Multi-Robot Routing Algorithms for Robots Operating in Vineyards

Thomas C. Thayer Stavros Vougioukas Ken Goldberg Stefano Carpin

***Abstract*— In this paper 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 known as the "team orienteering problem". 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 approaches informed by the domain we consider, and compare them against a general purpose heuristic formerly developed and widely used. In various 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 hundred of thousands of vertices in the graphs.**

I. Introduction

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. However, currently used irrigation systems lack the ability to adjust water delivery at a fine grain level, e.g., on a per vine basis or based on small zones. Because overstressing a vine may lead to inferior yield and even to vine death, growers tend to over-irrigate and avoid potential losses. Delivering "the right amount of water" remains an open challenge. This problem is particularly acute in California, where grape production has a strong economic impact, but freshwater availability is limited, and where recent droughts have made the situation even worse. The US - where freshwater usage in agriculture is estimated to be 85% - is not facing this problem alone; worldwide usage of freshwater in agriculture is thought to be around 70% [10], making agricultural water conservation a global challenge. If one looks at irrigation infrastructure from a control standpoint, there is 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 coarse. Equipping each vine (or small groups of vines) with an electrically actuated variable-rate emitter is unfeasible because of cost considerations and the necessity to withstand extremely harsh operating conditions for extended periods of time. Passive, variable-rate emitters mitigate this problem, but manually adjusting them is an approach that does not scale due to the sheer size of the vineyards and increasing labor shortage.

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 improving yield and quality. The objective of RAPID is to create a co-robot system whereby fleets of robots will navigate through vineyards to adjust passive emitters allowing a more targeted water delivery. In [6] 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 [2] we showed how this concept can be extended for mounting on a robotic arm. Ultimately, the robotic arm would be moved by a mobile platform emitter to emitter to perform required adjustments.

The motion of the mobile robot carrying the arm is subject to various constraints. In particular, the robot may not be able to follow a straight line when moving from one emitter to another because vines and irrigation lines prevent changing vine rows. This is illustrated in figure 1. Moreover, due to the limited onboard power supply, the robot needs to periodically return to the deployment site for a recharge or swap of its batteries. In [11] we recently showed that this problem is related to the *orienteering* problem and we proved that it remains NP-hard even when considering the 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. However, in [11] we only considered 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 are deployed. This problem is related to the *team orienteering problem* and is obviously as hard as the single agent case. From a practical perspective, there is an additional challenge. Because vines are densely grown, the width of each row is small, and with multiple robots operating in the same vineyard, 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 briefly discussed in section II, whereas the problem we consider is formalized in section III. Motivated by the intrinsic computational complexity, three different heuristics are presented in section IV, where we moreover shortly discuss a formerly developed method for this class of problems.

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 supported by the by USDA-NIFA under award number 2017-67021-25925 (NSF National Robotics Initiative). Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the U.S. Department of Agriculture.

[1]While at this stage of the project we use a small Husky robot, in a field deployment we anticipate that platforms with a larger footprint will be used.

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.

Section V compares the different solutions on various problem instances based on data we collected on a commercial vineyard, and shows that our proposed solution favorably scales with the size of the problem. Finally, conclusions are given in section VI.

II. Related Work

The problem considered in this paper is related to the classic *orienteering* problem whereby one agent needs to traverse a graph where each vertex has a reward and each edge has a 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 budget. If a vertex is visited multiple times, the associated reward is counted only once. This problem was originally introduced in [7] where it was proven to be NP-hard, and in [3] the authors showed that orienteering belongs in the and APX-hard class. There exist numerous variants of the orienteering problem, and we point the reader to [8] for a recent survey. Two main approaches are followed to tackle this problem. The first aims at developing heuristic approaches that may work well in practice, often informed by domain specific knowledge (the above mentioned survey provides numerous pointers). The second utilizes exact methods using branch-and-bound techniques, but applicability is limited to problem instances with a small number of vertices in the graph, i.e., less than 1,000 [5]. To put this number into context, the problem instances we consider may have more than 100,000 vertices. The team orienteering problem, (TOP) is a variant of the orienteering problem where multiple agents collect the rewards, while all agents are subject to the same budget constraint [4]. Evidently, this problem is as hard as orienteering, and then subject to the same computational challenges, although the cooperative nature of the problem calls for the development of specialized heuristics, like [12]. The problem we consider is also related to the multi-robot motion coordination problem, in particular because of our requirement of not 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 [9]. The multirobot path planning remains intractable even when restricted to problem instances defined on planar graphs, and heuristics are used in this domain as well [1], [14], [13].

III. Problem definition

The team orienteering problem is formally defined as follows: 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 cost and a reward. The cost is the sum of the costs of all edges along the path, while the 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 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 following two constraints: 1) each path has cost at most T_{MAX}; 2) if a vertex is visited more than once, either by the same or different agents, the associated reward is collected only once.

In our recent work [11] 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 emerging when robots navigate in a vineyard. Figure 2 show the structure of these graphs, and we refer the reader to [11] for the formal definition. 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 model the motion constraints for a robot operating in a vineyard, i.e., row swapping is only possible at either end of the row, but not in the middle of a row.

Orienteering heuristics and algorithms often require complete graphs, and an irrigation graph can obviously be extended into a complete graph by adding the missing edges with the 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 a graph $IG(m, n)$, $v_1, v_n \in V$ be two of its vertices and c, r be a cost and reward function on G. For a given constant T_{MAX}, find M routes of maximum 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

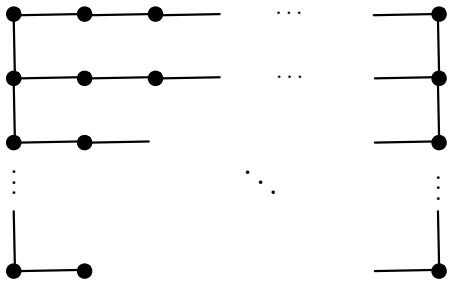

Fig. 2: Structure of an irrigation graph. The parameters m and n refer to the number of rows, and the number of vertices in each row, respectively.

collected only once if the vertex is visited multiple times.

In [11] we proved that the special case of IGTOP where $M = 1$ (single agent) and $c(e)$ is equal for all edges is NP-hard. Consequently, IGTOP is NP-hard as well. 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 homogenous; that is, they perform equally well in the same conditions.

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.

IV. Proposed solutions

A. Single Agent Greedy Partial Row Heuristic

The Greedy Partial Row Heuristic algorithm (GPR) we proposed in [11] only solves the single agent case, but can be used as a building block to solve the multi-agent case. We shortly summarize it in the following and refer the reader to [11] 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 (entering a row, visiting some number of vertices into the row, then turning around and exiting from whence it came, leaving some vertices in the row unvisited) from either side. A *feasible* vertex is a vertex such that there is enough budget to visit the vertex and then go 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 visit, the heuristic scores for all feasible paths are computed dividing the potential rewards by 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 greater of the two is then added to the tour if it is feasible and then marked as unfeasible so it does not enter the tour a second time. If not feasible, it is marked as such and the other is added to the tour then 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. In [11] we demonstrated that this heuristic is the best among those we considered for irrigation graphs, and it also outperforms general purpose heuristics proposed in the past. Therefore we use it to implement various multi-agent extensions to solve IGTOP instances.

```
1: Compute cumulative rewards from left and right
2: Mark all vertices as feasible
3: while feasible vertices exist do
4:     Compute full-row heuristics
5:     Find best full-row
6:     Compute partial-row heuristics for current side
7:     Find best partial-row
8:     if best full-row is better then
9:         if feasible then
10:            Append to tour
11:        Mark as unfeasible
12:    if best partial-row is better or best full-row is unfeasible
       then
13:        if feasible then
14:            Append to tour
15:        Mark as unfeasible
16:    if both unfeasible then
17:        Mark both as unfeasible
18: Append path from current vertex to ending vertex to tour
19: return tour
```

Algorithm 1: Greedy Partial-Row Heuristic (GPR)

B. Vineyard Sectioning

The simplest method of applying teamwork to orienteering on a graph consists in splitting the graph into a set of M sections and assign one agent to each section. This is the classic divide and conquere approach, and besides being extremely simple to implement, it can be very effective in some circumstances. For the case of IGTOP, each of the M agents is assigned a set of contiguous rows over which it is to solve the single-agent orienteering problem using the GPR algorithm.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 suboptimal 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 in the following, 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 verticies 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 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 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 heuristics of full-row and partial-row 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 confilcts, 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, then each tour will be concluded at the ending vertex if there are not time conflicts. The complexity of this algorithm is $\mathcal{O}(M \cdot m^2 n)$, again linear with respect to the

number of agents, and pseudocode is shown in Algorithm 5.

```
1: Compute cumulative rewards from left and right
2: while all tours not concluded do
3:     Reset conflictmap, blacklist, and vertex feasibility
4:     All tours = ∅
5:     while feasible vertices exist do
6:         Clear candidates
7:         for all agent_M do
8:             Compute heuristics of full-row and partial-row for
               current side
9:             Compute visiting time of full-rows and partial-rows
10:            for all full-rows without time conflicts do
11:                Find best full-row not in blacklist for agent_M
12:                Add to candidates
13:            for all partial-rows without time conflicts do
14:                Find best partial-row not in blacklist for agent_M
15:                Add to candidates
16:            if nothing added to candidates for agent_M then
17:                Tell tour_M to wait 1 unit
18:        while Candidates is not empty do
19:            Find candidate with greatest heuristic
20:            if best candidate is feasible then
21:                Append to tour_M
22:                Add vertices and times to conflictmap
23:                Mark as unfeasible
24:            else
25:                Add to blacklist
26:        Mark verticies blacklisted by all agents as unfeasible
27:    for all agent_M do
28:        if exists path to end vertex without time conflict then
29:            Append path to tour_M
30:            Add vertices and times to conflictmap
31:        else
32:            Tell tour_M to save more time for the end
33:    if all tours at ending vertex then
34:        return tours, conflictmap
```

Algorithm 5: Parallel GPR

E. Guided Local Search

To benchmark our algorithms, we use the Guided Local Search (GLS) Metaheuristic [12], 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. 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 commercial vineyard in central California with multiple blocks of roughly the same size. Most blocks were rectangular with $m = 240$ rows of $n = 500$ columns (vines) each (see Figure 3). These generate problem instances with 120,000 vertices in the graph.

Fig. 3: One of the vineyard blocks used to collect data for our experiments and the locations where soil moisture data was collected.

The reward $r(v)$ for each vertex v was calculated as $r(v) = |T - m(v)|$, where T is a constant indicating the desired soil moisture in the vineyard and $m(v)$ is the soil moisture at vertex v. This reward is the difference between desired moisture and actual moisture, thus revealing how underwatered or overwatered 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 with a linear interpolation in between samples, and extrapolated with a nearest neighbor for locations outside the convex hull of the locations of the samples. Other interpolation techniques, like kriging, could be used, but are inconsequential to the work discussed here. Figure 4 shows one example of reward map.

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 verticies (12×25)

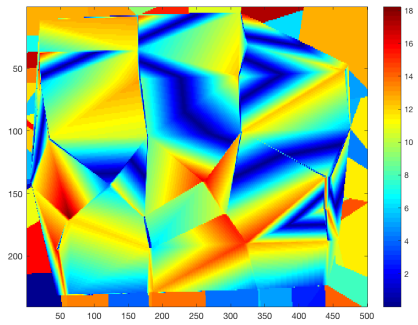

Fig. 4: Reward map for the same vineyard in Fig. 3.

because of computational constraints using GLS. For a thourough comparison, the number of agents and total budget (budget for all agents combined) was varied. For all four algorithms, no collisions occurred. This result was somewhat unexpected for GLS, as it is not designed to mitigate collisions in an irrigation graph. However, while our heuristics are guaranteed to produce collision free solutions irrespective of the graph size and the reward map, the same cannot be said for GLS.

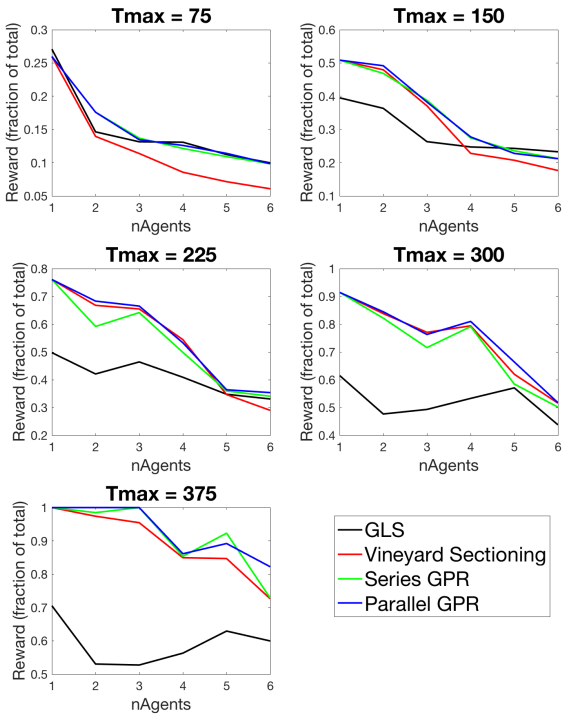

Fig. 5: Number of agents vs fraction of total reward collected for fixed budgets on a 12×25 graph. Note that some lines are overlapping in some intervals.

Figure 5 shows how the number of agents used to solve a IGTOP effects the overall outcome. 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 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 almost as well as the others.

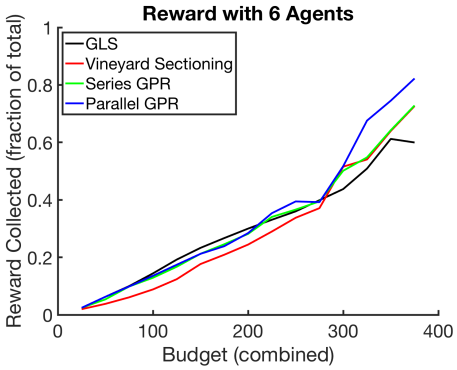

Fig. 6: Budget vs fraction of total reward collected for 6 agents on a 12×25 graph. Note that some lines are overlapping in some intervals.

In figure 6 we instead consider the case where 6 agents are used and the combined budget is increased. The performance of the three proposed algorithms shows similarity. Vineyard Sectioning, Series GPR, and Parallel GPR all at one point collect the highest reward, but usually Parallel GPR is on top. In many instances, the three algorithms are almost equal to each other in performance, and they stay neck and neck to each other in most cases. There is a maximum difference of 13.52% in collected reward between the three algorithms when $M = 6$ and $T_{MAX} \times M = 325$. When GLS is considered, the gap between the top performer and GLS maximizes at 47.25% when $M = 6$ and $T_{MAX} \times M = 375$. Note that GLS performs as well as the others when budgets are lower, but begins to fall behind as budgets climb higher.

The plots shown in figure 7 and figure 8 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 optimal algorithm should have a low residual as the fraction of total reward collected increases, with minimal spikes (signaling where graph structure prevents efficent 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 lots of high value

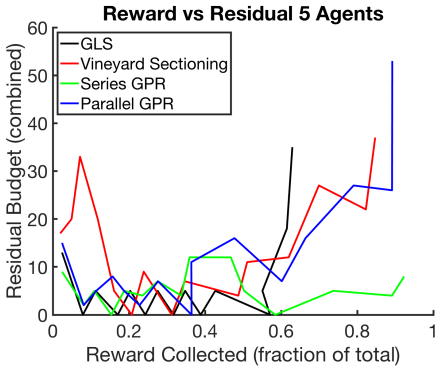

Fig. 7: Reward vs leftover budget after paths are built for 5 agents on a 12×25 graph. Note that some lines are overlapping in some intervals.

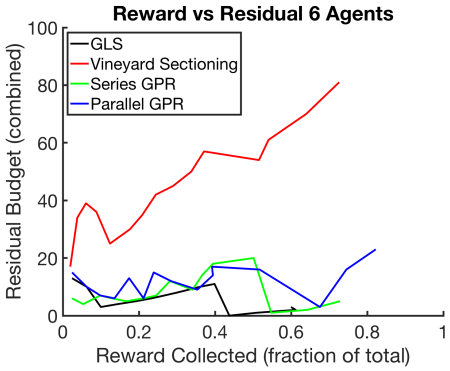

Fig. 8: Reward vs leftover budget after paths are built for 6 agents on a 12×25 graph. Note that some lines are overlapping in some intervals.

residuals, 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 residual budget increases greatly at a lower reward ceiling, signifying it is not as efficient as the other algorithms.

The second test compared the three proposed algorithms to each other on the full sized irrigation graph with 120,000 vertices (240×500). Again, both the number of agents and total budget were variable parameters 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 9 shows Series GPR and Parallel GPR are very close for much of the tested number of agents, while Vineyard Sectioning is considerably less efficient. Similarly to the 12×25 test, Parallel GPR performs slightly better than Series GPR most of the time, especially with larger budgets.

Figure 10 reveals that with the full sized graph and 50 agents, Series GPR and Parallel GPR perform nearly identically however, unlike in figure 6, Parallel GPR is nearly always the top performer. Vineyard sectioning also

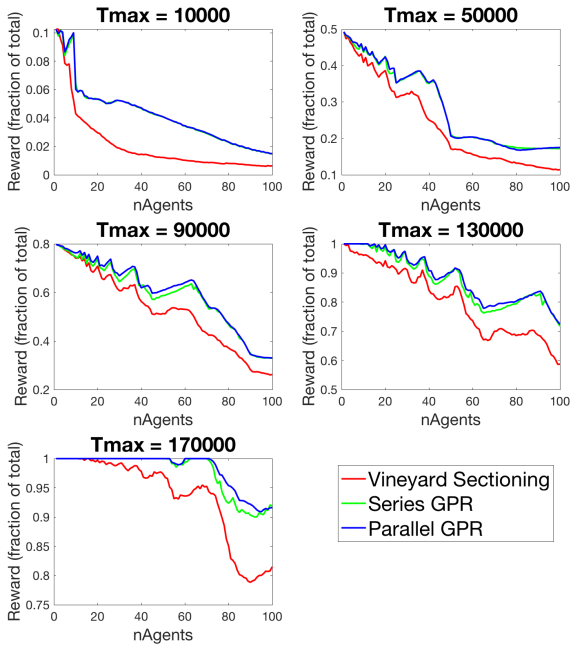

Fig. 9: Number of agents vs fraction of total reward collected for fixed budgets on a 240×500 graph.

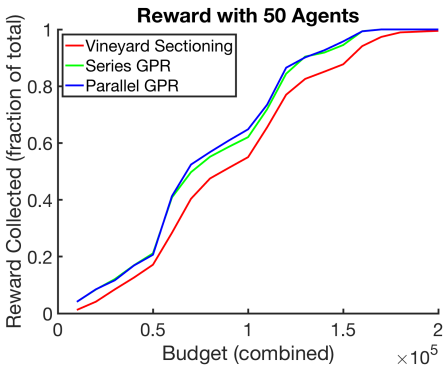

Fig. 10: Budget vs fraction of total reward collected for 50 agents on a 240×500 graph.

performed very well, however it never equals the reward collection performance of the other two algorithms. Specificly, with a budget of 150,000, Parallel GPR can collect 95.7% of the rewards and Series GPR can collect 94.5% of the rewards, but Vineyard Sectioning is only able to collect 87.7% 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 algorthims when only one agent is considered but diverging as more agents are added.

The residuals in figure 11 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

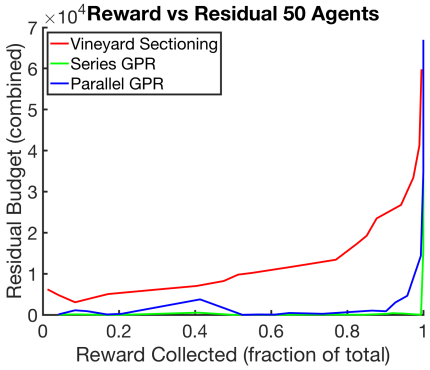

Fig. 11: Fraction of collected reward vs leftover budget after paths are built for 50 agents on a 240×500 graph.

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.

To analyze the robustness of the proposed algorithms, they were subjected to more tests of the same sizes as the first two tests (12×25 and 240×500) but with different moisture maps. The results of these tests are very similar to the results presented earlier, and the relative effectiveness of each algorithm remains consistent.

VI. Conclusions

In this paper we studied the problem of routing multiple robots within a vineyard - where movement is limited when a row is entered - for the application of precision irrigation. Recognising this problem as NP-hard, we presented three 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 and widely used. Two of the heuristic outperformed GLS in both reward and computation time. Future work in this domain will consider navigational and emitter adjustment uncertainties, and heterogeneous agents (i.e. humans and robots). Moreover, we will study the effect of different techniques to construct global moisture maps from a finite set of samples. Methods like kriging yield smoother maps and this could indicate alternative heuristics to explore. Efforts to deploy a fully working prototype in the field are also ongoing.

Acknowledgments

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 and Jose Manuel Gonzalez for assisting with data collection in the field.

References

[1] 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*, 2(4):1941–1947, 2017.

[2] R. Berenstein, R. Fox, S. McKinley, S. Carpin, and K. Goldberg. Adjusting indoor drip irrigation emitters with the toyota hsr robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2236–2243, 2018.

[3] 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*, 37(2):653–670, 2007.

[4] I. Chao, B. Golden, and E. Wasil. The team orienteering problem. *European Journal of Operations Research*, 88:464–474, 1996.

[5] M. Fischetti, J.J. Salazar Gonzalez, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

[6] D. Gealy, S. McKinkley, M. Guo, L. Miller, S. Vougioukas, J. Viers, S. Carpin, and K. Goldberg. Co-robotic device for autoamated tuning of emitters to enable precision irrigation. In *Proceedings of the IEEE Conference on Automation Science and Engineering*, pages 922–927, 2016.

[7] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logisitics*, 34(3):307–318, 1987.

[8] A. Gunavan, H. Chuin Lin, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[9] L.E. Parker. Path planning and motion coordination in multiple mobile robot teams. In Meyers R.A, editor, *Encyclopedia of Complexity and System Science*. Springer, 2009.

[10] G.D. Schaible and M.P. Aillery. Callenges for US irrigated agriculture in the face of emerging demands and climate change. In J. Ziolkowska and J.M. Peterson, editors, *Competition for Water Resources*, chapter 2.1.1, pages 44–79. Elsevier, 2017.

[11] T. 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*, pages 2221–2228, 2018.

[12] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operations Research*, 196:118–127, 2009.

[13] J. Yu. Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics and Automation Letters*, 1(1):33–40, 2016.

[14] J. Yu and S. M. LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.