Evaluation of RRTConnect for MSL and M2020 Arm Motion Planning

Valen Yamamoto
University of California, Irvine

Justin Huang
Mentor
Jet Propulsion Laboratory, California Institute of Technology

***Abstract*—ArmSketch is a tool used by rover planners to generate arm plans for the robotic arms on the MSL and M2020 rovers.The current motion planning algorithm it uses does not always produce arm paths that rover planners believe would be safe to execute on Mars. The RRTConnect path planning algorithm grows two rapidly-exploring random trees rooted at the start and end poses and tries to connect the two. We evaluated RRTConnect as a more general use motion planning algorithm for the rover arms. The paths produced by the RRTConnect algorithm are often long and have extraneous viapoints; path simplification algorithms post-process paths to return the most efficient paths based on the number of viapoints, the final path length, and the distance from the terrain. We evaluated the path planner with different parameters and path planning algorithms both for the metrics above and for algorithm execution time. Using a service called Proximeter to find the minimum distance between the rover arm and the terrain, we can impose a constraint for minimum distance from the terrain on generated paths. Our experiments show that the RRTConnect planner can reliably find paths with minimal viapoints and path length for the M2020 rover arm.**

I. Introduction

Rover planners use the program ArmSketch to do motion planning for the 5 DOF arms on the M2020 and MSL rovers. The current motion planning algorithm sometimes generates paths that rover planners believe to be unsafe to execute on Mars, particularly in situations with more complex constraints it was not designed for. The RRTConnect path planning algorithm is a more general path planning algorithm that can meet more of the rover planners' constraints such as keeping the arm at least 15 cm from the terrain and keeping the WATSON camera pointed below the horizon.

Additionally, rover planners prefer it when ArmSketch generates the minimum number of viapoints in order to keep command sequences simple. Because of the relatively open terrain on Mars, most paths can be planned with a single viapoint, if a viapoint is needed at all. In addition to the base RRTConnect algorithm, path simplification algorithms are needed to simplify paths to the most essential viapoints.

In ArmSketch, motion planning is run for every allowed combination of commands, which means the motion planning algorithm can run anywhere from once to tens of thousands of times; therefore, any motion planning algorithm should be able to run quickly as it may need to generate thousands of paths in a short amount of time.

II. The RRTConnect Path Planner

The RRTConnect path planner, as described in [1], grows two RRTs (rapidly-exploring random trees), one rooted at the starting pose, T_a, and one rooted at the end pose, T_b, to search the configuration space for a collision free path, as seen in Figure 1. For each iteration, the algorithm samples a random configuration q_{rand} and finds the closest configuration already in T_a, q_{near}. The algorithm then takes an epsilon (ϵ) sized step from q_{near} to q_{rand}, creating the configuration q_{new}, which is added as the child of q_{near} if there are no collisions along the path from q_{near} to q_{new}. After each iteration, the closest point in T_b to q_{new} is found and the algorithm attempts to connect the two points by taking ϵ sized steps towards q_{new} from the closest point in the other tree until either that point is reached or a collision is found along the path. T_a and T_b swap roles and the next iteration begins. With smaller ϵ values, the tree nodes tend to grow right up to obstacles before a collision occurs, leading to paths that hug obstacles. With large ϵ values, the algorithm is less likely to generate configuration near obstacles, as the longer steps have more opportunities to collide with obstacles, and is more likely to create new configurations far away from obstacles. This creates paths that stay far away from obstacles but are often longer than necessary.

A. Our Implementation

To save execution time, before the RRTConnect algorithm is run, the motion planner attempts to perform a "snap plan", or linearly interpolate, between the start and end poses. To determine whether a snap plan can be performed, the linear interpolation is collision checked at intervals set by the collision check step size, as seen in Figure 2. If the snap plan fails due to a collision somewhere inbetween the points, the RRTConnect path planning algorithm runs.

For our implementation of RRTConnect, we sampled q_{rand} in the rover arm's 5D joint space with the ϵ step size as an adjustable parameter with values between 0.1 and 5 that can also be turned off to run the algorithm without taking ϵ sized steps. MrCat, a tool that generates collision-free arm trajectories for the Mars rovers, checks for collisions with the terrain and with the rover. We check for collisions at certain points along the linearly interpolated path between two poses, Because new configurations and tree edges are subjected to collision checks, it is relatively simple to add on additional constraints to be checked at the same time.

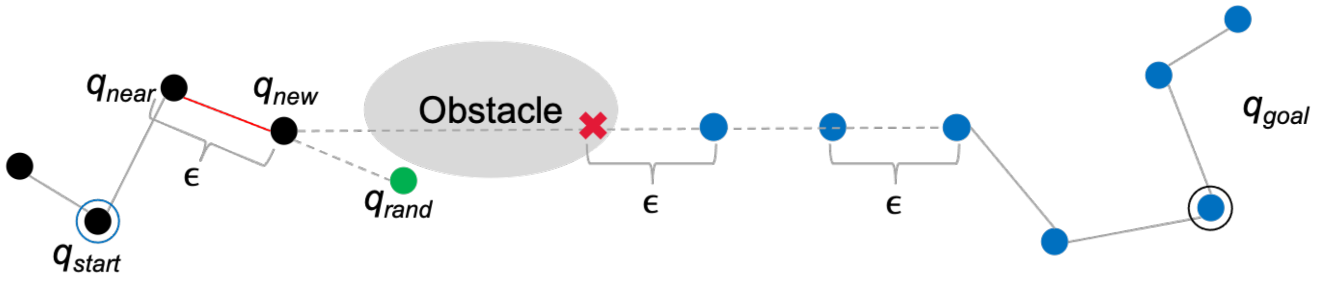

Fig. 1. Illustration of RRTConnect Algorithm. The algorithm samples q_{rand}, shown in red, and takes an ϵ size step towards it, creating q_{new} and adding the red edge. The other tree, shown in blue, tries to connect to q_{new}, taking ϵ sized steps towards it.

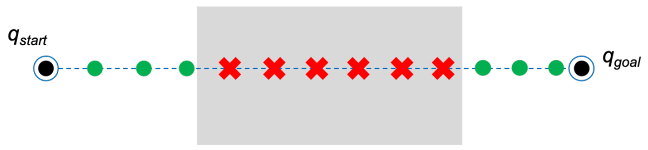

Fig. 2. Illustration of collision check process. Regularly spaced points, as set by the collision check step, along the line between two points are checked for collisions.

Because paths generated by the RRTConnect path planning algorithm are often longer than necessary and have extraneous viapoints, path simplification methods are needed. The following are path simplification algorithms we tested with the base RRTConnect algorithm:

- Naive: Beginning with the start pose, attempt to create an edge between the start pose and other viapoints in the order they appear in the path, accepting the collision-free edge between the start pose and the furthest along viapoint. Repeat with the accepted viapoint and continue until the last viapoint is connected.
- Binary Search: Much like the naive path simplification but the remaining viapoints are searched in a binary search-like manner for greater efficiency, where the first viapoint that is tried is the midpoint between the current point and the last point in the path. If the edge between the midpoint and the current point contains a collision, the viapoints between the current viapoint and the midpoint are checked in a similar fashion, otherwise the points between the midpoint and the end pose are checked.
- Divide and Conquer: Splits the path in half, simplifies each half, and combines the two simplified halves by trying to connect a pose from one half to a pose on in the other half. The base case returns a 2-pose path without any processing.
- Greedy Search: Connect every point on the path to every point after it with a directed edge and do a greedy graph search to find the next node that has the shortest distance to the goal pose and that does not have a collision along the edge.
- A*: Connect every point on the path to every other point after it with a directed edge and run a A* search-esque algorithm using a distance heuristic to find the path with smallest distance in joint space
- De Casteljau: Consider every pose along the path as a control point on a bezier curve and sample the midpoint of that bezier curve as the new single viapoint along the path. If there are no collisions along the edges between the start and end poses and the new viapoint, the algorithm can run again on the new path a second time only.
- De Casteljau Random: Like the first De Casteljau method, consider every pose along the path as a control point on a bezier curve but instead sample a random point from a normal distribution centered around the midpoint on the beizer curve as the new single viapoint. If the new path is collision-free, repeat a second time only.
- Hypercube: Consider every pose along the path as a coefficient of a Bernstein polynomial. The maximum and minimum of each joint is therefore an upper or lower limit on the path; new points are sampled from this "hypercube" to be the new single viapoint. If the path is collision-free, the algorithm can be repeated a second time only with the new path.
- Triangle Sampling: Given a path with a single viapoint, a new viapoint is sampled within the triangle formed by the original viapoints. If the new path is collision-free, the algorithm can be run again on the new path until 5 attempted viapoints create paths with collisions

Any new edges or viapoints tried during path simplification also require collision checking to ensure the arm does not hit any obstacles.

B. Proximeter

Proximeter is a tool that finds the minimum distance from the arm to the terrain. Originally, Proximeter was a part of HyperDrive, but in order to facilitate its use in other rover planning tools, we have pulled the core functionality out of HyperDrive into its own server. Using Proximeter, we can set a constraint on the minimum distance from the terrain by rejecting any q_{new} that would move the arm too close to the terrain. Proximeter can also check for collisions with the terrain but MrCat is still needed to check for self collisions. Rover planners prefer that the arm stay at least 15 cm from the terrain; this is the number we used in all of our experiments.

In the case that either the start or end pose is closer than the minimum allowed distance to the terrain, the minimum distance of viapoints is kept at the original set value, usually 15cm, but the minimum distance of points along the edges between viapoints is lowered to the minimum of the distances of the start and end poses from the terrain.

III. Experiments

The RRTConnect algorithm's main adjustable parameter is its ϵ step size. Because ϵ has an effect on the number of viapoints produced, it also has an effect on the effectiveness of path simplification algorithms. The addition of querying Proximeter has an effect both during the base RRTConnect algorithm and during path simplification; therefore, the three main tunable parameters are ϵ, the path simplification algorithm, and whether Proximeter is being used during execution.

To check how the algorithm would perform in different environments, we ran tests in two different environments:

- DevTT09 (Figure 3): A large rock in front of the rover. The arm moves from a target on one side of the rock to a target on the other side of the rock. Targets: achung_test_010, fromPointNormal_020
- Sol001351 (Figure 4): A small ridge in front of the rover. The arm moves through combinations of targets at the top, bottom, and on the face of the ridge. Targets: Nausaspoort_rp (top), Nauaspoort_Lower_Lip (lower lip), Grendraai_rp (bottom)

The best combination of ϵ and path simplification algorithm should work well in a variety of environments and a variety of tasks.

All Experiments were run on a 6 core Intel Xeon Gold 6128 CPU @ 3.40GHz running RHEL 7.9.

The algorithms are evaluated on the following metrics:

- Path Cost: Measured as distance traveled in joint space in radians. Shorter paths are better.
- Number of Viapoints: Measured as the number of points between the start and end poses. Less viapoints is better
- Algorithm Execution Time: Because the algorithms' running times are essentially dictated by the number of collision checks, the number of collision checks is sometimes used as a proxy for wall clock time. Smaller is better

A. Collision Check Step Size

The execution time of the base RRTConnect algorithm is dominated by the time it takes to do collision checks: 99% of the time spent executing the algorithm is spent doing collision checks. One way to reduce the number of collision checks is to increase the collision check step to do less collision checks. To find the maximum collision check step size before the algorithm started to skip over collisions, we swept over collision check steps from 0.0057 to 0.114 radians, running each collision check step size 20 times. All paths were then checked at a collision check step of 0.0057 and the number of unsuccessful paths was recorded.

B. Small ϵ Path Simplification

Depending on the ϵ value, paths created by RRTConnect either have extraneous viapoints or unnecessarily long paths. Path simplification is needed to clean up generated paths and optimize them for shorter path distances and fewer viapoints.

The naive, binary search, divide and conquer, greedy search, and A* algorithms work best when the ϵ is between 0.1 and 1.25 radians, creating anywhere between 5-30 viapoints. These paths tend to hug obstacles as the connect step of the RRTConnect algorithm steps until the path collides with the obstacle as the small ϵ keeps the path from moving too far away from the obstacle. Algorithms for small step sizes focus on picking the smallest subset of the original viapoints as opposed to sampling new points because there are enough known safe configurations to create a short path. To test these algorithms, we performed a sweep of over all the small ϵ path simplification algorithms for ϵ values between 0.1 and 1.2 and measured the number of collision checks to compare the time the path simplification algorithms would take to execute. The collision checks for only the path simplification algorithms and not the base RRTConnect algorithm were collected and each combination of ϵ and path simplification algorithm was run 5 times, each on a different random seed, on the DevTT09 RML.

C. Large ϵ Path Simplification

The De Casteljau inspired methods, the Hypercube sampling method, and the triangle sampling method work best with large ϵ values, which creates 1 or 2 viapoints far away from obstacles. Because the step size is so large and martian terrain is mostly open, almost any sampled point will be able to connect with the other tree; however, the sampled point may not create the most efficient path. Path simplification for paths created using a large ϵ focuses on creating or sampling new points constrained by the original viapoints as with so few viapoints, it is impossible to use the original viapoints. First, to compare the De Casteljau midpoint and random path simplification methods, 100 RRTConnect paths without any path simplification were generated without taking ϵ sized steps. Each De Casteljau algorithm was run on each of the 100 paths and the path costs and execution time in seconds was measured for the path simplification algorithm only.

To test all the large ϵ path simplification algorithms, we swept through these path simplification algorithms without taking ϵ sized steps, meaning any q_{rand} could be a q_{new} as long as it was collision free. Each algorithm was run 50 times on the DevTT09 RML with different random seeds and the execution time as wall clock time and the path cost were collected.

D. ϵ and Path Simplification Algorithm Sweep

To compare the performance of the RRTConnect path planning algorithm at different ϵ values with different path simplification algorithms, the algorithm is initially seeded with one of 50 predetermined random seeds. This allows direct comparison between path generated at different ϵ values

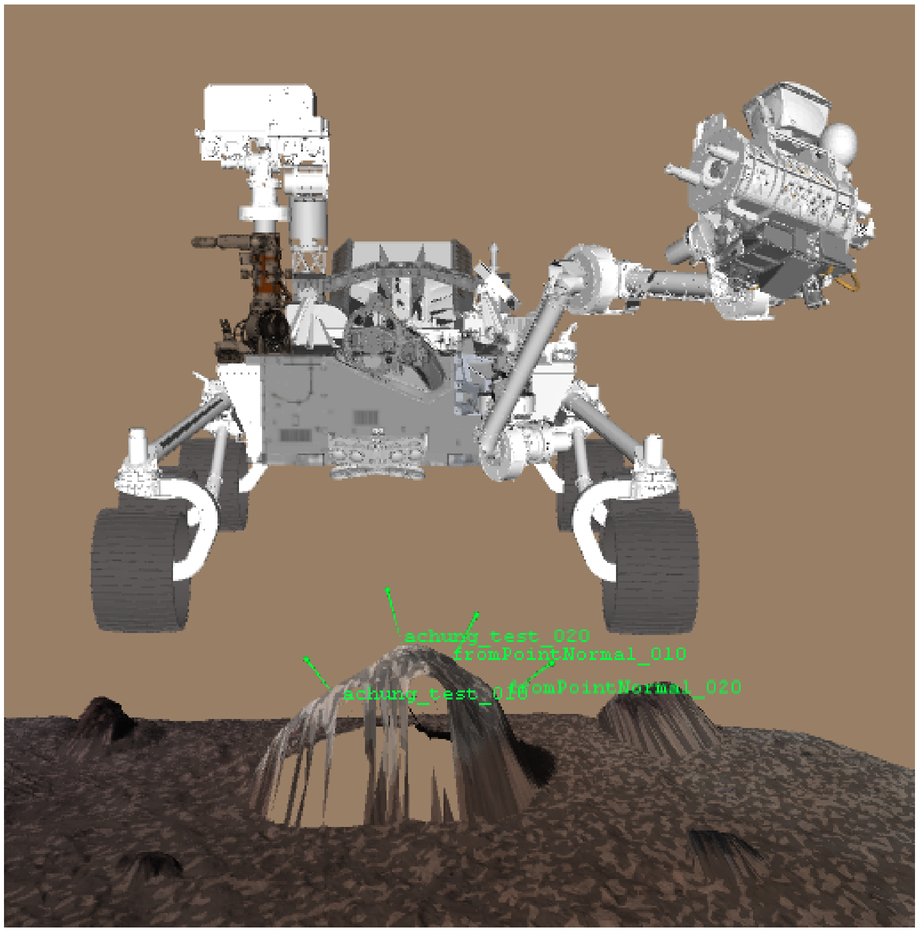

Fig. 3. Screenshot of the DevTT09 RML

as well as comparison of simplification algorithms for raw RRTConnect paths generated with the same ϵ step size and random seed.

For these tests, ϵ could assume a value of 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, or 5 to get a range of different RRT paths generated and see how path simplification algorithms performed with different numbers of viapoints. Each combination path simplification algorithm, ϵ, and random seed is used to plan each combination of two targets in each environment. The path cost, the number of collision checks for the combined RRTConnect algorithm and path simplification algorithm, and the number of viapoints generated were measured. In order to compare all path results across different planning scenarios, the results on each scenario were standardized and averaged per path simplification and epsilon comparison.

E. Random Retries

Path simplification requires additional collision checks, which increases the total algorithm execution time. To try and reduce the number of collision checks, the RRTConnect algorithm could just be rerun from scratch to see if a better path could be found. Because the terrain on Mars tends to be very open, almost any randomly sampled configuration could serve as the single viapoint along the path. Many configurations in the free space would create unnecessarily long paths, but rerunning the planner from scratch could potentially find a good path. The RRTConnect algorithm with path simplification returns a path cost of 2.7 planning around the DevTT09 rock, so in order to be comparable, the RRTConnect algorithm with random restarts should find paths that have a path cost of 3 or less. In order to test this, we ran the RRTConnect algorithm with up to 100 random restarts 40 times, each with a different random seed, and recorded the best path cost and the execution time of all the restarts needed. If the algorithm found a path with a cost less than 3, it could stop execution early and return the path.

F. Experiments with Proximeter

With Proximeter, paths can be constrained such as to not to get too close to the terrain. Both viapoints and points along the edges between viapoints need to be checked for distance in order to fulfill this constraint. Adding a query to the Proximeter server to get the minimum distance from the terrain adds an additional 0.012 seconds to every collision check. As such the execution time with Proximeter is 3 to 4 times longer than just doing collision checks with the terrain.

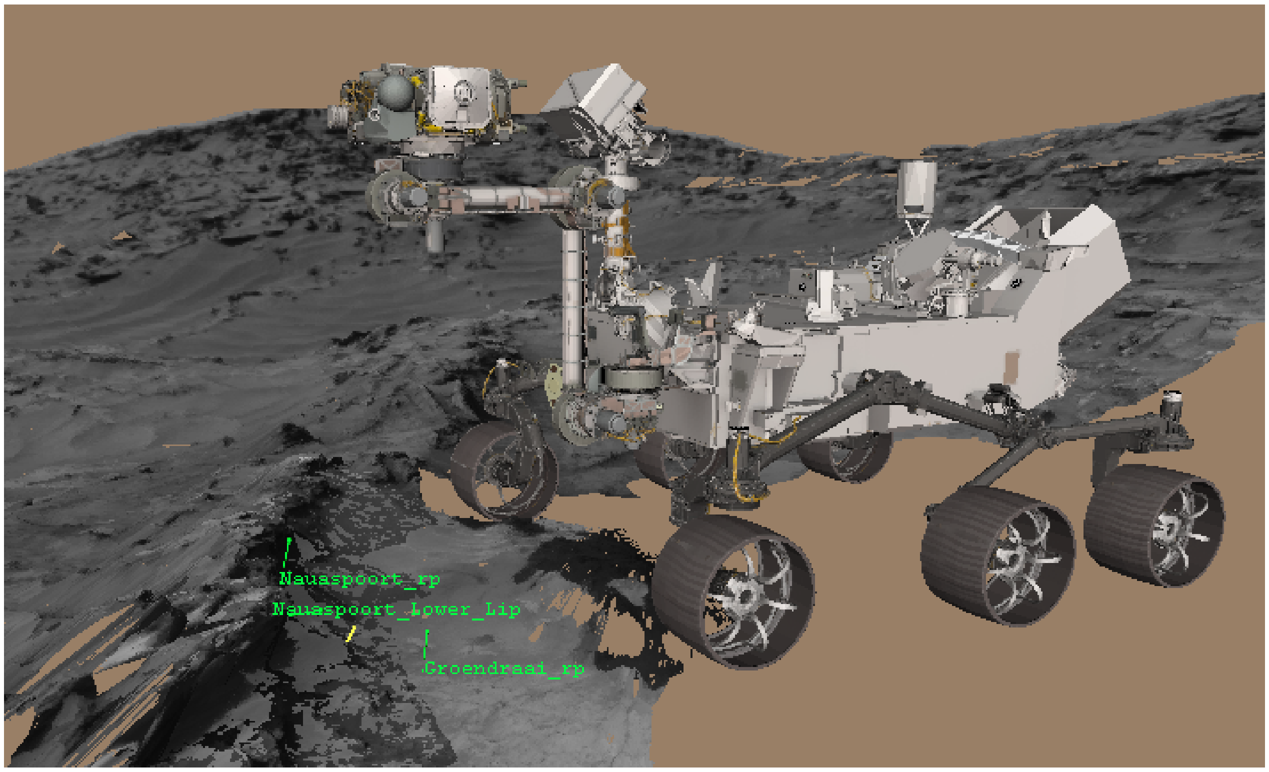

Fig. 4. Screenshot of the Sol001351 RML

The RRTConnect algorithm was run again on with the same 9 ϵ values with all the path simplification algorithms with the same 50 random seeds on the same RMLs. Similarly, we measured the path cost, the number of collision checks for both the RRT-Connect algorithm and the path simplification algorithm. To compare total performance across multiple planning scenarios, the results for each scenario were standardized and averaged per path simplification algorithm and epsilon combination for all scenarios.

IV. Results

A. Collision Check Step Size

The percentage of paths that were generated without skipping collisions at various collision check steps is shown in Figure 5. The algorithm starts to miss collisions at a collision check step of 0.057 and misses more as collision check step is increased. For the rest of the experiments, a collision check step of 0.045 is used.

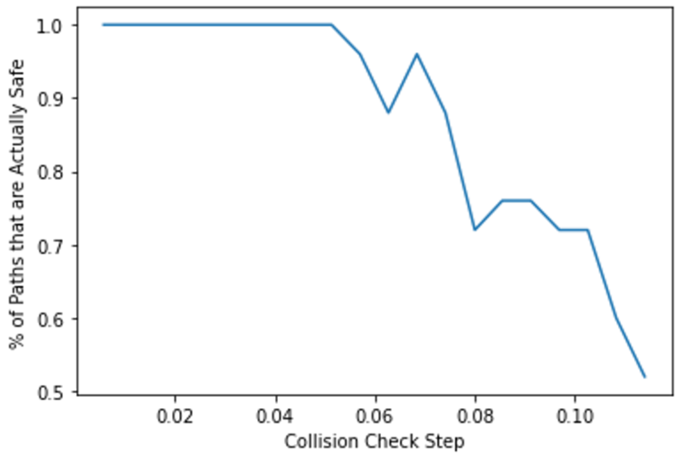

Fig. 5. The percentage of collision-free paths generated as collision check step is increased. Collisions begin to be missed at a collision check step of 0.57

B. Small ϵ Path Simplification

During the sweep of small ϵ path simplification algorithms, on average 20 to 25 viapoints are created by the RRTConnect algorithm for each path. The naive and binary search methods return paths with a single viapoint around 75% of the time. The binary search runs in around 60% of the time it takes to run the naive algorithm with essentially the same end path length because it runs less collision checks. The greedy search has the shortest wall clock time and runs in 59% of the time it takes to run the binary search algorithm; however, is likely to generate 2 or more viapoints rather than just the single viapoint preferred by rover planners. A* search also takes noticeably longer–over 2.2 times longer than the binary search algorithm–while also giving the shortest path cost. The divide and conquer algorithm has a longer wall clock time and does not consistently produce a single viapoint.

C. *Large ϵ Path Simplification*

In the comparison of the De Casteljau methods, the De Casteljau Random algorithm generally provided shorter paths than the midpoint De Casteljau algorithm. The distributions of the costs of the paths generated by the two simplification algorithm can be seen in Table I. A two-sample one-tailed t-test gave a pvalue of 0.001, meaning that the random algorithm gave shorter paths by a statistically significant margin. The speed of the algorithms were essentially the same, so the De Casteljau Random algorithm is better option than the midpoint algorithm for shorter paths.

Path Cost	Midpoint De Casteljau	Random De Casteljau
Mean	2.856	2.500
Std. Dev.	1.027	0.4929

TABLE I
COMPARISON OF THE DE CASTELJAU MIDPOINT AND DE CASTELJAU RANDOM PATH SIMPLIFICATION ALGORITHMS, RUN ON THE SAME 100 PATHS

For the test of large ϵ path simplification algorithms, paths with 1-2 viapoints were generated. The triangle sampling path simplification method produced shorter paths than the De Casteljau random and Hypercube sampling methods. All three algorithms were run on the same paths, and the paths created by the triangle sampling method had a mean cost of 2.27 and a standard deviation of 0.973 while the De Casteljau and Hypercube sampling paths had mean path costs around 2.58 and standard deviations of 0.93 and 0.58 respectively. A two-sample one-tailed t-test of the path costs of the triangle sampling generated paths versus the paths generated by the De Casteljau and Hypercube methods gave pvalues of 0.05 and 0.02, indicating that the paths generated by triangle sampling are significantly shorter than those created by other algorithms. However, the triangle sampling algorithm takes longer to run than the De Casteljau method, with a mean algorithm duration of 1.02 seconds compared to 0.649 seconds. This longer wall clock time is probably because the triangle sampling method requires more collision checks and does not have a cap on the number of new collision-free configurations sampled like the De Casteljau and Hypercube methods.

D. *ϵ and Path Simplification Algorithm Sweep*

A comparison of the number of collision checks for the sweep of all path algorithms with the best ϵ values for each path simplification algorithm on DevTT09 is in shown in Figure 6 and a comparison of the path costs for the best ϵ for each path simplification algorithm is shown in Figure 7 for the DevTT09 RML. Most path simplification algorithms converge on the same path cost except for the divide and conquer algorithm, which has a distribution of costs that is higher than the other path simplification algorithms. Of all the path simplification algorithms, the triangle sampling algorithm requires the most collision checks and the De Casteljau Random method has the second highest number of collision checks while the rest of path simplification algorithms produce most paths within 100 to 200 collision checks.

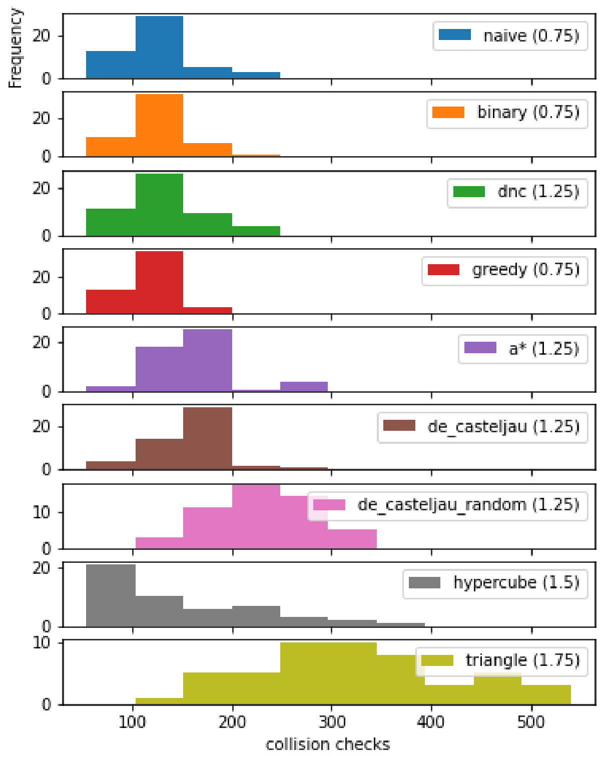

Fig. 6. A comparison of the number of collision checks for different path simplification algorithms planning around the DevTT09 rock. All path simplification algorithms except the De Casteljau Random and Triangle Sampling Methods generated paths within 100 to 200 collision checks.

A comparison of the 10 best performing combinations of parameters can be seen for DevTT09 in Table V, moving from the top to the bottom of the ridge in Sol001351 in Table VII, moving from the lower lip of the ridge to the top of the ridge in Table VI, and moving from the lip to the bottom of the ridge in Table VIII. The average standardized scores (subtract the mean, divide by the standard deviation, smaller values are better) of the 10 best performing algorithms can be seen in Table II. We choose the binary search path simplification algorithm with ϵ=0.5 as the best combination of parameters because it had the best performance in path cost and had good performance in both the number viapoints and the number collision checks. With these parameters, 141 out of 150 paths generated across both environments returned a single viapoint. However, many of the generated paths have minimum distances from the terrain that are below the 15 cm requested by rover planners.

E. *Random Retries*

In the test of using random restarts instead of path simplification, only 17 out of 40 trials where able to find a path with

Simplification Algorithm (ϵ)	Standardized Viapoints	Standardized Path Cost	Standardized Collision Checks
Binary (0.75)	**-0.29**	-0.44	-0.52
Binary (0.5)	***-0.29***	***-0.48***	*-0.40*
Greedy (1)	-0.24	-0.38	**-0.75**
Naive (0.75)	**-0.29**	-0.44	-0.30
A* (1.25)	**-0.29**	-0.33	-0.30
De Casteljau (1.5)	-0.24	-0.32	-0.38
De Casteljau (1.75)	-0.26	-0.29	-0.29
Hypercube (2)	-0.16	0.01	-0.48

TABLE II
AVERAGE STANDARDIZED METRICS FOR ALL PATH PLANNING SCENARIOS WITHOUT PROXIMETER

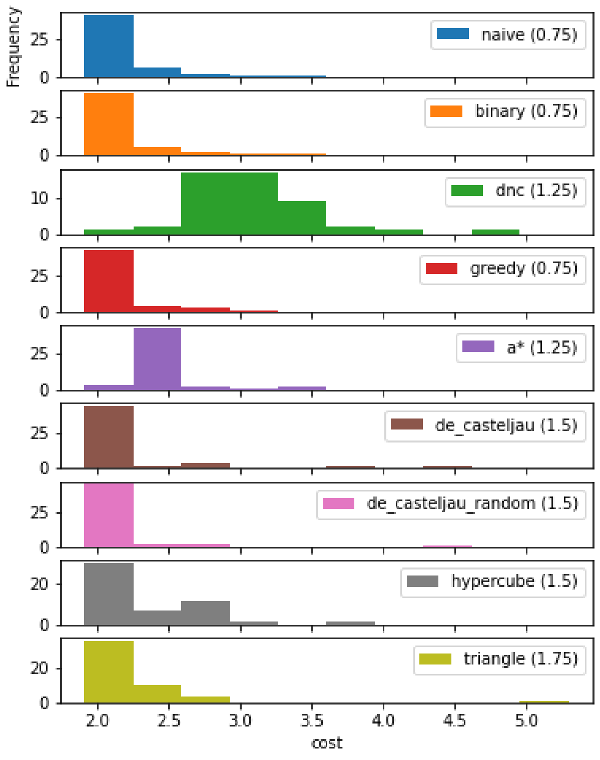

Fig. 7. A comparison of the path cost for paths generated by different path simplification algorithms planning around the DevTT09 rock. All path simplification converge on the same path cost except for the A* and Divide and Conquer (dnc) algorithms.

a cost equal to or less than the path generated by the current motion planning algorithm in ArmSketch. Additionally, the time taken to find these paths is prohibitively long, as seen in Table III. With path simplification, the RRTConnect algorithm takes between 0.5 and 2 seconds to run; a mean execution time of 24.91 seconds to find a comparable path is not fast enough to be a usable replacement in rover planners' workflow. Therefore, path simplification is still a better option both for minimal path cost and shorter execution time.

	Count	Algorithm Duration (s)	Path Cost (radians)
"Good" Paths	17	24.91 (16.701)	2.609 (0.265)
All Paths	40	54.07 (27.57)	5.17 (2.67)

TABLE III
RESULTS OF THE RRTCONNECT RANDOM RESTARTS TRIALS

F. Results with Proximeter

We performed a similar sweep of the parameters with the same random seeds with Proximeter turned on. A comparison of collision checks and path costs per algorithm can be seen in Figures 8 and 9. Paths generated using Proximeter mostly take between 100 to 200 collision checks, which is similar to the paths that were generated without Proximeter, and the path simplification algorithm with the most collision checks is similarly the triangle sampling method. Also similarly to the paths generated without Proximeter, all the path simplification methods converge on similar path costs except the divide and conquer method and, to a lesser extend, the A* algorithm.

With Proximeter, the plan from the lower lip to the bottom of the ridge in Sol001351 in the shoulder out configuration requires at least one viapoint to maintain the minimum distance from the terrain; a comparison of the ten best performing algorithms for this planning scenario can be seen in Table XIII. Additionally, similar comparison for DevTT09, moving from the top to the bottom of the ridge, moving from the top to the lower lip of the ridge, and moving from the lip to the bottom of the ridge in the shoulder in configuration on Sol001351 can be seen in Tables IX, XI, X, and XII. Standardized results across all planning scenarios can be seen in Table IV. We choose the combination of binary search and ϵ=1 because although it did not have the outright best in any category, it had the most consistently high performance in every category.

V. ANALYSIS

In all the trials, the RRTConnect never fails to find a path. All paths were found within 100 iterations of the algorithm.

The most important metrics for the RRTConnect planner are the path cost and the number of viapoints. With and without Proximeter, the RRTConnect planner with path simplification produces paths with a single viapoint on multiple terrains at a high enough frequency, over 90% both with and without Proximeter, that if a path with multiple viapoints were found, the planner could just be rerun from scratch with a high probability of a single viapoint result. All path simplification

Simplification Algorithm (ϵ)	Standardized Viapoints	Standardized Path Cost	Standardized Collision Checks
Binary (1)	*-0.32*	*-0.33*	*-0.45*
Binary (1.25)	**-0.33**	-0.27	-0.46
Naive (1.25)	**-0.33**	-0.27	-0.41
Greedy (1.25)	-0.25	-0.29	**-0.64**
A* (1.25)	-0.32	-0.31	-0.20
De Casteljau (2)	-0.23	-0.07	-0.36
De Casteljau Random (2)	-0.30	**-0.36**	0.57
hypercube (2)	-0.21	-0.10	-0.36

TABLE IV
AVERAGE STANDARDIZED METRICS FOR ALL PATH PLANNING SCENARIOS WITH PROXIMETER

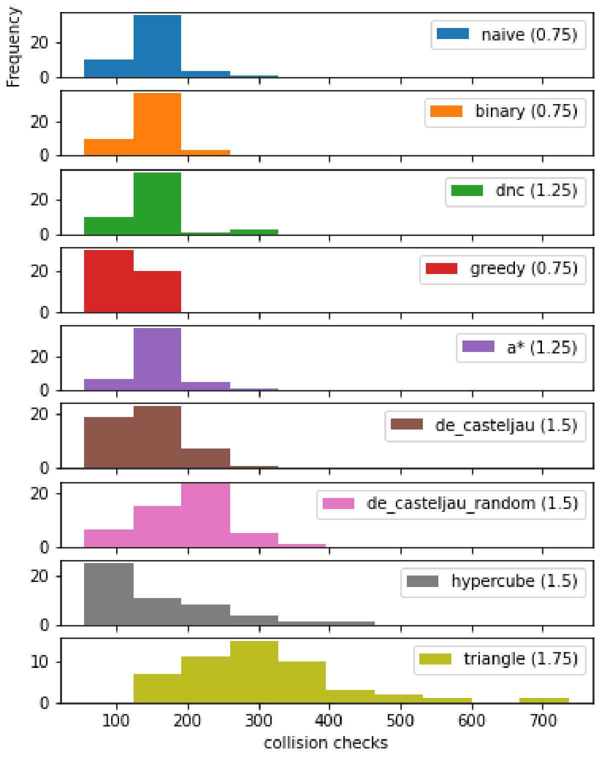

Fig. 8. A comparison of the number of collision checks for different path simplification algorithms using Proximeter, planning around the DevTT09 rock. This has a similar distribution to the results without using Proximeter, but the tail of Triangle Sampling method goes on longer.

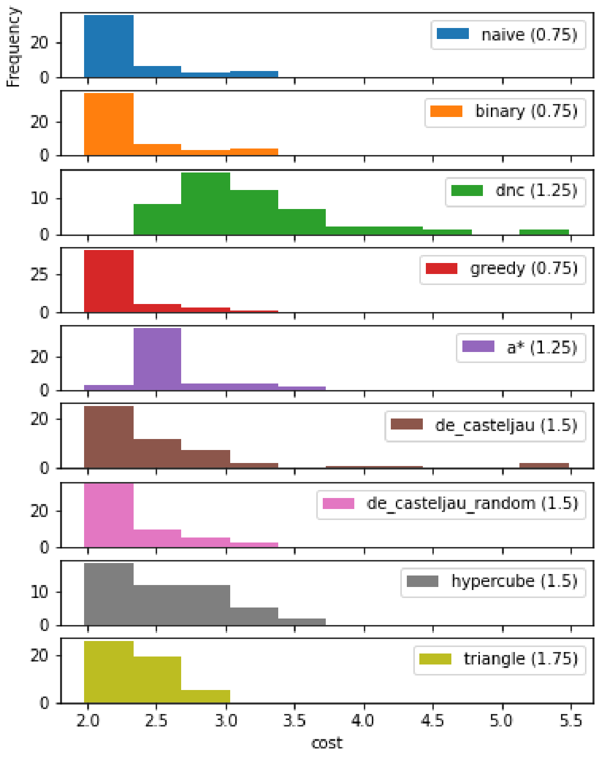

Fig. 9. A comparison of the path cost for paths generated by different path simplification algorithms using Proximeter, planning around the DevTT09 rock. Compared to the path costs generated without Proximeter, the large ϵ path simplification algorithms and Divide and Conquer (dnc) algorithms have more paths with larger path costs.

algorithms converged on similar path costs, though some differed in number of viapoints or number of collision checks needed to find that result. The number of collision checks needed for paths generated with Proximeter is only slightly higher than the number needed without Proximeter; however, the time spent calling both Proximeter and MrCat means that although the collision check numbers are similar, the time taken by the planner using Proximeter is much higher. Without using Proximeter, RRTConnect with path simplification takes around 1 second to run; with Proximeter, the algorithm can take between 2 and 5 seconds.

Without Proximeter, the small ϵ path simplification algorithms often produce paths too close to the terrain, as seen with the naive and binary algorithms with ϵ=0.5, where the mean minimum distance is 0.15 m but the standard deviation is 0.08, meaning a sizable proportion of paths get closer than 0.15 m. Using slightly larger step sizes, such as 0.75 and 1.25, usually generates paths that are far enough away from the terrain even without Proximeter but provides no guarantee that all paths generated will meet the minimum distance constraint. It is possible that Proximeter could be turned on for specific paths: if a path generated without Proximeter comes too close to the terrain, the rover planner could choose to use Proximeter at the cost of additional execution time spent querying Proximeter.

After all the experimentation, the binary search path simplification algorithm most consistently generated paths with small path costs and minimal viapoints in the least amount of collision checks, both with and without Proximeter, though the ϵ step size is larger with Proximeter. The RRTConnect path planner with binary search both consistently provides a single viapoint and has short paths in joint space.

VI. Further Work

The RRTConnect planner takes too long for large ArmSketch plans and work can be done to exploit parallelism to decrease the execution time. Additionally, it is possible that Proximeter could be queried less frequently than at every collision check and still meet the minimum distance constraint, which would greatly reduce the amount of time per collision check. Calls to Proximeter and MrCat could potentially be done in bulk as to decrease the amount of overhead in calling them.

Work is currently being done in ArmSketch to utilize parallelism. Currently, all combinations of actions in a sequence are checked in a single thread; parallelizing them could decrease the amount of wall clock time to find the best sequence and also make the larger execution time of the RRTConnect algorithm less noticable.

The RRTConnect algorithm could be ported to C++ as a part of Cat, the backend of MrCat, which could decrease the execution time hopefully through gains from C++'s better execution time as compared to Python and from being a few layers of indirection closer to where the collision check code is.

The constraint of keeping WATSON pointed below the horizon was not thoroughly explored before this report and be easily added to the process of generating new configurations. All proposed q_{new} configurations would be checked to ensure that both q_{new} and the path between q_{near} and q_{new} does not violate an orientation constraint before being added to the tree, similar to how Proximeter is used to keep a minimum distance constraint.

Given that all that is needed to run this motion planning algorithm is the ability to detect collisions with the terrain and the rover itself, it is possible that it can be deployed autonomously on future missions where ground-in-the-loop communications are more difficult than they are on Mars or if the robotic arm should do its own path planning without human intervention.

VII. Acknowledgements

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the JPL Education Office and the National Aeronautics and Space Administration (80NM0018D0004).

References

[1] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000.

APPENDIX

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (0.75)	1.02 (0.14)	2.13 (0.28)	129.56 (26.16)	0.19 (0.04)
Binary (0.5)	**1.00 (0.00)**	**2.02 (0.11)**	127.72 (22.61)	0.15 (0.06)
Greedy (1)	1.08 (0.27)	2.28 (0.32)	**120.00 (21.47)**	0.21 (0.04)
Naive (0.75)	1.02 (0.14)	2.13 (0.28)	130.04 (33.40)	0.19 (0.04)
A* (1.25)	1.02 (0.14)	2.48 (0.25)	158.44 (43.76)	0.22 (0.01)
De Casteljau (1.5)	1.06 (0.31)	2.20 (0.44)	162.68 (42.41)	0.19 (0.05)
De Casteljau (1.75)	1.06 (0.31)	2.24 (0.54)	176.44 (47.33)	0.20 (0.04)
Hypercube (2)	**1.00 (0.00)**	2.21 (0.26)	218.58 (77.82)	0.21 (0.03)

TABLE V
DEVTT09 ACHUNG_TEST_010 TO FROMPOINTNORMAL_020, NO PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (0.75)	1.04 (0.20)	3.32 (0.23)	221.02 (34.52)	0.07 (0.02)
Binary (0.5)	1.04 (0.20)	**3.28 (0.18)**	238.42 (42.03)	0.06 (0.03)
Greedy (1)	1.12 (0.33)	3.39 (0.32)	**181.42 (27.91)**	0.07 (0.02)
Naive (0.75)	1.04 (0.20)	3.32 (0.22)	244.28 (32.10)	0.07 (0.02)
A* (1.25)	**1.02 (0.14)**	3.46 (0.30)	229.06 (45.87)	0.07 (0.02)
De Casteljau (1.5)	1.18 (0.60)	3.47 (0.52)	225.22 (48.08)	0.07 (0.02)
De Casteljau (1.75)	1.14 (0.45)	3.56 (0.66)	234.76 (55.11)	0.07 (0.02)
Hypercube (2)	1.30 (0.46)	3.77 (0.49)	184.64 (135.03)	0.07 (0.01)

TABLE VI
SOL001351 NAUASPOORT_RP SO EU WD TO NAUASPOORT_LOWER_LIP SO EU WU, NO PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (0.75)	1.12 (0.33)	4.86 (0.14)	329.30 (33.14)	0.19 (0.07)
Binary (0.5)	1.14 (0.35)	**4.80 (0.14)**	383.40 (39.93)	0.17 (0.08)
Greedy (1)	1.64 (0.85)	4.90 (0.30)	260.76 (31.62)	0.19 (0.06)
Naive (0.75)	1.12 (0.33)	4.86 (0.14)	436.10 (56.09)	0.19 (0.07)
A* (1.25)	**1.04 (0.20)**	4.87 (0.24)	390.94 (77.51)	0.19 (0.07)
De Casteljau (1.5)	1.56 (1.16)	5.18 (0.76)	296.80 (56.68)	0.22 (0.07)
De Casteljau (1.75)	1.32 (0.87)	5.20 (0.82)	316.88 (58.38)	0.23 (0.04)
Hypercube (2)	2.10 (0.89)	5.78 (1.04)	**237.18 (181.16)**	0.22 (0.07)

TABLE VII
SOL001351 NAUASPOORT_RP SO EU WD TO GRENDRAAI_RP SO EU WU, NO PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (0.75)	**1.00 (0.00)**	4.11 (0.10)	268.22 (19.14)	0.07 (0.01)
Binary (0.5)	**1.00 (0.00)**	**4.09 (0.06)**	297.28 (30.22)	0.07 (0.01)
Greedy (1)	1.02 (0.14)	4.13 (0.15)	**218.54 (21.60)**	0.07 (0.01)
Naive (0.75)	**1.00 (0.00)**	4.11 (0.10)	315.52 (24.91)	0.07 (0.01)
A* (1.25)	**1.00 (0.00)**	4.20 (0.26)	312.60 (35.75)	0.07 (0.01)
De Casteljau (1.5)	**1.00 (0.00)**	4.15 (0.09)	322.88 (30.94)	0.07 (0.00)
De Casteljau (1.75)	**1.00 (0.00)**	4.17 (0.08)	335.44 (29.04)	0.07 (0.00)
Hypercube (2)	1.50 (0.61)	4.78 (0.64)	243.92 (122.45)	0.07 (0.01)

TABLE VIII
SOL001351 NAUASPOORT_LOWER_LIP SI EU WU TO GRENDRAAI_RP SI EU WU, NO PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (1)	1.08 (0.27)	2.54 (0.57)	141.16 (39.99)	0.22 (0.03)
Binary (1.25)	1.04 (0.20)	2.65 (0.52)	140.90 (34.39)	0.22 (0.00)
Naive (1.25)	1.04 (0.20)	2.65 (0.52)	**135.62 (35.53)**	0.22 (0.00)
Greedy (1.25)	1.12 (0.33)	2.62 (0.42)	137.02 (25.58)	0.22 (0.01)
A* (1.25)	1.04 (0.20)	2.58 (0.32)	162.82 (37.47)	0.22 (0.00)
De Casteljau (2)	1.12 (0.44)	2.80 (1.04)	167.32 (56.06)	0.22 (0.00)
De Casteljau Random (2)	**1.00 (0.00)**	2.39 (0.42)	219.36 (67.13)	0.22 (0.01)
Hypercube (2)	**1.00 (0.00)**	**2.39 (0.35)**	219.34 (83.93)	0.22 (0.00)

TABLE IX
DEVTT09 ACHUNG_TEST_010 TO FROMPOINTNORMAL_020, PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (1)	**1.00 (0.00)**	3.55 (0.27)	216.36 (33.25)	0.21 (0.06)
Binary (1.25)	1.02 (0.14)	3.74 (0.40)	219.68 (31.95)	0.26 (0.09)
Naive (1.25)	1.02 (0.14)	3.74 (0.40)	211.70 (37.09)	0.26 (0.09)
Greedy (1.25)	1.10 (0.30)	3.73 (0.36)	194.92 (27.08)	0.26 (0.09)
A* (1.25)	1.02 (0.14)	3.73 (0.36)	237.72 (39.23)	0.26 (0.09)
De Casteljau (2)	1.18 (0.39)	3.81 (0.66)	205.28 (77.34)	0.28 (0.11)
De Casteljau Random (2)	1.02 (0.14)	**3.54 (0.35)**	273.42 (75.79)	0.20 (0.07)
Hypercube (2)	1.38 (0.49)	3.89 (0.48)	**169.50 (97.74)**	0.31 (0.12)

TABLE X
SOL001351 NAUASPOORT_RP SO EU WD TO NAUASPOORT_LOWER_LIP SO EU WU, PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (1)	1.24 (0.43)	5.25 (0.36)	380.64 (67.50)	0.24 (0.02)
Binary (1.25)	1.42 (0.57)	5.01 (0.24)	466.98 (87.30)	0.23 (0.03)
Naive (1.25)	1.24 (0.43)	5.12 (0.25)	419.30 (60.62)	0.24 (0.02)
Greedy (1.25)	1.24 (0.43)	5.25 (0.36)	335.88 (47.30)	0.24 (0.02)
A* (1.25)	1.40 (0.53)	**5.01 (0.23)**	349.76 (53.25)	0.23 (0.03)
De Casteljau (2)	**1.22 (0.42)**	5.12 (0.25)	**329.38 (44.30)**	0.24 (0.02)
De Casteljau Random (2)	1.26 (0.44)	5.03 (0.20)	466.88 (66.44)	0.24 (0.02)
Hypercube (2)	1.24 (0.43)	5.25 (0.37)	403.52 (73.39)	0.24 (0.02)

TABLE XI
SOL001351 NAUASPOORT_RP SO EU WD TO GRENDRAAI_RP SO EU WU, PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (1)	1.04 (0.20)	4.38 (0.42)	282.26 (59.03)	0.19 (0.07)
Binary (1.25)	**1.00 (0.00)**	4.33 (0.35)	275.62 (44.03)	0.21 (0.07)
Naive (1.25)	**1.00 (0.00)**	4.35 (0.39)	292.52 (53.75)	0.20 (0.07)
Greedy (1.25)	1.08 (0.27)	4.27 (0.29)	**231.52 (27.08)**	0.19 (0.06)
A* (1.25)	**1.00 (0.00)**	4.31 (0.35)	322.26 (56.37)	0.19 (0.06)
De Casteljau (2)	1.10 (0.58)	4.38 (0.96)	337.90 (53.51)	0.20 (0.06)
De Casteljau Random (2)	**1.00 (0.00)**	**4.10 (0.07)**	494.46 (75.85)	0.14 (0.04)
Hypercube (2)	1.64 (0.66)	4.86 (0.63)	237.94 (139.69)	0.32 (0.12)

TABLE XII
SOL001351 NAUASPOORT_LOWER_LIP SI EU WU TO GRENDRAAI_RP SI EU WU, PROXIMETER

Simplification Algorithm (ϵ)	Viapoints (stdev)	Path Cost (stdev)	Collision Checks (stdev)	Distance (stdev)
Binary (1)	1.06 (0.31)	2.95 (0.74)	201.72 (66.34)	0.19 (0.07)
Binary (1.25)	**1.00 (0.00)**	3.01 (0.80)	194.82 (67.68)	0.20 (0.08)
Naive (1.25)	**1.00 (0.00)**	3.01 (0.80)	198.70 (81.72)	0.20 (0.08)
Greedy (1.25)	1.08 (0.27)	2.92 (0.69)	**173.62 (42.55)**	0.20 (0.07)
A* (1.25)	1.04 (0.20)	2.92 (0.69)	225.20 (80.29)	0.20 (0.07)
De Casteljau (2)	1.28 (0.76)	3.35 (1.78)	221.44 (57.68)	0.22 (0.08)
De Casteljau Random (2)	1.16 (0.62)	2.86 (1.55)	272.48 (67.36)	0.15 (0.05)
Hypercube (2)	1.06 (0.24)	**2.72 (0.79)**	264.76 (111.29)	0.19 (0.07)

TABLE XIII
SOL001351 NAUASPOORT_LOWER_LIP SO EU WU TO GRENDRAAI_RP SO EU WU, PROXIMETER