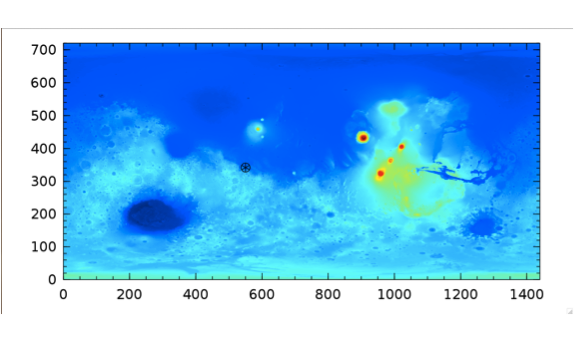
My project demonstrated various strategies for using dynamic programming to solve a single-vehicle routing problem. I was interested in analyzing and comparing the time complexity of value iteration, policy iteration, and the hybrid of the two: optimistic policy iteration. Additionally, I considered the applicability of the solutions to the controls question of which direction should the vehicle actually be moving to reach its destination. I chose to base my project’s maps around the Curiosity rover on Mars. All algorithms were able to determine the same control strategy given the same inputs, validating the theory (and, more importantly, their implementation). The results seemed to indicate that value iteration was the most successful algorithm to apply. Value iteration did not suffer from the time-cost of a large matrix solve, was not dependent upon prior knowledge of a proper policy, and would be easier to parallelize. There are several follow-up steps that could be taken from this work in both algorithm and problem development. One example is creating a traveling salesman problem over multiple vehicles, given the solutions to all of the routing sub-problems. Other examples include allowing for stochastic control or a parallel solver implementation. I included code for discounted total cost, but was dissatisfied with the solution results and discontinued my pursuits in that vein.

To make my project more relevant (and interesting), I decided to build the map from data of a real place. I was able to find high-resolution topography data for Earth, however, the abundance of natural obstacles (such as rivers) and man-made ones (such as roads and buildings) posed a significant problem. If I ignored them, the solutions would miss out on the more interesting solutions. If I tried to include them, I would be spending more time trying to re-encode the vast amount of data available than I would on the actual dynamic programming. Therefore, I decided to use a simpler world for my modeling. Mars seemed to fit my requirements nicely, and NASA provides good-quality data taken by the Mars Science Laboratory for the landing of the Curiosity Rover. See Figure 1 below. The map gives the median heights above a theoretical sea-level value, at a resolution of 4 pixels per degree (14 km per pixel).[[1]](#footnote-1)

This map of the planet of Mars provides several features that show up in the algorithm solutions and are thus worth pointing out here. One is the triad of three tall mountains (in the right-center). The second is the difference in height between the North Pole and South Pole.

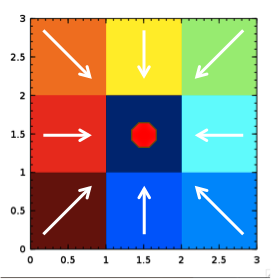
Figure : Mars Topography with site of Curiosity Rover marked



Source: http://pds-geosciences.wustl.edu/missions/mgs/megdr.html

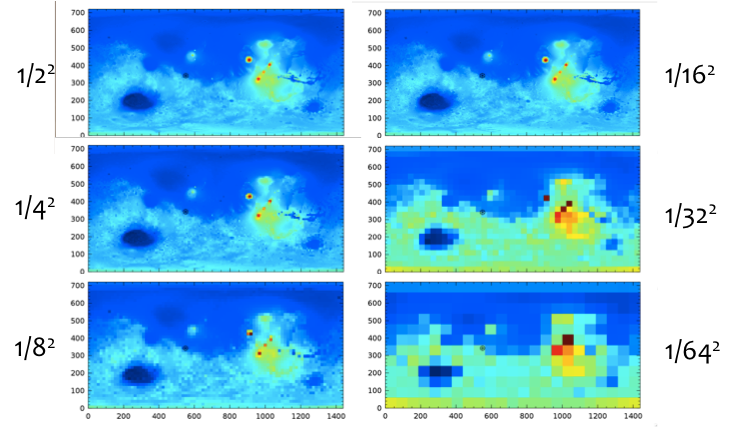
Because this map data already was gridded, I implemented my algorithms assuming this 2D discretization of the space. Additionally, I discretized the controls to be the 8 cardinal and ordinal directions (plus a zero-cost stopping control for the destination point) corresponding to travel to any adjacent square. See Figure 2 below for the color-coded key of valid control directions (I will do my best to also describe the result for those looking at a black and white printout).

Figure : Valid controls



Ignoring the details of the planet’s actual curvature and geometry, I arranged that the east and west boundaries were continuous such that travel from one edge to the other was continuous. I placed a teleporter at the poles (the top and bottom) to provide continuous vertical travel also. As mentioned previously, since the world was already gridded (using a simple cylindrical projection), I did not wish needlessly complicate matters by attempting to convert the map back onto a sphere. I used a Euclidean approximation for the distance between two adjacent map points – either 1 or (this is only approximate because the actual surface is curved, which alters this value based upon the latitude). To reduce runtime and investigate the performance of the algorithm at various problem sizes, I took the center pixel from each group of N2 (NxN). The results are illustrated below in Figure 3. It can be seen that the key features pointed out previously are still represented – to the extent possible – under this transformation. Thus, we expect that our solutions at the various resolutions should be comparable.

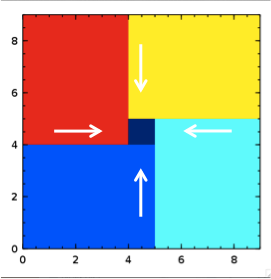
Figure : Successive Map Approximations



I tried several formulas for travel costs based upon a metric for the “roughness” of the terrain and a scaling constant that I felt gave a “good” set of solutions. For validation, I used a simple shortest-path formula , for which the solution can be easily visually inspected. Then I tried two variations of a formula that related the difference between the height of the current square and the height of the square arrived at after applying control u. In the first of these formula, I used a linear relationship . In the second formulation, I used a square relationship of , so that a large jump would be discouraged more strongly than a series of smaller hills.

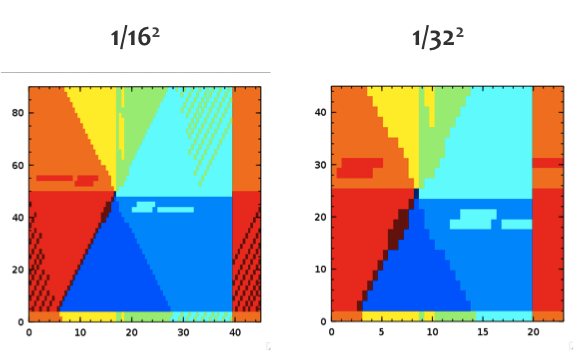
I initialized the algorithm with the very simple control pattern illustrated in Figure 4 below, which was guaranteed by construction to be a perfect control. Correspondingly, the cost structure was initialized to be the sum of the X and Y distances to the origin, which I expected to be a reasonable-close under-approximation of the current costs.

Figure : Initial State



For results, I we’ll first look at the Euclidean distance solutions to understand how to interpret the output. In Figure 5, I am showing the results of running the DP algorithm. (I also will here refer you back to the map color key in Figure 2 for decoding.) There is a asymmetry apparent in the structure of the solution. This, I believe, is due to the algorithm always selecting the lowest-numbered of the discrete options, when faced with equal cost choices, leading to somewhat broken looking and arbitrary boundaries. However, upon tracing several paths, it becomes apparent that all of these paths are equivalent -- e.g. there is no cost difference between going up-left then up versus going up then up-left.

Figure : Euclidean distance controls



Now that we have observed the algorithm working on our simple cost function, we look at the result of using the terrain-biased cost functions. Again, the dynamic programming algorithm works as desired, as can be verified by looking for the features in the map mentioned earlier guiding the selection of the controls. In Figure 6, the results are obtained from the cost function with a linear relationship to the difference in height. Note that we see the control is wrapping both east to west and north to south here. A few other landmark peaks can also be mapped on these maps by observing the regions outlined in a color that is more “out-of-place” than the region of that color expected based upon the results given in Figure 5.

In Figure 7, it is clear that the cost maps are still strongly dominated by the Euclidean distance to the target (with wrap-around visible to the east). This validates that the model (and by extension, the dynamic programming) is optimizing locally for the lowest cost direction, while globally converging towards the current location of the rover. The cost map is showing the cost for the rover to reach any other location on the surface of mars, just as the control map is indicating the appropriate control set for navigating the rover to a new location from its current location. Note the apparent 8-pointed structure to the cost map, caused by the discretization of the controls (and states).

Figure : Linear Cost Function, in terrain height difference

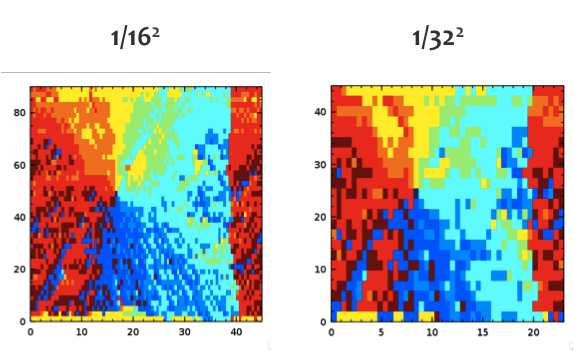
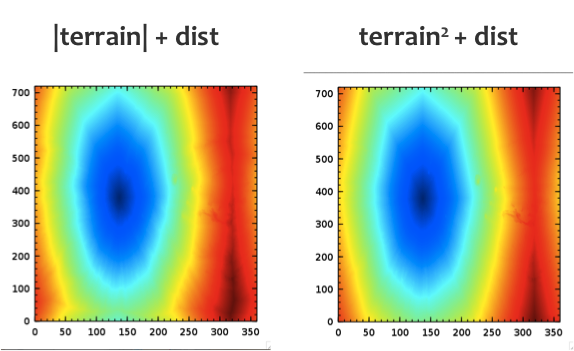


Figure : Cost Maps



The results shown in Figure 8 are seen to be similar to those in Figure 6. I have run the algorithm for a wider range of state discretization amounts (in steps of powers of 2) to provide a broader reference set for making comparisons. It can be observed that the average behavior of a group of blocks can (quite logically) be used as a good estimate for the macro behavior. Ideally there would exist an algorithm that was useful in the opposite direction. However, no simple algorithm is immediately obvious for taking the macroscopic controls (e.g. at the 1/322 level) and computing a block-wise approximation to the microscopic control set (e.g. at the 1/22) level.

We are now equipped to compare the performance and operation of several dynamic programming algorithms.

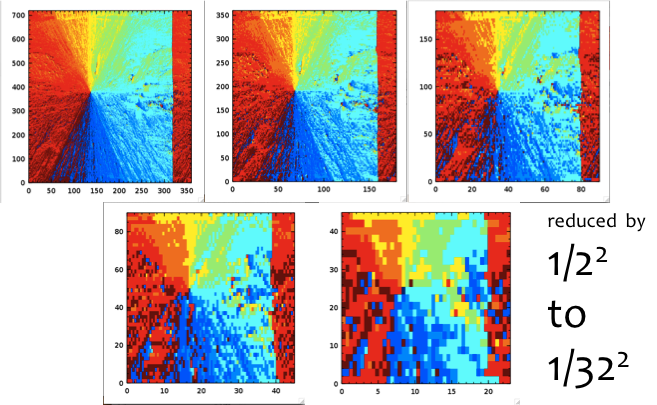


Figure : Height-difference-squared Cost Function

Figure : Computational Time cost of DP



Figure : Algorithmic cost of DP



Figure : Value of Optimistic Policy Iteration



1. http://pds-geosciences.wustl.edu/mgs/mgs-m-mola-5-megdr-l3-v1/mgsl\_300x/meg004/megt90n000cb.lbl [↑](#footnote-ref-1)