A NEW OFFLINE PATH SEARCH ALGORITHM FOR COMPUTER GAMES THAT CONSIDERS DAMAGE AS A FEASIBILITY CRITERION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SERHAT BAYILI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2008

Approval of the thesis:

A NEW OFFLINE PATH SEARCH ALGORITHM FOR COMPUTER GAMES THAT CONSIDERS DAMAGE AS A FEASIBILITY CRITERION

submitted by **SERHAT BAYILI** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Volkan Atalay
Head of Department, **Computer Engineering** _____

Prof. Dr. Faruk Polat
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. İ. Hakkı Toroslu
Computer Engineering Department, METU _____

Prof. Dr. Faruk Polat
Computer Engineering Department, METU _____

Prof. Dr. Göktürk Üçoluk
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Department, METU _____

Dr. Çağatay Ündeğer
Computer Engineering Department, Bilkent University _____

Date: 05.08.2008

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Serhat Bayılı

Signature :

ABSTRACT

A NEW OFFLINE PATH SEARCH ALGORITHM FOR COMPUTER GAMES THAT CONSIDERS DAMAGE AS A FEASIBILITY CRITERION

Bayılı, Serhat

M.S., Department of Computer Engineering

Supervisor: Prof. Dr. Faruk Polat

August 2008, 65 pages

Pathfinding algorithms used in today's computer games consider path length or a similar criterion as the only measure of optimality. However, these games usually involve opposing parties, whose agents can inflict damage on those of the others'. Therefore, the shortest path in such games may not always be the safest one. Consequently, a new suboptimal offline path search algorithm that takes the threat sources into consideration was developed, based on the A* algorithm. Given an upper bound value as the tolerable amount of damage for an agent, this algorithm searches for the shortest path from a starting location to a destination that would cause the agent suffer no more damage than the specified maximum. Due to its mentioned behavior, the algorithm is called Limited-Damage A* (LDA*). Performance of LDA* was tested in randomly-generated and hand-crafted fully-observable maze-like square environments with 8-way grid-abstractions against Multiobjective A* (MOA*), which is a complete and

optimal algorithm. It was found to perform much faster than MOA* with allowable sub-optimality in path length.

Keywords: Path search, pathfinding, path feasibility criterion, threat

ÖZ

TEHDİTLERİ DİKKATE ALAN YENİ BİR ÇEVRİM DIŞI YOL BULMA ALGORİTMASI

Bayılı, Serhat
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Faruk Polat

Ağustos 2008, 65 sayfa

Günümüz bilgisayar oyunlarında kullanılan yol bulma algoritmaları, bir çözümün optimal olma ölçütü olarak yalnızca yol uzunluğu ya da benzer bir değeri kullanırlar. Ancak bu tür oyunlarda genellikle birbirlerine düşman olan ve birimleri diğer tarafların birimlerine hasar verebilen taraflar yer alır. Dolayısıyla, bu oyunlarda iki nokta arasındaki en kısa yol, en güvenli yol olmayabilir. Bu nedenle, haritadaki tehdit unsurlarını da dikkate alan, A* tabanlı, optimal olmayan yeni bir çevrim dışı yol bulma algoritması geliştirilmiştir. Bu algoritma, müsamaha gösterilen hasar miktarının belirtilmesi sonrasında, haritada verilen iki nokta arasında, birimi bu miktarı geçmeyen derecede hasara maruz bırakacak bir yol bulmaya çalışır. Bu özelliği nedeniyle bu algoritma Sınırlı-Hasar A* (LDA*) algoritması olarak adlandırılmıştır. LDA*'ın performansı, rastlansal olarak üretilmiş olan tam gözlenebilir labirent benzeri ızgara soyutlamalı kare haritalarda, optimal ve tam bir algoritma olan Çok

Amaçlı A* (MOA*) algoritmasınınkiyle karşılaştırılmıştır. Geliştirilen algoritmanın, MOA*'dan çok daha hızlı olduğu ve izin verilebilir derecede optimal altı sonuçlar verdiği belirlenmiştir.

Anahtar Kelimeler: Yol arama, yol bulma, yol fizibilite kriteri, tehdit

To those people, who dare to follow their passions

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor Dr. Faruk Polat, who has always supported and guided me throughout this study.

I would also like to thank Dr. Erhan Karaesmen and my Ph.D. supervisor Dr. Ahmet Yakut in the Department of Civil Engineering, who helped me make this study possible.

I would like to thank TÜBİTAK for their financial support.

Finally, I would like to thank my friends, who have helped me throughout this study, and my family.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

CHAPTER 1

INTRODUCTION

Searching is among the basic, yet quite frequently performed actions in many problems. Simply defined, search is the action of testing a set of conditions on the elements of a domain. Despite its fairly simple definition, searching is not always a simple task. Its application to a domain is heavily dependent on the properties of that domain, and a search algorithm can be very complex, when the domain involves non-discrete properties or multiple objectives.

The path planning (or path search) problem is one of such cases. Its domain is a 2-dimensional (2D) or 3-dimensional (3D) map, where an agent is required to find a sequenced set of state transitions (solution path) for itself from its current location to a destination, or determine the nonexistence of such a sequence. A solution path, however, is required to satisfy certain conditions, such as having the minimal traveling cost (path length, fuel spent, travel time, etc.).

In general, it is not feasible to solve this problem by utilizing the domain properties naïvely, due to time or space constraints. Therefore, the search domain is generally abstracted, and simplified by discretization of the map into a uniform or non-uniform grid. More simplification can be achieved by considering the environmental characteristics such as observability and dynamism.

Depending on the time constraints, the path search algorithms can be classified into two groups as *offline* and *real-time*. Offline search algorithms are those that are not bounded by a time period, which generate total (and generally optimal) solutions for the specified start and goal locations. In these types of problems, the path is realized after a solution is found. Dijkstra's algorithm [1], A* algorithm [2] and all uninformed graph search algorithms [3] are among the examples of offline path search algorithms.

Real-time algorithms, on the other hand, are applied to the problems, where the agent is required to start navigation before having a total solution at hand, and generate one during navigation. Therefore, these algorithms are required to run along with the path execution, and distribute their run times discretely over a period of time, each session occupying only a fraction of a second. Due to their nature, they are not expected to generate a solution either within one session, or with optimal properties. On the other hand, real-time algorithms can handle dynamic and partially-known environments, as opposed to offline algorithms, for which all environmental properties must be static and fully known prior to path search.

According to how they generate solutions, the real-time path search algorithms can be classified into two groups as incremental and non-incremental. Incremental algorithms are aimed to generate an initial sub-optimal path within an initial small time period, start execution of that path, and repair it during subsequent sessions to make it closer to an optimal solution. The family of D* algorithms, such as D* [4], D*-lite [5], Focussed D* [6], TA* [7] and [8] are among those utilizing incremental approach.

Non-incremental algorithms do not generate an initial solution and try to optimize it, but just try to find a solution considering only the short-term choices and their heuristic values. In other words, they try to generate only the portion of

a solution path closest to the current location of the agent. Learning Real-Time A* (LRTA*) and Real-Time A* (RTA*) [9], Real-Time Edge Follow [10] and Real-Time Target Evaluation Search [11] are among the commonly used algorithms of this kind. Since such algorithms do not explore the search space deeply, they run much faster than the other types of algorithms. However, the solutions they generate can be very sub-optimal.

1.1 MOTIVATION AND SCOPE

All of the path search algorithms mentioned above are aimed to generate solutions that minimize the path length. In most cases, this is an adequate criterion. However, there are cases, where other optimization parameters are present along with the path length criterion. One of such parameters is the amount of damage that the agent may suffer, in the presence of threat sources that can apply damage to the agent.

One of the domains that require such optimization is computer games that involve opposing parties. Currently, only path length minimization is in consideration for the navigation task of mobile agents in such games. Yet, execution of a path without being able to reach its goal location is not a logical approach, and this fact should be considered by the artificial intelligence of a game. Therefore, the damage considerations should be integrated into the path planning algorithm of a computer game.

In such a case, the single objective optimization problem would turn into a multiobjective optimization problem over distance and damage. There are some algorithms developed for solving such problems. One of them is the Multiobjective A* algorithm (MOA*) [12]. It is a complete and optimal algorithm, when coupled with an admissible heuristic. However, it requires large amounts of both space and time, and is infeasible when space or time is limited.

The current literature involves other multiobjective path planning algorithms that are focussed on the safe navigation of agents within territories involving detectors [13, 14]. Such algorithms consider the probability of detection as the second optimization measure, which prohibits their usage in computer games, which use much simpler and direct methods for enemy detection, attack and damage calculation.

Observing the non-existing properties in the existing research, the need for an algorithm that could be used mainly in computer games was determined. In addition, it was observed that the safety requirements of the agents in such environments would depend on the task they were assigned to. Taking the time limitations in such games into consideration, an A*-based fast algorithm that finds a suboptimal path between a given starting location and a goal location within a predefined upper amount of damage was devised, and called Limited-Damage A* (LDA*). Its performance was tested against MOA* for randomly generated and hand-crafted maps of three different sizes, and the results are reported.

The following chapters in this thesis are organized as follows: Chapter 2 states the background and related work to this study. The proposed algorithm and the environmental properties on which it was tested are defined in Chapter 3. Test cases and their results are given in Chapter 4. Finally, the discussion of results, and the conclusions are provided in Chapter 5.

CHAPTER 2

BACKGROUND AND RELATED WORK

Searching basically constitutes of testing the elements of a candidate set of elements for compliance with a set of conditions, and forming a result set based on the attributes of the elements complying with the given set of conditions. Path search is a special type of search problems, where the aim is to find a set of consecutive (vertex, edge) pairs within a given graph structure that connects a start vertex to a goal vertex through such pairs. Here, the set of consecutive edges connecting two vertices through (vertex, edge) pairs is called a path, and the total cost of the edges in this path is called the cost of that path.

Determining the shortest path(s) between two nodes of a graph is one of the commonly encountered path search problems. Aim of such problems is to find a sequence of (vertex, edge) pairs, which connect the starting vertex to the goal vertex with the smallest path cost.

There are many algorithms proposed for solving such problems, each having different algorithmic complexities. Depending on their runtime characteristics, they are grouped into two, as *offline* and *real-time* path search algorithms. The offline search algorithms are those that are not bounded by a time period, and generate total (and usually optimal) solutions. Dijkstra's algorithm [1], A* algorithm [2] and all uninformed graph search algorithms such as Depth-First

Search [3], Breadth-First Search [3], and Iterative-Deepening Search [3], etc. are among the examples of offline search algorithms.

On the other hand, the real-time algorithms have limited runtime periods for each session (usually in the order of a few milliseconds), and execute for a number of discrete sessions. These algorithms can be grouped into two as incremental and non-incremental, depending on how they generate solutions.

Algorithms in the incremental group are those that aim to generate an initial sub-optimal path in the very first session(s), and repair it during the following sessions to make it closer to the optimal. The family of D* algorithms, such as D* [4], D*-lite [5], Focussed D* [6], TA* [7] and [8] are among the well-known real-time path search algorithms.

Non-incremental algorithms are those, which do not aim to generate an initial solution and later optimize it, but try to find a solution by considering only the short-time available choices. In other words, they try to generate only the portion of a solution path that is closest to the agent's location. Learning Real-Time A* (LRTA*) [9] and Real-Time A* (RTA*) [9], Real-Time Edge Follow [10] and Real-Time Target Evaluation Search [11] are among the commonly used algorithms of this kind. Since such algorithms do not explore the search space deeply, they run much faster than the other types of algorithms. However, the solutions they generate can be very sub-optimal.

2.2 A* SEARCH ALGORITHM

The A* algorithm [2] is one of the well-known search algorithm to make use of heuristic values. It can process directed or undirected graphs with non-negative edge costs. A* algorithm basically chooses to explore the seemingly best alternative at each step, instead of choosing it blindly. It does this by keeping a

list of vertices that are neighbor to the explored ones, with their path cost estimates (i.e. f(x) = g(x) + h(x)).

The algorithm keeps a list of unprocessed (open) vertices with their path cost estimates, f (x). In the beginning, this list contains the starting vertex only. At each step, it picks the seemingly best vertex from the list, checks whether it is the goal vertex. If it is not, then the algorithm calculates the path cost estimates of the neighbors of that vertex and puts them into the open list of vertices, with a reference to the currently explored vertex as the parent of the neighbor. This loop is run until the goal node is reached, or the list of unprocessed nodes is empty (i.e. no solution). If the goal node is reached, then the path from the starting vertex to the goal is generated by backtracking the parent pointers from the goal node.

If an admissible heuristic is utilized in the algorithm, it is proven to be complete and optimal, in the existence of a solution exists. Running time complexity of the algorithm depends on the chosen heuristic. In the worst case, it is exponential to number of nodes expanded in the shortest path. However, it runs in polynomial time, if the heuristic function satisfies the following condition:

$$\forall x \ \left| h(x) - h^*(x) \right| \leq O(\log h^*(x)) \tag{2.1}$$

where h^* is the exact cost from node x to the goal, termed as the optimal heuristic. In other words, in order to obtain polynomial running time complexity, the error of the utilized heuristic function should not grow faster than the logarithm of the optimal heuristic function.

Space complexity of the algorithm can be a problem in the worst case, since the number of nodes in the open list of nodes would grow exponentially. Some algorithms like Iterative-Deepening A* [15], Memory-Bounded A* [16] and

Recursive Best-First Search [17] are developed in order to cope with this problem.

A* is an offline graph search algorithm, and it must be run in a single session on a graph, whose properties are static and fully known. Considering this property, it is not possible to use this algorithm on very large graphs, if the time period reserved for the search algorithm is limited. Neither can it be utilized effectively in the cases, where the search environment is dynamic and exhibits changes with time. Another shortcoming of the algorithm is in the need of multiple-objective optimization, where the measurement in an objective cannot be expressed in terms of others.

2.3 MULTIOBJECTIVE A* SEARCH ALGORITHM

The A* algorithm is a complete and optimal solution for the cases where a single optimization criterion such as the path cost is in consideration. However, most of the real-world problems involve more than one optimization criteria, which are not convertible or expressible in terms of a combination of the others.

As a solution to such problems, the Multiobjective A* (MOA*) Algorithm [12] was developed, based on similar principles as A*, for the cases where more than one optimization criteria are involved. Like A*, it is complete and optimal, when used with an admissible heuristic function.

Unlike A* does, MOA* utilizes vectors of estimated path cost $F(n) = (f_1(n), f_2(n), \ldots, f_n(n))$, traveled path cost $G(n) = (g_1(n), g_2(n), \ldots, g_n(n))$ and heuristic estimation $H(n) = (h_1(n), h_2(n), \ldots, h_n(n))$ values for each solution candidate for a vertex n, as opposed to the scalar values utilized in A*. It keeps a set of OPEN (to be processed) nodes and a set of CLOSED (already processed) nodes. At each iteration of the algorithm, the best alternative is selected among a subset

ND of the OPEN set, which is formed by the elements of the OPEN set that are not dominated by any other element of this set and any of the discovered solutions.

After determining which vertex to explore through which path, it checks if the current vertex is in the set of goal nodes. If so, the current node and its path cost vector are added to the solution set and the iteration continues with selection of a new node. If, on the other hand, the explored vertex is not in the set of goal nodes, then its neighboring nodes are generated. At this step, the newly generated node n' is checked for being generated for the first time. If it is generated for the first time, then its path cost estimate vector F(n'), traversed path cost vector G(n') and the heuristic estimate vector H(n') are computed, and the newly generated neighboring node is added to the set of OPEN nodes. If the node is not explored for the first time, then there is a possibility that a path passes through this node with a non-dominated costs to other candidates. If this is found to be the case, then the node and its non-dominated cost vectors are taken into consideration in the following steps of the solution. The algorithm iterates over the above steps until the ND set is found to be empty. Then, the solution paths are generated by backtracking the edges from the goal nodes. Taken from [12], Figure 2.1 shows the steps of this algorithm. This algorithm, when accompanied with admissible heuristic functions, is optimal and complete.

One of the important properties of this algorithm is the non-dominance property of the elements in the solution set over one another, which is also utilized in the ND set. This property ensures optimality, and for a set Σ containing elements of compatible vectors of same type, non-dominance can be formulated as:

$$\left\{\Sigma_i \mid \Sigma_i^m > \Sigma_j^m, \Sigma_i, \Sigma_j \in \Sigma, i, j = 1, 2, ..., \|\Sigma\|, m = 1, 2, ... \|\Sigma_i\|\right\} = \varnothing \tag{2.2}$$

The current literature involves other multiobjective path planning algorithms that are focussed on the safe navigation of agents within territories involving detectors [13, 14]. Such algorithms consider the probability of detection as the second optimization measure, which prohibits their usage in computer games, which use much simpler and direct methods for enemy detection, attack and damage calculation.

0. Initialize by setting OPEN equal to a set containing only the start node and setting CLOSED, SOLUTION, SOLUTION_COSTS, SOLUTION_GOALS, and LABEL each equal to the empty set.
1. Find the set of nodes in OPEN, call it ND, that have at least one node selection function value that is not dominated by:
   1.1. the cost of any solution path already discovered (i.e., in SOLUTION_COSTS), nor by
   1.2. the node selection function values of any other potential solution represented by a node on OPEN.
2. Terminate or select a node for expansion.
   2.1. If ND is empty, do the following:
      2.1.1. Use the set of preferred solution path costs in SOLUTION_ COSTS and the LABEL sets. If any, to trace through backpointers from the goal nodes in SOLUTION_ GOALS to s.
      2.1.2. Place any solution paths in SOLUTION.
      2.1.3. Stop.
   2.2. Otherwise, do the following:
      2.2.1. Use a domain-specific heuristic to choose a node n from ND for expansion, taking goals, if any, first.
      2.2.2. Remove n from OPEN.
      2.2.3. Place n on CLOSED.
3. Do bookkeeping to maintain accrued costs and node selection function values.
4. Identify solutions.
   4.1. If n is a goal node, do the following:
      4.1.1. Add it to SOLUTION_ GOALS,
      4.1.2. Add its current costs to SOLUTION_COSTS.
      4.1.3. Remove any dominated members of SOLUTION_ COSTS.
      4.1.4. Go to Step (6).
   4.2. Otherwise, continue.

Figure 2.1. Outline of the Multiobjective A* Algorithm.

5. Expand n and examine its successors.
    5.1. Generate the successors of n.
    5.2. If n has no successors, go to Step 6.
    5.3. Otherwise, for all successors n' of n, do the following:
        5.3.1. If n' is a newly generated node, do the following:
            5.3.1.1. Establish a backpointer from n' to n.
                5.3.1.1.1. Set LABEL(n', n) equal to the nondominated subset of the set of accrued costs of paths through n to n' that have been discovered so far.
            5.3.1.2. Establish a nondominated accrued cost set, G(n') = LABEL(n', n),
            5.3.1.3. Compute node selection values, F(n'), using G(n') and the heuristic function values at n', H(n').
            5.3.1.4. Add n' to OPEN.
        5.3.2. Otherwise, n' was previously generated. so do the following:
            5.3.2.1. If any potentially nondominated paths to n' have been discovered. then, for each one, do the following:
                5.3.2.1.1. Ensure that its cost is m LABEL(n', n) and therefore in the current set of nondominated accrued costs of paths discovered so far to n'; that is, in G(n').
                5.3.2.1.2. If a new cost was added to G(n'), do the following:
                    5.3.2.1.2.1. Purge from LABEL(n', n) those costs associated with paths to n' to which the new path is strictly preferred.
                    5.3.2.1.2.2. If n' was on CLOSED, move it to OPEN.
6. Iterate.
    6.1. Increment iteration counter.
    6.2. Go to Step (l).

Figure 2.1 (cont'd). Outline of the Multiobjective A* Algorithm.

CHAPTER 3

PROPOSED ALGORITHM

3.1 MOTIVATION

Simply defined, computer games are the programs that interact with humans and aim to entertain them. In doing so, they are expected to behave intelligently, or human-like. This requirement of computer games poses many problems, which can be partially or fully dealt with Computer Scientific approaches and the current level of hardware technology.

One of such common problems is the navigation problem of a computer-controlled agent from a starting location to a destination on a two-dimensional (2D) or three-dimensional (3D) maze-like environment (or a map). In most cases, the agent is desired to find a path (if one exists) between these two given locations, which can be traveled in the shortest possible time or distance (travel cost).

There are many path search algorithms in the current Computer Science literature, which can handle various problems involving static or dynamic environments that are fully or partially known. These algorithms make use of regular or irregular square or hexagonal grids, or irregular polygonal grids in order to construct an environmental representation, which is then used in the path search.

In most cases, determining and traveling along the path with the minimum length is adequate for an agent to exhibit intelligent behavior, since an intelligent being would normally take the shortest path, or one close to the shortest in length. However, this is not an adequate behavior, when the environment involves other agents that pose sources of threat for the navigating agent for the agent's health.

Strategy games are among the most common cases, where this problem is encountered. These games involve two or more opposing parties, each having a number of agents, and a global aim. The aim of a party may vary from a simple task like safe navigation of an agent between two predefined points, to destroying all property of every other party.

Whatever the global aim of any party might be, their agents are required to navigate in the environment, and this problem has been solved by utilizing the available path search algorithms. However, these algorithms have a common drawback for strategy games: They only consider the travel cost as a measure of quality for the solutions they generate, and no other factors in the environment such as threat sources. Therefore, it is unknown if an agent will complete its navigation task successfully, without being destroyed.

As a result, the navigation problem can be turned into safe navigation problem, which involves finding out the shortest safe path between two locations in an environment. However, this problem of multiobjective optimization is a complex problem, in case both damage and distance are parameters of optimization, and it is in the class of NP problems. Therefore, it cannot be solved in small time intervals that can be allocated for paths search in computer games. Moreover, the safest path may be much longer than the shortest path, yet a low damage path being close to the shortest. Such cases can be common,

depending on the environment, so a logical decision maker is required in order to choose the optimal path among the optimal and sub-optimal solutions.

3.2 LIMITED-DAMAGE A* ALGORITHM

As mentioned above, the problem of determining the shortest and safest path between two points in an environment, which involves threat sources that can apply damage to the traveling agent, is a complex problem. It has exponential running time complexity, and cannot be solved within the time limitations of a computer game, which can be expressed in terms of a few ten milliseconds. Consequently, an algorithm is required that can generate a solution approximate to the exact solution within the time limitation.

Considering a special case of the problem at hand, where damage is not an optimization, but a decision criterion, an A*-based path search algorithm can be a solution. This special case of the general problem defines a maximum limit to the damage amount that an agent is allowed to withstand, and asks for the shortest path within this damage range.

Thus, the proposed A*-based algorithm utilizes a combination of the path distance and damage acquired from traveling along that path as the path cost, and determines the shortest path between a starting location and a destination within a given damage allowance for an agent. Therefore, it is called Limited-Damage A* (LDA*) algorithm.

3.2.1 Environmental Parameters

Environment involves the key parameters for testing the success of such an algorithm. It is easier to observe the performance of such an algorithm in a relatively simple environment. Therefore, a fully-observable, 2D square grid representation is chosen as the environmental representation. It is also easier to

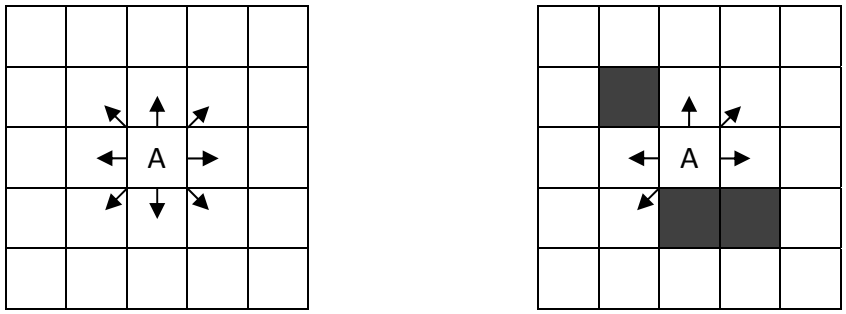
implement, compared to irregular polygon-based environments, and is commonly used among the path search algorithms.

Length and time units in the environment are chosen to be meters and time, respectively. Dimensions of the grid environment is defined by the number of rows and columns it has, and the length of a single cell's edge. Ends of the environment are closed, thus traveling out of the grid is not allowed. A cell in this environment is either empty, occupied by a block, or occupied by a threat source. Without loss of generality, the size of a cell, which is chosen to be 10 meters, is assumed to be greater than the sizes of the traveling agent, which is assumed to be negligible. A block in a cell is assumed to occupy some area at the center of a cell, and prohibit the agent from entering it only.

The agent is assumed to have a certain value of starting health (hit points, HP), which is chosen to be 100, and travel at predefined constant speed, which is chosen as 10 m/s, without acceleration and without any time delays for turning, stopping, etc. It is allowed to travel between adjacent empty cells in primary 8-directions, as seen in Figure 3.1. It is not allowed to enter the cells occupied by blocks or by threat sources. A single travel action between two adjacent cells is assumed to happen without any interruption, along the line segment connecting the centers of the cells, at the speed of the agent.

A threat source is assumed to be stationary and occupy an empty cell. It is chosen to have a predefined and constant radius of effect, and thus a circular area of effect (threat zone). It is assumed to behave like the turret-like structures in strategy games, and have a non-negative initial targeting time delay and a positive time delay between two shots, which are chosen to be 0 seconds and 1 second, respectively. A constant amount of damage is applied to the traveling agent by a single shot, which is chosen to be 1 HP of damage for the test cases. For simplicity, the damage of a shot is applied to the agent instantaneously,

without any travel time for the damaging unit such as a bullet or a rocket fired from a threat source. Thus, the agent is assumed to take damage if it stays in the area of effect of a threat source for more than the initial time delay of that source. It is assumed to experience more damage if it remained in the area of effect for a time period greater or equal to the shooting time delay of that source. A threat source is assumed to be in action as long as the agent is in its threat zone, and have its activity paused when the agent leaves that area. Therefore, an entry to a threat zone after an exit is assumed to resume the activity of that source at the time of re-entry from the point, when the agent had previously left the threat zone.



a. b.

Figure 3.1. Valid movements for the agent: a) 8-direction movement between empty cells, and b) a mixed configuration. “A” denotes the agent’s position.

Although blocking cells and threat sources can be placed anywhere on the map, the latter having arbitrary parameters, the grid is chosen to be more regular for the test cases. As mentioned above, the grid environment is chosen to be square, having equal number of rows and columns. In addition, it is chosen to consist of groups of cells, where a cell group is defined as a square group of adjacent cells with predefined odd number of rows and equal number of columns. An example is shown in Figure 3.2 for a grid environment consisting of 2x2 cell groups, each

of which are 5 cells long (thus a 5x5 grid). A cell group is assumed to be empty with a probability of 0.5, have a threat source at its center cell with a probability of 0.3, and involve blocking cells with a probability of 0.2.

If the cell group has a threat source, then its radius of effect is chosen to be equal to the center-to-center distance between the center cell of that group and the edge cell of that group plus 1 meter, as seen in Figure 3.3. The additional 1 m of distance is utilized so that an agent passing through the edge of a cell group would just enter the area of effect of a threat source, thus it would not be possible for it to pass through adjacent cell groups with threat sources, without taking any damage.

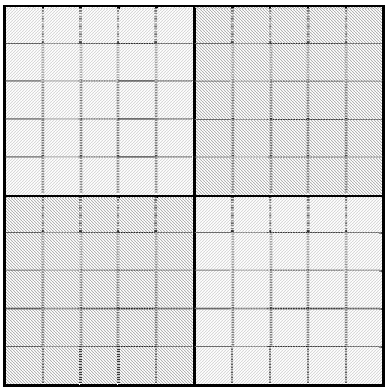


Figure 3.2. Representation of the global grid environment in 2x2 cell groups, each consisting of 5x5 cell grids.

The second option for a cell group is to involve blocking cells. Configuration of blocking cells in a cell group is important for the proper representation of the environment at hand. If the blocking cells are spread randomly over the grid, then it is be possible for the agent to sneak through the holes among the blocks. Therefore, three types of blocks are used in a cell group, each having equal

probability of appearance. The first is a plus-shaped block, which prevents horizontal and vertical movement through that cell group. The second and the third types are horizontal and vertical dash-shaped blocks, which prevent vertical and horizontal travel through that cell group, respectively. The types of cell blocks are shown in Figure 3.4.

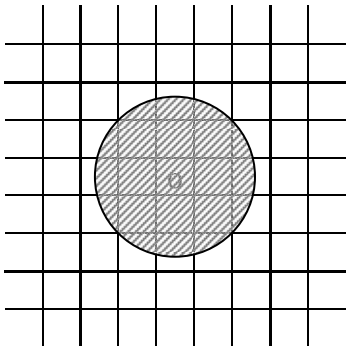
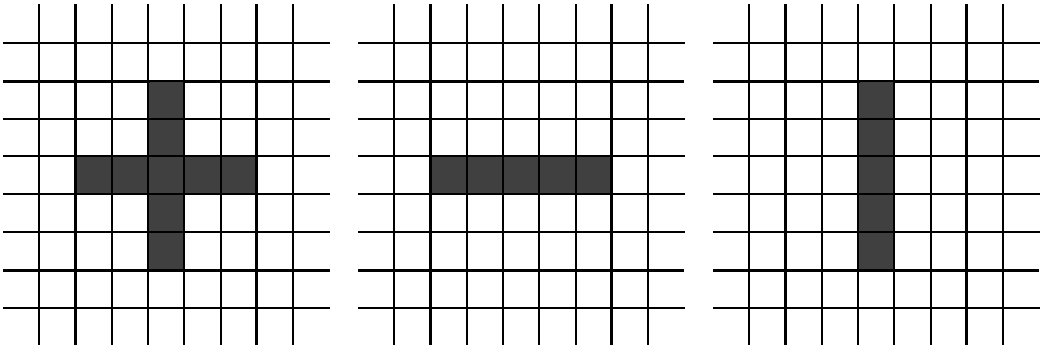


Figure 3.3. Representation of a threat source in a 5x5 cell group and its area of effect, which covers the centers of the middle cells at the edges of the group slightly.



a. b. c.

Figure 3.4. Types of cell blocks that can be utilized in a cell group, which prevent a) both vertical and horizontal movement b) only vertical movement, and c) only horizontal movement through the cell group.

3.2.2 Details of LDA* Algorithm

With the introduction of a limit in the damage parameter, the classical path length minimization problem has turned into a problem, where the goal is to determine the minimum-length path that must satisfy such a limitation. This new problem cannot be solved using naïve A*. However, with some modifications, A* can be an appropriate algorithm that can generate approximate solutions to this problem, running much faster than an optimal and complete algorithm.

It is best to apply this modification to the heuristic estimate part, h(x), used in A*. In this study, it is done by calculating the heuristic estimate, h(x), as the scalar sum of the distance heuristic estimate, $h_{di}(x)$, and the damage heuristic estimate, $h_d(x)$, each of which is multiplied by a factor. These factors are called the Distance Contribution Factor (DiCF), and the Damage Contribution Factor (DCF) according to the type of values they affect. Thus, h(x) becomes:

$$h(x) = DiCF \cdot h_{di}(x) + DCF \cdot h_d(x) \tag{3.1}$$

The distance heuristic estimate, $h_{di}(x)$, is chosen to be calculated as the straight line distance between the centers of the cell in consideration and the goal cell. On the other hand, damage heuristic estimate is determined by calculating how much damage the agent would suffer if it traveled along the same line. This calculation is done by computing the distance traveled in the effective area of each threat source and converted to damage by making use of the characteristic values of each threat source (initial shot delay, shot-to-shot delay, and damage of single shot) and the agent's navigation speed. Figure 3.5 illustrates this concept on an example configuration.

Other than the heuristic estimate values, the traveled path cost values must also be accounted for both distance and damage. This is done by keeping a record of

the distance traveled as g(x), and the distance traveled in each threat zone for each search state. Since damage is not an optimization parameter in the problem, it is not added into the total path cost, f(x), of the search states, but is rather used as a validity parameter for the state in consideration, as shown in Equation 3.2, where only the heuristic estimate involves a contribution of expected damage. Thus, at each exploration step, the LDA* algorithm chooses the most promising node among the fringe of the search tree by considering their total path cost estimates, f(x), and checks validity of that node by calculating and comparing the total suffered damage from the distance values traveled in threat zones with the allowed limit.

$$f(x) = g(x) + h(x) \tag{3.2}$$

where, h(x) is as defined in Equation 3.1

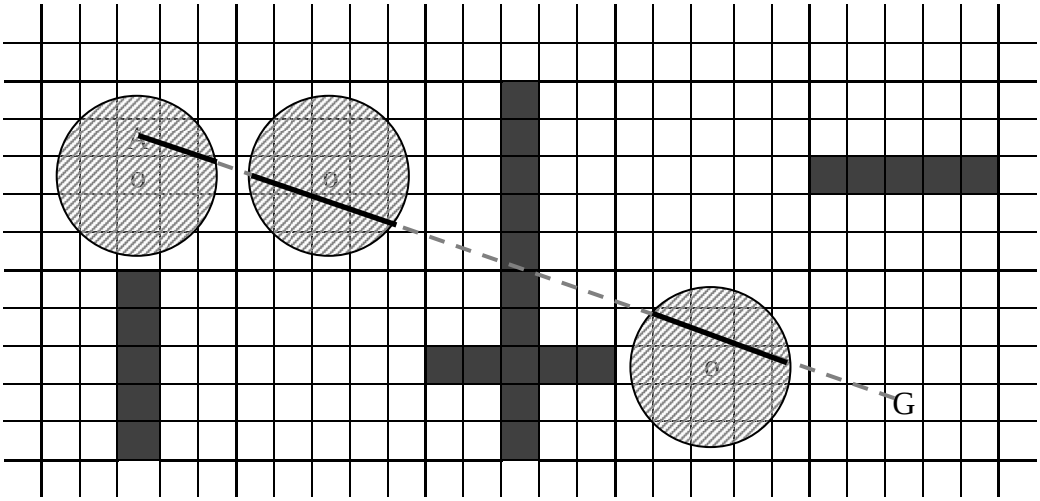


Figure 3.5. Center-to-center distance from the current cell explored by the algorithm (A) to the goal cell (G) (gray dashed line from A to G), and its portions that lie within the areas of effect of some threat sources (solid black lines) for an example configuration, as used in heuristic estimate calculation.

A* algorithm is optimal in graph search problems with proper heuristics, since it expands a node only once; therefore it is known to generate a solution with minimum number of search iterations. However, in the presence of a second optimization parameter, it is not possible to satisfy the optimality condition, unless the optimization parameters are dependent on each other. Moreover, expanding a cell only once can be a drawback for some configurations of the problem at hand. These situations involve the conditions, where exploring a cell from a parent with less suffered damage than one with higher damage (regardless of the path length) is crucial for being able to explore the search tree deeper, where a solution may be present. Therefore, a path repair mechanism is introduced into the proposed algorithm. This mechanism works when a node is extracted from the group of unexplored nodes and its children are being generated. It checks if the child had been generated previously, and changes the parent pointer of the child to the current node, if the child can be reached from the current node with either less damage and equal or less traveled path length, or with less distance and equal or less damage.

The contribution factors for distance and damage are also key factors that determine the behavior of the algorithm for the selection of the next node to be expanded, since they affect the weight of path length and damage in the heuristic. However, optimal values of the contribution factors DiCF and DCF depend on the environmental parameters, and there exist no universal optima. Therefore, their effect on the speed and success of the algorithm must be examined experimentally for various values and their optimal values for an environment should be determined based on such data. Values used in this study are presented in the next chapter, with the results of experimental tests.

CHAPTER 4

EXPERIMENTAL RESULTS

The algorithm presented in this work can be applied to various environments, each having different properties. In order to achieve high efficiency from the algorithm, its parameters must be fine tuned in accordance with the environmental parameters. However, it is not possible to enumerate all possible cases in this work. Therefore, environments with specific conditions are selected for experimental analysis.

Traversable area of the environment is chosen to consist of a square grid map, consisting of 7x7 cell groups, whose general properties are as defined in Chapter 3. Three different grid sizes as 28x28, 49x49 and 98x98 cells were used in the experiments to observe the change in the performance of the algorithm with increasing search space. Memory requirement of MOA* had been the determining factor on the selection of grid sizes to be used. During the tests, 98x98 grids were observed to occupy 200MB to 300MB of RAM, and 147x147 grids were unable to be solved with 768MB of allocated Java Virtual Machine memory. Consequently, the biggest grid size was chosen to be 98x98 for the tests. In all tests, the performance of LDA* was compared to the reference values of MOA*.

A notebook computer, with 1.6GHz Intel Dothan CPU and 1GB RAM was used for the tests. The CPU clock was set to constant 1.6GHz frequency, and no other

programs were run during the tests. Both LDA* and MOA* algorithms were implemented in Java 1.5 environment, and were run in Borland's "Turbo JBuilder 2007" integrated development environment under Windows XP SP2 operating system. Java virtual machine's memory was set to 512MB for all runs.

4.1 PRELIMINARY TESTS

Distance and damage contribution factors are the key factors affecting the performance and behavior of LDA*. Therefore, a set of preliminary tests were run on the environment defined above, in order to observe the best combination, and continue the rest of the analyses with those numbers.

In these tests, values of the Distance Contribution Factor (DiCF) were chosen to be 0.1, 0.5 and 1.0. Using the value of DiCF as 1.0 essentially indicates the usage of optimal distance heuristic of A* for the distance portion of the whole heuristic. On the other hand, Damage Contribution Factor (DCF) was chosen to vary as 0.0, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 40, 50, 60, 70, 80, 90, 100, 1000, 10000 and 100000. Very large and very small numbers were used for this factor, in order to observe the effect of dominance of one part of the total heuristic on the whole.

The test maps were generated randomly as explained in Chapter 3. Later, start and goal positions were selected at random among the non-occupied cells, with the condition of being different from each other. To achieve a high probability of the existence of a solution for the search, the allowed damage was set to 30 HP. MOA* was run on the defined configuration, and if any solutions had been found, LDA* was run on the same environment, with the allowed damage factor equal to the minimum-damage solution of MOA*. If LDA* was unable to find a result for a specific case, the allowed damage was incremented by 1 and the

algorithm was re-run. 200 different cases were tested for 28x28 and 49x49 grids, and 100 cases for the 98x98 grids.

By determining the possible minimum-damage for a configuration, and running LDA* on that configuration, it was aimed to measure the performance (success and failure rate) of LDA* under the worst case conditions, where the damage limitation was stated so tightly that only one solution existed. Figures 4.1, 4.2 and 4.3 show the variation of failure rate of LDA* with increasing DCF, for different DiCF values, for different grid sizes. The charts are limited at DCF=120, since the results tend to remain constant after DCF=100.

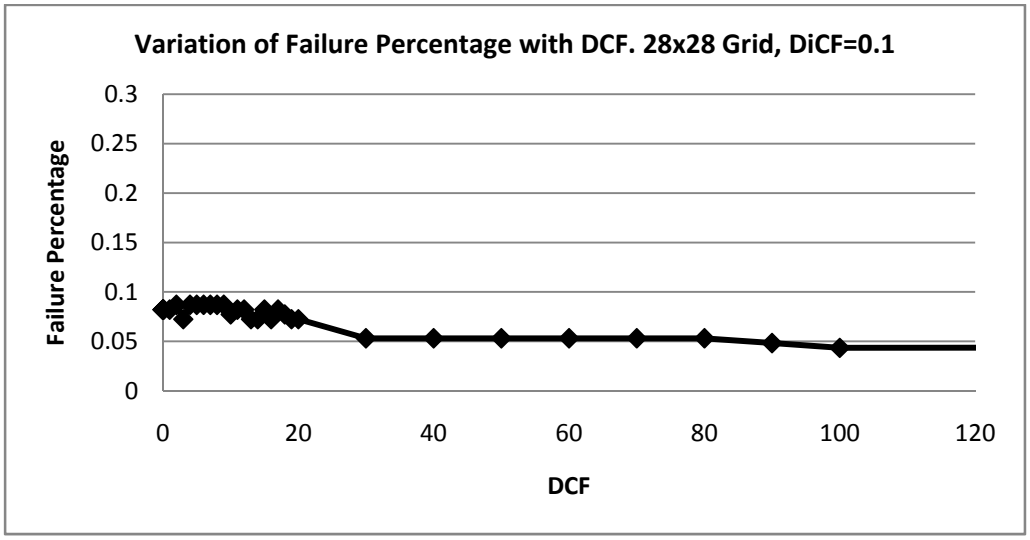
Observing these charts, LDA*'s failure rate seems to be slightly higher for small DCFs, especially in the region close to 0.0. It tends to decrease until DCF is in the range of 20 to 40, and for higher values the failure rate seems to remain minimal and constant. For the small grids (28x28), the maximum failure rate is observed, when DiCF is 1.0 and DCF is 0.0, which is about 12%. The minimum failure rate among these maps is about 5%, which is observed, when DiCF is 0.1, and DCF is greater than 40. For the medium sized grids (49x49), there is a slight increase in the maximum failure rate compared to the smaller grids. Failure rate is 15% and it is observed for the configurations, where DiCF is 1.0, and DCF is 0.0. The minimum failure rate tends to decrease slightly, down to 11% in such configurations. For the large grids (98x98), the maximum failure rate is about 29%, when DiCF=1.0 and DCF is 0.0. The value drops down to 11% when DiCF is 0.1 and DCF is 80. In general, there seems to be slight tendency for increased failure rates with increased DiCF values. LDA* seems to be quite successful, especially with success rates up to 95% for small maps, and 89% for medium and large grids with smaller DiCF values and DCF values greater or equal to 20.

In addition to success/failure rate analyses, the solution path lengths generated by successful runs of LDA* and MOA* were compared for varying DCF values. The results of this analysis are shown in Figures 4.4, 4.5 and 4.6 in terms of path length ratios. On the average, LDA* tends to generate approximately 0% to 3% longer paths than the exact solution for the small grids. This value varies up to 4% for the medium-sized grids and to 6% for the large grids, all of which are observed at high DCF values, greater than 80. Moreover, it is observed from these results that LDA* path lengths are closer to those of the exact solutions, especially in the 0-20 range of DCF, with slight increase towards 20. Here, better results are observed for average and maximum solution path length, when DiCF is 0.5, compared to the other results, where DiCF is 0.1 or 1.0.

The third comparison case of the preliminary analyses tests is about the CPU runtime ratios of LDA* and MOA*, which are presented in Figure 4.7, 4.8 and 4.9. Examining all cases, LDA*, on average, is observed to finish its run at 2% of the time spent by MOA* in the worst case, and at less than 0.5% in the best case. Considering all cases, average runtime ratio is observed to increase slightly, and deviation in it is observed to decrease slightly with increasing DCF values.

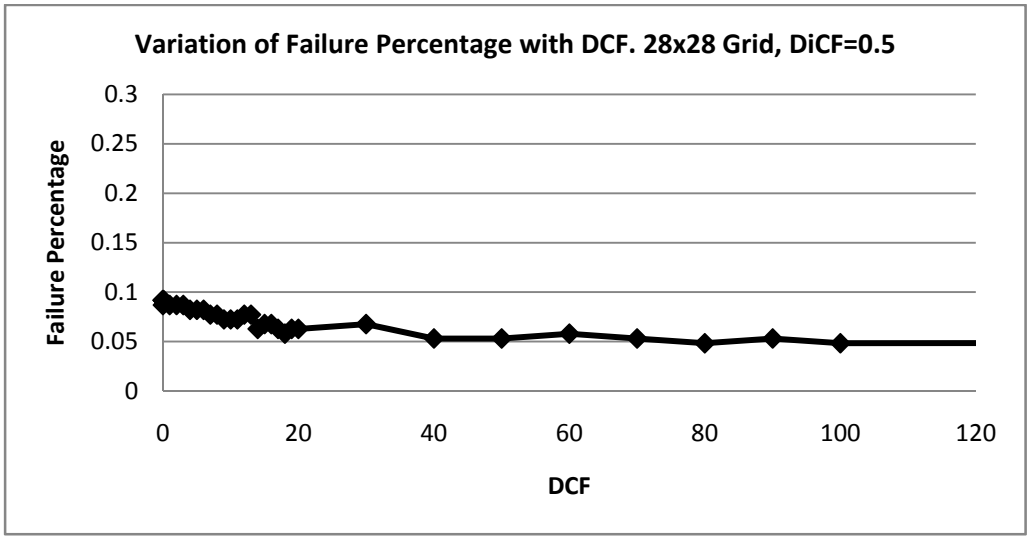
In the absence of a second optimization parameter as damage, MOA* and LDA* are very similar algorithms, if not the same. Therefore, for small distances and optimal heuristics, they are expected to exhibit similar runtime values. Considering some overhead involved in MOA* compared to LDA*, this is observable in Figures 4.7 through 4.9, and as expected, LDA* runtimes tend to decrease with increasing grid size. Moreover, for a given grid size, LDA* runtime ratio tends to decrease with increasing DiCF, since LDA*'s distance heuristic approaches the optimal distance heuristic as DiCF approaches 1.0.

It is observed from the preliminary test results that, worst case failure rate for LDA* is almost constant when DCF is around 20 or greater. In addition, the path length ratio of LDA* to MOA* tends to increase with increasing DCF values at the smaller DCF values, and stabilize at high values. Moreover, solution's average path length ratio is observed to be closer to 1.0, when DiCF is 0.5, than for the other values of it. Considering these results, DCF and DiCF values to be used in performance analysis tests of LDA* are chosen to be 19, 0.5, respectively. Results of the performance tests are presented in the next section.

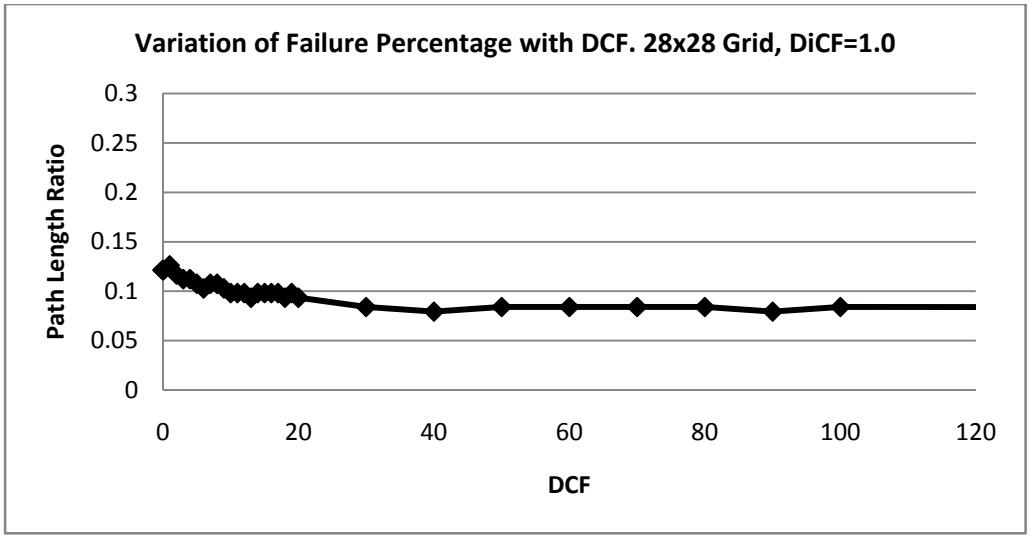


a.

Figure 4.1. Variation of failure percentage of LDA* with DCF for 28x28 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

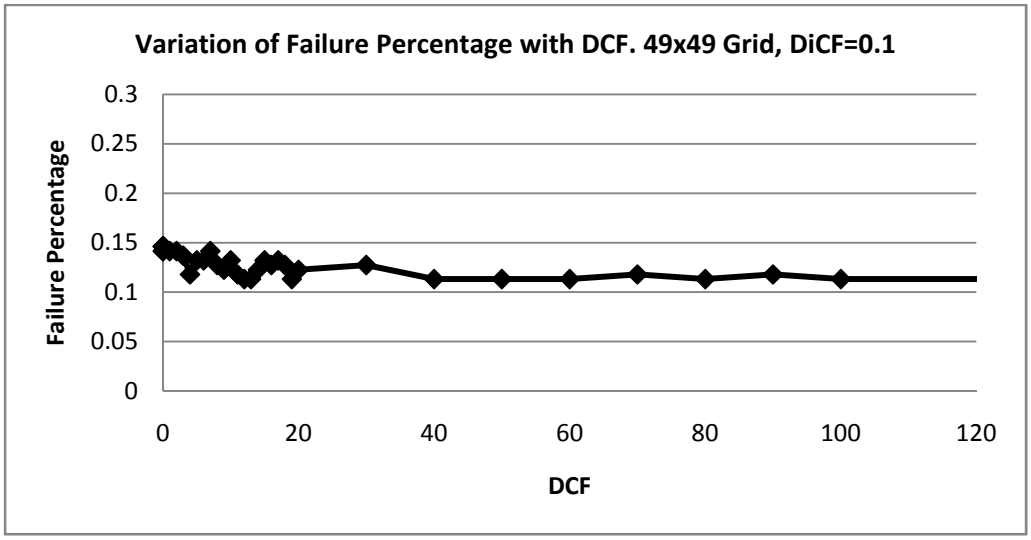


b.

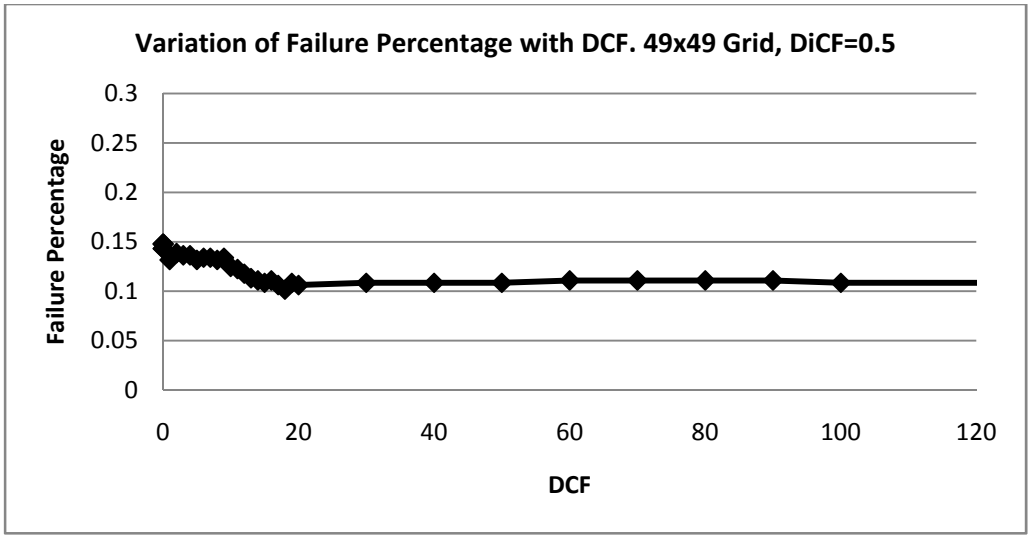


c.

Figure 4.1 (cont'd). Variation of failure percentage of LDA* with DCF for 28x28 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

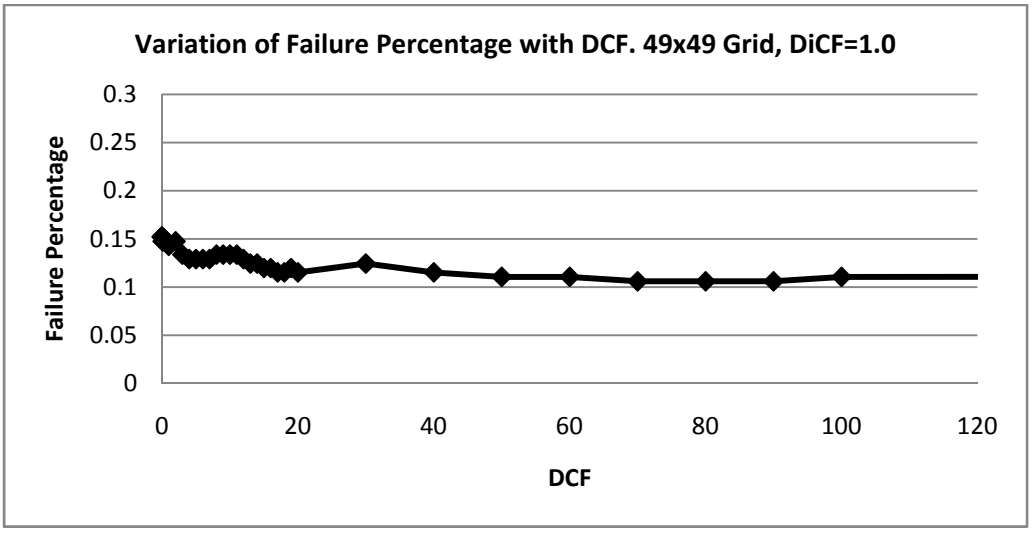


a.



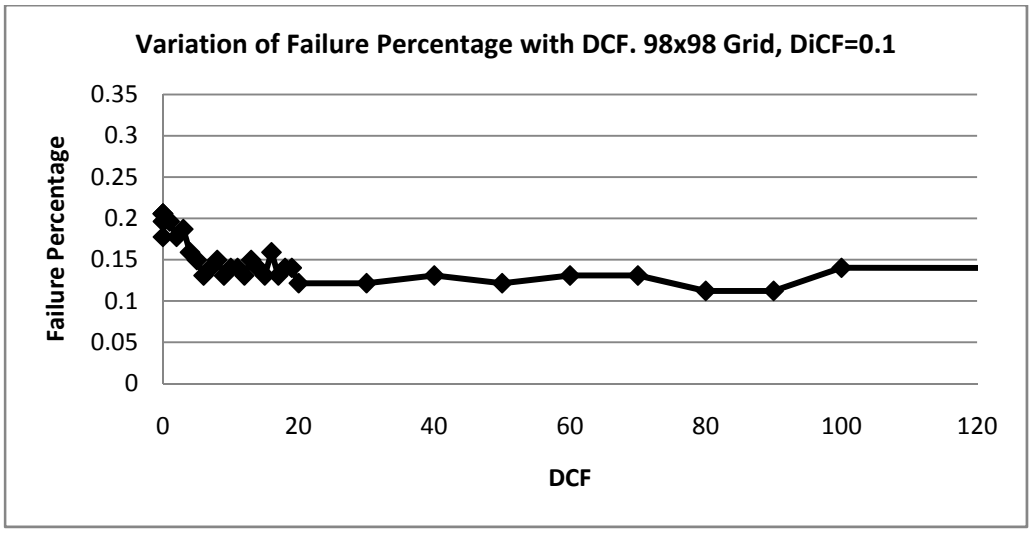
b.

Figure 4.2. Variation of failure percentage of LDA* with DCF for 49x49 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.



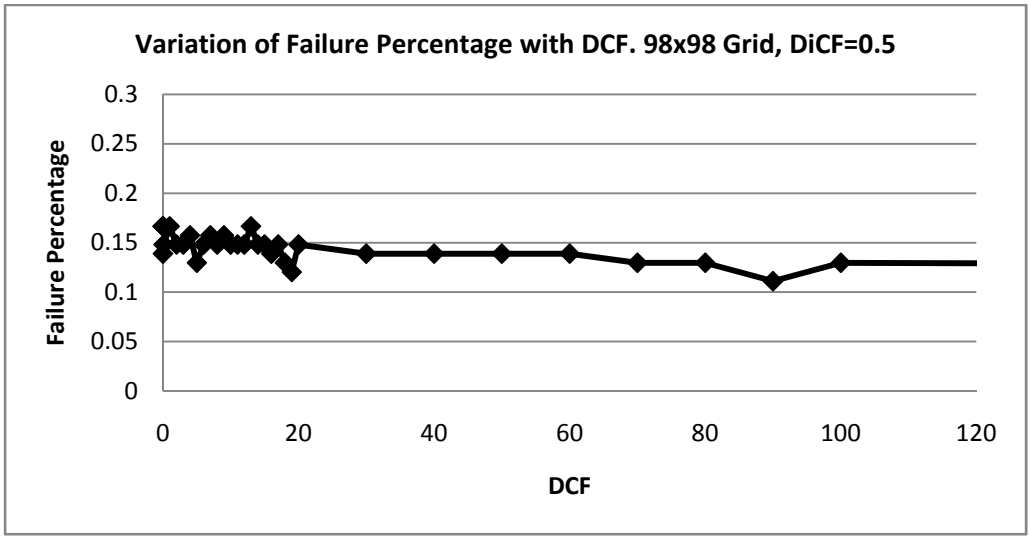
c.

Figure 4.2 (cont'd). Variation of failure percentage of LDA* with DCF for 49x49 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

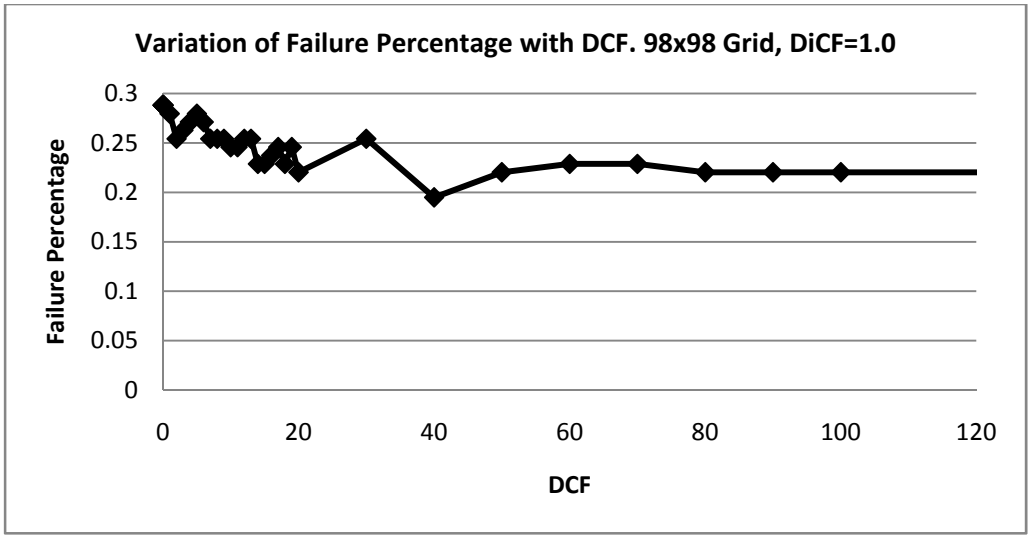


a.

Figure 4.3. Variation of failure percentage of LDA* with DCF for 98x98 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

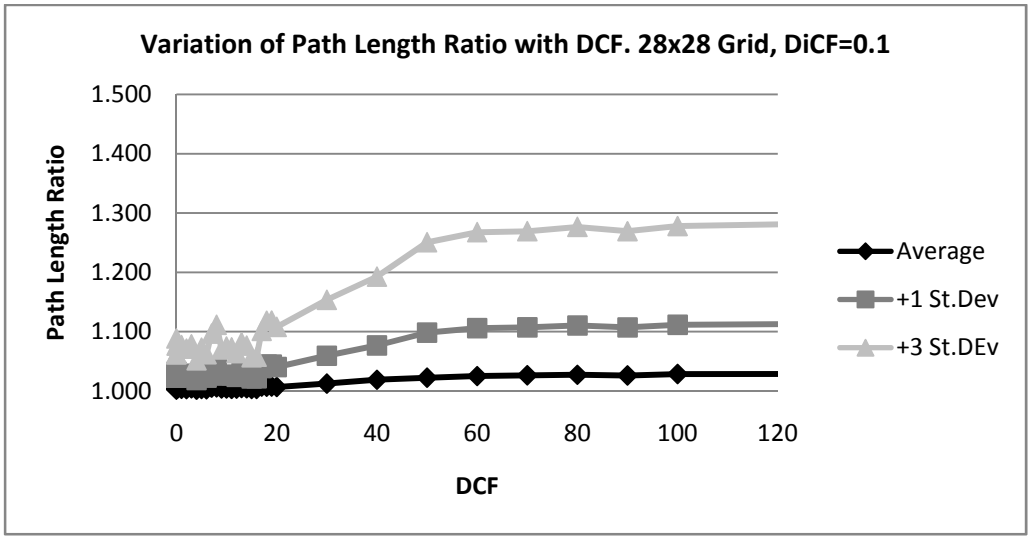


b.

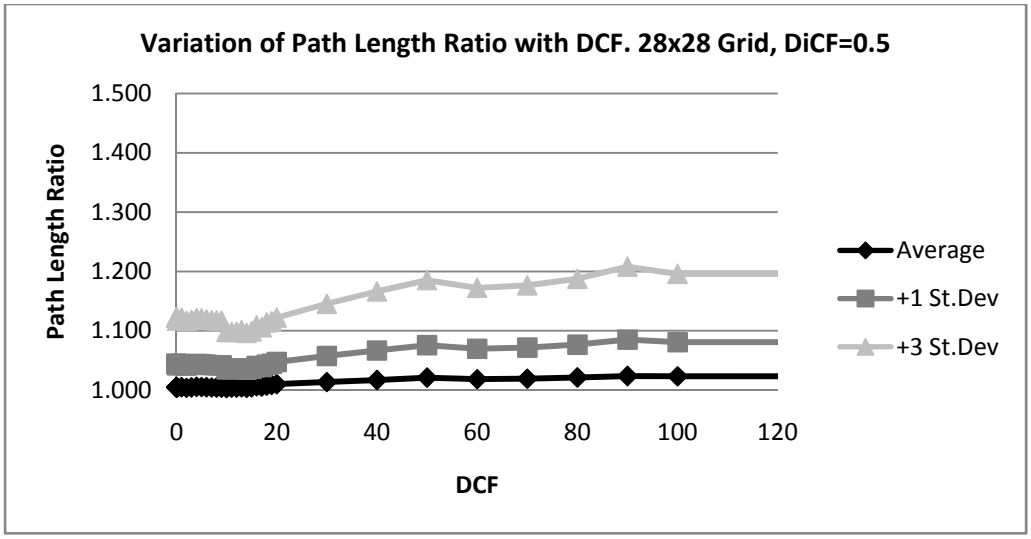


c.

Figure 4.3 (cont'd). Variation of failure percentage of LDA* with DCF for 98x98 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

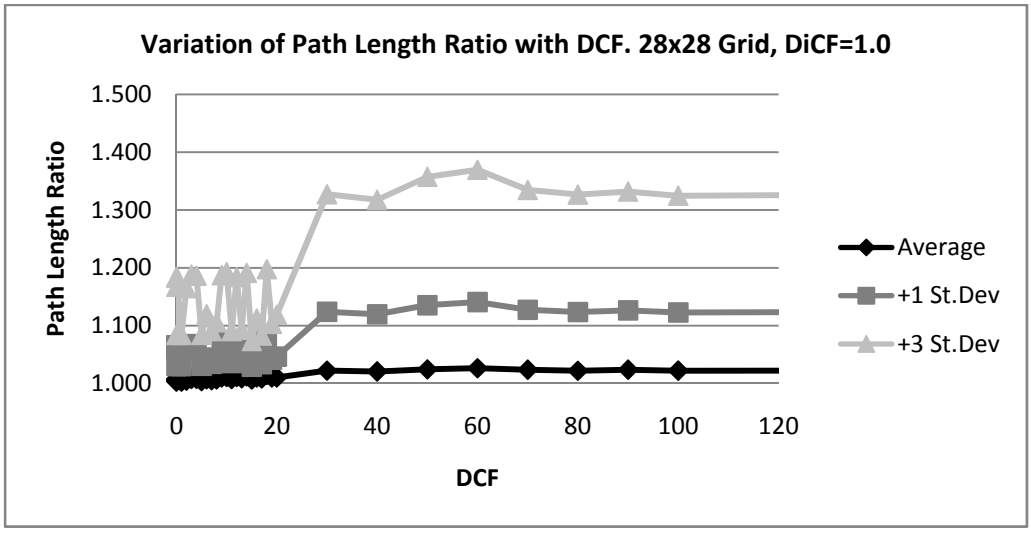


a.



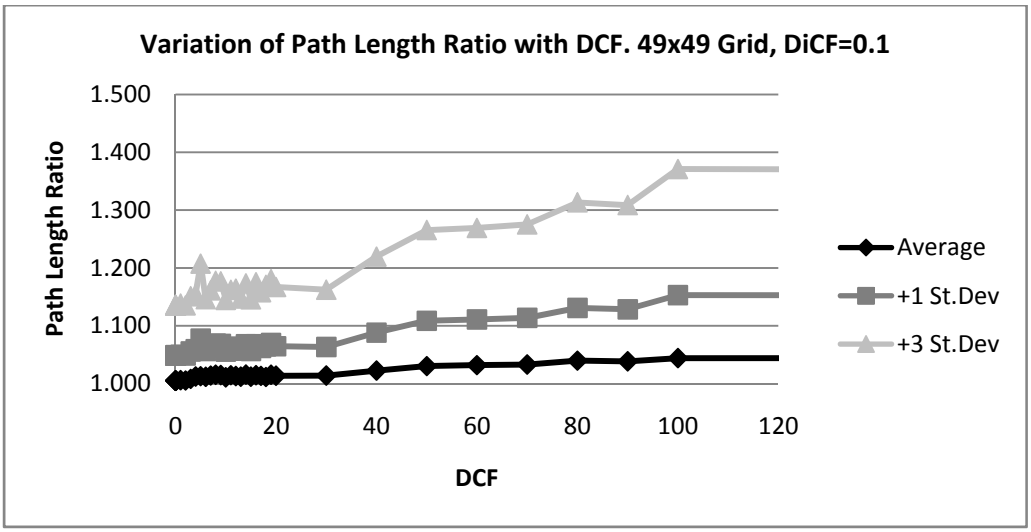
b.

Figure 4.4. Variation of solution path length ratio of LDA* to MOA* with DCF, for 28x28 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.



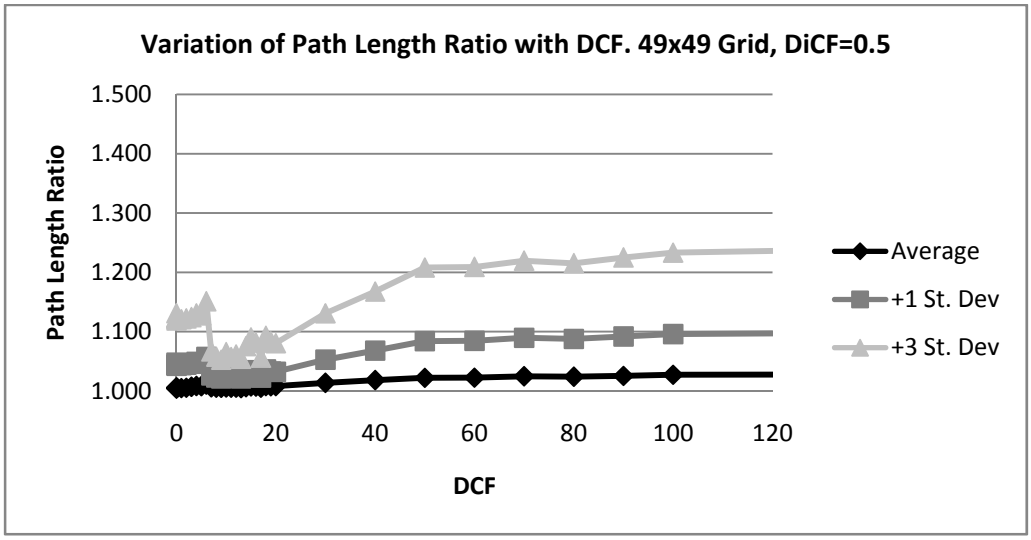
c.

Figure 4.4 (cont'd). Variation of solution path length ratio of LDA* to MOA* with DCF, for 28x28 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

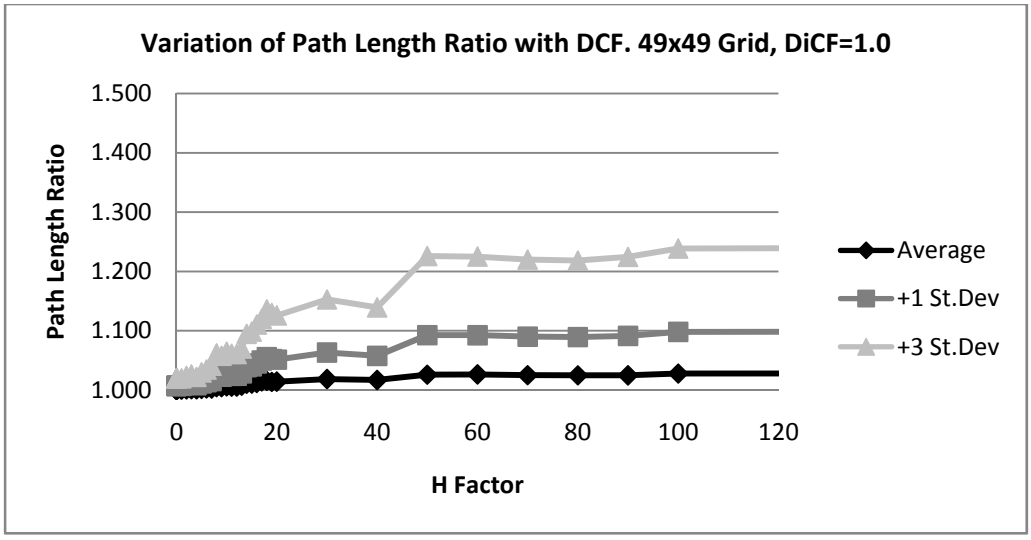


a.

Figure 4.5. Variation of solution path length ratio of LDA* to MOA* with DCF, for 49x49 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

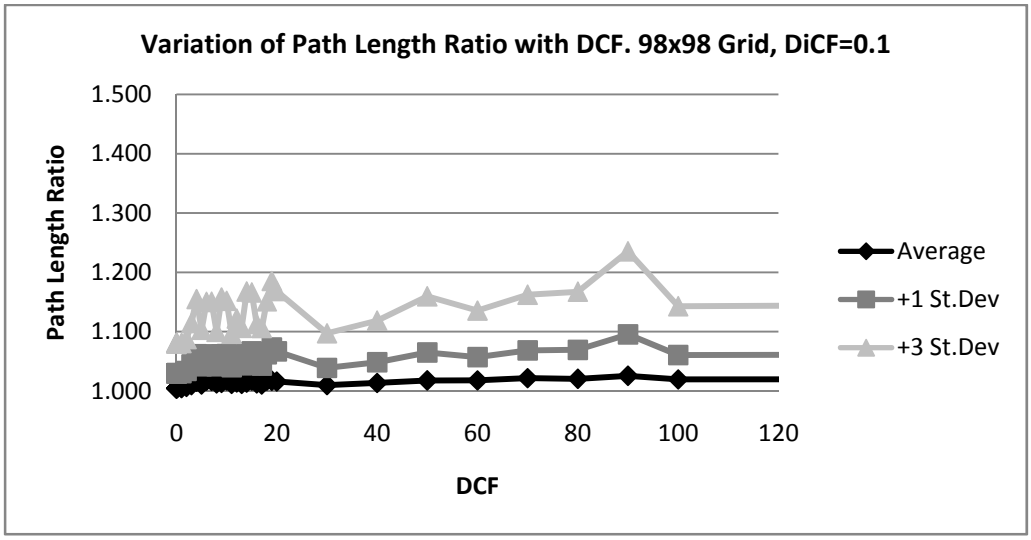


b.

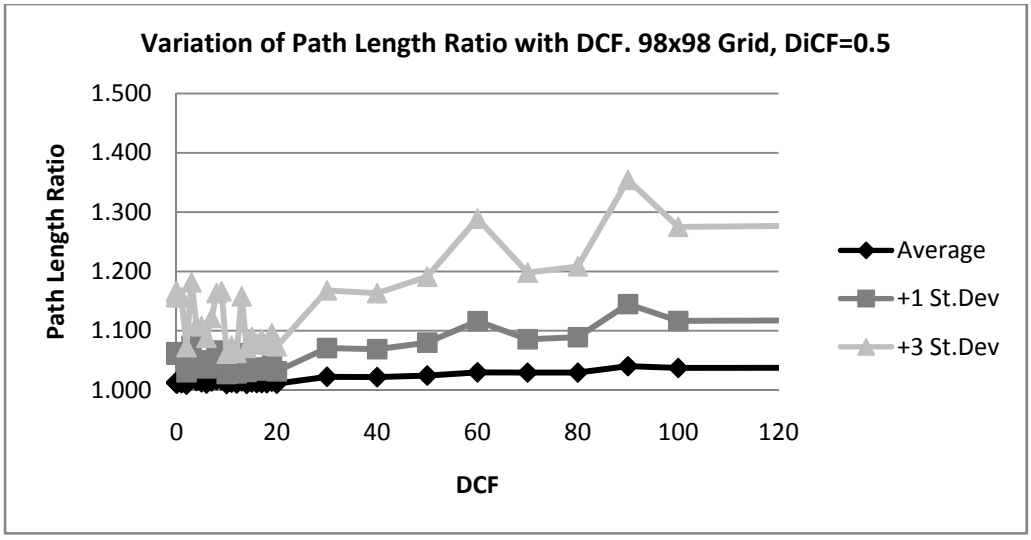


c.

Figure 4.5 (cont'd). Variation of solution path length ratio of LDA* to MOA* with DCF, for 49x49 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

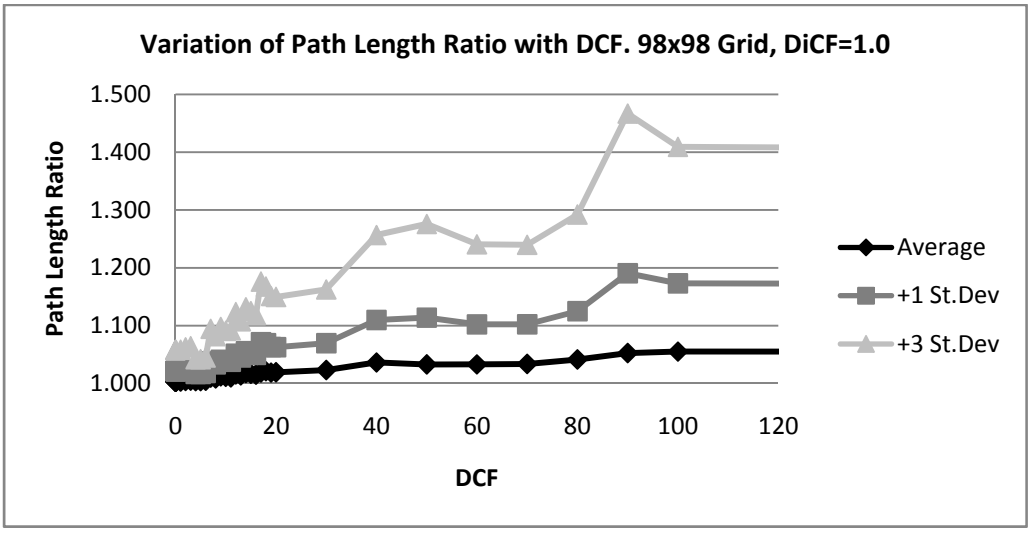


a.



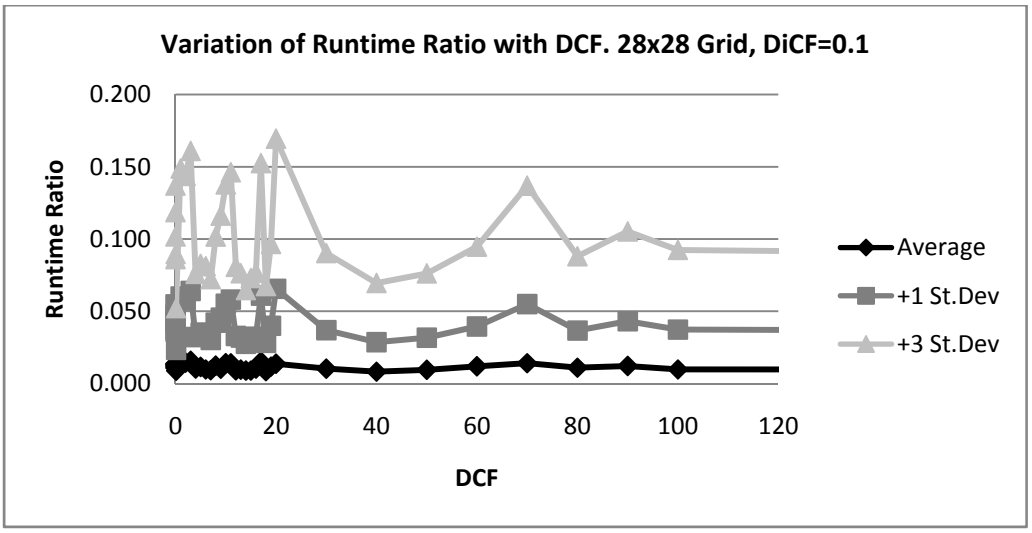
b.

Figure 4.6. Variation of solution path length ratio of LDA* to MOA* with DCF, for 98x98 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.



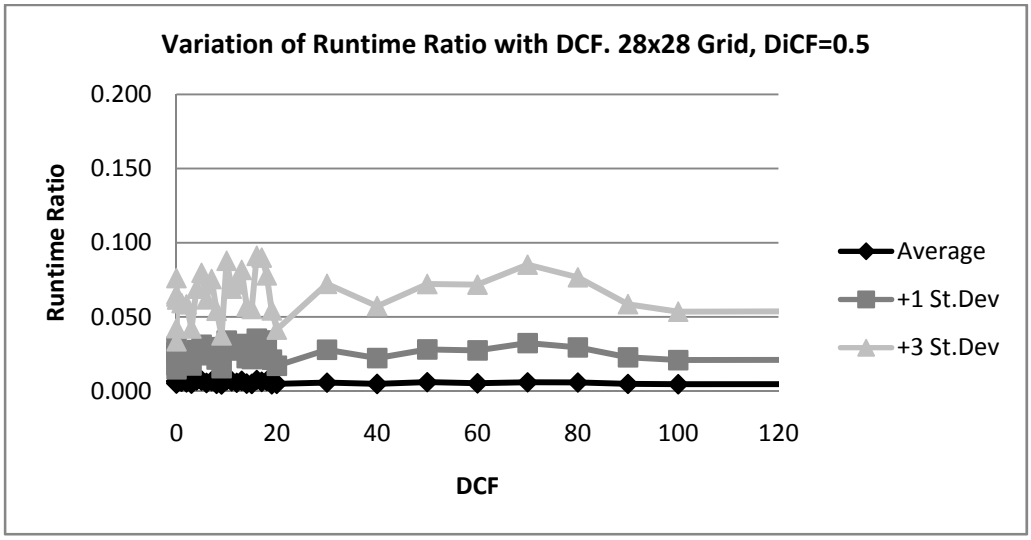
c.

Figure 4.6 (cont'd). Variation of solution path length ratio of LDA* to MOA* with DCF, for 98x98 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

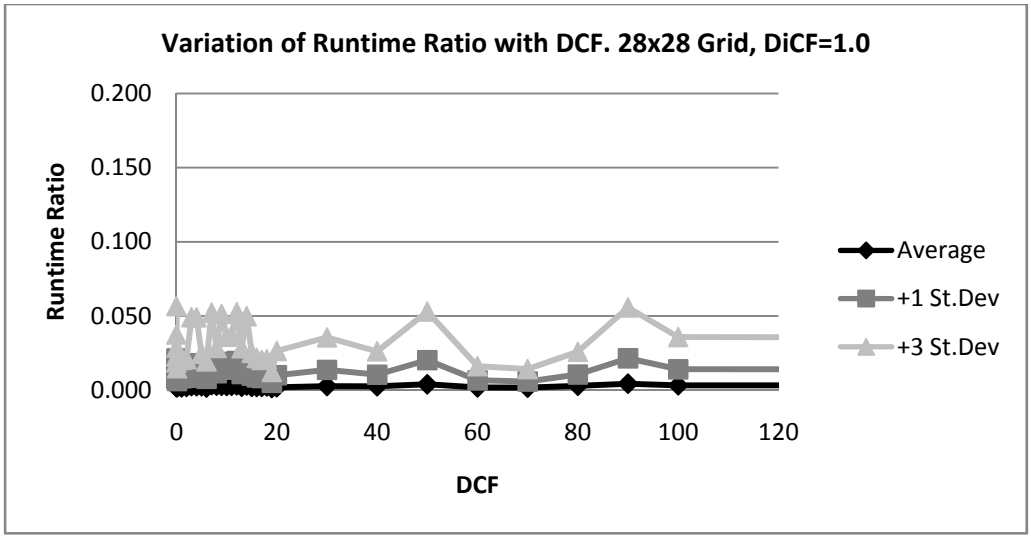


a.

Figure 4.7. Variation of CPU runtime ratio of LDA* to MOA* with DCF, for 28x28 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

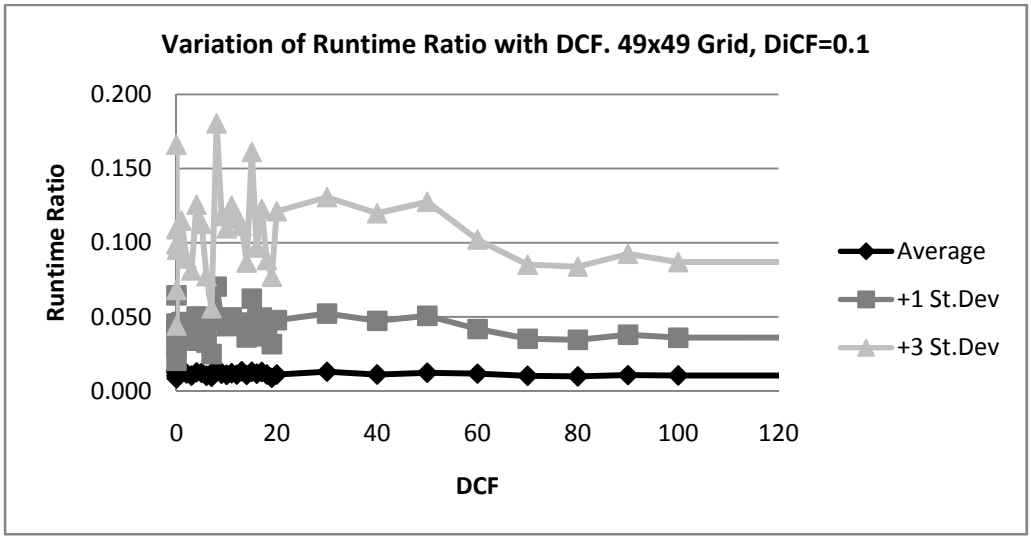


b.

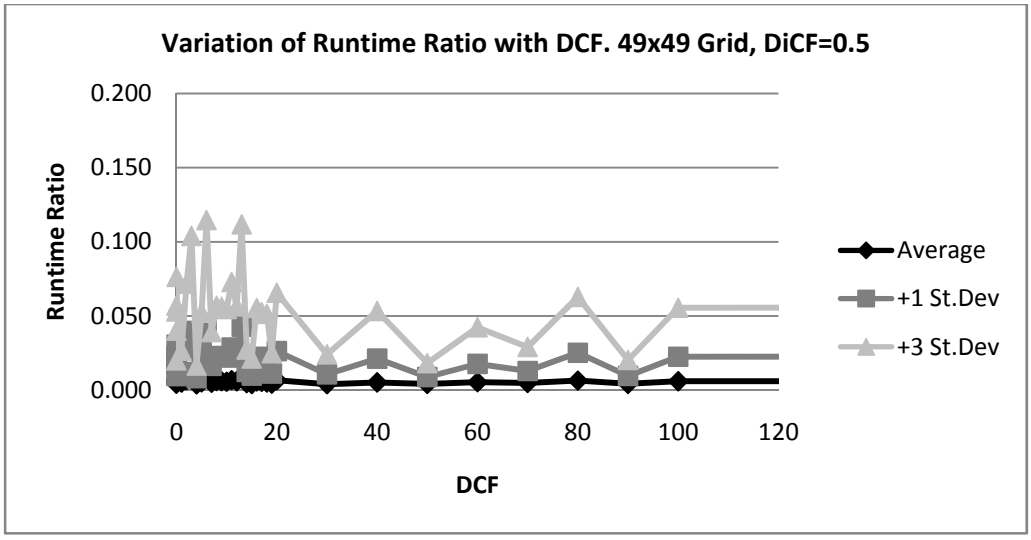


c.

Figure 4.7 (cont'd). Variation of CPU runtime ratio of LDA* to MOA* with DCF, for 28x28 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

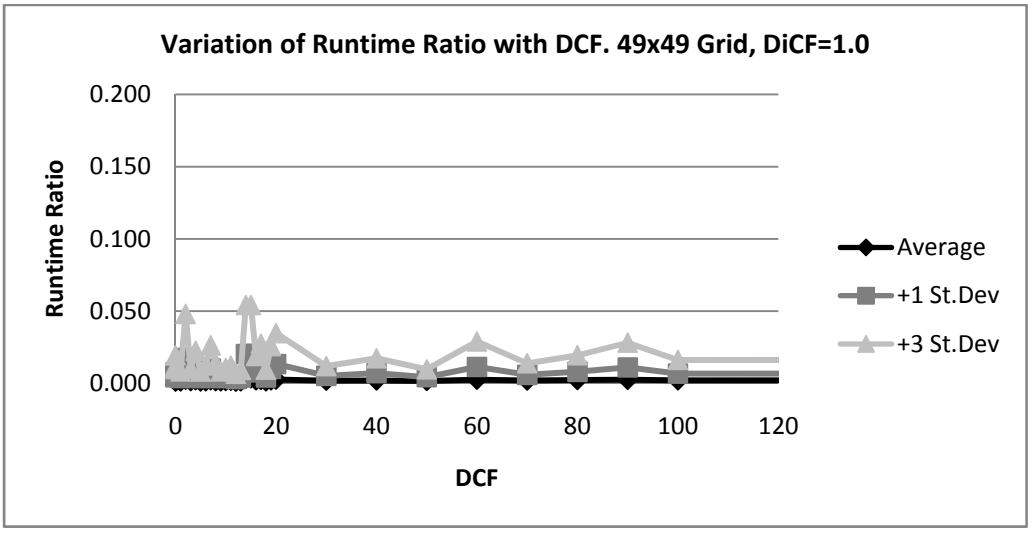


a.



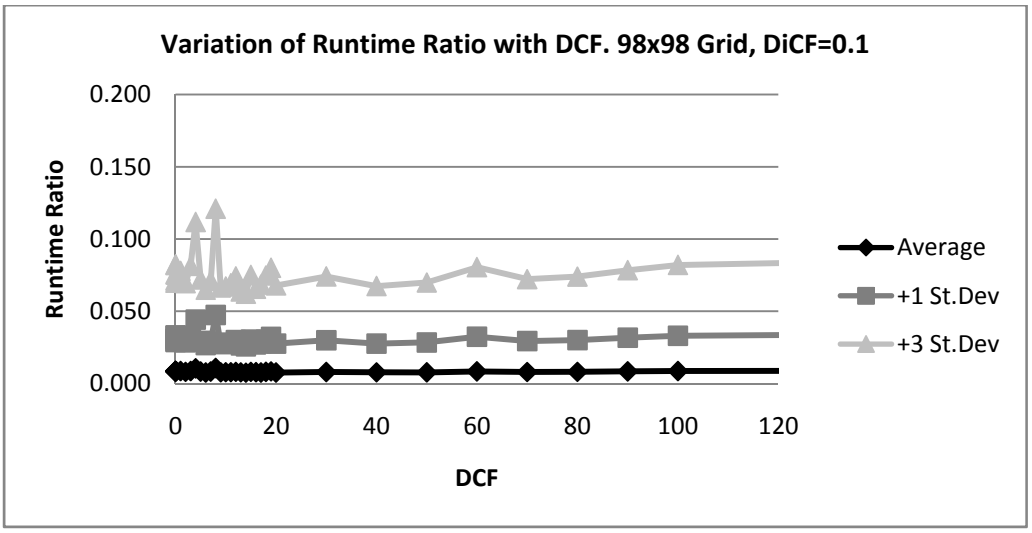
b.

Figure 4.8. Variation of CPU runtime ratio of LDA* to MOA* with DCF, for 49x49 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.



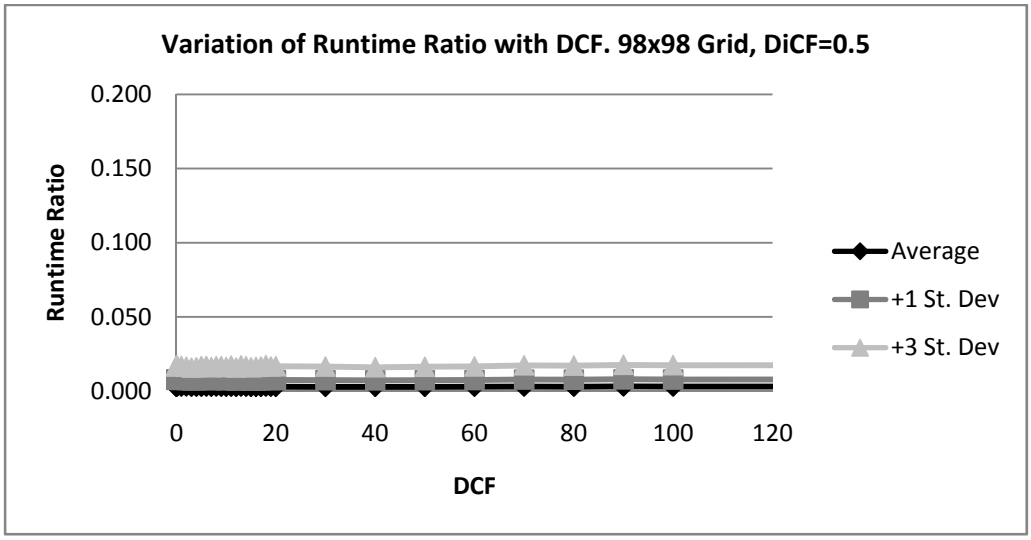
c.

Figure 4.8 (cont'd). Variation of CPU runtime ratio of LDA* to MOA* with DCF, for 49x49 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

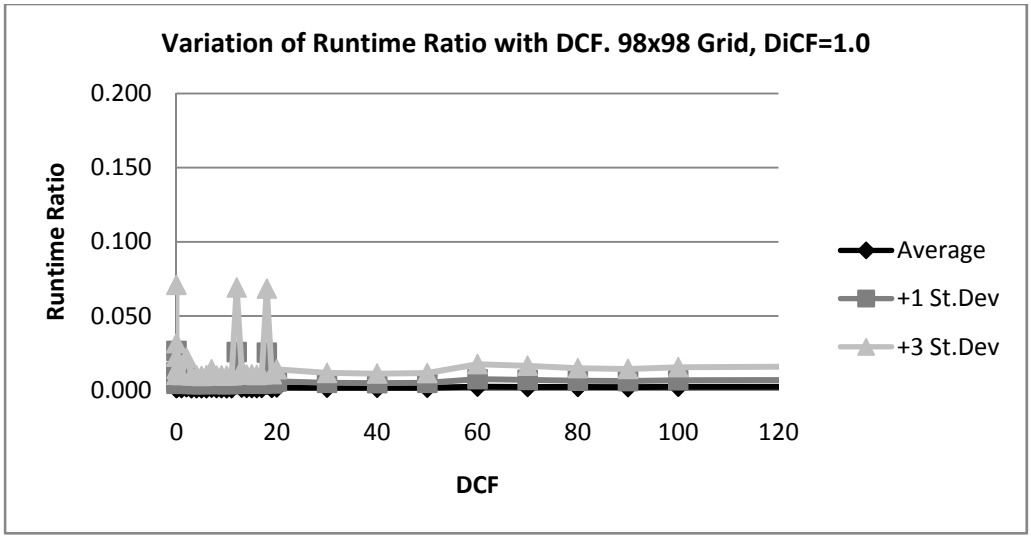


a.

Figure 4.9. Variation of CPU runtime ratio of LDA* to MOA* with DCF, for 98x98 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.



b.



c.

Figure 4.9 (cont'd). Variation of CPU runtime ratio of LDA* to MOA* with DCF, for 98x98 grids, for the worst case condition, where the allowed damage equals its minimum possible value for a configuration, for a) DiCF=0.1, b) DiCF=0.5, and c) DiCF=1.0.

4.2 PERFORMANCE TESTS

These tests are aimed to measure the path length and runtime performance of LDA*, compared to that of the exact solution algorithm MOA*. In these tests, the same environmental parameters are used, as defined in this chapter and in the previous chapter.

The tests were run on randomly generated grids, for constant DCF value of 19 and constant DiCF value of 0.5, where the allowed damage parameter varied from 0 to 50 in increments of 10. In other words, the extent of damage that the agent was allowed to suffer varied from none and 50% of its health. For each of the tests, a new map, a start cell, and a goal cell were generated at random. Later, the generated case was solved by MOA*. If a solution existed for the given case, then the same configuration was solved by LDA*, and the results were recorded. If a solution could not be found, then another map was generated, and the above mechanism was applied to the newly generated map. Thus, only the cases having legitimate solution(s) were analyzed.

In this set of performance tests, 400 28x28 maps were analyzed, for which LDA* and MOA* were run 10 and 4 times, respectively, on the same configuration, whose results were then averaged. For the 49x49 maps, 200 test runs were performed in a similar fashion, where LDA* was run 5 times and MOA* was run 2 times. Finally, 100 separate tests were performed on 98x98 maps, LDA* being run for 5 times, and MOA* being run for only once. Number of tests was reduced with increasing grid size due to the fast growing runtime characteristics of MOA*.

Test results were examined for runtime and path length characteristics. Later, they were plotted against the shortest exact solution path length (in number of cell groups), which was obtained from the maximum damage - minimum path

length solution of MOA* for the given allowed damage, in 4 categories: LDA* runtime, MOA* runtime, runtime ratio of LDA* to MOA*, and path length ratio of LDA* to MOA*.

Figure 4.10 shows the variation of LDA* runtime with increasing distance between start and goal cells for 28x28, 49x49 and 98x98 grids, respectively. It is observed that the results for the random input spans the path lengths from one cell group side length (7 cells) to twice the side length of the grid. As expected, the runtime increases with increased distance, and the average runtime, considering all allowed damage cases for a chart is observed to increase polynomially, with respect to the path length.

The three charts in Figure 4.10 are not drawn to the same scale, however for a specified path length, it is observed that LDA* runtime increases approximately 3 to 4 times when the grid size is doubled. As for A*, this behavior indicates a polynomial complexity for the average runtime LDA*.

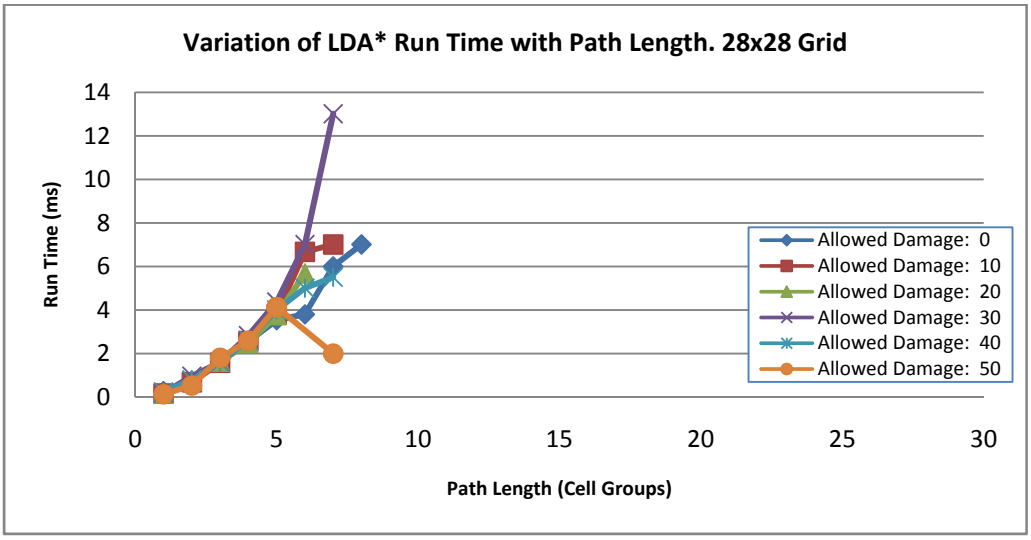
The runtime performance of MOA* for the same cases as LDA* is plotted in Figure 4.11. The first observation on these charts is that MOA* performs much better, when allowed damage is 0.0, compared to the cases, where allowed damage is greater than 0.0. Since, in this case, the multiobjective optimization problem boils down to a single objective optimization problem, this was an expected behavior, and it was also observed in the preliminary analysis test results. Considering these results, average runtime complexity of MOA* is observed to be in the polynomial order of path length. However, there are some extreme cases, as well, that are seen as sharp peaks in charts b and c of Figure 4.11. Moreover, the runtime complexity of MOA* is observed to be in less than exponential order of the map size. Consequently, further analysis is required in order to state the runtime complexity of MOA* in similar environments.

The runtime ratio of LDA* to that of MOA* is plotted in Figure 4.12. LDA* is observed to run in much less time than MOA*. It is observed that the time gained by utilizing LDA* in a similar problem instead of MOA* is more than 87% of the runtime of MOA*. In more detail, LDA* runs in less time than 4% of the running time of MOA* for small maps, less than 8% of it for the medium-sized maps, and less than 13% of MOA* for the large maps, when no damage is allowed to the agent. These are even less than 1% of the runtime of MOA* in the presence of damage, for paths longer than 3 cell groups' side length (21 cells). It is not possible to observe a definite correlation for the length of the path found by LDA* with increasing path lengths, however, LDA* outputs tend to remain less than values stated above.

The relative runtime curves are observed to exhibit ascending behavior with the path length of the best solution. Thus, LDA* runtime is observed to increase faster compared to the increase in that of MOA*, especially when allowed damage is 0.0. In addition, the relative runtime tends to increase for very small path lengths, since MOA* would run relatively faster, as it would not be keeping track of multiple solutions as much for a longer path case. As a result, LDA* performs best in terms of runtime, when allowed damage is greater than zero, and the length of the path to be searched is greater than 3 cell groups' side length (21 cells).

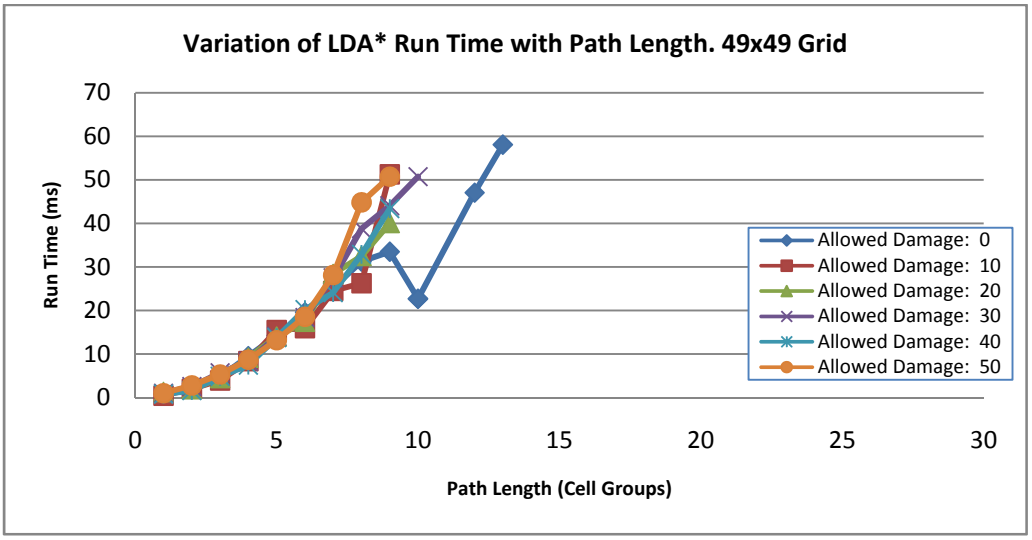
Variation of the ratio of the solution's path length found by LDA* to the shortest path found by MOA* is plotted in Figure 4.13. The results indicate that the paths found by LDA* are no greater than 5% of the best for small maps, about 8% for the medium-sized maps, and 10% for the large maps. Moreover, when allowed damage is 0.0, LDA* performs its best in terms of path length, and they are almost identical to the best path's length for small and medium-sized maps, and less than 2% longer than that of the best path for the large maps.

Finally, LDA* is observed to be very successful in finding a path within the given damage limit, for all cases. It was 100% successful in finding a solution with the given allowed damage values for both the small and the medium-sized grids. For the large maps, only one of the 100 cases was unable to be solved with the given allowed damage limit, which was solved after the limit was incremented by 2; from 10 to 12.

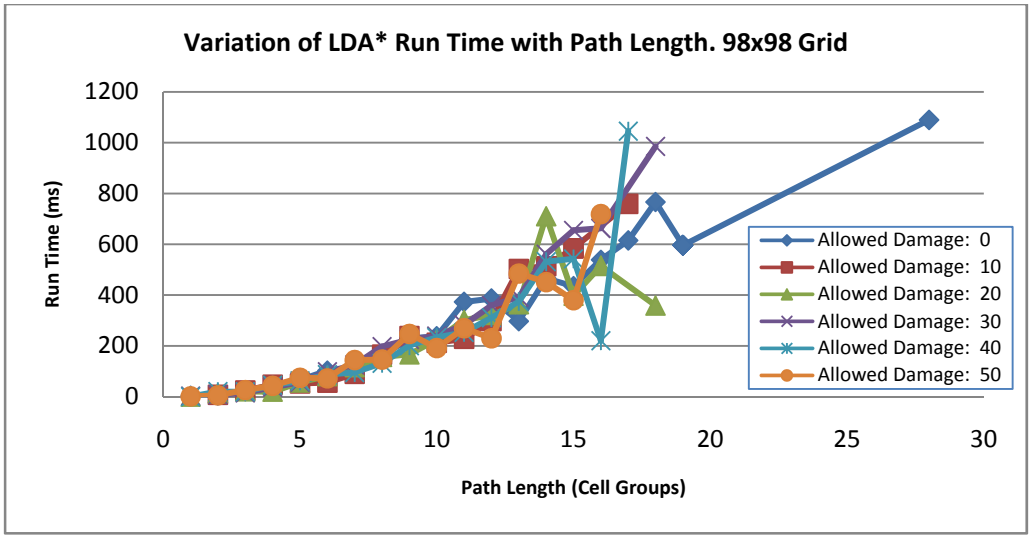


a.

Figure 4.10. Variation of CPU runtime of LDA* with path length, where DCF is 19, DiCF is 0.5, and the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.

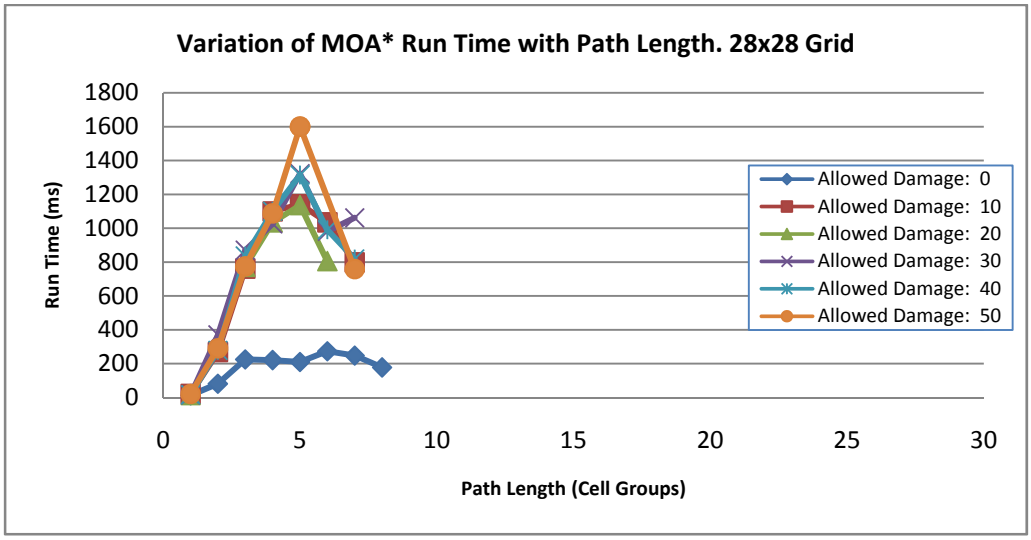


b.

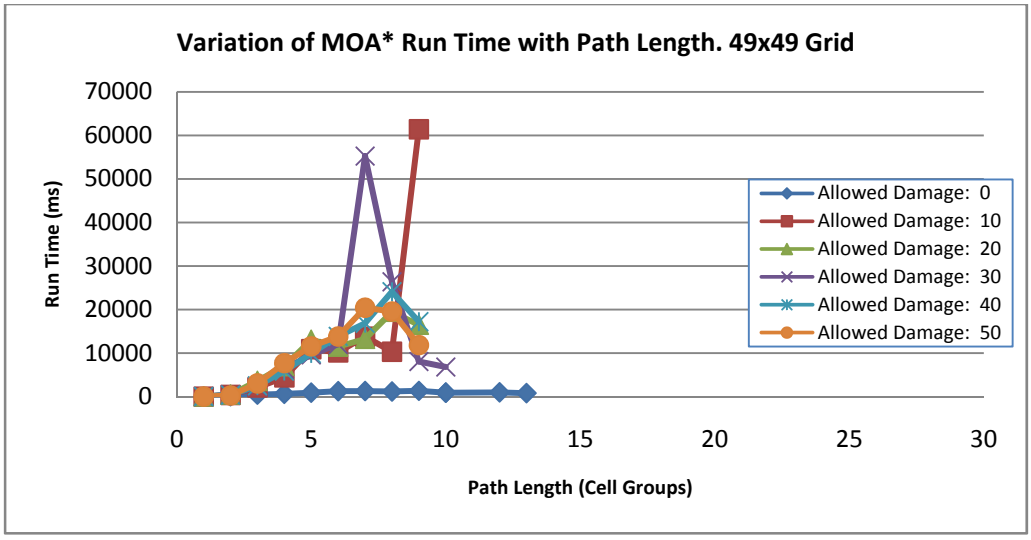


c.

Figure 4.10 (cont'd). Variation of CPU runtime of LDA* with path length, where DCF is 19, DiCF is 0.5, and the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.

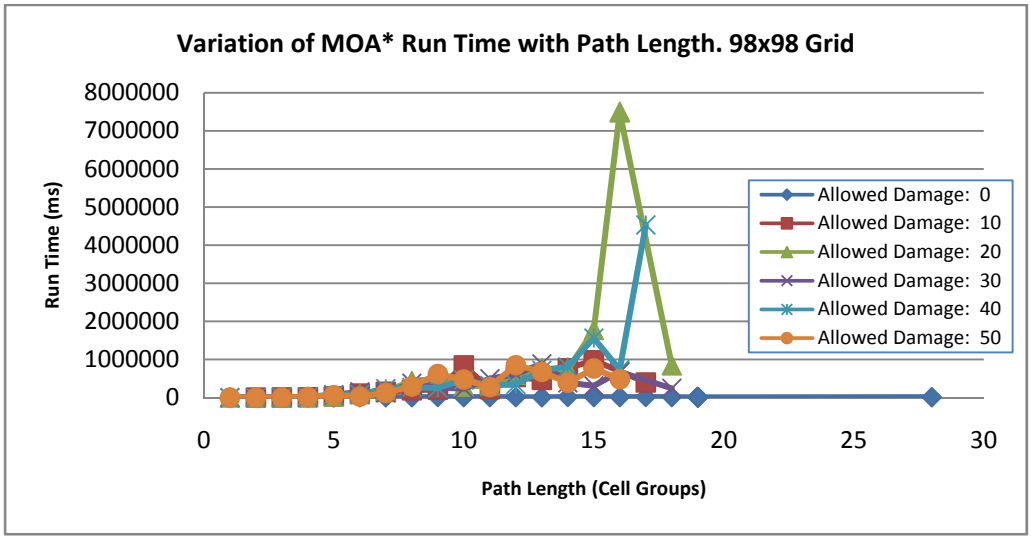


a.



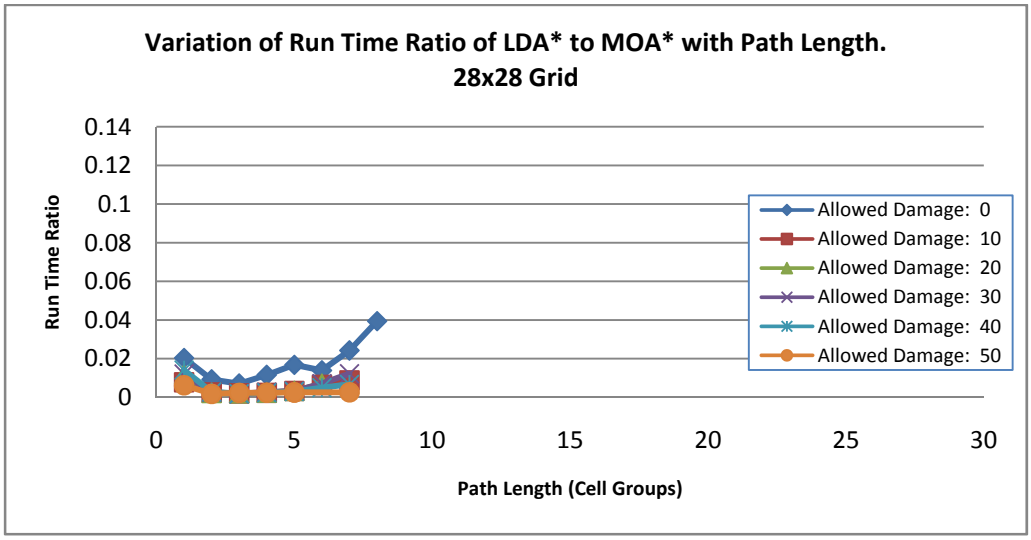
b.

Figure 4.11. Variation of CPU runtime of MOA* with path length, where the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.



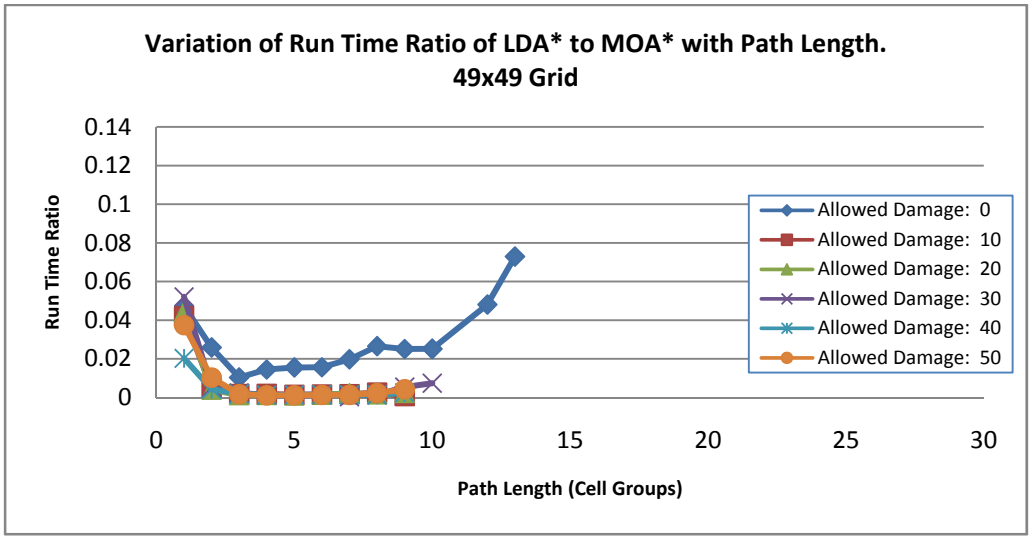
c.

Figure 4.11 (cont'd). Variation of CPU runtime of MOA* with path length, where the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.

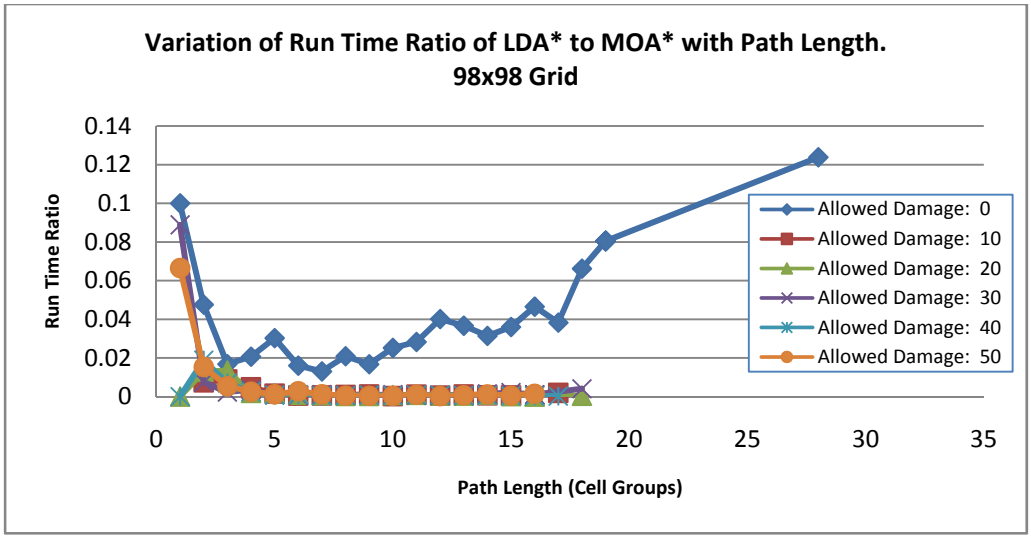


a.

Figure 4.12. Variation of CPU runtime ratio of LDA* to MOA* with path length, where LDA* DCF is 19 and DiCF is 0.5, and the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.

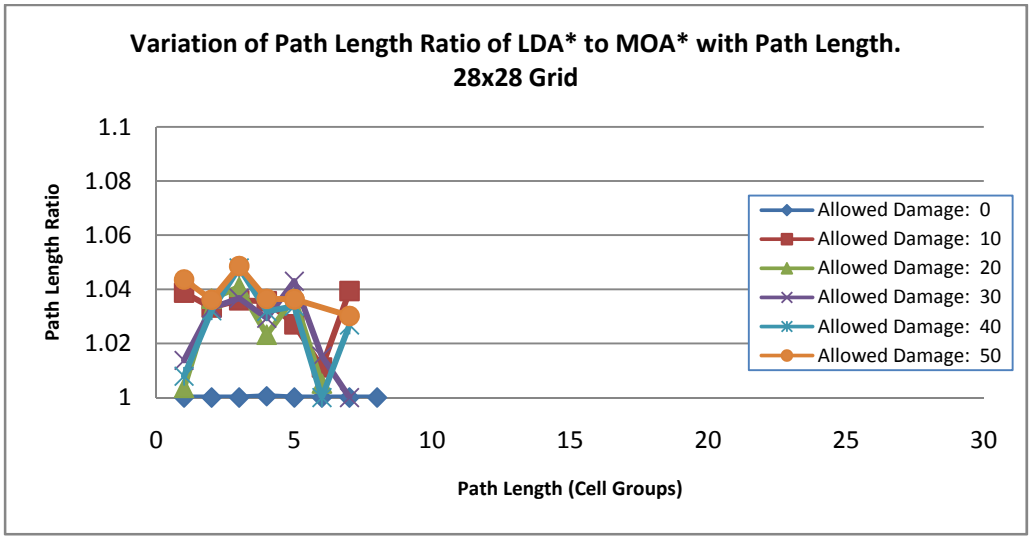


b.

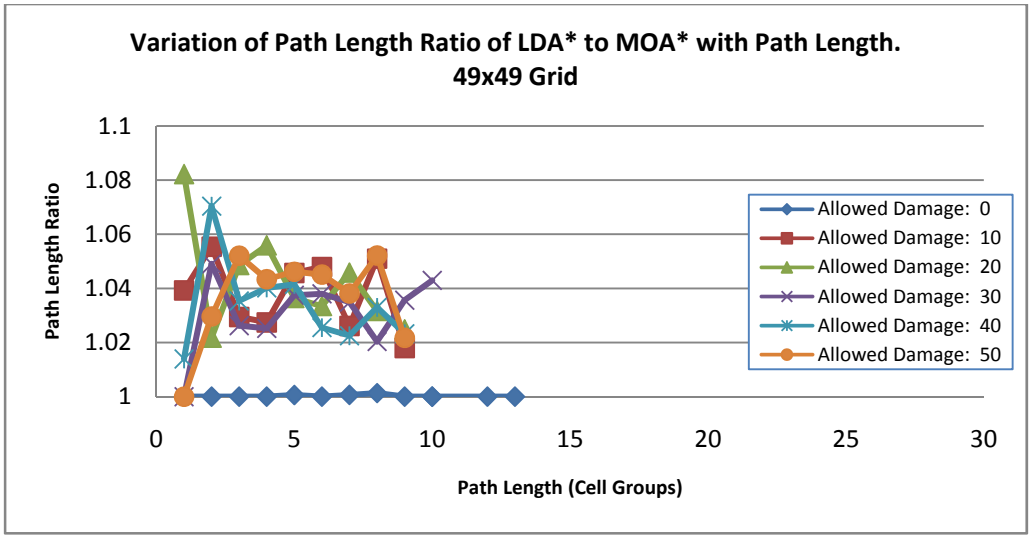


c.

Figure 4.12 (cont'd). Variation of CPU runtime ratio of LDA* to MOA* with path length, where LDA* DCF is 19 and DiCF is 0.5, and the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.

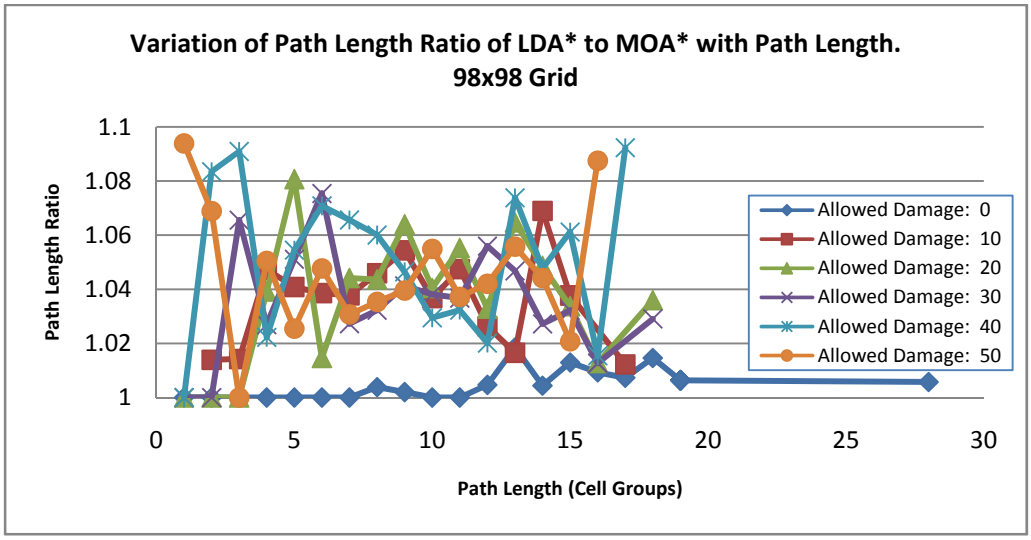


a.



b.

Figure 4.13. Variation of the path length ratio (solution path length of LDA* to shortest solution path length of MOA*) with path length, where LDA* DCF is 19 and DiCF is 0.5, and the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.



c.

Figure 4.13 (cont'd). Variation of the path length ratio (solution path length of LDA* to shortest solution path length of MOA*) with path length, where LDA* DCF is 19 and DiCF is 0.5, and the allowed damage varies from 0 to 50 in increments of 10, for a) 28x28 grids, b) 49x49 grids, and c) 98x98 grids.

4.3 SPECIAL CASE TESTS

LDA* performed well on the randomly generated test maps under the conditions defined in this chapter and in the previous chapter. However, presence of special cases in the test data is unknown, and therefore, no special data was obtained regarding the performance of LDA* in special conditions.

In order to measure the performance of LDA* in such conditions, four hand-crafted maps were prepared and tested using the algorithm, which utilized constant DiCF of 0.5, and varying DCF of 0 to 100000, as used in the preliminary test cases. MOA* was also run on the same setups, whose results were used as the optima. In these setups, MOA* is run twice, and LDA* is run for 5 times. During the data recording state, the values obtained from these runs

are averaged for LDA*. The test results and examples from generated solution paths are presented in this section. Threat source and agent characteristics used in these setups are identical to those used in the previous tests, unless otherwise is stated for a setup. The threat sources apply 1 HP of damage to the agent per 1 cell edge's length of distance, but their threat zones may be greater than those used in the previous tests.

The classical trap problem was selected as the first special case. The starting position in this case was put inside a large region, surrounded with blocks, having the exit from the region located away from the goal cell. No threat sources were used in this configuration. Consequently, the map shown in Figure 4.14 was prepared and tested. The shortest path solution of MOA*, the solution of LDA* are also shown in the same figure, and the test data are presented in Table 4.1. In this case, LDA* is able to find a path equivalent to the optimal, within 0.1% to 0.3% runtime of MOA*, and its runtimes are at most 8 milliseconds.

The second case has a setup, where all of the allowed damage points have to be spent in order to reach the unique solution with the shortest path. In this map, the agent is required to travel from the top-left corner of the map to the bottom-left corner. However, there is a wide threat zone in between these locations, which is slightly offset towards the center of the map. The radius of this threat zone is selected to be 200 meters (20 cells). In order to force the solution path to pass through the threat zone, a horizontal block is placed in the middle of the map that penetrates the threat zone, as shown in Figure 4.15.

Contrary to the preliminary tests, in case LDA* is unable to find a solution with the predefined allowed damage of 10, the allowed damage is not increased, since this case is aimed to measure the performance of LDA* under the given strict conditions. Numerical test results for this case are shown in Table 4.2.

Importance of DCF selection is once again observed from the test results, because only 2 solutions were generated for 37 different DCF values, the best being 9% longer than the optimal and having a runtime of 1.1% of MOA*. The second solution was 15% longer than the shortest path, and its runtime was about 0.5% of the MOA* run. The shortest path found by MOA*, and the best solution of LDA* are plotted in Figure 4.15. Since the best solution of LDA* is found when the value of DCF is 80, one can observe that the heuristic is governed by the damage part. This part takes into consideration the damage amount that the agent is expected to suffer by following the straight path from its location to the goal cell. As a result, the algorithm tries to find a location, from where the straight line drawn to the goal cell passes through outside of threat zones.

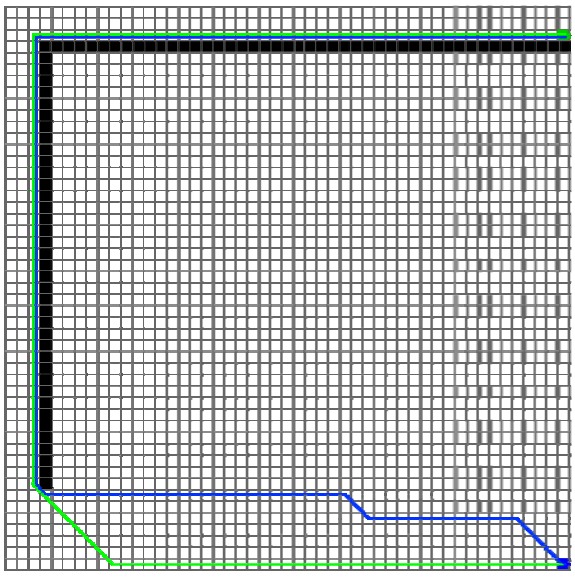


Figure 4.14. Trap map for case 1 with the start cell at the bottom right corner (blue), and the goal cell near the top right corner (green). Green path is the solution path of MOA* (shortest path), and the blue path is an equivalent solution by LDA*.

This behavior is observable in the final (bottom) half of the LDA*-generated path. After having traveled through the threat zone, the algorithm has chosen a path that would avoid the threat zone in the line of sight of the goal cell. Only after this point, the path actually approaches the goal cell. Although many runs of LDA* are run in order to reach a solution for this setup, the total time spent by LDA* is still less than 30% of the time required to run MOA* on the same setup. In addition, LDA* runs take very small amount of time as a maximum of 22 milliseconds.

The third special case is similar to the second case, in that it has a unique solution, and requires traveling through the far end of threat zones to reach the goal. However, there are two threat sources in this setup. The first one, which is closer to the start cell, is placed at one-third of the side of the grid, similar to the threat source in the second test case, and has an effective range radius of 100m (10 cells). The second threat source is placed towards the right side of the map, with an effective range radius of 103 meters. A horizontal blocking line is placed to the left of the second source, in order to force the algorithm to find a path that passes through the right side of this source. The described test setup is shown in Figure 4.16. The numerical test results for this case are also presented in Table 4.3.

According to the test results, LDA* was able to solve this case with high DCF values, which indicated a requirement for the dominance of damage over distance in the heuristic function. Since with increased dominance of damage, LDA* tries to travel around the threat zones instead of trying to pass through them, this was an expected behavior. In addition, the behavior of the algorithm about avoiding threat zones in the line of sight to the goal cell is also observable in the last portion of the LDA* path in Figure 4.16, where the path just leaves the second threat zone. Here, instead of trying to pass near the second threat zone diagonally in a leftward and downward manner, the path chooses to follow

the right border of the map, until a clear line of sight is established to the goal cell. Lengths of the paths generated by LDA* in this case are 5% longer than the optimum, and the runtime of the algorithm varies between 0.7% and 0.9% of that of MOA*. Like the previous setup, LDA* is run several times in order to reach a solution, along with many runs with no result. However, the total time spent by LDA* is still less than 20% of the runtime of MOA*, which indicates the performance of LDA*, compared to that of MOA*. LDA* runs take at most 42 milliseconds for this setup.

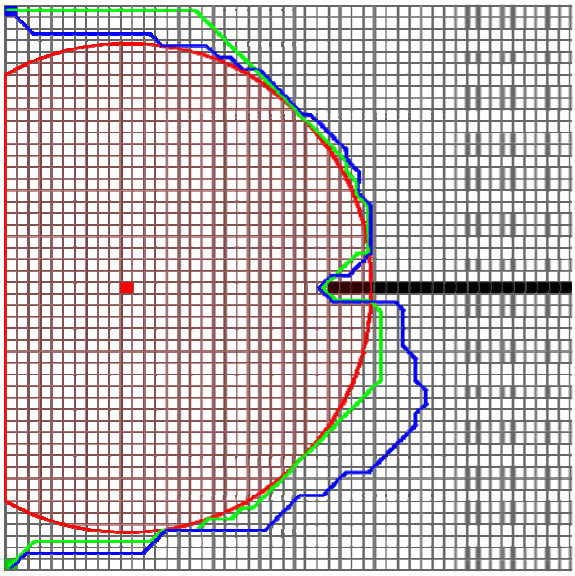


Figure 4.15. Map for case 2 with the start cell at the top left corner (blue), and the goal cell at the bottom left corner (green). Green path is the solution path of MOA* (shortest path), and the blue path is the best solution found by LDA*, with DCF = 80.

The last special case takes another strict case into consideration. In this case, the start cell is at the top left corner of the map, and the goal cell is hidden behind a threat source having a large radius of 200 meters and higher damage of 3 HP.

There is another threat source located below it, with a radius of 70 meters, and a regular damage characteristic of 1 HP. A horizontal blocking wall is placed to the upper right side of the map, where a path to the goal cell would otherwise pass. The configuration is shown in Figure 4.17, and the test results are presented in Table 4.4.

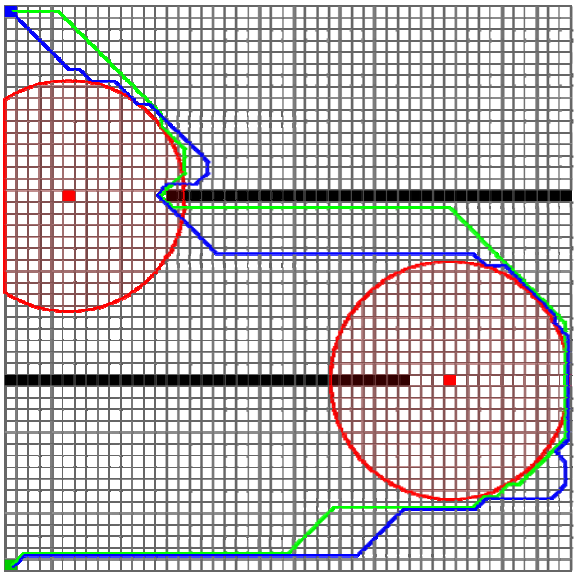


Figure 4.16. Map for case 3 with the start cell at the top left corner (blue), and the goal cell at the bottom left corner (green). Green path is the solution path of MOA* (shortest path), and the blue path is the best solution found by LDA*, with DCF = 1000, 10000 and 100000.

Like the previous two cases, this setup is a trap for a greedy heuristic, since in that case the algorithm would try to pass through the threat zone with higher damage, and fail. Again, there exists only one path to the goal cell, and it passes through the bottom cells of the map. The algorithm must maintain a no-damage condition until it reaches the bottom of the map, where it crosses the smaller threat zone. Otherwise, it would get stuck and be unable to yield any solutions.

Interestingly, LDA* is able to solve this case without any problems for all values of DCF. The optimal values, however, are observed when DCF is close to 0.0, where LDA* finds an equivalent path to the optimal solution. The ratio of the length of the paths generated by LDA* to the optimal path are at most 1.15, which indicate suboptimality up to the amount 15%. In this test setup, runtime of LDA* is in the range of 1% to 3.6% of that of MOA*, which indicate that LDA* would be a good choice, if the time available for the path search is limited to small values. The longest time period for this setup by LDA* is 22 milliseconds.

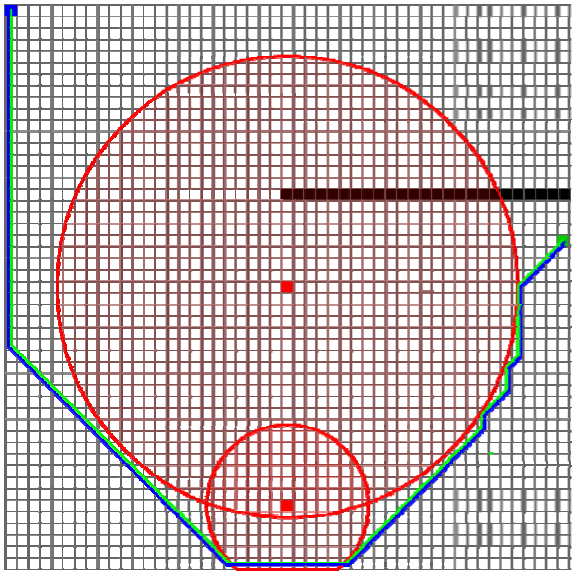


Figure 4.17. Map for case 4 with the start cell at the top left corner (blue), and the goal cell at the right edge (green). Green path is the shortest path obtained from MOA*, and the blue path is the best solution found by LDA*, with DCF = 0.0 through 0.1, and 2.

Table 4.1. Test results for special test case 1.

DCF	Case 1, Allowed Damage=10							
	LDA* Solution Damage	MOA* Solution Damage	LDA* Path Length	MOA* Shortest Path Length	LDA*/MOA* Path Length Ratio	LDA* Runtime (ms)	MOA* Runtime (ms)	LDA*/MOA* Runtime Ratio
0	0		1333.14		1.00	8.01		0.003
0.00001	0		1333.14		1.00	4		0.001
0.0001	0		1333.14		1.00	2		0.001
0.001	0		1333.14		1.00	4		0.001
0.01	0		1333.14		1.00	4		0.001
0.1	0		1333.14		1.00	6.01		0.002
1	0		1333.14		1.00	4		0.001
2	0		1333.14		1.00	6.01		0.002
3	0		1333.14		1.00	2		0.001
4	0		1333.14		1.00	2		0.001
5	0		1333.14		1.00	6.01		0.002
6	0		1333.14		1.00	6.01		0.002
7	0		1333.14		1.00	6.01		0.002
8	0		1333.14		1.00	2		0.001
9	0		1333.14		1.00	6.01		0.002
10	0		1333.14		1.00	2		0.001
11	0		1333.14		1.00	4		0.001
12	0		1333.14		1.00	4		0.001
13	0	0	1333.14	1333.14	1.00	4	2799.03	0.001
14	0		1333.14		1.00	4		0.001
15	0		1333.14		1.00	8.01		0.003
16	0		1333.14		1.00	2		0.001
17	0		1333.14		1.00	6.01		0.002
18	0		1333.14		1.00	2		0.001
19	0		1333.14		1.00	8.01		0.003
20	0		1333.14		1.00	4		0.001
30	0		1333.14		1.00	2		0.001
40	0		1333.14		1.00	4		0.001
50	0		1333.14		1.00	4		0.001
60	0		1333.14		1.00	6.01		0.002
70	0		1333.14		1.00	6.01		0.002
80	0		1333.14		1.00	6.01		0.002
90	0		1333.14		1.00	2		0.001
100	0		1333.14		1.00	4		0.001
1000	0		1333.14		1.00	8.01		0.003
10000	0		1333.14		1.00	4		0.001
100000	0		1333.14		1.00	4		0.001

Table 4.2. Test results for special test case 2.

DCF	Case 2, Allowed Damage=10							
	LDA* Solution Damage	MOA* Solution Damage	LDA* Path Length	MOA* Shortest Path Length	LDA*/MOA* Path Length Ratio	LDA* Runtime (ms)	MOA* Runtime (ms)	LDA*/MOA* Runtime Ratio
0	-		-		-	12.01		0.009
0.00001	-		-		-	8.01		0.006
0.0001	-		-		-	8.01		0.006
0.001	-		-		-	8.01		0.006
0.01	-		-		-	14.02		0.011
0.1	-		-		-	14.02		0.011
1	-		-		-	10.01		0.008
2	-		-		-	12.01		0.009
3	-		-		-	8.01		0.006
4	-		-		-	8.01		0.006
5	-		-		-	6.01		0.005
6	-		-		-	8.01		0.006
7	-		-		-	6.01		0.005
8	-		-		-	6.01		0.005
9	-		-		-	2		0.002
10	-		-		-	2		0.002
11	-		-		-	8.01		0.006
12	-		-		-	6.01		0.005
13	-	10	-	989.12	-	2	1301.88	0.002
14	-		-		-	8.01		0.006
15	-		-		-	8.01		0.006
16	-		-		-	6.01		0.005
17	-		-		-	8.01		0.006
18	-		-		-	6.01		0.005
19	10		1139.83		1.15	6.01		0.005
20	-		-		-	8.01		0.006
30	-		-		-	14.02		0.011
40	-		-		-	14.02		0.011
50	-		-		-	18.03		0.014
60	-		-		-	18.03		0.014
70	-		-		-	16.02		0.012
80	10		1080.83		1.09	14.02		0.011
90	-		-		-	22.03		0.017
100	-		-		-	20.03		0.015
1000	-		-		-	14.02		0.011
10000	-		-		-	12.01		0.009
100000	-		-		-	20.03		0.015

Table 4.3. Test results for special test case 3.

DCF	Case 3, Allowed Damage=10							
	LDA* Solution Damage	MOA* Solution Damage	LDA* Path Length	MOA* Shortest Path Length	LDA*/MOA* Path Length Ratio	LDA* Runtime (ms)	MOA* Runtime (ms)	LDA*/MOA* Runtime Ratio
0	-		-		-	18.03		0.005
0.00001	-		-		-	20.03		0.005
0.0001	-		-		-	16.02		0.004
0.001	-		-		-	24.03		0.006
0.01	-		-		-	18.02		0.005
0.1	-		-		-	26.04		0.007
1	-		-		-	8.01		0.002
2	-		-		-	12.01		0.003
3	-		-		-	18.03		0.005
4	-		-		-	22.03		0.006
5	-		-		-	14.02		0.004
6	-		-		-	14.02		0.004
7	-		-		-	30.04		0.008
8	-		-		-	20.03		0.005
9	-		-		-	24.03		0.006
10	-		-		-	10.01		0.003
11	-		-		-	10.01		0.003
12	-		-		-	8.01		0.002
13	-	10	-	1274.97	-	8.01	3870.57	0.002
14	-		-		-	8.01		0.002
15	-		-		-	12.02		0.003
16	-		-		-	22.03		0.006
17	-		-		-	10.01		0.003
18	-		-		-	10.01		0.003
19	-		-		-	12.01		0.003
20	-		-		-	14.02		0.004
30	-		-		-	26.04		0.007
40	-		-		-	30.04		0.008
50	-		-		-	30.04		0.008
60	-		-		-	40.03		0.010
70	-		-		-	18.02		0.005
80	-		-		-	20.03		0.005
90	-		-		-	42.06		0.011
100	-		-		-	36.05		0.009
1000	10		1334.97		1.05	36.05		0.009
10000	10		1334.97		1.05	28.04		0.007
100000	10		1334.97		1.05	32.05		0.008

Table 4.4. Test results for special test case 4.

DCF	Case 4, Allowed Damage=10							
	LDA* Solution Damage	MOA* Solution Damage	LDA* Path Length	MOA* Shortest Path Length	LDA*/MOA* Path Length Ratio	LDA* Runtime (ms)	MOA* Runtime (ms)	LDA*/MOA* Runtime Ratio
0	10		1017.4		1.00	18.03		0.029
0.0000	10		1017.4		1.00	20.03		0.033
0.0001	10		1017.4		1.00	12.02		0.020
0.001	10		1017.4		1.00	16.02		0.026
0.01	10		1017.4		1.00	10.01		0.016
0.1	10		1017.4		1.00	14.02		0.023
1	10		1029.12		1.01	14.02		0.023
2	10		1017.4		1.00	8.01		0.013
3	10		1023.26		1.01	8.01		0.013
4	10		1052.55		1.03	8.01		0.013
5	10		1040.83		1.02	16.02		0.026
6	10		1034.97		1.02	18.03		0.029
7	10		1052.55		1.03	6.01		0.010
8	10		1052.55		1.03	8.01		0.013
9	10		1052.55		1.03	12.01		0.020
10	10		1052.55		1.03	14.02		0.023
11	10		1052.55		1.03	14.02		0.023
12	10		1054.97		1.04	10.01		0.016
13	10	10	1066.69	1017.4	1.05	12.01	615.89	0.020
14	10		1054.97		1.04	12.02		0.020
15	10		1058.41		1.04	8.01		0.013
16	10		1052.55		1.03	10.01		0.016
17	10		1052.55		1.03	10.01		0.016
18	10		1052.55		1.03	8.01		0.013
19	10		1049.12		1.03	12.01		0.020
20	10		1052.55		1.03	18.03		0.029
30	10		1074.97		1.06	12.01		0.020
40	10		1060.83		1.04	14.02		0.023
50	10		1052.55		1.03	14.02		0.023
60	10		1052.55		1.03	14.02		0.023
70	10		1052.55		1.03	14.02		0.023
80	10		1052.55		1.03	10.01		0.016
90	10		1052.55		1.03	14.02		0.023
100	10		1052.55		1.03	18.03		0.029
1000	10		1174.97		1.15	22.03		0.036
10000	10		1174.97		1.15	20.03		0.033
100000	10		1174.97		1.15	16.02		0.026

CHAPTER 5

CONCLUSION

Pathfinding is a common problem of many computer games, and the current path search algorithms used in computer games consider the path length as the only optimality criterion. In this study, the amount of damage taken during the path execution was considered as the second criterion along with the path length. Consequently, an A*-based heuristic path search algorithm was developed in order to be used in pathfinding problems, where a static and fully-observable maze-like grid environment involved non-moving sources of threat that could inflict damage on the agent traveling through their areas of effect (threat zones, or ranges). Here, damage and agent's health were abstracted as numeric values.

This algorithm aimed to generate a suboptimal solution (a path from a starting location to a goal location), which would have a path length close (or equal) to the shortest path that would cause an agent a traversing that path to suffer an amount of damage less than or equal to the predefined damage limit. Thus, the algorithm was called as Limited-Damage A* (LDA*).

Three different sizes of 28x28, 49x49 and 98x98 cells were selected as the grid sizes of the environment to be used in testing the performance of the algorithm. In the first set of tests, 200, 200, and 100 runs were performed on small, medium-sized, and large maps, respectively. In these tests, Multiobjective A*

(MOA*) was run on randomly-generated maps in order to determine the optimal non-dominating solutions of a case. Later, LDA* was run on the same configuration, with the damage limit set equal to the minimum value obtained from MOA*, in order to observe the performance of LDA* in unique-solution cases. Moreover, in order to examine their effects in the performance of the algorithm, various values of the heuristic contribution factors of damage (DCF) and distance (DiCF) were utilized for each of the above configurations. Since LDA* is not optimal, CPU runtime and solution path length values were considered as the performance criteria in this set of tests, and they were compared to those of the relevant exact solution's computed by the Multiobjective A* (MOA*) algorithm. In addition, the worst case rate of failure (to find a solution) for a given setup was measured, since the LDA* is not complete.

LDA* was observed to generate paths being less than 3%, 4% and 6% longer than the length of the exact solutions', for small, medium and large maps, respectively. It was observed to run faster than 2% of the runtime of MOA* on average. Finally, the ability of the algorithm to find the unique solution for a given configuration was measured to range between 88-95% for small maps, between 85-89% for medium-sized maps, and between 71-89% for the large maps, which varied with both DCF and DiCF. Optimal values for DCF and DiCF for the defined environments were observed to be 19 and 0.5, respectively.

The best DCF and DiCF values obtained from the first set of tests were used in the second set of tests, which were performed to measure the performance of LDA* with varying damage limits and path lengths in randomly generated configurations. LDA* was observed to run at less than 4%, 8% and 13% of the relevant MOA* CPU runtimes, for small, medium-sized, and large maps, respectively, generating paths shorter than 5%, 8% and 10% more in length of

those of MOA*'s. Average success rate of LDA* in such cases was observed to be greater than 99.8% considering all of the second set of tests.

The third set of tests was composed of hand-crafted maps, which involved typical traps for the path search algorithms. Behavior of LDA* in such cases was observed with changing DCF, and shortest paths generated by LDA* were plotted against the optimal solutions for comparison.

Examining the results of the performed tests, LDA* is found to have impressively small runtime values, especially when compared to those of MOA*'s. Moreover, the average failure rate of the algorithm is determined to be very small (<0.2%) for random configurations, which do not necessarily have a unique solution. Path lengths of the solutions are observed to be approximately 10% longer than the optimal ones at maximum. Considering these factors, LDA* can be classified as a successful non-optimal and non-complete heuristic path search algorithm for environments having similar parameters to those defined for the test cases. Further research is required in order to improve its completeness property.

LDA* runtimes exceed 10-20 milliseconds for medium and large map sizes, and therefore the algorithm lacks the pace required for real-time applications such as real-time computer games, for large environments. However, it can be used effectively in small environments or when utilized in a multi-level hierarchical pathfinding scheme. Especially hierarchical pathfinding that can guide the search towards more exploration is expected to improve the success rate of LDA*'s finding a solution in unique-solution cases

On the other hand, LDA* can be utilized in games that do not have very small and limited runtime requirements, such as turn-based games, with slight or no scheme modifications. Other offline applications that have similar abstracted

environments such as military route planning applications may also make use of LDA*, either naïvely, or with appropriate modifications.

The problem of multiobjective pathfinding that incorporates expected (or projected) damage into path search is a complex issue. Many problems that have been researched in detail for single-objective path search, such as environmental dynamism, partial observability, hierarchical path search and real-time algorithmic pace are among the fields that are still open for research, when multiobjective pathfinding is of concern.

Consequently, researchers are highly encouraged to apply the basic principles of LDA* into incremental and/or hierarchical approaches in order to the algorithm to be able to handle partially observable and dynamic environments in real-time; mobile threat sources, mobile targets and dynamic gates, more realistic threat source attacks being among the properties of such environments. Moreover, utilization of non-uniform grid abstractions instead of uniform grids is also highly encouraged, since such an improvement would both shrink the search space, and contribute to the performance of the algorithm drastically.

REFERENCES

[1] E. W. Dijkstra. *A note on two problems in connection with graphs*. Numerische Mathematik, 1, 1959, page 269--271.

[2] Hart P. E., Nilsson N. J., Raphael B., *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions on System Science and Cybernetics, 4, 100-107, 1968.

[3] Russell S, Norvig P (1995) *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey

[4] Stentz, A. 1995. *Optimal and efficient path planning for unknown and dynamic environments*. International Journal of Robotics and Automation 10(3).

[5] Koenig, S. and Likhachev, M. *D* Lite*. In Proceedings of the National Conference on Artificial Intelligence, 2002.

[6] Stentz, A. *The Focussed D* Algorithm for Real-Time Replanning*. In Proceedings of Int'l Joint Conf. on Artificial Intelligence, August, 1995.

[7] D. Demyen and M. Buro, *Efficient Triangulation-Based Pathfinding.* Proceedings of the AAAI conference, Boston 2006, pp.942-947

[8] Koenig, S. and Likhachev, M. *Improved Fast Replanning For Robot Navigation in Unknown Terrain*. In Proceedings of the Int'l Conf. on Robotics and Automation, 2002.

[9] Korf R. E., *Real-time heuristic search*, Artificial Intelligence, 42, 189-211, 1990.

[10] Undeger, C., Polat, F. and Ipekkan, Z. *Real-Time Edge Follow: A New Paradigm to Real-Time Path Search*. In Proceedings of GAME-ON 2001, London, England, 2001.

[11] Undeger, C., Polat, F., *RTTES: Real-time search in dynamic environments,* Applied Intelligence, v.27 n.2, p.113-129, October 2007

[12] Bradley S. *Stewart and Chelsea C*. White III. *Multiobjective A** . Journal of the Association for Computing Machinery, 38:775--814, 1991.

[13] Qu, Y., Pan, Q., Yan, J., *Flight path planning of UAV based on heuristically search and genetic algorithms*, Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE , pp. 5, 6-10 Nov. 2005.

[14] Hallam, C., Harrison, K.J., andWard, J.A. (2001). *A multiobjective optimal path algorithm*. Digital Signal Processing, 11, 133–143.

[15] R. E. Korf. *Depth-first iterative-deepening: An optimal admissible tree search*. Artificial Intelligence, 27:97–109, 1985.

[16] P. P. Chakrabarti, S. Ghose, A. Acharya, and S. C. de Sarkar, *Heuristic Search in Restricted Memory*, Artificial Intelligence, vol. 41, pp. 197-221, 1989.

[17] R. E. Korf. *Linear-space best-first search*. Artificial Intelligence, 62(1):41–78, 1993.