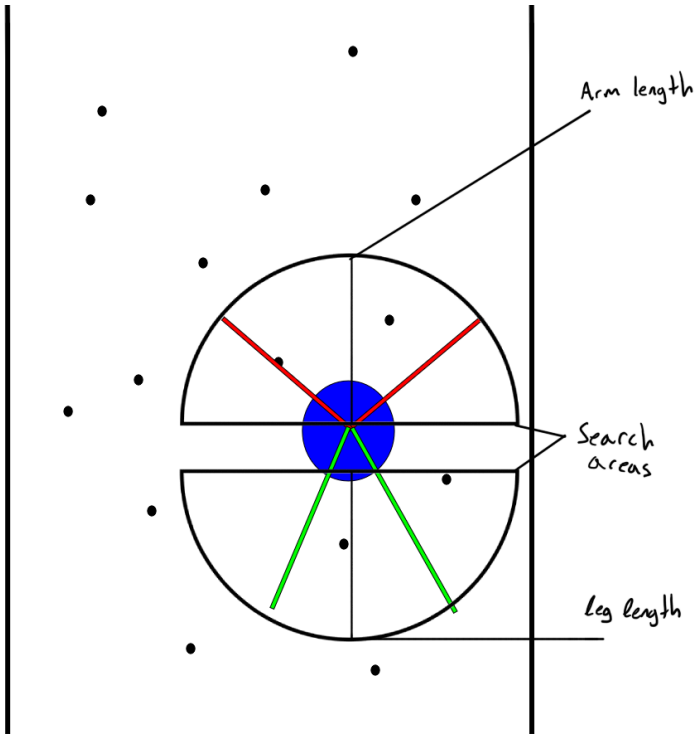
Path Planning Options

Goal: We wish to develop a mobile application that can effectively scan an image of a boulder problem and provide an optimal path for the user.

Problem: We need to solve the boulder problem using a path planning algorithm.

Inputs: Climbing point locations, user wingspan, leg length, user height, and user skill level.



The semicircles represent the search areas, red lines represent the arms, green lines represent the legs, the blue circle is the user. The idea initially thought of would involve the arms look in the valid search area for 2 points. Once 2 points have been found the user is moved to a new position equal distant between the 2 arms and "arm's length" below the highest point. Then the search begins for the legs if the legs have at lest 2 valid points within the search area then the movement would be considered a valid move upwards.

The way the path would be scored would involve distance traveled to the top, distance each step is needed to move upwards possibly scaled with user skill (larger steps may be more difficult but faster), and all steps are valid. Possibly some sort of time efficiency, stability, reach feasibility.

Baseline Overview further research needed

A*

Application: A graph based search algorithm designed to find the shortest path between 2 points by minimizing cost. We could modify this to look at pairs of points (each arm) as well as some way to verify each step is valid.

Pros: Efficient for simpler problems, well known, possibly the easiest, guarantees optimal path

Cons: Depending on how many possible paths there are it may explore a lot of unnecessary paths

This may be a good lightweight option that would be capable of running on a mobile device.

Rapidly Exploring Random Trees (RRT)

Application: path planning algorithm that builds a tree incrementally by exploring random samples in the search space

Pros: Lightweight, Fast, could provide real time updates on path generation

Cons: May not give an optimal path

This one seems to be a good lightweight choice while it may not guarantee an optimal path it will most likely give a good enough path since time is a important constraint

---

Non dominated Sorting Genetic Algorithm 2 (NSGA 2)

Application: Evolutionary multi objective optimization algorithm that rank solutions based on a Pareto dominance. It tries to focus on converging on a solution while also using "mutation" to add diversity.

Pros: Able to solve complex problems with multiple variables "quickly", can provide multiple solutions offering more diverse solutions to the user

Cons: Higher population sizes may make this too slow for our application, computationally heavy, may be too complicated the simple problem we have

Initially Tristans idea, could work would require a lot of fine tuning to get quick results which may hinder the quality of the outputs. From my experience while it did allow you to get multiple solutions to the problem it may be too costly, complex, and time consuming to implement and run. If this idea is considered there may be similar genetic algorithms that might be more lightweight.

Simulated Annealing (SA)

Application: A probabilistic optimization algorithm that explores solutions by allowing for worse solutions early on and gradually refining them. Suitable for large problem spaces where you may not care for an optimal solution rather than a good one. We could use it to gradually improve generated paths to create a "good enough" solution.

Pros: Lightweight, allows the escape of local minima

Cons: No guaranteed optimal path, converging on a more optimal solution may take time

This sounds like NSGA2 without all the "genetic" steps so this may be an option if we want to explore something similar with less workload. May still need more computing power and time than we want.

Dijkstra's Algorithm

Application: Graph search algorithm that guarantees finding the shortest path. We could use this to find the shortest path depending on how complex we consider an (optimal) path

Pros: Simple well known algorithm, easier to implement, lightweight

Cons: Slower than A*, inefficient on larger graphs

Summary

This is just an estimated ranking based on the limited research I've done. When ranking them I considered these 5 things important and simply just scored easiest/fastest based on their rank (fastest = 5 points slowest = 1 point ect)

Complexity to implement (easiest – hardest): Dijkstra's, A*, RRT, SA, NSGA 2

Execution time (fastest to slowest): RRT, A*, Dijkstra's, SA, NSGA 2

Computational Requirements (least to most): RRT, A*, Dijkstra's, SA, NSGA 2

Solution Quality (optimal = 2 good = 1): A*, Dijkstra's (both provide optimal solution tied for 1st), RRT, SA, NSGA 2 (these seem to provide a good enough solution so tied in second)

Online Resources (most to least): Dijkstra's, A*, RRT, NSGA2, SA

- Note: Dijkstra's and A* are often taught in school so there is lots of resources on it, RRT seems to be a popular path finding solution where we may find libraries that would be useful to us, NSGA2 and SA we may find less information on.

Ranking :

A: 18*

Dijkstra's: 18

RRT: 17

SA: 8

NSGA 2: 6

Final note I think we should consider RRT as it seems like a well scoring middle of the road similar to the original idea of using evolutionary computing.