Cool, Cool Mountain Orienteering Problem: Greedy Solution and Simulation for Distance Acceleration

Joey Vongphasouk
Colorado School of Mines
joeyvongphasouk@mines.edu

***Abstract*—To gain a better understanding of autonomous robot systems, we study and analyze a variation of the Orienteering Problem called the Linearly Accelerating Orienteering Problem (LAOP) where the robot is able to linearly accelerate across distance. This problem is modeled using a 2-dimensional grid map and with rewards associated with each cell. We then develop a greedy algorithm that accounts for the linear acceleration and continuously chooses the path with the most reward whilst respecting remaining distance. We show that this greedy solution is normalized, but not submodular nor monotonic, and thus show that no guarantees can be made as a result. This situation is then simulated using the Robot Operating System framework with the RViz simulation tool. Finally, we compare this algorithm with the general greedy solution for the Orienteering Problem through the ROS simulation.**

***Index Terms*—Orienteering Problem, Greedy Algorithm, Linear Acceleration, Robot Operating System, Jazzy Jalisco**

I. Introduction

Modern applications have seen an increasing reliance on automated tasks. Sales of robots have been shown to be increasing in many different industries, and studies have shown that fields with automated robots have seen general labor productivity growth and output cost reductions [1] [2]. These patterns showcase the reach and market share of applied robots in fields of autonomous work, and so furthering the understanding and knowledge of robots and their capabilities will further improve the industries and general lives. To improve our knowledge of autonomous robots, the team will look to document the exploration of applied robotics in a variation of the Orienteering problem that considers linear acceleration.

Cool, Cool Mountain is a level in the video game Super Mario 64 where the player is placed in an icy environment, allowing for the player to accelerate across straight distances [3]. One of the challenges the player faces is to gather as many coins as possible given a time limit, where different areas of a map have more coins to gather than other areas. This challenge served as inspiration for this paper, providing a basis and a reasonable starting point for exploration in path planning. The goal of this paper is to create a path planning algorithm using this scenario that maximizes the reward gained in the path from a defined starting position to a defined ending position.

Figure 1 depicts an abstracted version of the challenge found in the game. The player must collect as much coins within a set time limit and end at the goalpost. The coins are also

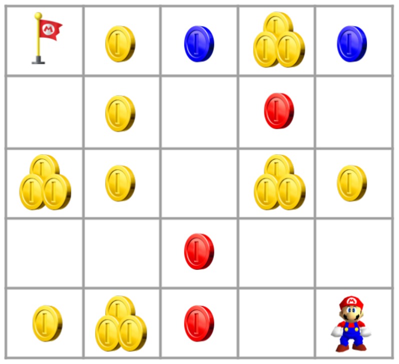
Fig. 1. Example traversable and reward map in Cool, Cool Mountain

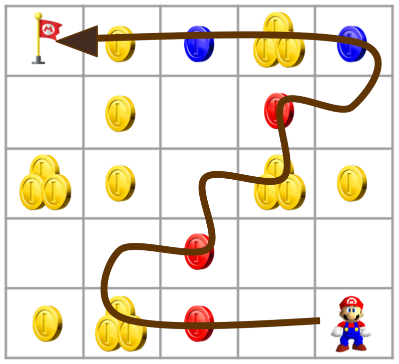
Fig. 2. Greedy plan without respect to linear acceleration

unevenly distributed across the map. Figure 2 illustrates the greedy approach for collecting as many high-valued coin cells within 14 cells. However, this approach fails to meet the time limit imposed on the player as the player stops and turns 8 times, slowing the player down. Figure 3 illustrates another greedy approach within 14 cells, but reaches the end goal within the time limit as the player stops and turns only 4 times, and can accelerate across continuous distances.

Additionally, the Robot Operating System (ROS) is explored in this project. ROS is a widely-used framework that combines each aspect of the robotics development cycle into an environ-

Dr. Micah Corah, Department of Computer Science at Colorado School of Mines

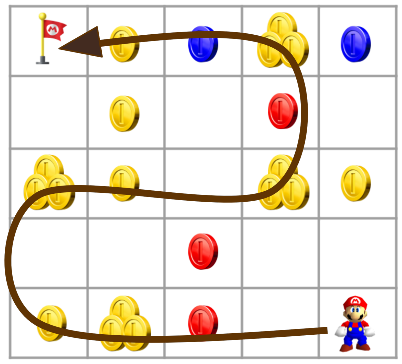

Fig. 3. Greedy plan with respect to linear acceleration

ment that can easily implement all parts. Furthermore, ROS has tools that can simulate robots in a controlled environment without needing the actual physical components for its robot implementation. This framework will be used in this paper to simulate the greedy solution approaches found in this paper.

II. Related Work

A. Orienteering Problem

Golden et al. first introduce the Orienteering Problem [4]. Closely related to the Generalized Traveling Salesman Problem, the Orienteering Problem is a path optimization problem where a vehicle aims to maximize the total reward gained from visiting a subset of known locations within a given time or distance. Many variants of this problem have been described in recent years and our paper aims to do something similar. We modify the original Orienteering Problem introduced by Golden such that the vehicle is placed on a 2-dimensional grid of cells and may accelerate across linear distances. This is different from Golden's original approach, which viewed the problem as a graph $G = (V, E)$ and both the vertices V representing points of interest and edges E representing distance between vertices were static.

Other recent works have extended the Orienteering Problem by introducing vehicle constraints. Penicka et al. introduce the Dubins Orienteering Problem, which modifies the Orienteering Problem by placing the situation onto a 2-dimensional reward grid and introducing both a time constraint and a turning constraint onto the vehicle [**?**]. This added constraint restricts paths by removing possible edges the vehicle can take at a given node dependent on the current heading angle. Penicka et al. use a Variable Neighborhood Search algorithm to address this issue. Our paper similarly uses a 2-dimensional reward grid to model the environment, but our paper uses a different constraint where the preceding distance traveled can affect the time used to get to that point.

Another Orienteering Problem Variation considers starting time. Fomin et al. have developed the Time-Dependent Orienteering Problem where the time cost from traveling between two nodes is dependent on the time for which the vehicle enters the connected node [6]. This is solved using a greedy algorithm where time is divided into measurable intervals and each time interval is greedily performed on. Our paper similarly uses a greedy approach to obtain a maximized path, however our solution utilizes a reward grid and edges are modified by distance traveled from a starting point rather than modified by the current time interval.

B. ROS

Another point of interest for related work is the implementation of autonomous algorithms in ROS. Due to the team's inexperience with ROS, many resources have been consulted to gain an adequate enough understanding to simulate the project. Lentin Joseph provides a book on ROS programming and discusses many of the inner workings of the ROS framework [7]. This source provided the team some starter programs explaining key concepts for which we had expanded on to fully realize the project simulation. Our usage of ROS will be different from the ones seen here as we will be looking to simulate using Rviz rather than Gazebo which was used in Joseph's book. The team had also consulted multiple online tutorials to gain a fuller understanding of ROS [8] [9].

Koubâa also goes into depth on ROS implementations regarding algorithms and simulations [10]. In the written book, Koubâa looks to analyze different papers regarding the implementation of various projects on various ROS distributions. Most of these papers abstract out basic movements of the robot in favor of more complex decisions and algorithms. Whilst our paper is similar in implementing a created algorithm, the ones shown in Koubâa's paper do not go into the actual specifics of ROS implementation, which we will be including.

III. Problem Statement

The LAOP problem is formally defined as per the following statements. The environment is a 2-dimensional grid map G of size $m \times n$. Each cell in the grid map is associated with a non-negative reward and can be obtained by the reward map $RMAP$ of similar size $m \times n$ through the usage of $w = RMAP(x, y)$, where w represents the reward gained at cell (x, y). The reward map is pre-determined and will be known before the simulation. An example grid and reward map can be found in Figure 1. The vehicle is placed at the starting position $s \in G$ and must end at the ending position $e \in G$ within the max time t_{max}. The vehicle may only travel to the cells directly horizontal or vertical to the cell it is currently occupying. The robot will also linearly accelerate in either horizontal or vertical directions as it travels. This means that the travel time to the next cell in its current direction will decrease as the robot moves in that direction.

The objective is to find a path from the starting position s to the ending position e that is within the time budget t_{max} and maximizes the amount of total reward covered from each cell. Mathematically, let $P = \{(x_s, y_s), (x_0, y_0), (x_1, y_1), ..., (x_e, y_e)\}$ which represents a path from the starting position $s = (x_s, y_s)$ to the ending position $e = (x_e, y_e)$. Additionally, let the time taken to traverse path P be t_P, and let the set of all possible paths be Ω. Then the objective can be defined with the respected time constraint:

$$\max_{P \subseteq \Omega} \sum_{(x_i, y_i) \in P} RMAP(x_i, y_i)$$

$$t_P \leq t_{max}$$

The LAOP objective is now defined.

IV. Approach

The general approach that the team has come up with is a greedy algorithm that chooses the next best path with the most reward. Additionally, the reward is modified such that the consecutive distance traveled affects the reward gained from choosing the current path. This approach keeps the project in scope given the team's provisional knowledge on related algorithms and its implementations in ROS.

A. Reward Modification Helper

The problem statement looks for the best path that the vehicle can take such that the path gives the most reward. As a result, the reward map and function is modified to then be based on the sum of all rewards from the map to that point. Refer to Algorithm 1: CalculateRewards for the implementation of the rewards. To both adhere to the limited movement of the problem and let the robot travel as much as possible in straight lines without turns, the robot is assumed to travel in two lines to the cell wanted. This then allows for two possible paths to the cell from the current position of the robot for which to calculate the reward to that cell. Both paths are taken into consideration and returned.

Additionally, the paths from each straight line are divided by $\log(Dist + 1)$, where $Dist$ is the distance of that straight line. This division captures the linear acceleration, where the increase in $Dist$ to the overall calculation is decreased over the distance traveled. An example can be referred to in Table 1, which shows the time taken to get to the specific cell which relies on the $\log(DistX + 1) + \log(DistY + 1)$ calculation. In this table, it can be seen that the increase in travel time from the start to a desired cell slowly decreases the further the designated cell is. For example, the difference in travel time from (0, 1) and (0, 2) is 0.38 units, whilst the difference between (0, 5) and (0, 6) is 0.29 units. This concludes the modification in rewards at each cell.

TABLE I
Table of Times to Reach Specific Coords

(x, y)	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)
(0, 0)	START	1.44	1.82	2.16	2.49	2.79	3.08
(1, 0)	1.44	1.82	2.16	2.49	2.79	3.08	3.37
(2, 0)	1.82	2.16	2.49	2.79	3.08	3.37	3.64
(3, 0)	2.16	2.49	2.79	3.08	3.37	3.64	3.91
(4, 0)	2.49	2.79	3.08	3.37	3.64	3.91	4.17
(5, 0)	2.79	3.08	3.37	3.64	3.91	4.17	4.43
(6, 0)	3.08	3.37	3.64	3.91	4.17	4.43	4.68

Performed on a uniform time 7x7 map

Algorithm 1 CalculateRewards
Input: CURR_P, cell, R_MAP
Output: xy_rw, yx_rw

```
1: xy_rw = 0
2: yx_rw = 0
3: horizontal_dist = $|cell.x - CURR\_P.x|$
4: vertical_dist = $|cell.y - CURR\_P.y|$
5: xy_rw += $\frac{\sum_{y=CURR\_P.y}^{cell.y} R\_MAP[CURR\_P.x][y]}{\log(vertical\_dist+1)}$
6: xy_rw += $\frac{\sum_{x=CURR\_P.x}^{cell.x} R\_MAP[x][cell.y]}{\log(horizontal\_dist+1)}$
7: yx_rw += $\frac{\sum_{x=CURR\_P.x}^{cell.x} R\_MAP[x][CURR\_P.y]}{\log(horizontal\_dist+1)}$
8: yx_rw += $\frac{\sum_{y=CURR\_P.y}^{cell.y} R\_MAP[cell.x][y]}{\log(vertical\_dist+1)}$
9: return xy_rw, yx_rw
```

B. Greedy Algorithm for Linearly Accelerating Paths

Following other greedy implementations, Algorithm 2: GreedyPath, is the algorithm used to greedily choose the next best path for the vehicle to take. Looping through each cell within the grid, the algorithm first decides if the cell seen will be able to still reach the end goal if taken. This is done in a similar manner to the linear distance calculation in the reward helper function. If so, the loop then calculates the reward from both paths to the cell and checks to see if the reward at such cell is greater than the current best reward. This operation then replaces the best path and corresponding variables.

The algorithm is ran until the returned BEST_TIME is 0. At this point, the algorithm paths to the end, as no more cells can be reached in conjunction with the end cell.

Algorithm 2 GreedyPath
Input: CURR_P, END_P, R_MAP, CURR_T, MAX_T
Output: BEST_PATH, BEST_TIME

```
1: BEST = 0
2: BEST_PATH = []
3: BEST_TIME = 0
4: for all cell ∈ R_MAP do
5:     T_taken = CURR_T + $\log(|CURR\_P.x - cell.x| + 1)$ + $\log(|CURR\_P.y - cell.y| + 1)$ + $\log(|cell.x - END\_P.x| + 1)$ + $\log(|cell.y - END\_P.y| + 1)$ < MAX_T
6:     if T_taken < MAX_T then
7:         xy_rw, yx_rw = CalcRw(CURR_P, cell, R_MAP)
8:         if xy_rw > BEST then
9:             BEST = xy_rw
10:            BEST_PATH = XY-path
11:            BEST_TIME = T_taken
12:        end if
13:        if yx_rw > BEST then
14:            BEST = yx_rw
15:            BEST_PATH = YX-path
16:            BEST_TIME = T_taken
17:        end if
18:    end if
19: end for
20: return BEST_PATH, BEST_TIME
```

C. Submodularity

It can be reasoned that the problem statement and algorithm is not submodular. This is due to how the algorithm handles the problem environment and how linear acceleration can vary the time taken to cross different edges. This property violates the diminishing returns property that submodularity upholds, which states that the marginal gain from adding to a set should decrease as that set grows bigger. Or mathematically, for a set function g, submodularity requires that for any subsets $A \subseteq B \subseteq \Omega$ and $C \subseteq \Omega \setminus B$, then $g(C|A) \geq g(C|B)$ holds true.

Submodularity may not be able to hold with consideration of linear acceleration, as adding a new edge to a larger path may result in a better outcome than adding that same edge to a shorter path. That is, the vehicle may be accelerating at such a high rate that the modified reward from part A of adding that path to the larger set can be better than a robot accelerating at a much slower rate. Therefore, we can conclude that this algorithm and problem environment is not submodular.

D. Monotonicity

The algorithm can also be shown to be non-monotonic. A set function g is monotonic if the reward of adding an element to the set will always stay the same or increase in reward gained. Or mathematically, for any subsets $A \subseteq B \subseteq \Omega$, then $g(A) \leq g(B)$.

Adding another cell to a given path may decrease the reward given at that path due to how the modified reward helper calculates using distance as a dividend. Thus, if the average reward of adding that element does not make up for the added distance, then the division will make the overall value of going through that path lower. Therefore, it is shown that the problem and algorithm is not monotonic.

E. Normality

The algorithm is normalized, as the set function here to calculate the path reward is 0 when the path is empty.

F. ROS Simulation

The simulation uses the ROS2 Jazzy Jalisco framework and a custom workspace and Python package. Additionally, the simulation was created using the RViz2 3-dimensional visualization tool. This was chosen in favor of the Gazebo ROS visualization tool as such was complicated to setup and better aligned with the team's goal in better understanding the autonomous robot development cycle. Additionally, multiple online sources have helped the team develop the project to a state where a simulation can be made [7] [8] [9].

To then compare simulation results, both a normal greedy reward algorithm and the modified path reward algorithm were implemented. Both simulations had been given a max time t_{max} of 40 units. Additionally, a heat map of rewards (where warmer colors have higher rewards) was used along with a marker array to show the path that the robot had taken. The start cell is at (0, 0) which is located at the closest corner in Figure 4, whilst the end cell is located at the farthest corner.

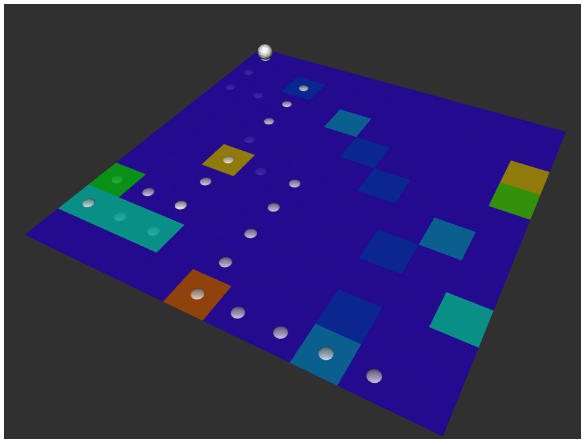
Fig. 4. Greedy path without modified reward metric

V. Results

The analysis of the algorithm above shows that the set function used to greedily calculate the path is neither submodular nor monotonic. As a result, no provable guarantees can be made relating the efficiency of this algorithm to that of the optimal route. However, the team was able to create a simulation using the ROS2 framework and RViz2 visualization tool. Figure 4 shows a simulation with an approach that greedily chooses based on the highest reachable reward. This simulation had returned a score of 34 when ran in the simulation. Figure 5 shows a simulation with an approach that greedily chooses based on the highest reward path. This implementation had returned a score of 42. This shows that the algorithm had given a better result when accounting for linear acceleration than through a normal reward metric.

Additionally, the ROS2 visualization and code had given much insight to the team on how to implement and simulate theoretical algorithms. This simulation required the usage of multiple publisher types across different topics, as well as the connection of such to a backend compuational method. Further demos were also created whilst creating the simulation, which had explored subscribers, servers, and various package manipulation methods. In total, the team had learned much about ROS and its usages in autonomous systems.

VI. Future Work

A. Physical Implementation

Using the ROS framework allows for easy translation between a simulated environment and a physical environment. Whilst the paper's main goal is to provide an algorithm and simulation that considers linear acceleration for the Orienteering Problem, implementation in physical environments may provide further insight into actual behaviors of the robot. Physical factors that were not considered, such as robot surface coverage/physics, maximum velocity, and uneven surfaces may affect the robot's calculation of the next best path. These factors should be considered when expanding this work into

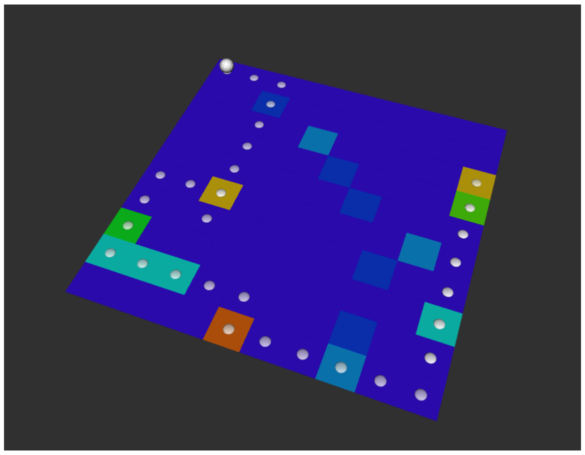

Fig. 5. Greedy path with modified reward metric

physical implementation and may affect the overall algorithm for when creating the next best path.

B. Different Algorithms

The topic of changing linear acceleration in the Orienteering Problem has not been as explored as other variations. This is inferred to be because of the inability to gain any guarantees without the problem being submodular and monotonic. However, using different algorithms can still be a viable option in simulation. This may provide more insight into implications on both the algorithm shown above, and the newer implemented algorithm.

VII. Conclusion

With the increasing reliance on autonomous solutions within multiple industries, it becomes important to have a general understanding of implementing such systems and its algorithm. The team aims to further their understanding in this field by considering a variation of the Orienteering Problem. The Linearly Accelerating Orienteering Problem (LAOP) is introduced where the Orienteering Problem is generalized and modified such that the vehicle can linearly accelerate across distance. Modeling in a 2-dimensional environment, a greedy solution is introduced whereby the reward is modified to instead represent the reward gained through the path in relation to the distance. Additionally, the distance relation is modified to increase at a slower rate the more distance is traveled in order to model the problem.

However, empirical results using ROS and RViz visualization demonstrate that the proposed algorithm performs better than a standard greedy approach that does not account for linear acceleration. This highlights the potential for further development of specialized algorithms tailored to real-world autonomous systems. Through this project, the team has achieved its original goal of obtaining a higher understanding of autonomous systems and the algorithmic implementation and simulation of problems.

Acknowledgment

I would like to thank Dr. Micah Corah and Sharfi Rahman from the Colorado School of Mines for reviewing this project under the CSCI 598D; SP TPS: Autonomous Sensing and Perceptions class.

Thank you for teaching such an interesting class on sensing and perception of robots! Even though I came into this class wanting to learn something else, I'm glad I continued forward to learn such interesting topics!

References

[1] G. Graetz and G. Michaels, "Robots at Work," The Review of Economics and Statistics, vol. 100, no. 5, pp. 753–768, Dec. 2018. [Online]. Available: https://doi.org/10.1162/rest_a_00754

[2] M. Albonico, M. Dordević, E. Hamer, and I. Malavolta, "Software engineering research on the Robot Operating System: A systematic mapping study," Journal of Systems and Software, vol. 197, p. 111574, Mar. 2023, doi: https://doi.org/10.1016/j.jss.2022.111574

[3] Nintendo, Super Mario 64. Kyoto, Japan: Nintendo, 1996.

[4] B. Golden; L. Levy; R. Vohra The orienteering problem., 1987, 3,pp. 307-318. doi: 10.1002/1520-6750(198706)34:3¡307::AID-NAV3220340302¿3.0.CO;2-D.

[5] R. Penicka, J. Faigl, P. Vana, and M. Saska, "Dubins Orienteering Problem," IEEE robotics and automation letters, vol. 2, no. 2, pp. 1210–1217, 2017, doi: 10.1109/LRA.2017.2666261.

[6] F. V. Fomin and A. Lingas, "Approximation algorithms for time-dependent orienteering," Information processing letters, vol. 83, no. 2, pp. 57–62, 2002, doi: 10.1016/S0020-0190(01)00313-1.

[7] L. Joseph and A. Johny, Robot operating system (ROS) for absolute beginners: robotics programming made easy, 2nd edition. New York, New York: Apress Media LLC, 2022. doi: 10.1007/978-1-4842-7750-8.

[8] Robotics Back-End, "ROS2 Tutorial - ROS2 Humble 2H50 [Crash Course]," YouTube. Aug. 31, 2022. Available: https://www.youtube.com/watch?v=Gg25GfA456o

[9] "Tutorials — ROS 2 Documentation: Jazzy documentation," Ros.org, 2021. https://docs.ros.org/en/jazzy/Tutorials.html (accessed Dec. 07, 2024).

[10] A. Koubâa, Ed., Robot Operating System (ROS): The Complete Reference (Volume 7), 1st ed. 2023. Cham: Springer International Publishing, 2023. doi: 10.1007/978-3-031-09062-2.