Spencer McClure
CS460 - AI
2 November 2012
Project #2

1. The first heuristic function I used was a euclidean distance from the state to the average of all the finish states. Due to the large quantity of finish/goal states and the nature of the area they cover in the test racetracks, the average of all their locations will always fall far enough beyond the 'finish line' (where the open race track turns into finish/goal states) to maintain that all the heuristics based on this location are consistent (as far as the non-goal states heuristics are concerned).

It is important to note that the euclidean distance by itself doesn't represent an admissible heuristic. It doesn't estimate how many steps we are away, just how far away we are. To use the euclidean distance as a heuristic, one needs to take into account how much distance one can travel in one turn. Our top speed is 30, so at first I used that, but the actual heuristic function is the euclidean distance divided by 42. Thirty really isn't an accurate representation of the max distance our race car can travel in euclidean terms, which would be Sqrt(30*30+30*30) = approximately 42.

In order for a heuristic to be consistent, h(n) <= C(n,p) + h(p) or, rather, h(n) - h(p) <= C(n,p). For any two states, n and p, h(n) - h(p) is the difference between the two euclidean distances from the same node. In the best case, the distance between n and p actually is h(n) - h(p). But most of the time, the actual distance will be greater. If one considers h(n)-h(p) as the best-case number of steps at max speed to get from n to p (in euclidean terms), then it is impossible for the C(n,p) to ever be less than h(n)-h(p) because in the best case they are equal, else h(n)-h(p) <= C(n,p).

The second heuristic function I used was similar to the first, only this time it considers the acceleration limitations of the problem. It calculates the optimal speed vector from the current location to the goal, and then begins accelerating as much as possible towards that speed vector, tracking its location, and occasionally rechecking its angle and adjusting its path accordingly. The entire time, I count the number of steps I take to accelerate closer to the goal. To optimize this heuristic function for time, I only recheck its angle and distance after a number of steps. That number is determined by how close the point was to the goal when it last started accelerating. If it is far away, it is allowed more leeway to accelerate without rechecking its location, while once it gets close it may have to check every move. Once it is within 84 units away, however, I stop and add one or two to the heuristic counter depending on the exact movement vector and exact distance away.

This problem relaxation could be described as removing all black squares, but still paying attention to the acceleration requirements, and once close, just assuming you're moving max speed in the absolute correct direction. The simplification of arrival combined with the lack of illegal moves lend this algorithm its consistency. Since every point can accelerate to the goal in the similar fashion, it is clear that h(n) - h(p) <= C(n,p) since the cost of C(n,p) could never be more than a direct acceleration towards n from p, especially since once within a large vicinity of n, the function basically assumes heuristic 1 and just predicts as if you are moving at max

velocity in the optimum direction which is correct if the situation is optimal, but most likely is an underestimate. This heuristic can certainly do no worse than the first heuristic in any case, as the base cases are the same once they are close, but this second heuristic does better when the points are far away, so it is safe to say the second heuristic dominates the first.

2.)

	Heuristic	Tiebreaker	Runtime (sec)	# of Nodes Expanded	Solution Cost	Average Car Speed
Track 1	h1	t1	2.16719	94468	71	9.625
‘’	h1	t2	2.1856	94468	71	9.625
‘’	h2	t1	1.12769	64050	71	9.63889
‘’	h2	t2	1.1226	64081	71	9.6528
Track 2	h1	t1	1.79783	92976	52	2.13208
‘’	h1	t2	1.84781	92976	52	2.13208
‘’	h2	t1	1.4567	64050	53	2.1111
‘’	h2	t2	1.176	56745	54	2.09091
Track 3	h1	t1	1.40193	67448	48	9.12245
‘’	h1	t2	1.4048	67448	48	9.12245
‘’	h2	t1	0.73595	35260	48	9.10204
‘’	h2	t2	0.7502	35224	48	9.12245

As you can see there is one anomaly above: heuristic two did not find the optimum path for racetrack 2! This is due to my approximation of the finish combined with the relatively small size of racetrack 2. Since the average final location is substantially different than where the finish line actually begins (relative to the size of the map and locations of valid states immediately before the finish), the heuristics fail to remain consistent in this example and it favors southernmost nodes more than it should at the end of the path (resulting in 1 or 2 more steps in the found solutions). If I had more time, I would not use the same approximation for the finish, and instead use a final location closer to where the goal states meet the regular states. As you can see, in the other racetracks, it is not an issue and the second heuristic consistently performs better.

The top_speed value does not affect the number of moving vectors checked in these racetracks because there is not enough freedom of movement to allow the car to ever reach the max

velocity of 30. Since the max value is never actually used to restrict the cars movement, it doesn't affect the runtime. If however, one used a very wide open racetrack where topspeed could be reached, it would decrease run-time because the solutions could contain less steps overall, and would thus be discovered faster with a good heuristic. Every step would still only consider 9 next moves, so runtime would not increase, per step, but since overall there could be more possible solutions, in a breadth-first search (A* with h=0), the increased topspeed would increase in runtime.

3. In racetrack #3, my search picks the bottom path. It still expands nodes in both paths, though it may eventually stop expanding the nodes from the slower path due to it finding the solution faster through under-way. The main reason for this is that the car would need to decelerate to near 0 just to enter the top-most path, and then it would need to pretty much remain around 0 until it clears the narrow portion. The g values for this path (and thus the f values to be considered in the heap) simply grow too fast, and the other path beats it out.

4. Please look at the table above for the results discussed below of the different tie-break considerations. As you can see the tiebreak option had no effect at all with heuristic 1.

With heuristic 2, however, in racetrack1, the first tie-break option (taking higher g) did better by a couple of expanded nodes, but in basically the same time. In racetrack3, the second tie-break option did better. In one tiebreak scenario, one prefers to take nodes closer to the start point (lower G, higher H) and in the other it prefers to take nodes closer to the goal state (higher G, lower H). Its obvious that the latter is the better of the two (it is better to pick nodes close to the goal than far from the goal), though in some situations (apparently such as in racetrack 1), it doesn't necessarily result in the optimal path, and an extra 32 nodes needed to be explored.

5. Since h1 and h2 are both consistent, it is known that neither one underestimates, and that neither one fails to satisfy $hx(n) - hx(p) <= C(n,p)$. If one took that as $Max(h1(n), h2(n)) - Max(h1(p), h2(p)) <= C(n,p)$, then it is clear that the first h(n) will be as large as can be, either h1(n) or h2(n). Lets call the max h(n) term 'hx(n)'. It follows that if hx(p) is from the same heuristic as hx(n), the new heuristic must be consistent because hx is consistent because its the same as just using one of the heuristics. If hx(p) is from the other heuristic, however, it still must satisfy $Max(h1(n), h2(n)) - Max(h1(p), h2(p)) <= C(n,p)$. This must be the case because even if one takes the heuristic from the other heuristic function, that means that the other heuristic function had a greater h(p) than the one you took, causing the difference between the first max and the second max to be less than if you used just one heuristic, guaranteeing that it is just as consistent as either of the two alone.

The benefits of such a heuristic function would be that a stronger heuristic is produced. It maximizes the strengths of both heuristics without ever compromising solution validity. On the other hand, it also would result in a more time consuming heuristic function, because it would involve calculating two heuristic functions for every state, rather than one. If one had quick, diverse heuristic functions, it would be very beneficial, but if one needed to perform many computations for the heuristic functions, it may not be worth the tradeoff.