A* Search

Problem Statement

Given

1. A map “assets/elevation.data” which provides the elevation of each point on the map at a pixel resolution;
2. A set of no-go areas “assets/overrides.data”, i.e., locations which are not driveable such as rivers or marshes;
3. Three locations on the map:
    1. Initial location of the vehicle (i.e., Location A);
    2. First location of interest which the vehicle must drive to (i.e., Location B)
    3. Second location of interest which the vehicle must drive to *from* A (i.e., Location C)
    4. That is: Requested Journey: A → B → C

Find

1. The shortest driveable path to go from A → B → C;
2. Time required to drive along such a path in nominal time
3. Total distance travelled along A → B → C in nominal terms.

Assume:

1. The vehicle “rover” can travel 1 unit (specifically, 1 pixel) in 1 unit of time
2. The vehicle “rover” can only travel (a) forward (b) reverse (c) along diagnals
3. The vehicle’s “rover’s”engine produces constant power.

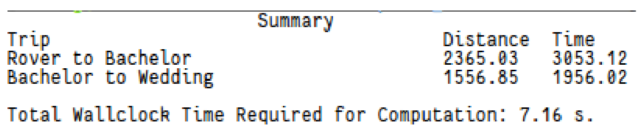
Optimize:

1. Time to search the path;
2. Distance travelled and corresponding time required.

Summary of Solution

Algorithm Used:	*A**
Heuristics:	*Euclidean Measure*
Wallclock Time:	*~7.16s. Measured only for computing all paths. Details below.*
RAM Usage:	*Depends on the architecture of the target processor.*

A summary of computation is provided at the terminal like so:



```
                          Summary
Trip                                        Distance   Time
Rover to Bachelor                           2365.03    3053.12
Bachelor to Wedding                         1556.85    1956.02

Total Wallclock Time Required for Computation: 7.16 s.
```

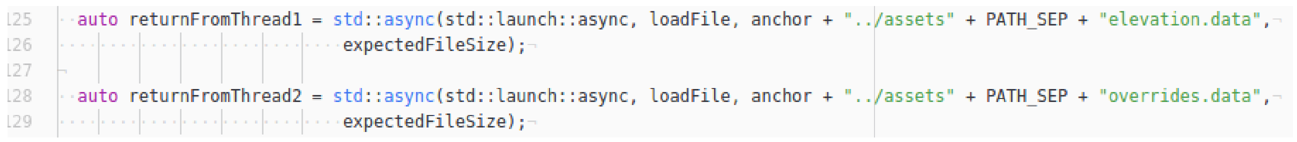
Note
1. The time reported in the summary is the 'nominal-time' as mentioned in the problem statement (i.e., reported travel time is normalized to 'island-time')
2. The distance is reported in terms of 'nominal-units' as mentioned in the problem statement.
3. The reported computation time is for a tunable parameter "weight" set to 1.0 (from rover's initial location to the location of the bachelor, and 10.0 for the path between the bachelor's location to the wedding). See the section of Heuristics for more details. Weights greater than 1.0 leads to increasingly suboptimal paths (i.e., A* optimality guarantee is violated increasingly), but decrease computation time.
4. The final path also depends on the regions prohibited from travel, as decided by the masks (e.g., OF_WATER_BASIN) as provided in the setup.

Design Overview

Entry Point for Execution: main.cpp

Step 1

Loading of Elevation and Override data is performed in parallel in order to reduce the overall computation time:

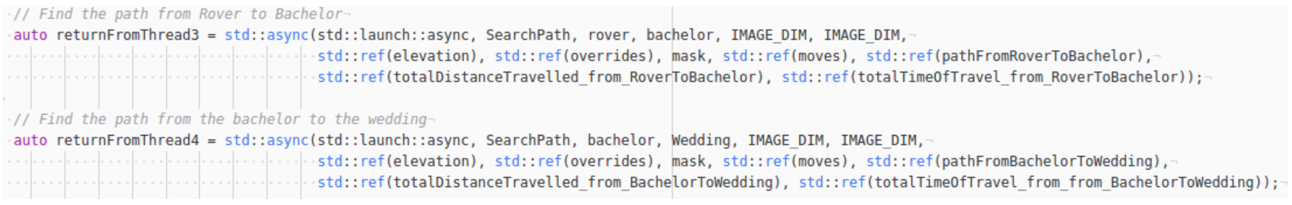


```cpp
  auto returnFromThread1 = std::async(std::launch::async, loadFile, anchor + "../assets" + PATH_SEP + "elevation.data",
                       expectedFileSize);

  auto returnFromThread2 = std::async(std::launch::async, loadFile, anchor + "../assets" + PATH_SEP + "overrides.data",
                       expectedFileSize);
```

Step 2

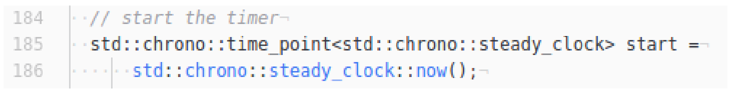
The following paths are computed in parallel:
1. From Rover's initial location to the location of the bachelor;
2. From the location of the bachelor to the location of the wedding



```cpp
// Find the path from Rover to Bachelor
auto returnFromThread3 = std::async(std::launch::async, SearchPath, rover, bachelor, IMAGE_DIM, IMAGE_DIM,
                                    std::ref(elevation), std::ref(overrides), mask, std::ref(moves), std::ref(pathFromRoverToBachelor),
                                    std::ref(totalDistanceTravelled_from_RoverToBachelor), std::ref(totalTimeOfTravel_from_RoverToBachelor));

// Find the path from the bachelor to the wedding
auto returnFromThread4 = std::async(std::launch::async, SearchPath, bachelor, Wedding, IMAGE_DIM, IMAGE_DIM,
                                    std::ref(elevation), std::ref(overrides), mask, std::ref(moves), std::ref(pathFromBachelorToWedding),
                                    std::ref(totalDistanceTravelled_from_BachelorToWedding), std::ref(totalTimeOfTravel_from_from_BachelorToWedding));
```

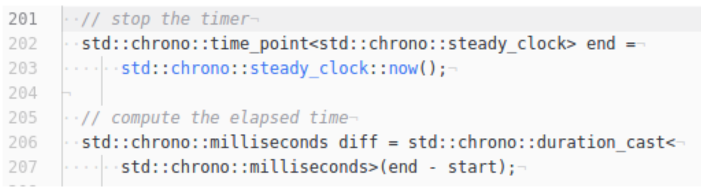
Step 3
The overall time to compute the paths is estimated by a call to chrono library like so:



```cpp
// start the timer
std::chrono::time_point<std::chrono::steady_clock> start =
    std::chrono::steady_clock::now();
```

(before launching the first thread that computes the path from the Rover's initial location to the location of the bachelor.

The overall time required to compute the paths is then deduced like so:



```cpp
// stop the timer
std::chrono::time_point<std::chrono::steady_clock> end =
    std::chrono::steady_clock::now();

// compute the elapsed time
std::chrono::milliseconds diff = std::chrono::duration_cast<
    std::chrono::milliseconds>(end - start);
```

Detailed Design

Attempt was made to keep the overall solution as modular as possible.
Specifically:

1. The call to the solver is made in "main.cpp" by launching the "SearchPath" function with the necessary paramters. See "SearchPath.hpp" for more details about parameters.
2. The "SearchPath" itself uses the class "Astar" for computing the paths. See "AStar.hpp" for details about the class.

Therefore, the "SearchPath" function wrapper encapsulates the core A* algorithm, enabling simultaneous searches for multiple paths, as implemented in this project.

Overview of Software Components

1. ***Wrapper "SearchPath"*** encapsulates the A* path search algorithm, and initializes a Astar object with the necessary parameters. See the "SearchPath.hpp" for details.

2. **Class AStar**: Provides the core functions required to perform path search using A* heuristic. The choice of A* was motivated by Sebastian Thrun's lecture in the third semester of "Self-Driving Car Nano-Degree Program", and by the fact that the algorithm was successfully deployed on "Junior" autonomous vehicle from Stanford University in DARPA's Urban Challenge. See "AStar.hpp" for details on the class itself.
3. **Class "Node":** Provides a convenient abstraction to express the location on the map. Currently supports only two coordinates <x,y>. Provides necessary operators (e.g., equality) to compare locations, as required by the A* algorithm.
4. **Class Heuristics:** Provides functions that may be used to compute the heuristic ("h") as required by the A* algorithm. Currently, only Euclidean Distance measure is provided. All algorithms are required to be optimistic (i.e., never overestimate the true distance between two points) by A*. Therefore, it is possible to just recompute paths using a different heuristic cost function without requiring any modifications to either the wrapper "SearchPath", or the core class "Astar".
5. **Class CostGrid:** Templated class provding matrix-like abstraction, together with convenient access functions. Used by "Astar" to store various data.
6. **Various Unit Tests** as part of the development process.

Execution

1. **Parallelism:** Search for paths from Rover's initial position to the location of the bachelor, and from the location of the bachelor to the location of the wedding are computed in parallel.
2. **Parallelism:** Search for all explorable neighbors are also in parallel:

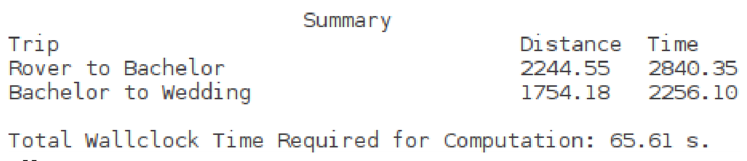
```
// For each explorable neighbor
for (Node thisNeighbor : allNeighbors) {


 std::future<void> t = std::async(std::launch::async, processThisNeighor, cheapestNode, thisNeighbor, std::ref(aStar), weight, end);

}  //for (std::vector<Node> thisNeighbor : allNeighbors)
```

Tunables

1. **Weight:** A tunable parameter "weight" is introduced which forces the path-search to possibly select less favorable paths but with reduced computation time. The provided implementation has:
    1. Weight = 1.0, for estimating the path between Rover's initial position to the location of the bachelor;
    2. Weight = 10.0, for estimating the path between the location of the bachelor and the wedding.
    3. With these weights, the computation time is 65.61 seconds, compared to ~7 seconds when weights are kept as 100.0 for both path searches.

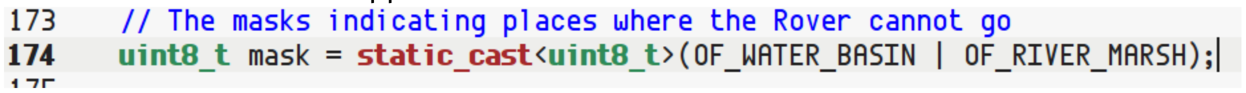


```
                         Summary
Trip                                    Distance  Time
Rover to Bachelor                       2244.55   2840.35
Bachelor to Wedding                     1754.18   2256.10

Total Wallclock Time Required for Computation: 65.61 s.
```

Illustration 1: Performance with weights 1.0 and 10.0 for paths. Compare the times and distances to the case above when weights for both paths were set to 100.0

2. **Mask:** The choice of regions where the Rover is prohibited can be supplied by setting the mask like so in main.cpp:



```
// The masks indicating places where the Rover cannot go
uint8_t mask = static_cast<uint8_t>(OF_WATER_BASIN | OF_RIVER_MARSH);
```

Heuristics

1. **Distance Estimation:** Once the final path is computed as a sequence of coordinates, the total distance is estimated by adding the Euclidean distance between two successive coordinates, and accumulating the distance.
2. **Time Estimation**:The following heuristic is followed:
    1. For movement between two points on the map with no net change in elevation: t = d (island-seconds), where d is the island-distance travelled.
    2. For movement between two points on the map with a net change in elevation of *h*, t = d/(1-sin(θ)), where θ = tan^{-1}(h/d). This heuristic assumes that the force generated by the Rover's engine remains constant throughout the journey. The scaling factor of 1-sin(θ) is based on the notion that effective force when going up an elevation with grade θ is F.sin(θ), where F is the constant force generated by the Rover's engines.

Notes

1. By intent,the time required to transform the path computed by AStar class to the format suitable for use with the visualizer is not counted, as this step does not form a core of the path-search. The provided transformation steps in "main.cpp" are mainly done to reduce the overall time to print the final bitmap.
2. The A* algorithm is described in Semester 3 of "Self Driving Car Nano-Degree Program".

Final Result

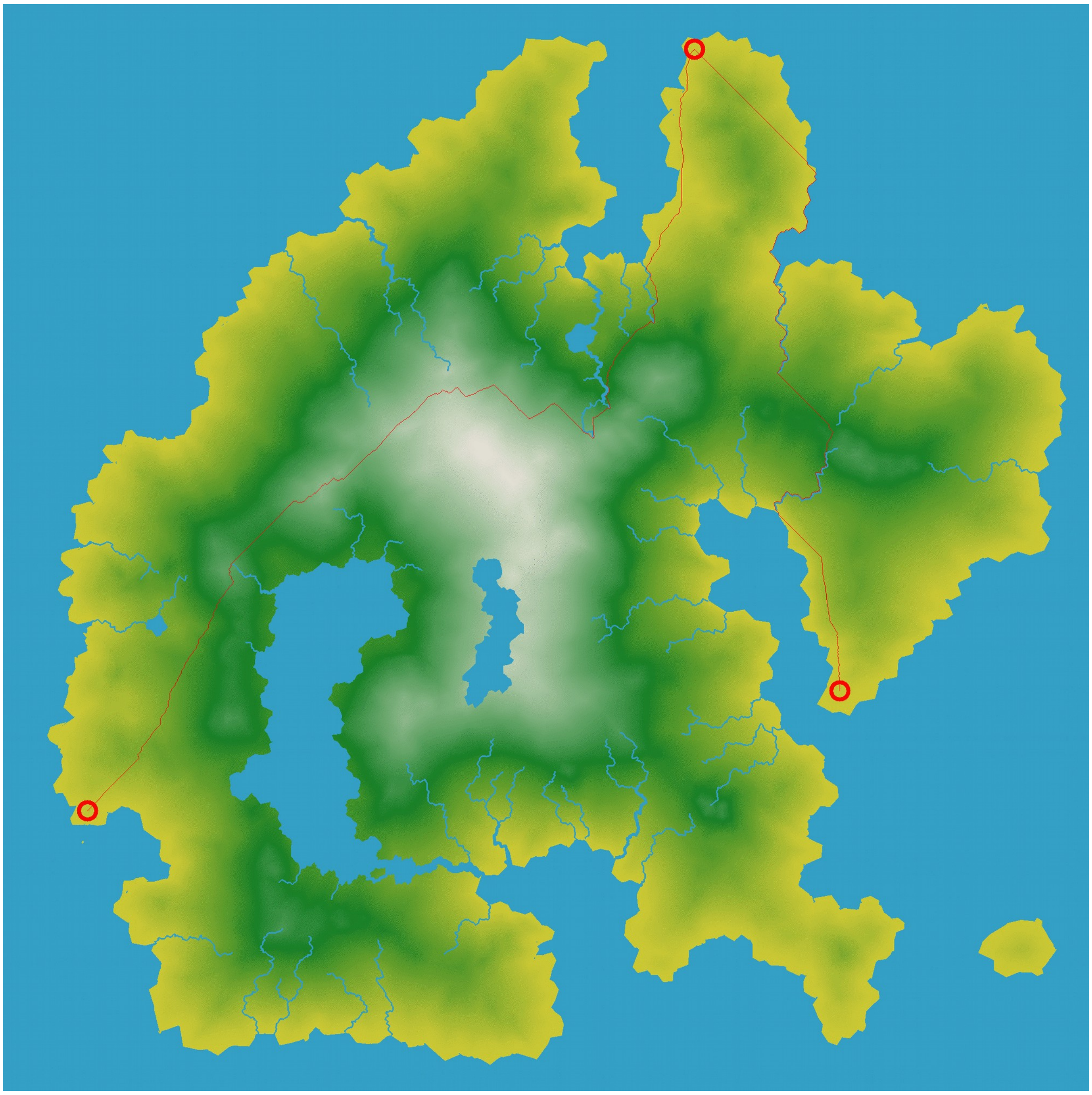


Illustration 2: Final Path. Weights: 1.0 for path from Rover's initial location to the Bachelor's location (A optimal), and weight = 10.0 from Bachelor's location to Wedding Location (Not optimal).*

(Provided for reference. Generated by the executable)

Related Summary:

Trip	Distance	Time
Rover to Bachelor	2508.59	3240.93
Bachelor to Wedding	1760.88	2287.81

Total Wallclock Time Required for Computation: 221.87 s.

Corresponding weights:

1. weight: 1.0: Rover’s Initial Position to the Bachelor’s Location

2. weight: 10.0: Bachelor’s location to the location of the wedding.

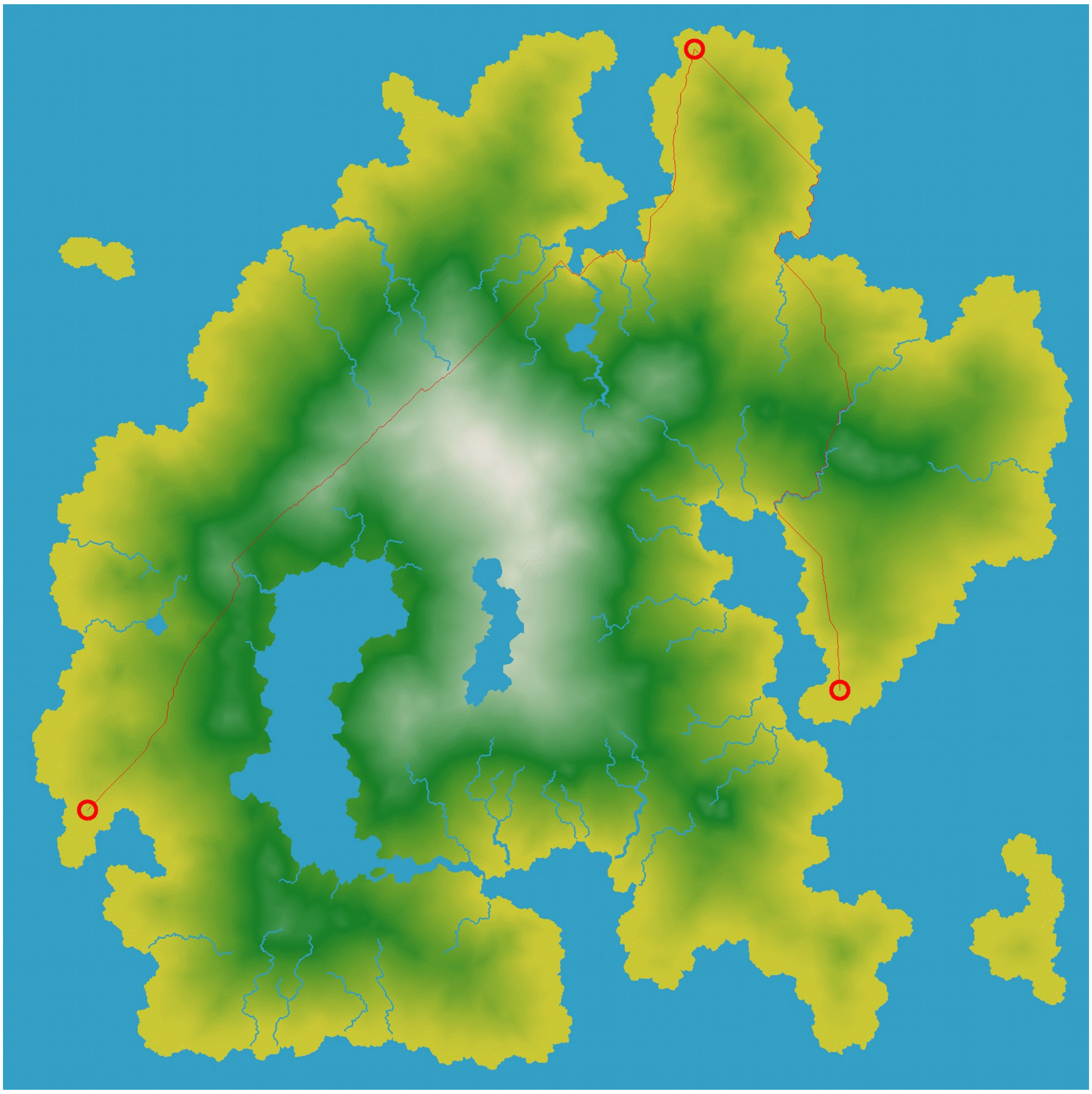


Illustration 3: Solution when Areas with Zero Elevation are (1). Not Painted Blue, and (2). Are Considered Drivable.