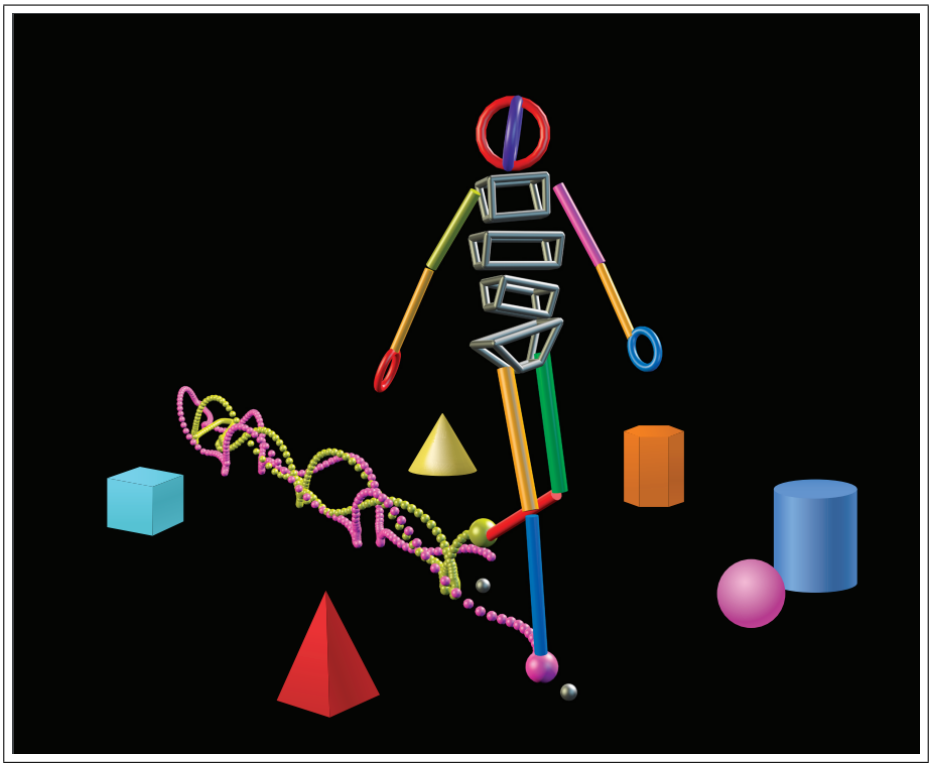
Algorithmic Robot Motion Planning - 236901

Homework #2: Sampling-Based Motion Planning



Name	ID	Email
Niv Ostroff	212732101	nivostroff@campus.technion.ac.il
Yotam Granov	211870241	yotam.g@campus.technion.ac.il

Faculty of Computer Science
Technion - Israel Institute of Technology

Winter 2022/23

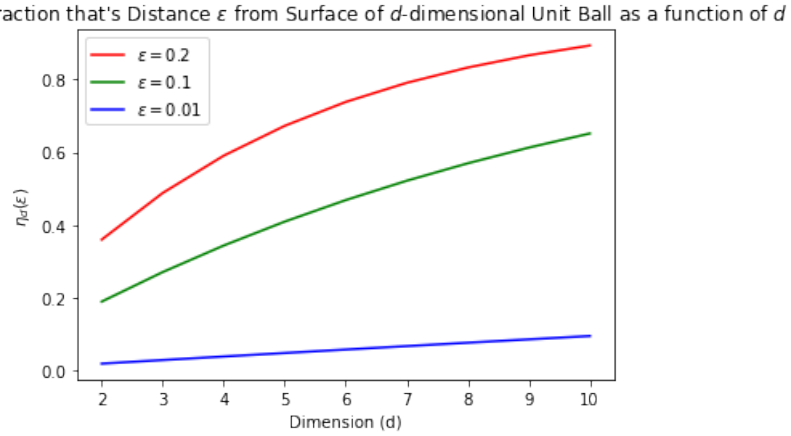
(1) Distribution of Points in High-Dimensional Spaces

Define the fraction of the volume that is ε distance from the surface of a d-dimensional unit ball as $\eta_d(\varepsilon)$.

Plot $\eta_d(\varepsilon)$ as a function of d for $\varepsilon = 0.2, 0.1, 0.01$ for $d = 2...10$.

Discuss the implications to reducing the connection radius required for a sampling-based algorithm to maintain asymptotic optimality.

Solution: Our plot for $\eta_d(\varepsilon)$ as a function of d looks as follows:



We see that $\eta_d(\varepsilon)$ increases monotonically with d $\forall d \in [2, 10], \forall \varepsilon \in \{0.2, 0.1, 0.01\}$. This means that as we increase the number of dimensions, a larger fraction of the ball's volume will be within ε distance from the surface of the ball. This conversely implies that as we increase the number of dimensions, a smaller fraction of the ball's volume will be within $1 - \varepsilon$ distance from the center of the ball.

Suppose we have a sequence of consecutively intersecting d-dimensional unit balls, and the distance between the centers of consecutive balls in the sequence (i.e. the connection radius) is some quantity r. Let's assume that this sequence of balls contains the optimal solution for a motion planning problem, and the center of every ball is a point located on the optimal path.

To converge to the optimal solution, we need to sample enough points so that we get samples which are "very close" to the centers of every ball. The volume in the ball that is "very close" to the center of each ball (i.e. balls of radius $1 - \varepsilon$) decreases with dimensionality d, and the probability of sampling a point inside a specific ball's volume from the entirety of free space is proportional to the ratio of the ball's volume to the volume of the free space - i.e. it decreases with that ball's volume, which in turn decreases with d. To counteract this result, we need to increase the number of balls in our sequence (i.e. decrease the connection radius), which will increase the probability that enough samples will be "very close" to the optimal solution.

Thus, we see that the increase in dimensionality requires a smaller connection radius in order to sustain asymptotic optimality for sampling-based motion planning methods.

(2) Tethered Robots

We consider a 2D point robot translating amidst polygonal obstacles while being anchored by a tether to a given base point p_b. The robot may drive over the cable, which is a

flexible and stretchable elastic band remaining taut at all times. We study the problem of constructing a data structure that allows to efficiently compute the shortest path of the robot between any two given points p_s, p_t while satisfying the constraint that the tether can extend to length at most L from the base.

Part 2.1

Describe the structure of a path from a start to target configuration (robot location + tether description).

Solution:

The path structure we will implement is a simple (x, y) location list of the 2D robot over the path along with the description of the tether, which we will implement to be the tether's homotopy class.

Part 2.2

Suggest an efficient way to encode the tether's description. Note that the robot can be at the same location but with completely different tether descriptions (e.g., circling once or twice around an obstacle) and what we need to define is the homotopy class of the tether.

Solution:

In class, we learned the h-signature, a way to encode the tether's curves to check if two curves are in the same homotopy class. Since in our case the tether is taut, it is enough to return a set of 2D points (x, y) which describe every point where the tether is at an angle in order to define the tether's homotopy class. This way, if we have the robot at the same location but once circling an object once and another time twice, we will be able to differentiate between the two since our point list encodes different values.

Part 2.3

The homotopy-augmented graph G_h of a graph $G = (V, E)$ encodes for each vertex of G, all homotopy classes that can be used to reach the vertex using a tether of length L. Describe an approach to compute the homotopy-augmented graph of a given graph using a Dijkstra-like algorithm. Describe the nodes, how they are extended and when the algorithm terminates.

Solution:

We start by constructing a graph G_h, which includes all of G's vertices and none of its edges. Our homotopy graph algorithm will look as follows:

- Let p_s be the starting node for the homotopy algorithm
- Create a priority list Q, initially containing only p_s
- Set $d(p_s) = 0, d(v) = \infty$ for every $v \neq p_s$
- While Q:
  - Remove the vertex v with the smallest distance from p_s
  - For every neighbor w of v:

  - If the distance to $d(w)$ through v is smaller than the current $d(w)$, update $d(w)$ and add w to Q. In this step, we must make sure that the distance is not greater than the length L.
  - When adding w to Q, we update the set of homotopy classes that we can use in order to reach w. We can do that by adding the homotopy class which signifies the shortest path from p_s to w to the current set of homotopy classes of w.

In our algorithm, the nodes are the original vertices of graph G. The code terminates when Q is empty, which signifies that all homotopy classes have been computed.

Part 2.4

Now we will now use the notion of a homotopy-augmented graph to efficiently answer queries solving the motion-planning problem for a point tethered robot. A query is given in the form of two points p_s, p_t and the h-invariant w_s describing the tethered placement at p_s. In order to compute a path (if one exists) between p_s and p_t with an original tether placement defined by w_s, we need to traverse the homotopy-augmented graph G_h^{vis} (the homotopy-augmented graph of the visibility graph).

- What new vertices do we need to add the homotopy-augmented graph G_h^{vis} to account for p_s and p_t?
- What new edges do we need to add the homotopy-augmented graph G_h^{vis} to account for p_s and p_t?
- How can we use the newly-constructed graph to efficiently solve our motion planning problem?

Solution:

The new vertices we need to add to the graph are simply 2D points to represent p_s and p_t. Each of these points should be connected to all nodes in the graph that are visible from p_s and p_t, respectively. Thus, the added edges will be all edges connecting vertices visible to p_s to node p_s, and all edges connecting vertices visible to p_t to node p_t. We can do this using the visibility graph algorithm implemented in the first homework. Now, we can use a Dijkstra algorithm to compute the shortest path between the two points, while meeting the constraints about the length of the tether and the h-invariant.

(3) Motion Planning: Search and Sampling

Part 3.1 - A* Implementation

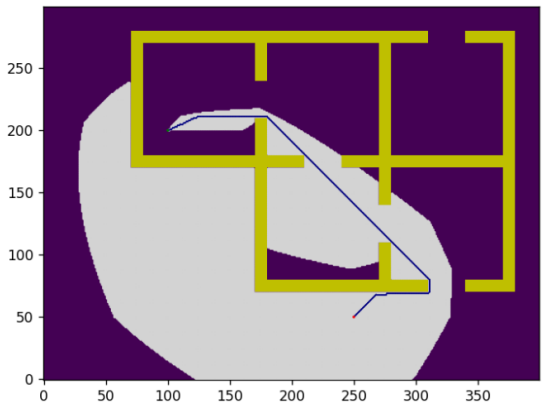


Figure 1: For the $\varepsilon = 1$ case, the cost is 349.103.

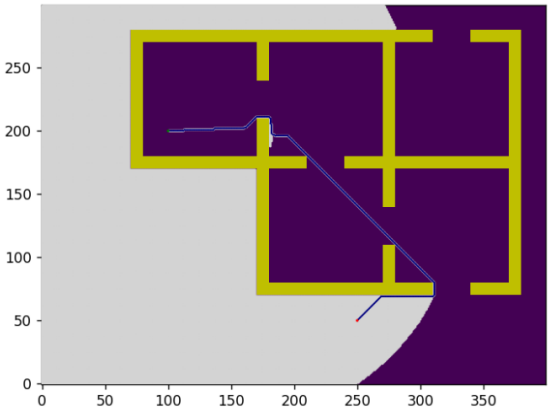


Figure 2: For the $\varepsilon = 10$ case, the cost is 355.546.

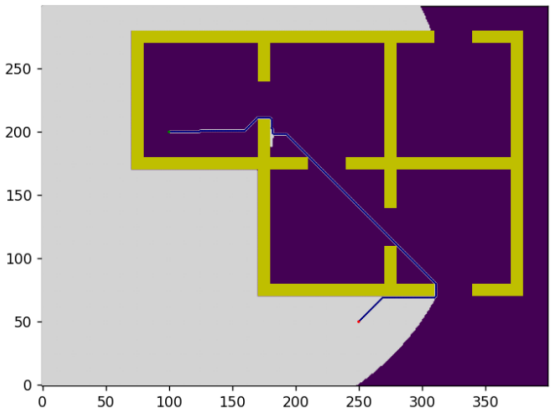


Figure 3: For the $\varepsilon = 20$ case, the cost is 354.960.

We can see that the solutions for the three different ε values are quite similar in terms of cost, with $\varepsilon = 1$ proving a slightly cheaper solution, meaning giving a lower weight to the Euclidean heuristic value resulted in the shortest path. In terms of efficiency, $\varepsilon = 1$ had the lowest amount of expanded nodes, at 43726, while $\varepsilon = 20$ had the highest amount, at 50683 expanded nodes.

Part 3.2 - RRT Implementation

The following two figures show our results when running the RRT algorithm on Map 2, where the variables are the goal biasing percentage, the extension mode, and the step size for the extensions. For Map 2, we chose to set the step size to $\eta = 10$ when the extension mode is E2.

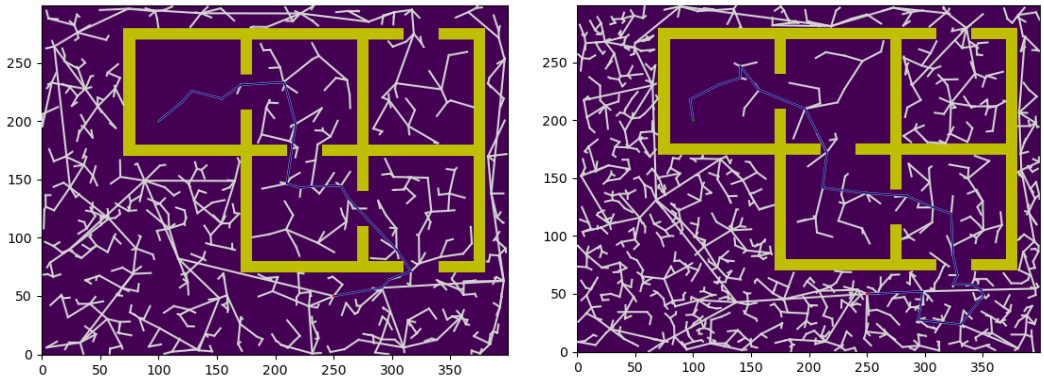


Figure 4: RRT algorithm acting on Map 2 in extension mode E1 with 5% goal biasing (left) and 20% goal biasing (right).

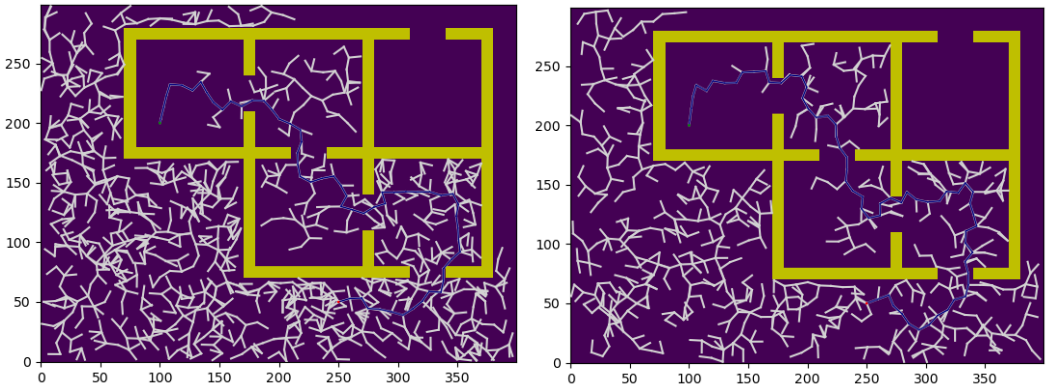


Figure 5: RRT algorithm acting on Map 2 in extension mode E2 with step size $\eta = 10$, with 5% goal biasing (left) and 20% goal biasing (right).

The following table summarizes the results for the 4 cases shown in the figures above (averaged over 10 trials each):

Map	Ext. Mode	Goal Bias	Step Size	Avg. Num. of Iterations	Avg. Time [sec]	Avg. Cost
M2	E1	0.05	-	572	0.785	532.377
M2	E1	0.2	-	899	2.098	570.022
M2	E2	0.05	10	2026	9.562	522.062
M2	E2	0.2	10	2360	11.458	514.446

Figure 6: Table containing the results for the experiments with RRT acting on Map 2.

These results indicate to us that smaller goal biasing percentages produce lower computation times (without saying much about the total cost), and we see that using the second extension method (E2) is much more temporally expensive than the first one (this scales against step size - larger step size causes smaller computation times, up to some upper bound), yet produces lower-cost paths. We indeed expected a time-cost tradeoff between the two extension modes, and to choose a preference between them would require more information about our logistical constraints - if we have enough time, then it is clear that the second extension mode is preferable since it produces lower-cost paths.

Part 3.3 - RRT* Implementation

We will run the RRT* algorithm on Map 2, using the second extension mode (E2) with step size $\eta = 10$ and 5% goal biasing - the first choice guarantees higher quality solutions, while the latter two choices ensure that we reach solutions in a reasonable amount of time (we've decided to pursue average runtimes below 30 seconds, and average costs below 550, as our tradeoff aim). The only parameter we will vary here is the number of nearest neighbors k considered for rewiring by the algorithm. We first let k be a constant value, and $k \in \{2, 3, 5, 10\}$. Running RRT* with these k values on Map 2, we obtained the following 4 graphs:

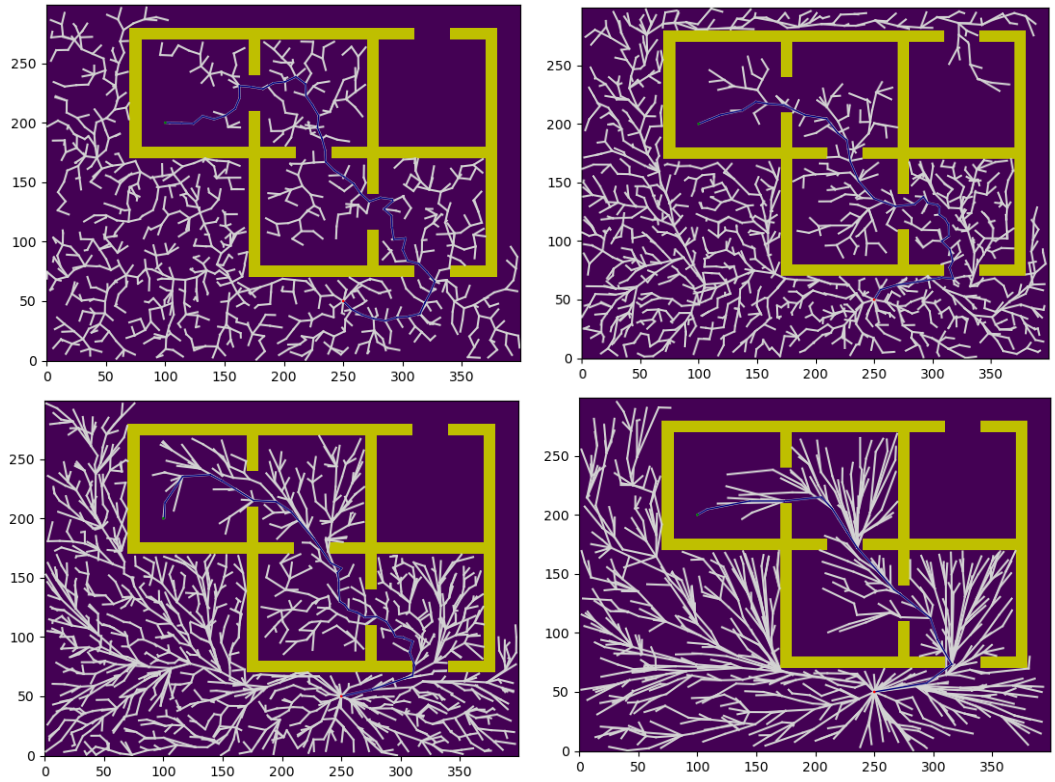


Figure 7: RRT algorithm acting on Map 2 in extension mode E2 with 5% goal biasing and step size $\eta = 10$, and with $k = 2$ (top left), $k = 3$ (top right), $k = 5$ (bottom left), and $k = 10$ (bottom right).*

We also ran RRT* on Map 2 where k is no longer constant and instead varies according to the function $k = floor[2 \cdot log(n)] \propto O(log(n))$. Doing so, we obtained the following graph:

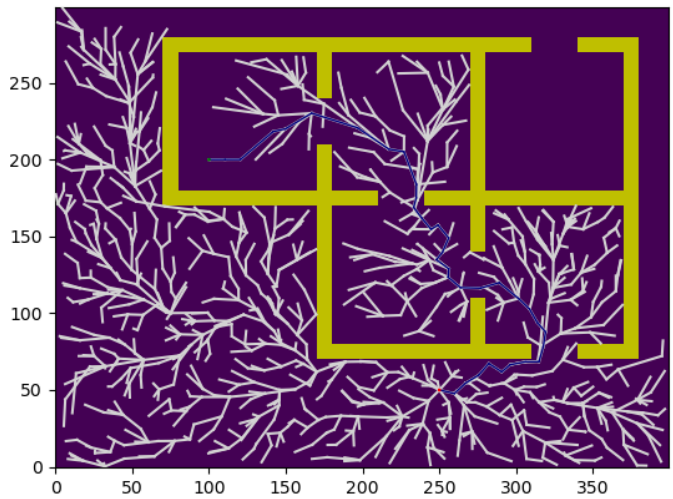


Figure 8: RRT algorithm acting on Map 2 in extension mode E2 with 5% goal biasing and step size $\eta = 10$, and with $k = floor[2 \cdot log(n)]$, where n is the number of nodes in the tree per iteration.*

The following table summarizes the results for the 5 cases shown in the figures above (averaged over 10 trials each):

Map	Ext. Mode	Goal Bias	Step Size	k	Avg. Num. of Iterations	Avg. Time [sec]	Avg. Cost	Avg. Num. of Rewirings
M2	E2	0.05	10	2	2069	14.969	527.959	522
				3	2325	21.320	456.445	1281
				5	1906	13.661	407.661	1615
				10	2051	17.376	366.811	2511
				floor[2*log(n)]	1905	12.870	408.883	1631

Figure 9: Table containing the results for the experiments with RRT acting on Map 2 with varying k values.*

We clearly see that increasing the value of k (when it's constant) causes the number of rewirings to increase while causing the cost to decrease - the effect on the computation time is unclear from this data. The $k \propto O(log(n))$ case proved to be the fastest, requiring the least number of iterations of all and returning path costs and rewiring attempts similar to that of $k = 5$. We also note that the value of $k = floor[2 \cdot log(n)]$ was almost always between 2 and 5 (though it obviously starts at 0, before quickly increasing).

The cost values here are almost all lower than what we were able to obtain with RRT in the previous section for Map 2 (the only outlier is the $k = 2$ case, which performed worse than the E2 trials for RRT - it is unclear why this occurred, and is likely a statistical anomaly), and this exemplifies the superiority of the RRT* algorithm for cases like this (though it is more time-consuming, as expected).

Finally, we can plot the success rate to find a solution as a function of time and the quality of the solution obtained as a function of time for one representative run:

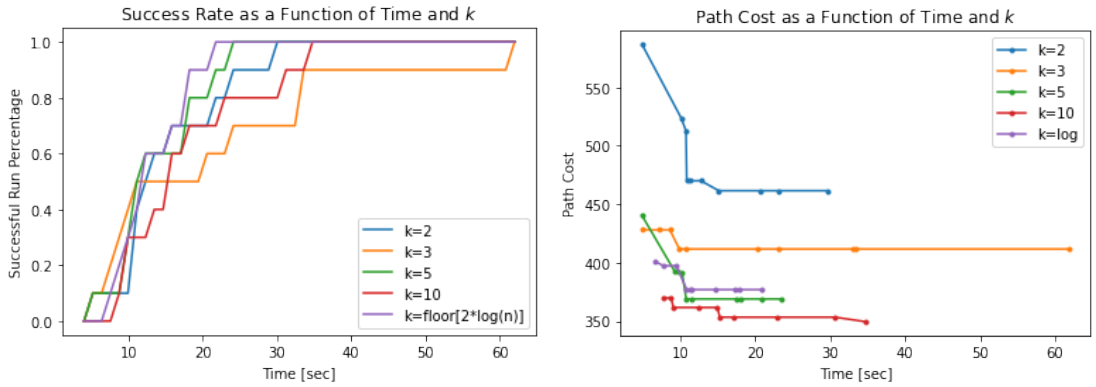


Figure 10: Plots showing the success rate as a function of time and k (left) and the path cost as a function of time and k (right) for the RRT algorithm acting on Map 2 in extension mode E2 with 5% goal biasing and step size $\eta = 10$, over 10 trials for each k.*

It's difficult to reach many conclusions looking just at the left graph, as there is no clearly monotonic relationship between k and the time needed to reach a success rate of 1 - though we note that the $k = floor[2 \cdot log(n)]$ case is the fastest to do so (i.e. it has the best minimal time required to reach a solution across all trials). We note that the $k = 5$ case reaches a solution in the shortest amount of time among all trials (4.90 seconds), closely followed by the $k = 3$ case (4.95 seconds). The $k = 3$ case had one outlier trial that required over a minute to reach a solution, and this caused it to be the last case to reach a solution across all 10 trials. It also tends to 1 at what appears to be the slowest

pace (i.e. slope) overall - followed by the $k = 10$ case - while the $k = floor[2 \cdot log(n)]$ case displays the best pace (followed by the $k = 5$ case).

In the right graph, we clearly see that higher k values produces lower-cost paths, and the $k = 10$ case was ultimately able to produce the lowest-cost path of all (with a cost of around 349.6). We also note that the $k = 10$ case took the longest amount of time to reach its best path across the 10 trials, which makes sense since it required a much larger amount of rewirings than the other cases. We see that the $k = 2$ displayed the worst performance in terms of path cost reached across all times t, while taking the second-longest amount of time to reach its best path (after the $k = 10$ case). We see that the $k = floor[2 \cdot log(n)]$ case performed similarly to the $k = 5$ case (slightly worse best-cost paths reached, though) while needing the least amount of computation time across its 10 trials.

It thus appears that to us that intermediate k values (roughly between 4 and 5) are generally the best for this case, and the logarithmic case presented a unique time advantage without too much penalty on the cost. In situations where minimizing time is more important to us, we might choose the logarithmic case, whereas for situations where minimizing cost is more pertinent to us, we might choose the $k = 5$ or the $k = 10$ cases instead.