1-1-2013

Path Planning with a Lazy Significant Edge Algorithm (LSEA)

Joseph Polden
University of Wollongong, jpolden@uow.edu.au

Zengxi Pan
University of Wollongong, zengxi@uow.edu.au

Nathan Larkin
University of Wollongong, nlarkin@uow.edu.au

Stephen Van Duin
University of Wollongong, svanduin@uow.edu.au

Path Planning with a Lazy Significant Edge Algorithm (LSEA)

Abstract

Probabilistic methods have been proven to be effective for robotic path planning in a geometrically complex environment. In this paper, we propose a novel approach, which utilizes a specialized roadmap expansion phase, to improve lazy probabilistic path planning. This expansion phase analyses roadmap connectivity information to bias sampling towards objects in the workspace that have not yet been navigated by the robot. A new method to reduce the number of samples required to navigate narrow passages is also proposed and tested. Experimental results show that the new algorithm is more efficient than the traditional path planning methodologies. It was able to generate solutions for a variety of path planning problems faster, using fewer samples to arrive at a valid solution.

Keywords

lsea, planning, algorithm, path, edge, significant, lazy

Disciplines

Engineering | Science and Technology Studies

Publication Details

Polden, J., Pan, Z., Larkin, N. & Van Duin, S. (2013). Path Planning with a Lazy Significant Edge Algorithm (LSEA). International Journal of Advanced Robotic Systems, 10 (198), 1-8.

INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

Path Planning with a Lazy Significant Edge Algorithm (LSEA)

Regular Paper

Joseph Polden[1,*], Zengxi Pan[1], Nathan Larkin[1] and Stephen Van Duin[1]

1 Defence Material Technology Centre, Faculty of Engineering, University of Wollongong, Australia
* Corresponding author E-mail: jwp973@uowmail.edu.au

Received 21 Jun 2012; Accepted 18 Sep 2012

DOI: 10.5772/53516

Abstract Probabilistic methods have been proven to be effective for robotic path planning in a geometrically complex environment. In this paper, we propose a novel approach, which utilizes a specialized roadmap expansion phase, to improve lazy probabilistic path planning. This expansion phase analyses roadmap connectivity information to bias sampling towards objects in the workspace that have not yet been navigated by the robot. A new method to reduce the number of samples required to navigate narrow passages is also proposed and tested. Experimental results show that the new algorithm is more efficient than the traditional path planning methodologies. It was able to generate solutions for a variety of path planning problems faster, using fewer samples to arrive at a valid solution.

Keywords Path Planning, Lazy Evaluation, Probabilistic Roadmap (PRM), Bridge Test

1. Introduction

The general path planning problem is given as; "planning a collision free path for a robot made of an arbitrary number of polyhedral bodies among an arbitrary number of polyhedral obstacles, between two collision free positions of the robot" [1]. Path planning methods developed during earlier years approached this problem by generating explicit representations of the robots surrounding environment, so that it can be navigated via the use of mathematical algorithms. However, as path planning applications became more complex, these planners began to struggle with the amount of data required to process a valid solution.

A major breakthrough in the field of path planning came with the development of probabilistic path planning methods, such as the Probabilistic Roadmap (PRM) planner [2]. These planners utilize a randomized sampling based approach, which builds a simplified model of the robot's free space. This removes the high computational load involved in calculating an explicit representation of the environment. The random nature of sampling the environment ensures that probabilistic planners have a quality known as probabilistic completeness: if a solution is possible the planner will find it, provided that the time frame is not finite. Probabilistic methods, however, suffer from the fact that they cannot explicitly recognize if a solution will be geometrically impossible.

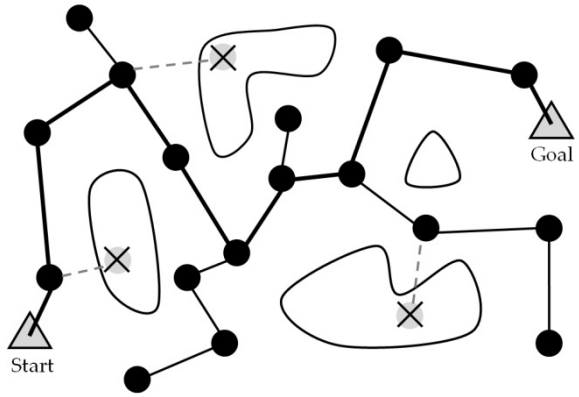

Figure 1. A network, or roadmap, graph ($\dot{G}$) represents the C_{free} space of a robots environment.

Probabilistic planners have been utilized extensively since their development and have been applied to a variety of different applications. The research presented in this paper focuses on the development of new, more effective, sampling methods which are implemented on a traditional lazy probabilistic path planner [3]. This is done through an iterative process of roadmap analysis and enhancement that intelligently guides the construction of the network graph. The overall result is a planner with a similar operation to traditional methods, but that is able to solve path planning queries in complex environments more quickly.

2. Background and Literature Review

Probabilistic methods generally operate on the premise of a robot and its associated configuration space (C_{space}). Loosely termed, C_{space} is the group of all kinematically feasible configurations of a robot about its workspace. For path planning applications, it is common to separate C_{space} into two distinct subsets; C_{free} and C_{forbid}. C_{free} represents the free configuration space of the robot: a group of configurations that do not clash with the surrounding environment. Conversely C_{forbid}, the forbidden configuration space, represents a collection of robot configurations that are. Generally speaking, probabilistic methods try to capture a simplified representation of the robot's C_{free} space. Once the planner has a sufficient understanding of the regions the robot can move about in without clashing, a series of clash free motions to solve a given path planning query can be generated.

An important tool utilized by a variety of probabilistic planners is an undirected network graph, commonly referred to as the Roadmap ($\dot{G}$), as shown in Figure 1. The nodes of $\dot{G}$ are used to represent individual C_{free} robot configurations captured by the sampler and the interlinking edges are used to represent a clash free motion between them. $\dot{G}$ can be used to solve a given path planning query by linking start/goal configurations (q_{start} & q_{goal}, respectively) to $\dot{G}$ and then using traditional graph searching techniques to trace a path from q_{start} to q_{goal}. This path can then be translated into a series of clash free motions used to direct the robot's subsequent movements. By utilizing $\dot{G}$ as a simplified representation of C_{free} we can solve path planning problems without having to generate an explicit representation, which can be costly to compute. Currently, network based planners are used in industry to solve path planning problems for a variety of applications including; industrial manipulators [3], mobile robot navigation [4], assembly tasks [5] and computer game animation. This is a testament to the flexibility, power and ease of use of probabilistic methods.

PRM planners were an early development in probabilistic methods that made effective use of the roadmap graph structure. PRM planners remain popular even today. They are effective for use in complex problems, whilst remaining relatively simple and robust in their operation. PRM planners operate in two distinct phases; the construction of $\dot{G}$, and querying $\dot{G}$ to solve the path planning problem. The first phase consists of building $\dot{G}$ by incrementally sampling random robot configurations for a clash. Samples found to lie in C_{free} are added to the map as network nodes. Subsequent motions between these nodes are also tested for clashing and are added as edges if they lie entirely in C_{free}. This process continues iteratively until a certain density of nodes/connections is achieved. The roadmap is then stored for querying. The query phase involves linking q_{start} & q_{goal} to the stored roadmap and then utilizing graph searching techniques to return a path between them. If no path is available, further sampling can be done. If the environment remains unchanged, the roadmap can be utilized multiple times to solve different path planning queries. Experiments show that the PRM planner is very effective. With the basis of its operation reinforced by probabilistic completeness, it proved to be both fast and reliable even when in use in environments featuring high dimensionality [2], or robots with many degrees of freedom (DOF) [6].

The probabilistic concepts and network structure used in PRM planners provide an effective framework to solve complex path planning problems. However much research has been devoted to improving the details of how these roadmap style planners operate. Observations of the PRM planners operation noted that the majority of computation time is devoted to clash checking robot configurations. In order to reduce the number of clash checks required to generate a solution, many heuristic sampling techniques were developed to replace basic random sampling strategies. Many of these heuristic sampling methods attempt to generate a portion of samples nearby the boundary of clash objects in the workspace. This was done explicitly using geometric information [7] or by using paired samples of specific clash criteria [8]. Other attempts involve partitioning the workspace in order to categorize the different regions in

terms of their complexity [9], [10]. If the sampler knows the 'difficult' regions, it is able to bias its sampling to these spaces, in order to more effectively utilize computational resources. Another focus of research centres on probabilistic methods' weakness in sampling narrow passages in C_{space}. Random sampling schemes employed by traditional PRM planners greatly reduce the likelihood of sampling multiple configurations inside these narrow passages. An effective method of addressing this is the bridge test, as shown in Figure 2 [11]. The bridge test will sample a pair of configurations. If these both fail, the midpoint is tested. If this midpoint is C_{free}, only then is it inserted into $\mathcal{G}$. Bridge testing proves effective at adding samples directly in narrow passages, not just at the boundaries of clash objects and is easily implemented into the structure of network based planners.

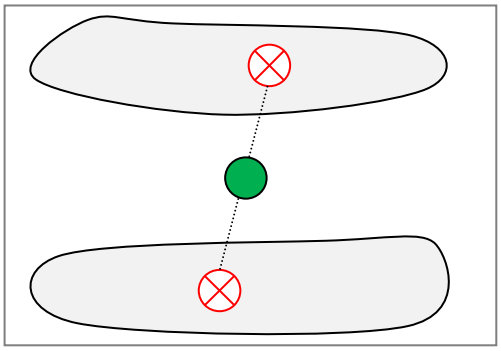

Figure 2. Bridge test criteria.

The 'Lazy-PRM' planner [3] provided a simple and robust method for reducing the number of clash checks to solve a given query. From a high level standpoint, the Lazy-PRM algorithm operates in a similar fashion to its predecessor, the traditional PRM. However a significant difference in operation comes from applying a delayed, or 'lazy', evaluation of clash. The algorithm initially constructs a kinematically sound roadmap, which is assumed to be entirely clash free. Then, repeatedly, the shortest available path through the roadmap is sent to the clash checker for evaluation. If a clash is detected on the path, the offending element (node/edge) is removed from $\mathcal{G}$. If sufficient elements are removed, causing the start and goal configurations to become disconnected, an expansion phase is initiated. The expansion phase adds more nodes in order to reconnect the roadmap. The search then continues iteratively until a continuous clash free path is found, or it is deemed that no solution is available. The Lazy-PRM planner effectively reduces the amount of samples required to solve path planning queries. By delaying the clash checks, redundant sampling of regions that could never provide a solution are removed, saving on computational expenses.

The PRM algorithm spends a large portion of its calculation time pre-processing the roadmap, making it more suited for applications in which a road map can be queried multiple times. Tree based algorithms were developed for applications that needed a solution generated rapidly for single use. [12] developed a notable variant of this style of planner, which grows its $\mathcal{G}$ outwards from an initial root node, one vertex at a time, towards the goal configuration. The growth of the tree can be controlled in a variety of different ways and many approaches have been investigated. Multiple tree approaches have also been made in which two trees are grown simultaneously, with the goal of connecting them together to solve the given query [13].

Recent research focuses on developing planners for specific or complex applications, such as for use on robots with non-holonomic constraints [14], hyper redundant manipulators [6], robots operating in dynamic environments [4], or to generate paths that are more optimal [15]. It is important to note that recent research focuses predominantly on the modification of some component of the probabilistic methodology, rather than on entirely new planning methods; a testament to the reliability and power of probabilistic methods in general.

3. Lazy Significant Edge Algorithm (LSEA)

This section is devoted to describing a new method of lazy path planning. The method offers improvement over the traditional Lazy-PRM planner by reducing the amount of clash checks required to arrive at a solution and is also more effective at navigating difficult regions of C_{space}.

3.1 Algorithm Overview

LSEA, like many other path planners, is an iterative based planner. It constructs an initial roadmap which is then analysed and augmented in a repetitive manner until a termination criteria is met. The algorithm is terminated if a solution to the given query is found, or if no solution is found within a given timeframe. The pseudo code of LSEA is shown in Figure 3. The planner begins by generating a roadmap of kinematically viable nodes and edges, which are initially assumed to be clash free. The algorithm searches for the shortest path through $\mathcal{G}$, which is tested for collision. If a clash is found on this path the subsequent edge/node is removed from $\mathcal{G}$ and the next shortest path is searched for. If no path is returned, the roadmap graph has been reduced to a discontinuous state and a roadmap expansion phase is initiated. The expansion phase re-joins the discontinuous components of the roadmap by intelligently adding nodes to areas of interest. A portion of nodes in this step are generated through uniform random sampling as well, so as to maintain the probabilistic completeness of the algorithm.

1: generate $\mathcal{G}_{init}$
2: **Main Loop**
3: return shortest path, P_s, through $\mathcal{G}_{init}$
4: **If** P_s exists: → clash check P_s
5: **If** ($P_s \in C_{free}$): → solution found. exit **Main loop**.
6: **else** clash detected in P_s → remove element.
7: **else** P_s does not exist: comp (q_{start}) =/= comp (q_{goal})
→ Expansion Phase (see detail)
8: **End Loop**

Expansion Phase
1: Collect all edges, E_{forbid}, in $\mathcal{G}$ ($E_{forbid} \in C_{forbid}$)
2: **for each** $\in E_{forbid}$:
3: add $E_{forbid}(i)$ back into $\mathcal{G}$.
4: **if** comp (q_{start}) = comp (q_{goal})
→ add E_i to significant edge group, SE.
5: **end for**
6: **for each** $\in$ SE
7: distribute N_{samp} nodes about midpoint of SE(i)
8: **end for**
9: lazy bridge test: generate nodes.
10: connect all expansion nodes to $\mathcal{G}$ (component-*n* strategy)
11: **return** main loop

Figure 3. Pseudo code of LSEA operation

The expansion phase is the most important stage of this algorithm. It uses the component connectivity information of the existing roadmap to determine which regions of the workspace have not been successfully navigated by the robot. Roadmap edges which pass through un-navigated clash objects, named significant edges, are used to bias the sampling strategy. The overall effect of the expansion phase is a scheme of sampling that adds more samples to regions of C_{space} that have not been successfully navigated, effectively reducing the amount of redundant sampling carried out. In addition to the roadmap expansion, a type of 'lazy' bridge test (LBT) is implemented. LBT is only activated in certain instances and uses prior clash information to reduce the computational expense of traditional bridge testing techniques. Details of the algorithms' components and their methods of operation are detailed in the following chapter sections.

3.2 Construction of Initial Roadmap

To build the initial roadmap ($\mathcal{G}_{init}$), N_{init} configurations are generated using uniform random sampling about the robots C_{space}. These nodes are then connected to the nearest N_{neighb} nodes using the *component-n* [16] connection strategy. If an edge is not kinematically feasible (e.g., out of joint or orientation limits) it is not included in $\mathcal{G}_{init}$. At this stage, all nodes and edges in $\mathcal{G}_{init}$ are assumed to be clash free. As the algorithm progresses, the values of N_{init} and N_{neighb} will have a significant effect on both the optimality of the returned path and the time taken to find a solution and so should be selected carefully for the given path planning problem.

3.3 Selection of shortest path and clash test

There exist many different methods of selecting the shortest path through a connected network graph, each with varying levels of complexity and performance. LSEA utilizes the A^* search algorithm [17] to return the shortest path through $\mathcal{G}$ between q_{start} & q_{goal}.

When the shortest path through the roadmap is found, each element in the path is checked for clashes. All nodes along the path are tested first, followed by the edges. If any node/edge is found to be clashing, the process is terminated immediately and the offending element is removed from $\mathcal{G}$. If it is a node that is removed, all edges that are connected to the node are removed as well. If the entire path is clash free, we have found a solution to the given query and the path planning process is successfully completed. To save processing time, all results of clash tests are stored in the roadmap structure, so that these elements do not need to be tested again over future iterations.

3.4 Iterative Expansion Phase

In a complex path planning problem, $\mathcal{G}_{init}$ will not be sufficient to solve the given search query. When the query fails, we are left with a disconnected roadmap consisting of at least two separate components. Upon reaching this condition, we initiate the expansion phase of LSEA. The roadmap is augmented by analysing its current state and placing new samples in areas that have not yet been navigated by the robot. These areas are found by utilizing roadmap component connectivity information to determine a collection of significant edges about the roadmap. The expansion phase then actively samples about these significant edges. Following this, if certain conditions are met, lazy bridge testing is carried out in an attempt to place clash free nodes in narrow passages of the workspace.

3.4.1 Significant edge search.

To determine a list of significant edges, we initially collect a group of candidate edges; edges that have been removed from $\mathcal{G}$ (note that edges that were removed with nodes are not considered as candidates). For each of these edges, we place them back into $\mathcal{G}$ and check to see if the start and goal queries have been rejoined into the same component. If a candidate edge satisfies this condition we have successfully defined a significant edge. Significant edges always occur about a clash object that has not been navigated by the robot yet. If an edge is obstructed by a clash object, but does not reduce the component count (refer detail *iv* in figure 4), the clash object has effectively already been navigated by the robot. Once a collection of significant edges has been determined, we randomly select a portion of them to be used as the basis for sampling during the expansion phase.

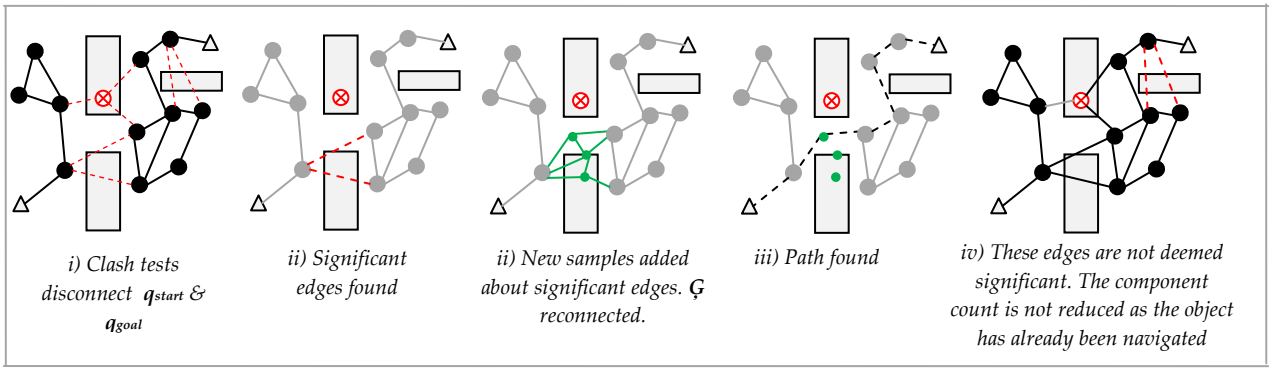

Figure 4. Graphic overview of the LSE algorithm operation

N_{samp} samples are placed about each significant edge via the use of a bivariate distribution [3], centred at the mid-point of the edge. The shape/size of the distribution is controlled by the length and direction of the edge. The newly distributed nodes are then connected to Ģ in the same manner as during the construction phase. After all significant edges have been sampled, N_{rand} nodes are also distributed randomly about the roadmap, in order to ensure probabilistic completeness and help the expansion of the roadmap over future iterations. In some rare instances no significant edges are present in Ģ, which indicates that whilst the current query has failed, the general exploration of C_{free} is progressing. In these instances, random sampling carried out as normal to re-connect the roadmap and the algorithm progresses as normal. Once the expansion phase is completed, the roadmap's connectivity is returned to a sufficient state to continue the planners operation. The expanded roadmap is passed to the shortest path search and the algorithm continues its iterations. To reduce cyclic behaviour in roadmap expansion, significant edges are not used multiple times. If a significant edge has been defined once, it is not utilized by expansion phases over future iterations.

3.5 Lazy Bridge Tests (LBT)

Bridge testing is a proven method of effectively navigating complex regions and narrow passages [11]. The bridge test operates by testing pairs of samples, within a given distance of one another, continually until both samples are found to be clashing. Once this criteria is met, the midpoint of these samples is tested and only if this configuration is C_{free} is it then placed in Ģ. Bridge testing ensures that samples are placed in narrow regions of the configuration space, however the method requires that three successive clash tests of a specific criteria must be met in order to place only one single node in Ģ.

In order to increase the efficiency of the traditional bridge test, a lazy method of bridge testing is implemented in LSEA. It operates by first collecting all nodes in Ģ that failed their clash test, or lie in C_{forbid}. For each pair of nodes in this set (within distance D_{bridge} of each other) a sample is generated at the midpoint. If the mid-point lies in C_{free}, we have satisfied the bridge criteria and this sample is added to Ģ. By utilizing previously determined C_{forbid} samples, the number of clash checks required to produce a bridge sample in Ģ is reduced. The Lazy Bridge test requires a sample of previously clash checked nodes, so it is implemented after each iteration of the expansion phase. To avoid cycles, once a bridge between two C_{forbid} samples has been found, it is never used again. In certain instances, LBT is not needed. At the end of each iteration of the main algorithm, if a certain ratio of failed samples to good samples produced is not met, LBT is not carried out.

4. Experiment and Results

4.1 Experimental Setup

Three path planning problems, each featuring varying levels of complexity, were created to test LSEA. To evaluate performance, LSEA was benchmarked against the Lazy-PRM path planner [3]. Both algorithms were run 100 times in each environment and the results were averaged. Both algorithms utilize the same set of variables that control the operation of the planner. For each environment, these variables were optimized (see Table 1.) and implemented before the testing process. Both planners made use of the *component-n* [16] neighbour connection strategy, which provided better results for both planners in each of the given environments. The effectiveness of lazy bridge testing techniques was carried out separately. In the *medium-scatter* environment, LBT was compared to traditional bridge testing as a means to effectively sample inside narrow passages. All algorithms and environments were developed and tested in the MATLAB programming language.

Each of the three environments is a 2D plane scattered with randomly distributed objects to navigate. The robot is a planar robot with three degrees of freedom (*x, y* and *rotation Rz*). The *sparse-scatter* environment (Figure 5. detail i) features oddly shaped objects, sparsely distributed about the configuration space. The robot is

relatively free to move about this environment and the robots orientation is not critical in order to pass between many of the objects. The second environment, *medium-scatter*, has more objects dispersed throughout. Their positions create several narrow passages the robot has to navigate in order to solve the given path planning query. The final environment has many smaller clash objects *densely-scattered* at random. The density of the clash objects drastically restricts the robot's motion and the orientation of the robot is critical in order to navigate through many sections of the workspace.

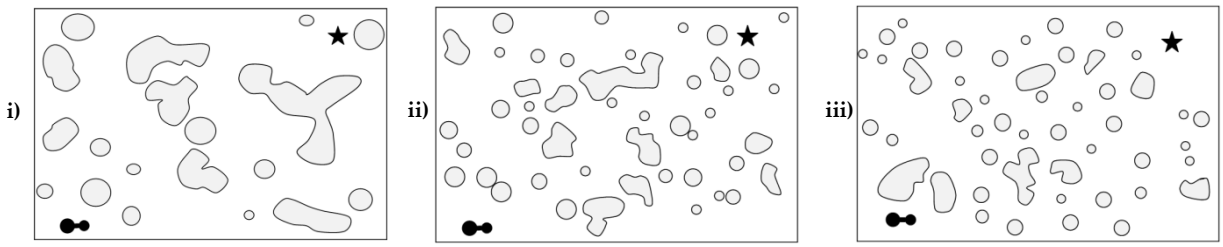

Figure 5. The testing environments, the robot is shown in the bottom left corner, and the goal is the black star in the top right. Detail: i) Sparse-Scatter ii) Medium-Scatter iii) Dense-Scatter

4.2 Results

4.2.1 Results of LSEA

The box plots in Figure 6 display the times taken per query for both planners. The whiskers encapsulate the entire spread of the data, the shaded box represents the lower quartile and the lighter shaded box above shows the upper quartile of the data. The boundary between these boxes represent the median time taken to complete the tests. Using the box plots, we can see that LSEA outperforms the traditional lazy planner in every environment. The improvement was smaller in the sparse scatter environment, but became more apparent in the medium and densely scattered environments.

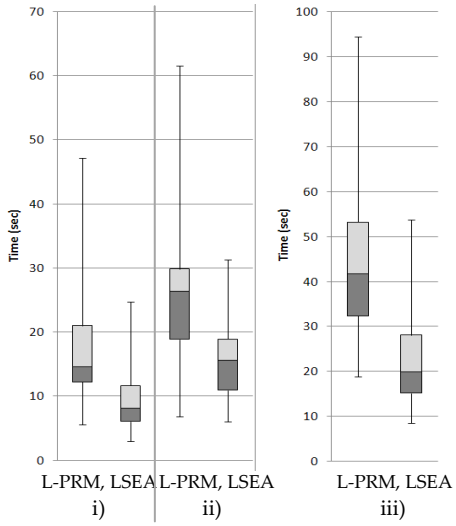

Figure 6. Box plots of results. i) Sparse-scatter ii) Medium-scatter and iii) Dense-scatter.

In the sparsely-scattered environment LSEA provided some degree of improvement. It was observed that the sparseness of the obstacles meant that both planners were able to solve the majority of the problems with the initial roadmap alone, few expansion iterations were required. Both algorithms use the same methods for generating $\mathcal{G}_{init}$, so their results are fairly similar. Regardless, the average time to find a solution was reduced by 40% and the spread of the results was also reduced somewhat. The advantages of LSEA became more apparent in the remaining environments. In the *medium-scatter* test, LSEA reduced the average time to reach a solution by 52% and the spread of data was reduced significantly. This result can be attributed to LSEA distributing its sampling workload towards areas that had not effectively been navigated by the robot. The traditional Lazy algorithm was observed redundantly sampling regions already navigated, slowing the average time to reach a solution. This same phenomenon was observed to a greater degree in the *densely-scattered* environment. The average time taken to solve the problem was reduced by 57%. LSEA was more intelligently able to discern which regions had been cleared by the robot and then push the sampling strategy away from these areas. Results from the testing are presented in Figure 7. The reduction in time for LSEA can be directly attributed to the reduced number of clash checks done in order to solve each path planning query. The roadmaps generated to solve the problems had fewer nodes and the percentage of roadmap nodes that were clash checked (shown in the brackets) was also lower, providing a more 'lazy' solution.

4.2.2 Results of LBT

To test the effectiveness of Lazy Bridge testing, another testing scheme was created using the *medium-scatter* environment. This environment was used as it features scattered obstacles, as well as narrow passages which need to be navigated, providing a good environment to test the overall performance of LBT. To gain both an

absolute and relative understanding of LBT effectiveness, two tests were created. In the first, the *medium-scatter* environment was used to test LSEA with and without LBT. LBT was found to improve the time taken to solve the query by 33.1%. When the planner is using LBT, fewer samples are required, because they are more effectively used during the process. The second test compared the effectiveness of LBT against traditional bridge testing techniques. Once again, the lazy method showed improved performance over the traditional method. A 17.8% reduction in time was observed and on average, 2934 fewer clash checks were required to reach a solution. These results can be attributed to the fact that LBT utilizes previously calculated clash results, reducing the overall amount of clash tests required to generate valid bridge samples. Granted, LBT has to substitute many more distance checks between nodes. However, the relative cost of these distance checks is small compared to the cost of the extra clash checks required when carrying out the traditional bridge testing. If LBT was employed on a more complex system, which requires a longer time to check the clash status of samples, a greater reduction in time would be observed.

100 tests. Sparse.	Lazy-PRM	LSEA
AveTime (s)	15.2	9.1
Ave Clash Checks	66898	44525
Nodes Used (% CC)	308 (78%)	186 (75%)

i)

100 tests. Medium.	Lazy-PRM	LSEA
AveTime (s)	28.8	13.9
Ave Clash Checks	120712	65348
Nodes Used (% CC)	385 (84%)	241 (80%)

ii)

100 tests. Dense	Lazy-PRM	LSEA
AveTime (s)	43.1	18.3
Ave Clash Checks	177903	83522
Nodes Used (% CC)	444 (90%)	249 (85%)

iii)

Figure 7. Tabulated results for each environment: i) Sparse-scatter ii) Medium-Scatter iii) Dense-Scatter

5. Discussion

LSEA was developed as an improvement over the traditional Lazy-PRM planner [3]. From a high level standpoint, both algorithms operate in a similar fashion. The main operational difference between LSEA and Lazy-PRM comes from the roadmap expansion phase. Lazy-PRM carries out its roadmap expansion by adding additional samples to regions deemed likely to be at the boundary of clash objects. LSEA takes this expansion method a step further by adding its samples to regions nearby clash objects that have not yet been navigated by the robot. By doing this a solution can be discovered faster, as no time is wasted by continuing to sample about a clash object that has already been cleared.

The trade off to this approach is that whilst a solution can be found faster, there is the possibility that the optimality of the returned path may not be as good. If, for instance, the LSEA planner navigates a certain obstacle poorly, the likelihood of adding more samples to this region and improving this local path over future iterations is low. Whereas, if the Lazy-PRM navigates a certain object poorly, there is a greater chance that future iterations of the algorithm will continue to place samples about this object, which could improve the local path. This effect can be reduced considerably via post process optimization of the path solution.

The observed performance advantage LSEA has over Lazy-PRM is directly attributed to the reduced amount of clash checks required to solve a path planning query. In the 2D test environments used, these clash checks are relatively simple and can be carried out very rapidly (~0.0001 sec). If LSEA planner was implemented in a more complex system that requires a much longer time to process a clash check, we can expect the performance advantage of LSEA to be even higher.

Variable	Value
N_{Init}	70,150,150
N_{Neighb}	5,5,5
# Seeds/SE's per iteration	10,10,10
N_{samp}	2,2,2
N_{rand}	10,10,10

Table 1. Variables used for each of the three environments (sparse, medium, dense) for both LSEA and Lazy-PRM during testing.

6. Conclusions and Future Work

In this paper, a new path planning algorithm that utilizes connectivity information to bias the expansion of the roadmap was presented. The algorithm was tested in three environments featuring different levels of complexity. Its performance was benchmarked against a traditional Lazy-PRM planner. In each test LSEA outperformed its counterpart considerably, despite having a similar method of operation. It was able to solve path planning queries faster, utilizing fewer samples to achieve its solution. In more difficult environments, it was able to process a valid solution twice as fast as its traditional counterpart. A lazy method of bridge testing was also proposed and tested. The lazy method was observed to sample narrow passages more effectively, utilizing fewer clash checks in order to generate valid bridge samples.

Future work regarding LSEA involves adapting its operation to plan paths for 6DOF articulated manipulators operating in a 3D environment. In such a system, clash checking algorithms place an even larger burden on computational resources. It is envisaged that

when LSEA is implemented on such a system, the reduction in required clash checks will further increase the performance advantage LSEA has over traditional Lazy-PRM methods. The LSEA algorithm is still in early stages of development. Many performance modifications are being investigated and further tuning of the algorithm for increased performance is to be carried out

7. Acknowledgments

This work was supported by the Defence Materials Technology Centre (DMTC), which was established and is supported by the Australian Government's Defence Future Capability Technology Centre (DFCTC) initiative.

8. References

[1] Tsianos K, Sucan I, Kavraki L (2007) Sampling-Based Robot Motion Planning: Towards Realistic Applications. Computer Science Rev. 1: pp 2-11.

[2] Kavraki L, Svestka P, Latombe JC, Overmars M (1996) Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. IEEE Trans. On Robotics and Automation. 12: pp 566-580.

[3] Bohlin R, Kavraki L (2000) Skin Path Planning Using Lazy PRM. Proc. IEEE Int. Conf. on Robotics & Automation. California, USA.

[4] Jaillet L, Simeon T (2004) A PRM-based Motion Planner for Dynamically Changing Environments. IEEE Int. Conf. on Intell. Robots & Systems. 2: pp 1606-1611, Sendai, Japan.

[5] Thomas U, Iser R (2010) A new Probabilistic Path Planning Algorithm for (Dis)assembly Tasks. ISR 41st International Symp. On Robotics. Munich, Germany.

[6] Lanteigne E, Jnifene A (2011) Small Tree Probabilistic Roadmap Planner for Hyper-Redundant Manipulators. Lec Notes in Comp. Science. Vol 6752. pp 11-20.

[7] S.A. Wilmarth, N.M. Amato, P.F. Stiller (1999) MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. IEEE 1999 Int. Conf. on Robotics and Automation, pp 1024-1031.

[8] Boor V, Overmars M, Stappen F.V.D (1999) The Gaussian Sampling Strategy for Probabilistic Roadmap Planners. Proc. of the 1999 IEEE Int. Conf on Robotics & Automation. USA.

[9] Rantanen M (2011) A Connectivity-Based Method for Enhancing Sampling in Probabilistic Roadmap Planners. J. Intell Robot Syst. 64: pp 161-178.

[10] Klasing K, Wollherr D, Buss M (2007) Cell-based Probabilistic Roadmaps (CPRM) for Efficient Path Planning in Large Environments. ICAR. 13th Int. Conf. on Adv. Robotics. pp 1075-1080.

[11] Sun Z, Hsu D, Jiang T, Kurniawati H, Reif J (2005) Narrow Passage Sampling for Probabilistic Roadmap Planning. IEEE Trans. On Robotics. 21: pp 1105-1115.

[12] Kuffner J, LaValle S (2000) RRT-Connect: An Efficient Approach to Single-Query Path Planning. Proc. IEEE Int. Conf. on Robotics & Automation. pp 995-1001.

[13] Sanchez G, Latombe JC (2003) A single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. Springer Tracts in Adv. Robotics. 6: pp 403-417.

[14] Cheng P, Shen Z, LaValle S (2000). Using randomization to find and optimize feasible trajectory for nonlinear systems. Allerton Conf. on Communications, Control, Computing, pp 926-935

[15] Nieuwenhuisen D, Overmars M, (2004) Useful Cycles In Probabilistic Roadmap Graphs. Proc. IEEE Int. Conf. Robotics & Automation. 1: pp 446-452

[16] Geraerts R, Overmars M (2005) Sampling and node adding in probabilistic roadmap planners. Robotics and Autonomous Systems. 54: pp 165-173.

[17] Latombe JC (1991) Robot Motion Planning. Kluwer, Boston, USA.