Anthony Le

ROB 534: SDM

Winter 2017

02-20-2017 (4 PM)

HW #2: Optimization and Informative Path Planning

Questions

1. Performance Guarantees
   1. For any K, the performance guarantee relative to optimal for the greedy solution is:

F(Agreedy) ≥ (1-1/e) max|A|≤K F(A), where F(A) is the submodular objective function. This means that the result of the greedy algorithm as compared to the optimal solution, which is NP-hard to achieve, is at most a constant factor of ~63% of optimal. Thus, the greedy algorithm gives near-optimal solution.

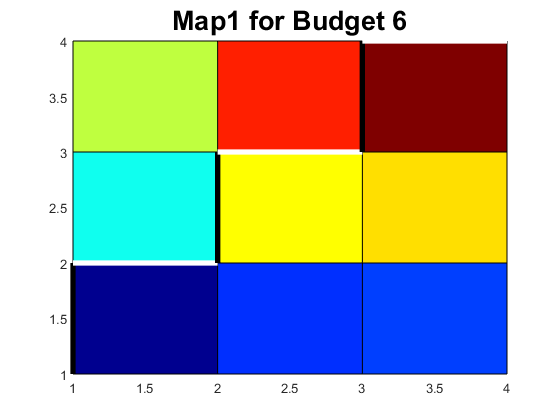
* 1. Assuming that the optimal solution maximizes the total quantity F(P)/C(P), there is no performance guarantee for any K samples taken by the vehicle, where the vehicle greedily selects the best locations and connects them using an approximation to TSP (NP-complete problem). F(P) and C(P) are submodular on their own but submodularity is destroyed when they are divided (or multiplied) together. Basically, with any combination of F(P) and C(P) together, there is no performance guarantee anymore. This greedy algorithm approach described performs poorly in an environment where the information/gains spatially vary and do not correlate such as monitoring/sampling across a large lake without know the model of the lake and the lake correlations.

1. Optimization
   1. (see attachment)
   2. The integrality gap between the integer solution and the non-integer solution was 3/4. The integrality gap is the maximum ratio between the solution quality of the integer program and the non-integer program so for approximation using linear programming, closely related integer programming problems are NP-hard. Therefore, you can reduce any NP-complete optimization problem to an integer linear program, relax it to a linear program by removing the integrality constraints, solve the linear program, and round the linear program solution to a solution to the original problem. Overall, by removing the integrality constraints, the variables are allowed to take on non-integral values.

Programming Assignment

Step 1

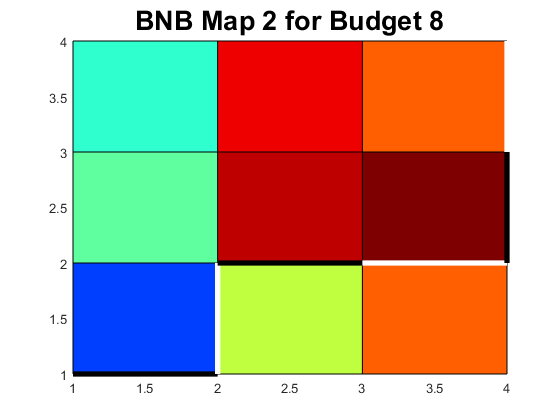
1. To ensure that the robot reaches the goal within the budget, you must filter out moves (in the 4 cardinal directions or remain still) at each node it reaches that will result in a greater number of remaining upcoming moves required to reach the goal (Manhattan distance) compared to the remaining number of moves in the budget.



Information Quality for Greedy Solver



1. The bounding strategy I used was based on computing the upper bound for information path planning inspired by a feature selection technique referred to in [2]. The upper bound was calculated by determining a set of nodes, the reachable set, that can be reached by a feasible path, given the current node, goal node, and budget. Thus, value of the upper bound was the sum of the information gain of each node in the reachable set at a given node, specifically, the current node the robot was present at. For every possible node leaving the start node, nodes were removed from the reachable set if there was no feasible path that could reach the node and the goal node with the budget and the upper bound was recalculated. The recursively continued until either the upper bound was lower than the objective function value by some path that was already examined or the budget was expired. I was able to run the algorithm with a budget of 20 in reasonable time. I believe the algorithm can run with even large budgets but the run time will increase exponentially as the budget increases.



Information Quality for BNB Solver

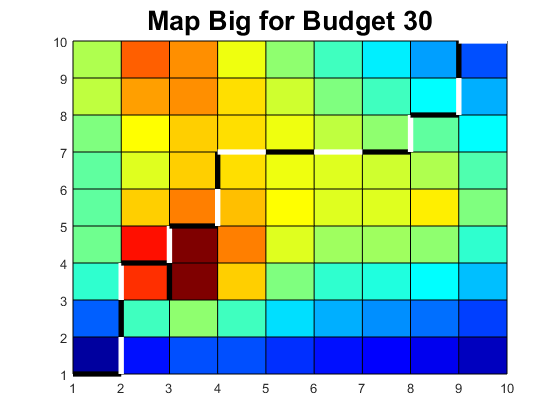
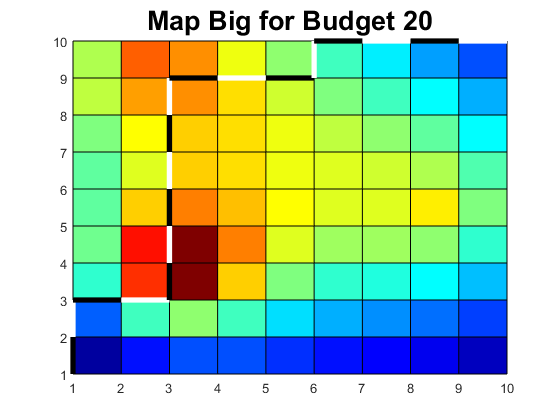


\*all run times are in seconds

The solution quality compare to the greedy solver for budget 6 and 8 for both maps are the same. However, the run time lengthens drastically as the budget is increases besides the unusual decrease in run time for a particular budget (shown above).

Step 2

1. The algorithm I developed resembles a similar mechanism to random-restart hill climbing. Basically, the algorithm can only move in the 4 cardinal directions (N, W, S, E) or remain still and only looks one step ahead. From the current node, the algorithm determines all possible nodes that can be reached based on the criteria mentioned above as well as only considering nodes that are within the map. These are stored in a set of feasible nodes of the particular current node and then, the algorithm picks a node from the set of feasible nodes to move to. This procedure will repeat for each current node that it reaches until either the budget expires or reaches the goal node. However, I ensured the algorithm will reach the goal node before the budget expires using the same trick I used in the greedy solver where I filtered out any moves that lead to nodes that will result in a greater number of remaining upcoming moves required to reach the goal (Manhattan distance) compared to the remaining number of moves in the budget. A path is found and the total information gain of the path is evaluated. If the information quality of this newly found path is better than the previously found path, then the path and the path’s information quality updates for the best. The algorithm repeats and finds paths for k times, where k is the number of times you want this for loop to run before settling with the best path and best information quality of that path. To ensure optimality (or near optimality), I set k = 1000000.





\*all run times are in seconds

Complete Table of Information Quality



\*all run times are in seconds

Discussion

1. The advantage of the discrete variant versus the continuous variant of this type of problem is that in discrete space, each node is defined and a move will take you to a defined node with a distinct edge length and set information gain while expending the budget accordingly per move. The budget can be determined based on the size of the map since the nodes are discretely laid out. Thus, with a known budget and a discrete space to path plan and collect information, the goal is ensured to be reached. While in a continuous space, a budget is difficult to determine since nodes are not defined and a move can lead to anywhere in the space with any edge length depending on your algorithm’s mechanism. Even with directing an algorithm through the continuous space towards the goal, the minimum number of moves to reach the goal varies so it is not always guaranteed to reach the goal before a particular budget expires. However, in continuous space, a smoother and higher information quality path can be planned (with a generous or nonexistent budget; basically performing an RRT) because the algorithm can plan the best path (with the freedom of planning anywhere) that allows it to collect the best information instead of only sampling information at set nodes, especially if none of the nodes in a discrete space lay on the global maximum and only some nodes lay near it. Overall, a discrete algorithm is more appropriate for situations where the map is discretized, and a budget is set. On the other hand, a continuous algorithm is more appropriate for situations where no budget is set, the map is larger (if the space is like the space in this problem), and the space has higher dimensionality (more complex in terms of information quality related to state configurations).
2. When designing my algorithm, I wanted to only explore and expand to nodes that are feasible to reach (not off the map) and I wanted to make sure the algorithm planned towards the goal to ensure that it reached the goal before the budget expired where I filtered out moves (in the 4 cardinal directions or remain still) at each node it reached that would result in a greater number of remaining upcoming moves required to reach the goal (Manhattan distance) compared to the remaining number of moves in the budget. Once a set of feasible nodes was determined, I wanted to the algorithm to randomly select a node to fix a new edge to as oppose to the greedy algorithm where it fixed an edge to the node with the highest information quality out of the set of feasible nodes. However, the algorithm needed to run enough times to plan all possible paths in order to find the optimal (or near optimal) solution. Thus, I incorporated exploration in the algorithm since it planned a different path each k loop and updated for the best solution. In comparison with other algorithms taught in this course, my algorithm explores similarly to a directed RRT but in discrete space where my algorithm randomly picks the next node to expand to just like how RRT randomly samples a node in the space to steers the direction of the new fixed edge. Another algorithm from the course that my algorithm compares to is the epsilon-greedy strategy with random restarts where my algorithm selects the best solution out of trials of solutions found like how epsilon-greedy selects the best solution for a proportion of 1 – epsilon of the trials. Furthermore, in epsilon- greedy, each solution is selected at random (with uniform probability) for a proportion of epsilon similar to how my algorithm finds a solution based on selecting the next node from the current node by random (with uniform probability as well).