2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Routing Algorithms for Robot Assisted Precision Irrigation

Thomas C. Thayer Stavros Vougioukas
Ken Goldberg Stefano Carpin

Abstract— **When robots navigate through vineyards to perform irrigation adjustments, an optimization problem emerges whereby robots are tasked with performing adjustments having the highest cumulative outcome within a given temporal budget due to limited battery charge. To this end, the robot needs to reach a set of spatially distributed sites, and the specific structure of the vineyard imposes various constraints on possible motions. In this paper we first demonstrate that this type of orienteering problem remains NP-hard even for the restricted class of graphs associated with precision irrigation. Then, we devise and analyze two greedy heuristics informed by the problem we consider. Finally, these algorithms are evaluated on settings associated with a commercial vineyard and we show that our methods favorably compare to solutions proposed in the past.**

I. Introduction

Agriculture is the major consumer of freshwater in the world, with some studies estimating that more than 70% of

freshwater is used in this domain. In the US this percentage is even higher, likely around 85% [17]. Despite extensive use of water resources, many commercial orchards still rely on irrigation systems that cannot control water use on a per-tree basis. This problem is particularly daunting in vineyards, where the quality of the final product is highly correlated with the amount of water absorbed by the vine. Ideally, vineyards should be subject to localized *stress irrigation*, customizing the amount of water delivered to each vine. The appropriate amount of water depends on a variety of factors, including the soil characteristics, age of the vine, sun and wind conditions, and season, to name a few. However, over-stressing a vine would result in inferior quality grapes, and potentially even death of the vine, with a consequent multi-year economic loss. For this reason, growers and farm managers often tend to over-irrigate with the double drawback of wasting water and obtaining an inferior product. RAPID (Robot Assisted Precision Irrigation Delivery) is a multi-year, multi-university project aiming at developing a robotic system capable of making fine grain adjustments to water delivery, ideally on a per-plant level, with the objective of reducing water use while preserving the quality of the final product. RAPID will explore: 1) novel sensing infrastructures based on remote and terrestrial sensing; 2) learning-based models to infer current moisture levels in 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 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.

soil and suggest adjustments to be made to the irrigation infrastructure; 3) passive variable rate water emitters and robotic actuators that can adjust them; 4) mobile platforms that carry the actuation system through the vineyard to perform the adjustments formulated by the decision support system. Figure 1 visualizes this concept.

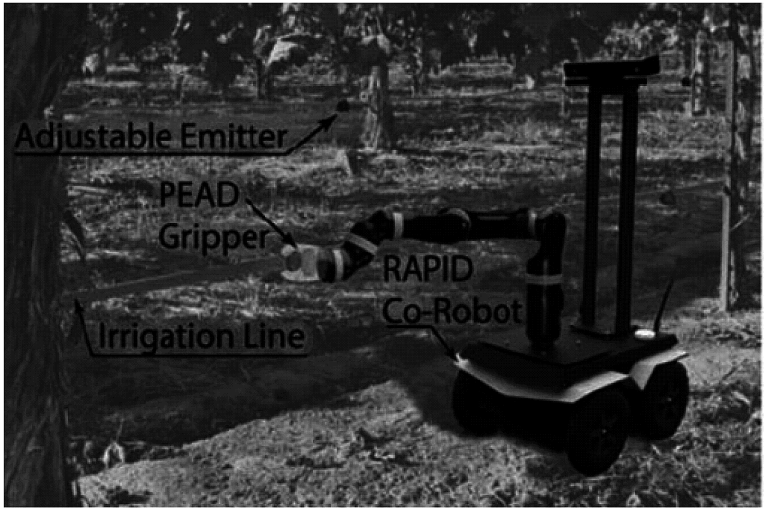

Fig. 1: RAPID concept: a robot performing adjustments on water emitters placed on irrigation lines. Note that irrigation lines prevent the robot from moving across plant rows.

In [12] the design of a portable emitter actuation device (PEAD) was presented, while a first version of the learning based decision infrastructure is still being developed and tested. In this paper we address a routing problem associated with actuating the water emitters. In commercial vineyards, once a robot enters a row it is constrained to move forwards or backwards inside the row; sideways motions to adjacent rows are not possible. This is due to the fact that, as

shown in Figure 1, irrigation lines are above the ground and prevent moving across rows. Moreover, robots can operate only for a limited amount of time due to having a finite battery capacity. We assume that the decision support system indicates how the water flow should be adjusted for each water emitter in the vineyard. Large adjustments indicate significant mismatches between the desired soil moisture and the measured (or inferred) soil moisture. Therefore, the robot should ideally adjust those emitters first. However, since these can be spatially distributed over the vineyard, an optimization problem emerges, whereby the robot has to decide which subset of emitters to adjust subject to the motion constraints imposed by the field structure and the bounded distance it can travel.

In this paper, we show that this problem can be cast as a special instance of the Orienteering Problem, where

978-1-5386-3081-5/18/$31.00 ©2018 IEEE

the graph structure is defined by the motion constraints associated with navigating the vineyard. We formally prove that, even though the graph has a very regular structure and we consider special cases where travel costs are constant (a reasonable approximation in vineyards, since all vines and emitters are uniformly spaced), the problem remains NP-hard. Motivated by this computational bottleneck, we then propose and examine two heuristic approaches and compare

them to the optimal solution (for small problem instances) and other heuristics. Our findings show that our proposed algorithms provide high quality solutions, but are much faster and do not depend on a careful selection of parameters governing their performance.

The rest of this manuscript is organized as follows. Related work is presented in Section II, whereas in Section III we formally define the problem and prove its NP-hardness. Our proposed routing algorithms are described in Section IV and experimentally validated in Section V. Finally, conclusions are given in Section VI.

II. Related Work

A. *Orienteering*

Precision irrigation adjustment is related to the *Orienteering Problem* originally introduced in [13] and also known as the "bank-robber-problem." In orienteering, one is given a graph where every edge has a weight and every vertex has a reward. A path in the graph collects the rewards of all traversed vertices, but if a vertex is visited multiple times its associated reward is collected only once. A path also incurs a cost equaling the sum of costs for all traversed edges[1]. The problem is to determine a path maximizing collected rewards subject to a given bound on the total path cost. The orienteering problem was shown to be NP-hard in [13] and APX-hard in [4]. Numerous variations of the original problem have been proposed through the years and

the reader is referred to [14] for a recent comprehensive review. Due to its intrinsic complexity, various heuristic approaches have been proposed, though the design of a general purpose, efficient heuristic remains a challenge [14]. Approximation algorithms have been extensively studied, where the produced path obeys the bound constraint and collects a guaranteed fraction of the optimal reward, i.e., a c-approximation produces a path with reward Π/c where Π is the optimum maximum reward. As shown in [1], the unrooted version of the problem, where the start vertex of the path is not fixed, is related to the k-TSP problem, and for this version of the problem a polylogarithmic and a 2-approximation has been known for a while [1], [11]. The first constant factor approximation for the rooted version of the problem, where the start vertex is fixed, was given in [4], where a 4-approximation is given. This algorithm was then improved in [2] where a 3-approximation was given for the more general case with both the start and end vertices are given. To the best of our knowledge, the best known algorithm is a $(2+\varepsilon)$-approximation given in [6] with a

[1]Contrary to vertices, every time an edge is traversed its cost is incurred.

running time of $n^{\mathcal{O}(1/\varepsilon^2)}$ where n is the number of vertices in the graph. For the special case of planar graphs with full connectivity, a $(1+\varepsilon)$-approximation is known [7], but is not applicable to our case. Alternatively, starting from the exact

formulation of the orienteering problem based on integer programming, specialized branch-and-cut methods have been proposed as well [9]. Such algorithms can deal with instances where the number of vertices is less than 1000, whereas in our domain we deal with graphs whose size is two orders of magnitude larger.

B. Robotics in precision agriculture

Mechanization and automation have been used in agriculture field operations for a long time. However, in recent years there has been a marked increase in the use of robotic systems for precision agriculture in a variety of domains, such as information gathering [19], yield estimation [3], pick assistance [18] and many others. The most similar work to ours is presented in [5], where routing problems for orchard operations are discussed. However, their problem differs from ours because they do not deal with a limited budget and are therefore interested in finding the optimal permutation for the rows to cover the entire orchard. In our case, due to the limited budget we also need to select a subset of vertices to cover that is compatible with the given budget, thus generating a different combinatorial problem.

III. Theoretical background and computational complexity

Classic textbooks such as [8] and [10] provide extensive introductions to the basics of computational complexity and

graph terminology, and we assume the reader is familiar with it.

We start defining a specific class of graphs, i.e., graphs that are bipartite, planar, and have degree at most 3. We indicate this class of graphs as $\mathcal{BP}3$. The following theorem due to Atai et al. [15] establishes the hardness of the Hamilton circuit problem for graphs in $\mathcal{BP}3$.

Theorem 1: The Hamilton circuit problem for bipartite planar graphs with maximum degree 3 is NP-complete.

An immediate consequence of this theorem is that the decision version of the TSP problem is NP-complete on $\mathcal{BP}3$.

Lemma 1: The decision version of the TSP problem is NP-complete on $\mathcal{BP}3$.

Proof. Recall that the decision version of the TSP problem asks whether for a given weighted graph and bound T there exists a tour of cost at most T. The classic reduction from Hamilton circuit to TSP is applicable to graphs in $\mathcal{BP}3$. That is, let $G = (V, E)$ be an instance of the Hamilton circuit problem. We then build an instance of the TSP with the same graph where all edge costs are 1 and we set $T = |V|$. The answer to the TSP instance is yes if and only if there exists a Hamilton circuit. ■.

For $\mathcal{BP}3$ graphs we next introduce two functions, i.e., a cost function $c : E \rightarrow \mathbb{R}_{\geq 0}$ and a reward function r :

$V \to \mathbb{R}_{\geq 0}$. Next, let us define the orienteering problem for a special class of graphs.

> **Constant Cost Orienteering on $\mathcal{BP}3$ Problem (CCOBP3P)**: let $G = (V, E)$ be a graph in $\mathcal{BP}3$, r a reward function and c a constant cost function, i.e., $c(e) = k$ for each $e \in E$. Moreover, let v_1, v_n be two vertices in V. For a given constant T_{MAX}, find a route of maximum reward from v_1 to v_n of cost no greater than T_{MAX}.

In the next theorem we show that even when restricted on this specific class of graphs the orienteering problem remains NP-hard.

Theorem 2: The CCOBP3 problem is NP-hard.

Proof: The proof is by reduction from the TSP problem on $\mathcal{BP}3$. Accordingly, let $G = V(V, E), c, T$ be an instance of the decision version of the TSP problem, where G is a $\mathcal{BP}3$ graph, c is the weight function for the edges, and T is the bound. Without loss of generality, let us assume that all costs of the edges in E are natural numbers. We build an instance of $CCOBP3P$ as follows: We start building a graph $G' = (V', E') = G$ and assign a reward $r(v) = 1$ to all vertices. Next, for each edge $e = (v_i, v_j) \in E$ with cost $c(e) = p > 0$ we introduce $p - 1$ new vertices in V' and p edges between v_i and v_j. The reward of each newly introduced vertex is set to 0 and the cost of each newly introduced edge is 1 (see Fig. 2 for this construction). Finally,

we set $T_{MAX} = T$.

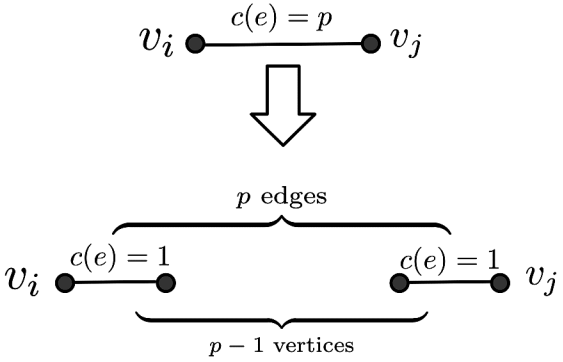

Fig. 2: Transformation

Note that the graph we just built is still planar and has maximum degree 3. However, it may not be necessarily bipartite. That is, for example, the case where the weight of and edge e between v_i and v_j is even, and therefore an odd number of vertices is introduced between v_i and v_j. This problem is easily remedied by introducing an extra vertex with 0 reward and two extra edges with 0 cost. This new graph is therefore also in $\mathcal{BP}3$. It follows that an instance of the CCOBP3P graph we just built has a solution of score T_{MAX} if and only if the answer for the TSP problem is yes. Therefore CCOBP3P is NP-hard. ■.

As a corollary of the above theorem, we can conclude that the orienteering problem with unconstrained costs is NP-hard on $\mathcal{BP}3$ graphs.

Having established the complexity of the orienteering problem for the $\mathcal{BP}3$ class of graphs, we now introduce a new class of graphs that is useful for the routing problems we consider in this paper.

Let $R(m,n)$ be the set of $m \times n$ couples of integers of type (i,j) with $1 \leq i \leq m$ and $1 \leq j \leq n$. In the following, without loss of generality[2], we assume $m, n \geq 3$. We consider an undirected graph $IG(m,n) = (V,E)$ where the set of vertices is $V = R(m,n)$ and the set of edges E is defined as follows:

- each vertex of type (i,j) with $1 < j < n$ has two edges, one towards $(i,j-1)$ and the other towards $(i,j+1)$.
- each vertex of type $(i,1)$ with $1 < i < m$ has three edges, i.e., one towards $(i-1,1)$, one towards $(i+1,1)$, and one towards $(i,2)$. Symmetrically, each vertex of type (i,n) with $1 < i < m$ has three edges, i.e., one towards $(i-1,n)$, one towards $(i+1,n)$, and one towards $(i,n-1)$.
- we add four additional edges: one between vertex $(1,1)$ and $(1,2)$; one between vertex $(1,1)$ and $(2,1)$; one between (m,n) and $(m-1,n)$ and one between (m,n) and $(m,n-1)$.

Figure 3 displays the structure of this graph. IG graph stands for *irrigation graph* because its structure captures the motion

constraints that a robot will face when navigating a vineyard or an orchard with an installed irrigation infrastructure (see also Figure 4). In a vineyard the robot can only move along rows, but to switch row it has to first reach either end of the row, i.e., a vertex of degree 3.

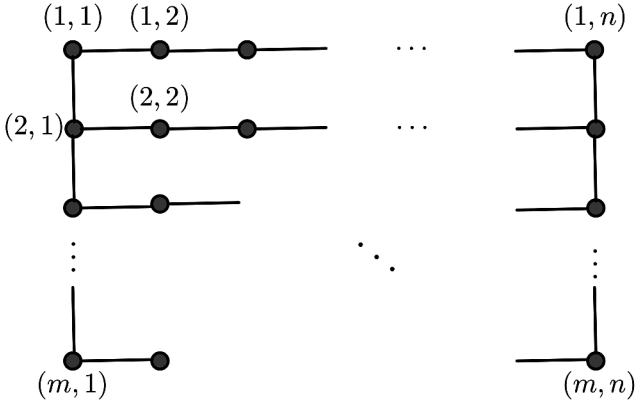

Fig. 3: Structure of the $IG(m, n)$ graph.

The reader should note that this graph is planar, and by construction each vertex has a degree of at most 3. Moreover, the graph is bipartite if one partitions the vertices between those for which $i + j$ is even or odd. Next, we introduce two functions, c (cost) and r (reward). The cost function c associates a non negative cost to each edge, i.e., $c : E \rightarrow$

$\mathbb{R}_{\geq 0}$, while the reward function associates a non negative reward to each vertex, i.e., $r : V \rightarrow \mathbb{R}_{\geq 0}$. For IG graphs, we define two related *orienteering* problems.

[2]If this is not the case, then the problem we introduce later on becomes trivial.

IG Orienteering Problem (IGOP): 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 a route of maximum reward starting at v_1 and ending at v_n of cost no greater than T_{MAX}.

IG Constant Cost Orienteering Problem (IGCCOP): let G be a graph $IG(m, n)$, $v_1, v_n \in V$ be two of its vertices and r be a reward function and c be a constant cost function on G, i.e, $c(e) = k$ for each $e \in E$. For a given constant T_{MAX}, find a route of maximum reward starting at v_1 and ending at v_n of cost no greater than T_{MAX}.

Theorem 3: The IGCCOP and IGOP problems are NP-hard.

Proof: The proof is immediate, observing that the hardness of IGCCOP follows from Theorem 2, while IGOP is NP-hard too because IGCCOP is a special case of IGOP. ■.

IV. ALGORITHMS

Having established the hardness of the orienteering problem on IG graphs, it is clear that a brute force approach is not applicable because typical instances we consider may have more than one hundred thousand nodes (see section V). In fact, commonly used benchmarks to test orienteering algorithms are based on instances with about 500 nodes. As an alternative, one could try using or adapting known approximation algorithms, like the one presented in [6]. However, such algorithms are complex to implement, and achieve at best a $(2+\varepsilon)$ approximation factor with complexity $n^{\mathcal{O}(1/\varepsilon^2)}$. Their value, therefore, appear to be limited in our case. Instead, we designed two heuristic approaches tailored to the specific problem at hand, and later on compare them with a general purpose heuristic that has been proposed in literature and is widely used.

In the following, let v_s be the start and ending vertex, i.e., each path computed by the algorithms we present will start and end at v_s. We assume that $T_{max} < mn$, otherwise a full row-by-row sweep of the graph is possible and the solution is therefore trivial (recall that all edges cost 1). In general we could assume $T_{max} = kn$ for a known $k < m$, i.e., the budget allows to cover at most k rows. Note that, however, the robot will generally cover less than k rows because part of the budget is in general spent to move between non-adjacent rows.

A. Greedy Row Heuristic Algorithm

The first heuristic we consider implements a greedy approach producing a path that selects a subset of rows to traverse. Algorithm 1 sketches the pseudocode for the strategy we propose. The algorithm keeps track of the already spent budget T (initially set to 0 in line 2) and of the current position of the robot v (initially set to v_s in line 3). For each of the m rows, a cumulative reward R_i is preliminarily computed, where $R_i = \sum_{j=1}^{n} r(i,j)$ is the reward collected by completely traversing the i-th row (line 4), and all rows are initially marked as feasible (line 5). The algorithm then enters a loop adding feasible rows to the path (loop at line 6). At each iteration the algorithm tests each row for feasibility, and if this is not the case it is marked as unfeasible (lines 7 to 9). A row is feasible if it is possible to move the robot from the current vertex v to the row, traverse the entire row, and still have enough budget to return to v_s. This computation uses the already spent budget T and the current vertex v. The rewards of the remaining rows are then adjusted by dividing them by the cost it takes to reach the row from v and traverse it entirely (line 11, where $c(v,i)$ is the cost of going from the current vertex v to row i and then traverse it). The row with the highest adjusted reward is then selected (line 12), the path is adjusted (line 13 and 14), and the spent budget and current position are updated (line 15). The row added to the path is also marked as unfeasible (line 16), so it will

not be considered again in the following iterations. Once no rows pass the initial test discarding unfeasible rows, the main loop terminates, and the the robot returns to the starting point v_s (line 17). Note that by construction this is always possible because the robot will not select a row to traverse unless there is sufficient budget to return to the starting point. Therefore the algorithm always produces a valid tour starting and ending at v_s and of cost less than T_{MAX}.

Algorithm 1 Greedy Row Heuristic

1: $V_{tour} = \emptyset$
2: $T = 0$
3: $v = v_s$
4: **for** $i \leftarrow 1$ **to** m: $R_i \leftarrow \sum_{j=1}^{n} r(i,j)$
5: **for** $i \leftarrow 1$ **to** m: $feasible_i \leftarrow$ **true**
6: **while** there exists feasible rows **do**
7: **for** $i \leftarrow 1$ **to** m **do**
8: **if not** feasible(i, T, v) **then**
9: $feasible_i \leftarrow$ **false**
10: **for all** feasible rows **do**
11: $R'_i \leftarrow R_i / cost(v, i)$
12: $nextrow \leftarrow \arg\max R'_i$
13: add path from v to $nextrow$ to V_{tour}
14: add all vertices in $nextrow$ to V_{tour}
15: update v and T
16: $feasible_{nextrow} \leftarrow$ **false**
17: add path from v to v_s to V_{tour}
18: **return** V_{tour}

After at most $T_{MAX}/n = k$ iterations of the main loop, the algorithm terminates because at each iteration the used budget increases by at least n units. Each iteration has complexity $\mathcal{O}(m)$ since all rows are examined at every iteration. Hence, recalling that $k < m$, the complexity of this algorithm $\mathcal{O}(mn + km) = \mathcal{O}(mn + m^2)$. For the common case of square growing blocks, $m = n$ and therefore the previous expression is $\mathcal{O}(m^2)$, i.e., the algorithm is linear in the number of vertices in the graph.

B. Greedy Partial-Row Heuristic Algorithm

Our next heuristic is designed by observing that moisture is a continuous phenomenon in the soil, and wet or dry regions can span multiple rows. In terms of rewards for the irrigation graph, this means that we often have clusters

of vertices in adjacent rows that have similar rewards. For example, this is evident when observing Figure 5, where we see clusters of vertices associated with higher rewards (warmer colors). When such high-reward clusters are located near to either side of the graph, it may be convenient to visit just one part of the row and then come back without collecting any more rewards - so as to switch to the adjacent row - rather than continue the entire way through the row before moving to the adjacent row. The name *Partial-Row*

Heuristic describes this behavior and Algorithm 2 sketches its pseudocode.

To implement this strategy, we preliminarily compute two reward values for each vertex in the graph (loop starting at line 4). $L(i, j)$ is the cumulative reward collected if the robot starts from the left side of the row and stops at the j-th vertex in the row, whereas $R(i, j)$ is the cumulative reward if the robot starts from the right and stops at the j-th vertex. The greedy partial-row algorithm then works similarly to the previous algorithm, but each of the R, L rewards is adjusted in two different ways. The first scales rewards by the cost of entering the a row, reaching vertex (i, j) and then coming back, as in the loop at line 12. The R and L rewards are scaled by different travel costs accounting for the fact that they can be reached coming from the right side of the grid, or the left side of the grid. The second, shown in the loop at line 18, scales the reward of each full row by the cost of entering from either the left side or the right side. Next, if the robot is positioned at the right end side of the graph ($v_j = n$) we select the best scaled reward from those entering the rows from the right (line 22), whereas if the robot is located at the left end we select the best scaled rewards from the left (line 24). The tour is then updated together with the relevant variables.

As in the full row case, the algorithm stops iterating when there is not enough budget to ensure that a robot can move to a new vertex and still go back to v_s. The complexity of this

algorithm is similar to the previous one, with the difference that now at each iteration all vertices must be re-evaluated, so each iteration has complexity $\mathcal{O}(mn)$ and the algorithm is $\mathcal{O}(mn + kmn) = \mathcal{O}(m^2 n)$.

C. S-Algorithm

The *S-Algorithm* is one of the most used and effective general purpose heuristic algorithms to solve the orienteering problem [21], and we therefore use it as a benchmark against our heuristic approaches. The S-Algorithm is a stochastic algorithm (hence the name) implementing a Monte Carlo approach whereby many routes respecting the given budget are generated, and the one associated with the highest reward is eventually returned. The algorithm builds a tour starting from the start vertex and iteratively adds a new vertex to the tour until the residual budget allows only to return to the start vertex v_s. Assuming the last vertex added to the path is v_i, to decide the next vertex a *desirability* function $A(j)$ is computed for every vertex v_j not yet visited. The desirability

Algorithm 2 Greedy Partial-Row Heuristic

1: $V_{tour} = \emptyset$
2: $T = 0$
3: $v = v_s$
4: **for all** $v(i, j) \in V$ **do**

$R(i,j) \leftarrow \sum_{l=j}^{n} r(i,l)$

$L(i,j) \leftarrow \sum_{l=1}^{j} r(i,l)$
$feasible_{i,j} \leftarrow$ **true**
while there exists feasible vertexes **do**
  for all $v(i,j) \in V$ **do**
    if not feasible(i,j,T,v) **then**
      $feasible_{i,j} \leftarrow$ **false**
  for all feasible verticies **do**
    $R'_{i,j} \leftarrow R_{i,j}/cost(v, v(i,j), v(i,n))$
    $L'_{i,j} \leftarrow L_{i,j}/cost(v, v(i,j), v(i,1))$
  for $i \leftarrow 1$ **to** m **do**
    if not feasible(i,T,v) **then**
      $feasible_i \leftarrow$ **false**
  for all feasible rows **do**
    $R'_{i,1} \leftarrow R_{i,1}/cost(v, v(i,1))$
    $L'_{i,n} \leftarrow R_{i,n}/cost(v, v(i,n))$
  if $v_j = n$ **then**
    $next \leftarrow \arg\max R'_{i,1}, R'_{i,j}$
  else
    $next \leftarrow \arg\max L'_{i,n}, L'_{i,j}$
  add path from v to $next$ to V_{tour}
  if $next_j \neq 1$ or n **then**
    add path from $next$ to $v(i,v_j)$ to V_{tour}
  update v and T
  $feasible_{pathtonext} \leftarrow$ **false**
add path from v to v_s to V_{tour}
return V_{tour}

is defined as

$$A(j) = \left\{ \frac{c(v_j) + E}{d(v_i, v_j)} \right\}^r$$

where $d(v_i, v_j)$ is the length of the shortest path between v_i and v_j, r is a parameter, and

$$E = a[T_{MAX} - T - d(v_i, v_j) - d(v_j, v_s)] \cdot n(v_j).$$

In this last expression, a is a parameter, T is the budget consumed so far, and $n(v_j)$ is a *neareness* measure defined as $n(v_j) = \sum_{v \in V} \frac{r(v)}{d(v, v_j)}$. Essentially, $A(j)$ is large for nodes with large rewards that are close to the current vertex and would leave more residual budget. After the desirability of each vertex has been computed, the next node added to the tour is randomly selected using a roulette-wheel method utilizing the desirability to compute the selection probability. The performance of the algorithm in terms of collected reward depends on various parameters, the most critical of which is the number of different paths to be generated. There is an obvious linear dependency between this number and the overall time spent by the algorithm, and our experience shows that for IG graphs the S-algorithm needs to generate tens or hundreds of paths to produce a viable result.

The cost of generating a single path is $\mathcal{O}(mnl)$ where l is the number of iterations before the budget T_{MAX} is fully spent. Of course, this is a random variable, and an expected value is difficult to determine with a formal analysis. We

observed that this is typically in the order of m, and therefore

the complexity to produce a single path is approximately $\mathcal{O}(m^2n)$ with an overall complexity of $\mathcal{O}(Cm^2n)$ where C is the number of sample paths generated by the Monte Carlo method. Note that the time complexity of generating a single sample path is the same as the overall complexity of the partial-row heuristic we presented in the previous subsection.

Unless otherwise stated, we set the parameters $a = 1.0$, $r = 4.0$ and $C = 3000$ as specified in the original text [21] for our experiments.

V. Experiments

We tested the algorithms we developed using a concrete routing problem associated with a commercial vineyard. To this end, soil water measurements were manually collected on the ground with a manual probe (Hydrosense HS2P by Campbell Scientific) equipped with a GPS receiver providing the location where the sample was taken. Figure 4 shows the vineyard and the locations of the measurements. At each site, multiple measurements were taken in close proximity to each other and were then averaged in postprocessing.

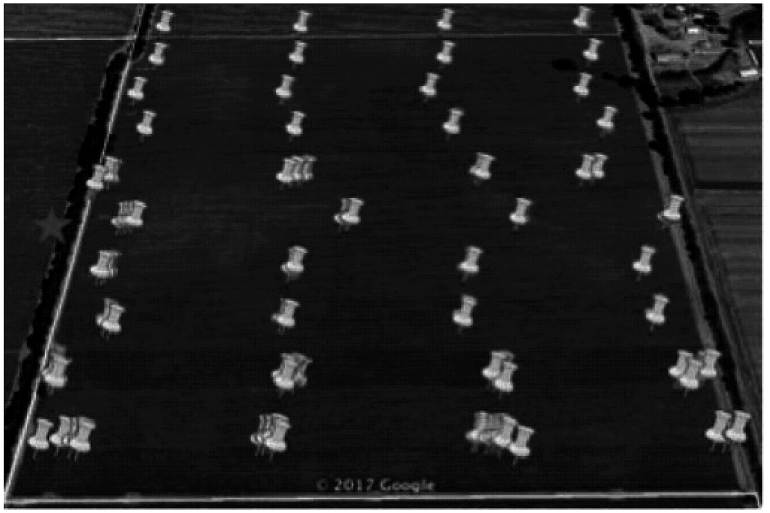

Fig. 4: Location of samples collected in a vineyard located in Snelling, CA. Multiple pins are displayed at each location because multiple samples were connected on the field and then averaged in postprocessing. Note the tree rows extending from left to right. To switch tree rows, the robot must first reach one of the two roads at either side of the block. The red star shows the location where the robot starts from and needs to return to, i.e. the location of vertex v_s.

Due to the vast extent of the vineyard, samples are taken at a limited number of locations, and a moisture map is reconstructed for each point in the field using a fitting

algorithm. In our experiments we used a linear function to interpolate within sampling locations and a nearest neighbor to extrapolate outside the area where samples were collected. Other approaches, such as krigging, are of course possible but inconsequential to evaluate the strengths and limitations of the algorithms we consider. The graph associated with this vineyard has the same structure shown in figure 3 with 500 columns and 240 rows, for a total of 120,000 vertices. The reward associated with each vertex is $r(v) = |T - m(v)|$ where T is a user supplied constant, i.e., the desired uniform soil moisture level, and $m(v)$ is the moisture at vertex v as inferred by the interpolation algorithm. In essence, the reward associated with a vertex is the mismatch between the actual and desired soil water content and vertices with a high reward indicate sites that are either overwatered or underwatered. Figure 5 shows the spatial distribution of the rewards. Warmer colors indicate vertices with higher values for $r(v)$. In all our tests the robot starts at the left end side of the vineyard, midway through the field, i.e., in correspondence of row 120 (location marked with a red star in Figure 4). Within the allotted budget, the robot needs to return to the starting point so that it can be retrieved or the battery can be swapped.

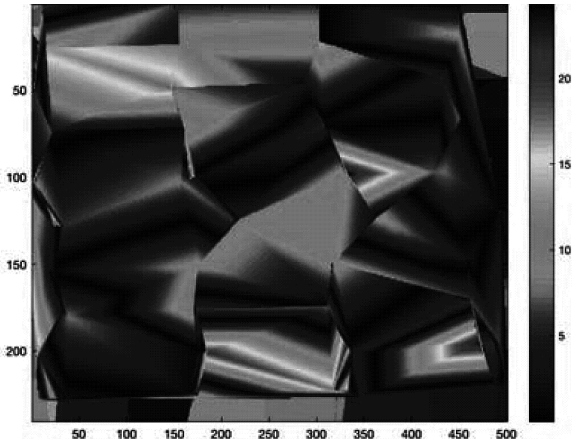

Fig. 5: Distribution of the rewards $r(v)$ for the graph associated with the vineyard shown in Figure 4.

In the first test we compare the various heuristics with the optimal solution, determined using a well known constrained integer programming formulation (see e.g., [14]). Note that the number of binary optimization variables scales quadratically with the number of vertices in the graph and this severely limits the size of the problems we can study. To solve the integer program, we use the freely available SCIP solver [16]. Specifically, within reasonable time[3] we can solve only problem instances with up to 8 rows and 12 columns i.e., settings considering only 96 emitters, whereas a

vineyard block in our case has 120,000 emitters, and a typical vineyard ranch consists of tens of blocks. Figure 6 contrasts the total reward collected by the various algorithms as a function of the allocated budget. In this case the S-algorithm was run with $C = 3000$. Note that for budget values greater than or equal to 110, the reward plateaus because all emitters can be reached and adjusted. The chart shows that for this limited size, the partial row heuristic and the stochastic heuristic are relatively close to the optimal solution. The chart also hints that using approximation algorithms is not an appealing proposition because heuristic approaches seem to largely overcome the guaranteed $(2 + \varepsilon)$ bound without incurring the corresponding time complexity (see discussion in Section II and Section IV).

[3]Even for slightly larger problems, the integer programming solution would not be completed even after more than 24 hours of computation on an i7 quadcore processor running at 4.2 GHz.

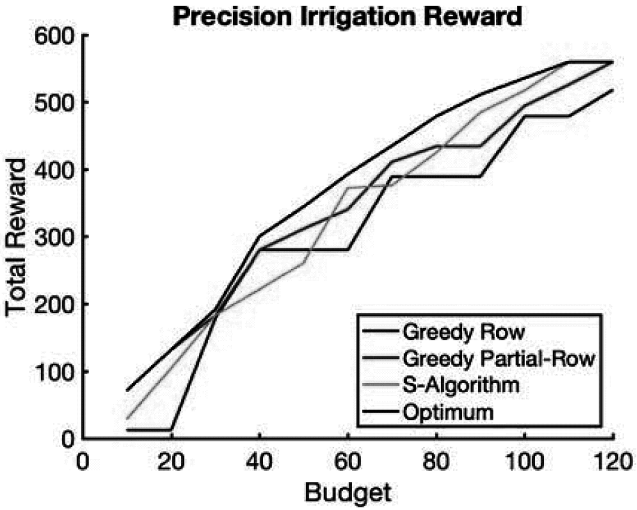

Fig. 6: Comparison with the optimum for a small problem instance.

Figure 7 displays the paths produced by the different algorithms for the small case with a budget of 60. The plot confirms the ability of the partial row algorithm to cover just certain sections of the rows. A similar behavior is also displayed by the optimal solution and the S-algorithm, while the greedy row algorithm only manages to cover the top and bottom rows.

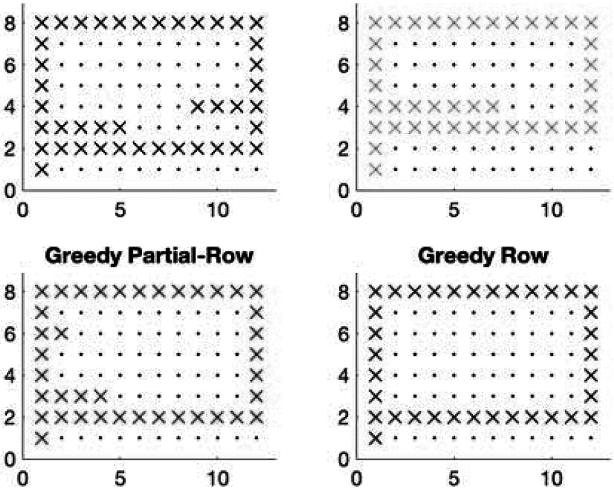

Fig. 7: Comparison between the different paths produced by the algorithms (starting vertex in this case is $(1, 1)$).

For larger problem sizes, the optimal solution cannot be computed within a reasonable time frame and we therefore compare only the various heuristics. Figure 8 shows the accumulated reward as a function of the travel budget for the three heuristic algorithms for a 60×60 patch. To speed up the computation of the S-algorithm in this case we reduced the number of samples to $C = 500$.

Note that in this case with a larger graph and a smaller number of samples, the greedy partial row algorithm pre-

forms better than the S-algorithm in terms of total reward.

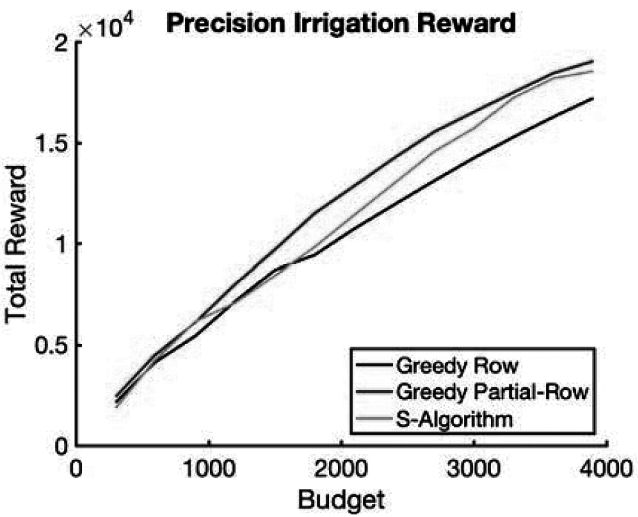

Fig. 8: Reward for the different algorithms as a function of the budget.

However, to put this result into perspective one also should consider that the S-algorithm is approximately C times slower than our heuristic. Figure 9 and 10 show the results for the full 240×500 graph for the two algorithms we proposed as a function of the budget. Results are averaged over three reward maps for the same vineyard generated by soil moisture measures collected at different times throughout the

growing season. Since different maps give different overall reward, we plot the normalized reward, i.e., the ratio between the collected reward and the total possible reward. The plot shows that for larger budgets the discrepancy between the two algorithms tends to increase and therefore the partial row greedy algorithm should be preferred.

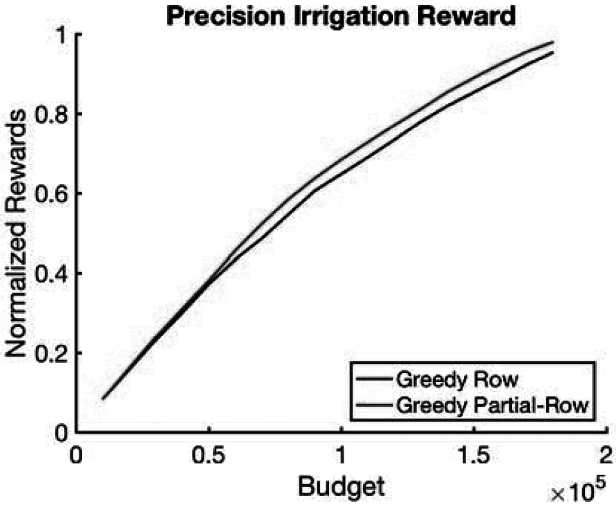

Fig. 9: Normalized average collected reward as a function of the budget for the full 240×500 irrigation graph.

Figure 10, in particular, shows the average unspent budget by the two algorithms, i.e., the budget still available once the robot returns to the starting point. The chart shows that

that the partial row algorithm not only picks a subset of nodes yielding a higher reward, but also is more effective in using the available budget. Both algorithms we proposed are greedy, and it is possible to build special cases where the algorithms perform very poorly. For example, one can

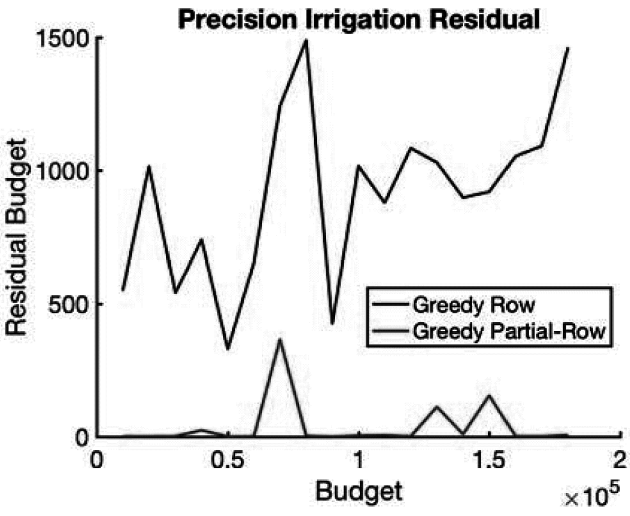

Fig. 10: Average unused budget as a function of the budget for the full 240×500 irrigation graph..

build a rewards map that will cause the algorithm to produce paths alternating between different extremes of the map, e.g.,

first visit one vertex in the top row, then one in the bottom row, then back in the top row, etc. In this case most of the budget would be wasted moving between vertices without collecting rewards. However, due to the fact that rewards are determined by the underlying soil moisture map and that this is a continuous physical phenomenon, such pathological situations are unlikely to happen in practice, i.e., once the robot visits a location because it has high reward, nearby locations will have similar rewards and will then be visited before moving to other areas.

VI. Conclusions

In this paper we studied a routing problem that emerges when autonomous robots are deployed for precision irrigation where motions are constrained once a robot enters a row. We showed that the problem is NP-hard and then presented two domain-specific heuristics. One of them outperforms the widely used S-algorithm heuristic for orienteering, while also being much faster. In the future we anticipate expanding this research in various directions. First, we will consider the case of multiple robots servicing the same vineyard and address the additional constraint that two robots cannot completely travel the same vine-row in opposite directions. Second, we will study how to introduce additional costs (e.g., latency) in the same domain, as well as uncertainties in navigation and emitter adjustment.

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 also thank Jose Manuel Gonzalez and Carlos Diaz Alvarenga for assisting with data collection in the field.

REFERENCES

[1] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28(1):254–262, 1999.

[2] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 166–174, 2004.

[3] S. Bargoti and J.P. Underwood. Image segmentation for fruit detection and yield estimation in apple orchards. *Journal of fields robotics*, 34(6):1039–1060, 2017.

[4] 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.

[5] D. Bochtis, H.W. Griepentrog, S. Vougioukas, P. Busato, R. Berruto, and K. Zhou. Route planning for orchard operations. *Computers and electronics in agriculture*, 113:51–60, 2015.

[6] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, 8(3), 2012.

[7] K. Chen and S. Har-Peled. The orienteering problem in the plane revisited. *SIAM J. on Computing*, 38(1):385–397, 2008.

[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2009.

[9] 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.

[10] M.R Garey and D.S. Johnson. *Computers and Intractability. A guide to the theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[11] N. Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 396–402, 2005.

[12] 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.

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

[14] 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.

[15] A. Itai, C.H. Papadimitriou, and J.L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[16] S.J. Maher, T. Fischer, T. Gally, G. Gamrath, A. Gleixner, R.L. Gottwald, G. Hendel, T. Koch, M.E. Lübbecke, M. Miltenberger, B. Müller, M.E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, D. Weninger, J.T. Witt, and J. Witzig. The SCIP optimization suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin, 2017.

[17] 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.

[18] 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 fields robotics*, 34(6):1140–1159, 2017.

[19] P. Tokekar and V. Isler. Large scale image mosaic construction for agricultural applications. *IEEE Robotics and Automation Letters*,

1:295–302, 2016.

[20] D. Tseng, D. Wang, C. Chen, L. Miller, and K Goldberg. Learning to infer ground drainage conditions from aerial agricultural images for automating precision irrigation. In *Proceeding of the IEEE International Conference on Robotics and automation*, 2018 (under review).

[21] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.