2007.
[19] L. Ke, C. Archetti, and Z. Feng, "Ants can solve the team orienteering problem," *Computers and Industrial Engineering*, vol. 54, pp. 648–665, 2008.
[20] S.-W. Lin, "Solving the team orienteering problem using effective multi-start simulated annealing," *Applied Soft Computing*, vol. 13, pp. 1064–1073, 2013.
[21] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden, "A guided local search metaheuristic for the team orienteering problem," *European Journal of Operational Research*, vol. 196, pp. 118–127, 2009.
[22] ——, "Metaheuristics for tourist trip planning," in *Metaheuristics in the Service Industry*, K. Sörensen, M. Sevaux, W. Habenicht, and M. J. Geiger, Eds. Springer Berlin Heidelberg, 2009, vol. 624, pp. 15–31.
[23] L. E. Parker, "Path planning and motion coordination in multiple robot teams," in *Encyclopedia of Complexity and System Science*. Springer, 2009.
[24] J. Banfi, N. Basilico, and F. Amigoni, "Intractability of time-optimal multirobot path planning on 2d grid graphs with holes," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 1941–1947, 2017.
[25] J. Yu and M. LaValle, "Optimal multiroot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.
[26] J. Yu, "Intractability of optimal multirobot path planning on planar graphs," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 33–40, 2016.
[27] S. Vougioukas, "Agricultural robotics," *Annual review of control, robotics, and autonomous systems*, vol. 2, pp. 339–364, 2019.
[28] A. Barrientos, J. Colorado, J. d. Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente, "Aerial remote sensing in agriculture: A practical approach to aera coverage and path planning for fleets of mini aerial robots," *Journal of Field Robotics*, vol. 28, no. 5, pp. 677–689, 2011.
[29] Z. Li and V. Isler, "Large scale image mosaic construction for agricultural applications," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 295–302, Jan. 2016.
[30] S. Bargoti and J. P. Underwood, "Image segmentation for fruit detection and yield estimation in apple orchards," *Journal of Field Robotics*, vol. 34, no. 6, pp. 1039–1060, 2017.
[31] G. Riggio, C. Fantuzzi, and C. Secchi, "A low-cost navigation strategy for yield estimation in vineyards," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2018, pp. 2200–2205.
[32] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis, "Design, integration, and field evaluation of a robotic apple harvester," *Journal of Field Robotics*, vol. 34, no. 6, Sep. 2017.
[33] T. Botterill, S. Paulin, R. Green, S. Williams, J. Lin, V. Saxton, S. Mills, X. Chen, and S. Corbett-Davies, "A robot system for pruning grape vines," *Journal of Field Robotics*, vol. 34, no. 6, pp. 1100–1122, 2017.
[34] Y. Zhang, Y. Ye, Z. Wang, M. E. Taylor, G. A. Hollinger, and Q. Zhang, "Intelligent in-orchard bin-managing system for tree fruit production," in *IEEE International Conference on Robotics and Automation Workshop on Robotics in Agriculture*, 2015.
[35] D. Bochtis, H. Griepentrog, S. Vougioukas, P. Busato, R. Berruto, and K. Zhou, "Route planning for orchard operations," *Computers and Electronics in Agriculture*, vol. 113, pp. 51–60, 2015.
[36] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter, "Hamilton paths in grid graphs," *SIAM Journal on Computing*, vol. 11, no. 4, pp. 676–686, Nov. 1982.
[37] M. A. Oliver and R. Webster, "Kriging: A method of interpolation for geographical information systems," *International Journal of Geographical Information Systems*, vol. 4, no. 3, pp. 313–332, 1990.
[38] T. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin, "Multi-robot routing algorithms for robots operating in vineyards," in *Proceedings of the IEEE Conference on Automation Science and Engineering*, 2018, pp. 7–14.