Knowledge-Based Systems 24 (2011) 501–512

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Limited-Damage A*: A path search algorithm that considers damage as a feasibility criterion

Serhat Bayili [a], Faruk Polat [b,*]

[a] Zibumi Game Studio, Galyum Block, No. 15/4, ODTU Teknokent, 06531 Ankara, Turkey
[b] Department of Computer Engineering, Middle East Technical University, 06531 Ankara, Turkey

ARTICLE INFO

Article history:
Received 9 October 2009
Received in revised form 23 December 2010
Accepted 23 December 2010
Available online 1 January 2011

Keywords:
Agent search
Path search in virtual environments
Offline path planning
Heuristic search
Multiobjective search

ABSTRACT

Pathfinding algorithms used in todays computer games consider the 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 based on the A* algorithm was developed, which takes the threat zones in the game map into consideration. Given an upper limit as the tolerable amount of damage for an agent, this algorithm searches for the shortest path from a starting location to a destination, where the agent may suffer damage less than or equal to the specified limit. Due to its behavior, the algorithm is called Limited-Damage A* (LDA*). Performance of LDA* was tested in randomly-generated maze-like grid-based environments of varying sizes, and in hand-crafted fully-observable environments, in which 8-way movement is utilized. Results obtained from LDA* are compared with those obtained from Multiobjective A* (MOA*), which is a complete and optimal algorithm that yields exact (best) solutions for every case. LDA* was found to perform much faster than MOA*, yielding acceptable sub-optimality in path length.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Path planning (or path search) problem is one of the common problems in computer games, where an agent is required to find a sequenced set of state transitions (solution path) for itself from its current location to a destination on a map, or determine nonexistence of such a sequence. However, a solution path in such problems is required to satisfy certain optimality conditions, such as having the minimal traveling cost (path length, fuel spent, travel time, etc.).

There are many offline and real-time algorithms that can solve path search problems by taking path length as the only optimality criterion for the search. A* algorithm [1] is a well-known example of such offline algorithms, and the family of D* algorithms [2–4], RTA*, LRTA* [5], RTEF [6] and RTTES [7] are examples of incremental/real-time algorithms.

In case there are no threats on the environment (map) that may prevent the agent from reaching its goal position, these algorithms are adequate. However, this is not the case in most computer games, since almost always there exist opposing parties that aim to destroy each others agents [8]. Considering the importance of safe navigation of the agent to its goal position, the shortest path may not always be a feasible one. Therefore, damage considerations should be integrated into the path searching task.

As a result, an A*-based algorithm that finds a suboptimal path between a given starting position and a goal position within a predefined upper amount of damage (to the agent) was devised, and called Limited-Damage A* algorithm (LDA*). Its performance was tested against the complete and optimal reference algorithm Multiobjective A* (MOA*) [9] in randomly generated maps of three different sizes, in hand-crafted case study maps and the results were reported. The following sections are organized as follows: Section 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 Section 3. Test cases and their results are given in Section 4. Finally, the discussion of results, and the conclusions are provided in Section 5.

2. Background and related work

Currently, there are many proposed path search algorithms, having different algorithmic complexities. Depending on their runtime characteristics, they can be 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 to run, and generate total (and usually optimal) solutions. Dijkstra's algorithm [10], A* algorithm [1] and all uninformed graph search algorithms such as Depth-First Search, Breadth-First Search, and Iterative-Deepening

* Corresponding author.
E-mail address: polat@ceng.metu.edu.tr (F. Polat).

0950-7051/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2010.12.009

Search [11], etc. are among the examples of offline path search algorithms. On the other hand, the category of real-time algorithms have limited execution periods for each session (usually in the order of a few milliseconds), and execute for a number of discrete sessions in order to generate solutions and optimize them. These algorithms can be grouped into two as incremental and non-incremental, depending on their characteristics in generating solutions.

Algorithms in the incremental group are those that aim to generate an initial sub-optimal path in the very first run session (s), and repair it during the consequent sessions to make it closer to the optimal. The family of D* algorithms, such as D* [2], D*-lite[3], Focussed D*[4], TA* [12] and Focussed Dynamic A* Lite[13] are among the well-known incremental real-time path search algorithms.

Non-incremental algorithms are those, which aim to generate a partial solution by considering only the short-term available choices. In other words, they try to generate only the part of a solution path that is closest to the agents location. Learning Real-Time A* (LRTA*) and Real-Time A* (RTA*) [5], Real-Time Edge Follow [6] and Real-Time Target Evaluation Search [7] 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 in terms of path length.

Other than possible execution time limitations, space (memory) complexity of a path search algorithm can also be a problem in large environments. Some algorithms like Iterative-Deepening A*[14], Memory-Bounded A* [15] and Recursive Best-First Search [16] are developed in order to cope with this problem.

The above algorithms consider the path length as the only optimality criterion. However, there can be more than one optimality criteria in a search. As a solution to such problems, the Multiobjective A* (MOA*) Algorithm [9] was developed, based on similar principles to A*. Like A*, it is complete and optimal, when used with an admissible heuristic function. There are also other multi-objective path planning algorithms that are focussed on the safe navigation of agents within territories involving detectors [17,18]. 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.

3. Proposed algorithm

In most computer games such as real-time strategy games [19,20], the problem of determining a path that is the shortest and the safest one between two points in a map, where there exist threat sources that can harm the traveling agent, is a complex one. It cannot be solved within the allowed path search algorithm execution time limit of such a computer game, where the execution time is usually in the order of a few ten milliseconds. Consequently, an algorithm that can generate a solution approximate to the exact solution within the time limitation is needed for such games, if the agents are required to move autonomously within threat zones.

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 represent a solution to this problem. This special case defines a maximum limit to the amount of damage that an agent is allowed to take, and asks for the shortest path within this damage limitation.

In the presence of a feasibility criterion (damage limit), path length optimality does not guarantee a valid solution. Therefore, simple A* is unable to generate solution to the problem at hand, and an algorithm incorporating path feasibility into length-minimization is required.

LDA* does not consider threat zones as obstacle objects, but as passable regions. However an agent traveling through such a zone would suffer damage depending on the damaging characteristics of the threat source and the amount of time spent within the threat zone. Therefore, LDA* tries to find a sub-optimal path, on which the agent would suffer damage within the limits. In doing so, it utilizes a path repair mechanism, which is not present in regular A*. During the search, this mechanism favors a safer (less-damage inflicting) path to a cell in exploration, if there exists any. As a result of this mechanism, LDA* is able to discover low-damage paths that generate suboptimal path lengths.

Thus, the proposed A*-based algorithm utilizes a combination of the path length and damage taken from traveling along that path as the path cost, and determines the shortest path between a starting position and a destination within a given damage allowance for an agent. Therefore, it is called Limited-Damage A* algorithm (LDA*). The proposed algorithm can be used for path planning in computer games, robotics and virtual simulations.

3.1. 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 (map). A grid-based map is also easier to implement, compared to irregular polygon-based maps, and is commonly used among the path search algorithms.

Length and time units in the environment are chosen to be meters and seconds, respectively. Dimensions of the map are defined by the number of rows and columns it has, and the length of a single cells edge. Edges of the map are closed, thus traveling out of the map is not allowed. A cell in this environment can be empty, blocked, or occupied by a threat source. Only the empty cells are assumed to be traversable by the agent. Without loss of generality, the size of a cell, is chosen to be 10 m, which is assumed to be greater than the negligible size of the traveling agent.

The agent is assumed to have a certain value of initial health (hit points, HP), which is chosen to be 100. It is assumed to travel at predefined constant speed of 10 m/s, without acceleration and without any time delays for turning, stopping, etc. The agent is allowed to travel to adjacent empty cells in primary 8-directions, as seen in Fig. 1. It is not allowed to enter the blocked cells or the cells occupied by threat sources. The agent is assumed to travel between adjacent cells along the line segment connecting the centers of those cells.

A threat source is assumed to be stationary and to occupy an empty cell. It is assumed to have a predefined and constant circular area of effect (threat zone), where it can damage the agents. It is assumed to behave like the turret structures in strategy games, which fire bullets or rockets to the enemy agents within their ranges. Such structures have a non-negative initial targeting time delay after the entry of an agent to the threat zone and a positive

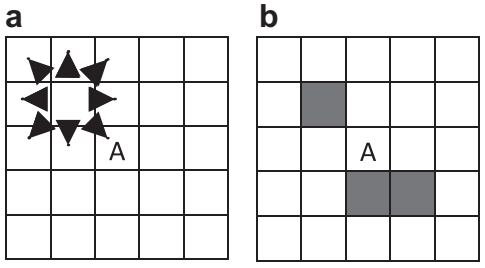

Fig. 1. Valid movements for the agent: a) 8-direction movement between empty cells, and b) a mixed configuration. "A" denotes the agent's position.

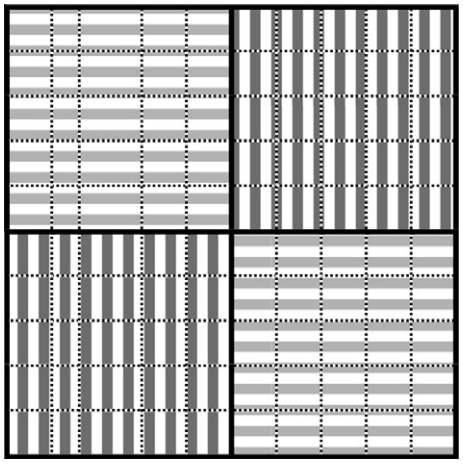

Fig. 2. Representation of a map in 2×2 cell groups, each consisting of 5×5 cell grids.

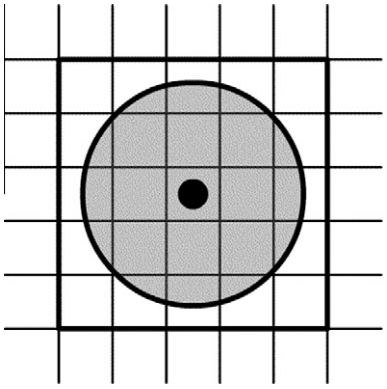

Fig. 3. Representation of a threat source in a 5×5 cell group and its area of effect. An agent passing through the middle cells in the edges would take damage.

time delay between two consecutive shots. These delay values are chosen to be 0 s and 1 s for the conducted experiments, respectively. All shots are assumed to be successful; i.e. any shot damages the agent. A constant amount of damage is applied to the agent by a single shot, which is chosen to be 1 HP of damage for the experiments. Damage caused by a shot from the threat source is applied to the agent instantaneously. Therefore, the agent takes damage, if it resides within a threat zone for more than the initial targeting delay of that threat source.

Maps used in the experiments are chosen to be square, having equal number of rows and columns. Maps are uniformly composed of square sub-regions called "cell groups." A cell group is composed of an odd number of cells on each side. An example is shown in Fig. 2 for a map consisting of 2×2 cell groups, where the cell group width is 5 (thus it is a 5×5 grid). A cell group is assumed to be empty with a probability of 0.5, has a threat source at its center cell with a probability of 0.3, and involves blocking cells with a probability of 0.2.

If the cell group has a threat source, then the radius of the threat zone of the threat source is chosen to be 1 m longer than the center-to-center distance between the center cell of that cell group and an edge cell in the middle of its edge, as seen in Fig. 3. The additional 1 m of distance is utilized so that an agent passing through the edge of a cell group would always enter the area of effect of a threat source, and take some damage.

The second option for a cell group is to involve blocked cells. Configuration of blocked 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 traversable regions among the blocked cells. Therefore, three types of blocked cell configurations are used for cell groups, each having equal probability of existence. The first is a plus-shaped one, which prevents horizontal and vertical movement through that cell group. The second and the third types are horizontal and vertical dash-shaped ones, which prevent vertical and horizontal travel through that cell group, respectively. The types of cell blocks are shown in Fig. 4.

3.2. Details of LDA* Algorithm

With the introduction of a limit in the damage parameter, the classical path length minimization problem turns into a problem, where the goal is to determine the minimum-length path that satisfies the damage limit criterion. This new problem cannot be solved using regular A*. However, with some modifications, A* can be an appropriate algorithm that can generate approximate solutions, running much faster than an optimal and complete path search 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*) respectively, according to the type of values they affect. Thus, $h(x)$ becomes:

$$h(x) = DiCF \times h_{di}(x) + DCF \times h_d(x) \tag{1}$$

The distance heuristic estimate, $h_{di}(x)$, is chosen to be calculated as the length of the line segment from the center of the cell in consideration to that of the goal cell. On the other hand, damage heuristic estimate is determined by calculating the amount of damage the agent would suffer if it traveled along the same line segment. Characteristic values of each threat source (initial shot delay, shot-to-shot delay, and damage of single shot) and the agents navigation speed are used in this calculation. Fig. 5 illustrates this concept on an example configuration, where the cell marked with "S" is the starting position, the one marked with "A" is the currently explored cell by the algorithm, and the cell marked with "G" is the goal position. In this case, the distance heuristic estimate is the length of the line segment between "A" and "G," and the damage heuristic estimate is the amount of damage the agent would take, if it traveled on this line segment. Note that the solid lines on the segment indicate parts of the path, where the agent would take damage. Along with the heuristic estimates, previously 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 $g(x)$ as in A*, in addition to the total distance traveled in threat zones for each candidate path. Since the suffered damage is not an optimization parameter in the problem, it is not added into the total path cost estimate, $f(x)$, yet it is stored separately for each candidate path to serve as a legality criterion; i.e. if the total amount of accumulated damage for the state in consideration exceeds the allowable value, then that path cannot be a solution. Therefore, the total path cost estimate $f(x)$ involves the previously traveled path length $g(x)$ and the heuristic estimate composed of the expected path length and the contribution of expected damage on that path, as shown in Eq. (2). Paths exceeding the allowable damage limit are pruned from the search tree.

$$f(x) = g(x) + h(x) \tag{2}$$

where, $h(x)$ is as defined in Eq. (1).

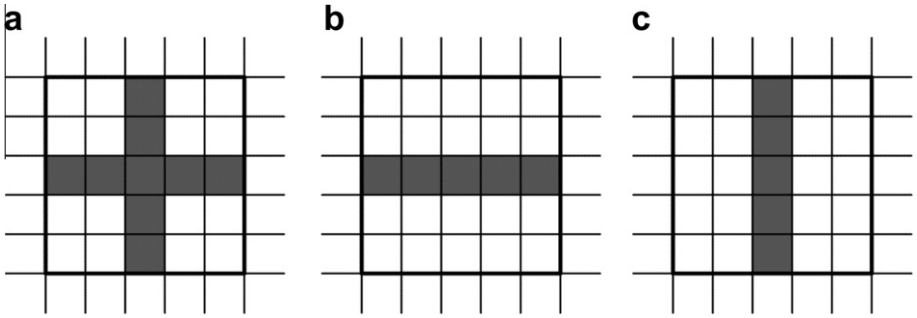

Fig. 4. Types of cell groups that involve blocked cells.

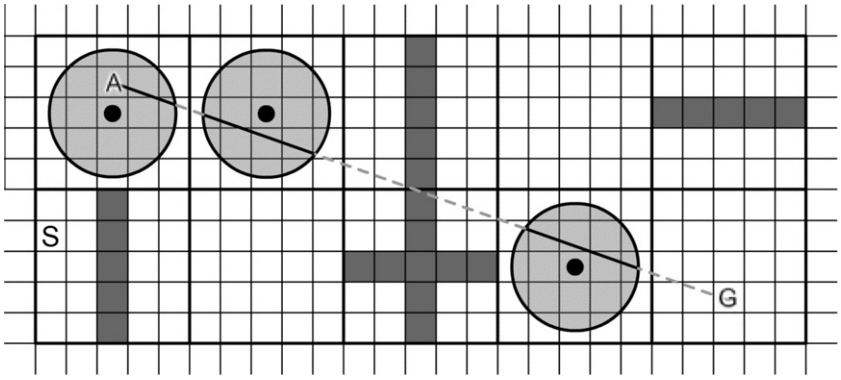

Fig. 5. Center-to-center distance from the current cell explored by the algorithm (A) to the goal cell (G) (gray dashed line from A to G). Length of line segment AG and the segments that lie within threat zones (solid black lines) are used in heuristic estimate calculation.

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). Making the exploration from a configuration with less damage 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 (as in A*) and its children are being explored. It checks if the child to be explored was previously explored, 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-smaller traveled path length", or with "smaller traveled path length 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 calculations. 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 section along with the results of experimental tests.

4. Experimental results

The algorithm presented in this work can be applied to various environments, each having different properties. In order to achieve better efficiency from the algorithm, its parameters must be fine tuned in accordance with the environmental parameters. Yet, 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, made of square cell groups having side lengths of 7 cells. Properties of the cell groups are as defined in Section 3. Three different map sizes of 28×28, 49×49 and 98×98 cells were used in the experiments to observe the performance of the algorithm with increasing search space. Memory requirement of MOA* had been the determining factor on the selection of map sizes used in the experiments. During the experiments, 98x98 grids were observed to occupy 200 MB to 300 MB of RAM, and 147×147 grids were unable to be solved with 768 MB of allocated as Java Virtual Machine memory. Consequently, the biggest grid size was chosen to be 98x98 for the tests, considering the software and hardware limitations. In all tests, performance of LDA* was compared to that of the reference values of MOA*.

A notebook computer, with 1.6 GHz Intel Dothan CPU and 1 GB RAM was used for the tests. The CPU clock was set to constant 1.6 GHz frequency, and no other programs were run in the background during the tests. Both LDA* and MOA* algorithms were implemented in Java 1.5 environment, and were run in Borlands "Turbo JBuilder 2007" integrated development environment under Windows XP SP2 operating system. Java virtual machines memory was set to 512 MB for all tests.

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 maps defined above, in order to observe the best combination, and continue the rest of the analyses with those values.

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 (distance or damage) on the whole.

The test maps were generated randomly as explained above. Start and goal positions were selected at random among the empty cells, where the two 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 LDA* was re-run. 400 different cases were tested for 28×28, 200 for 49×49 maps, and 100 cases for the 98×98 maps.

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. Fig. 6 shows the variation of failure rate of LDA* with increasing DCF, for different DiCF values, for different map 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 maps (28×28), 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 maps (49×49), there is a slight increase in the maximum failure rate compared to the smaller maps. Failure rate is 15% and it is observed for the configurations, where DiCF is 1.0, and DCF is 0.0.

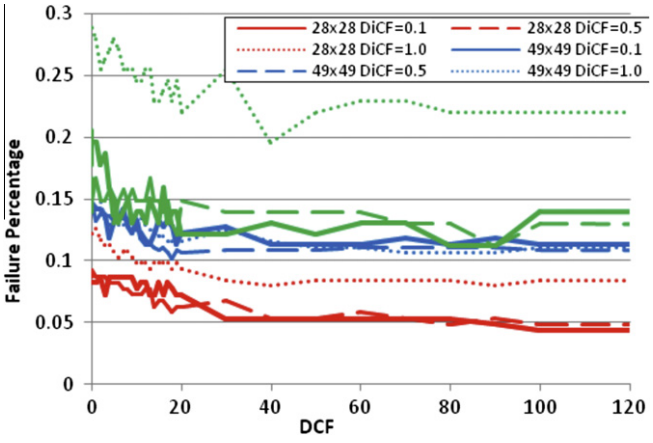

Fig. 6. Failure percentage in preliminary tests for unique-solution cases.

The minimum failure rate tends to decrease slightly, down to 11% in such configurations. For the large maps (98×98), 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 maps 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 Figs. 7a-c 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 maps. This value varies up to 4% for the medium-sized maps and to 6% for the large maps, 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 Figs. 8a-c. 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 Figs. 8a-c, and as expected, relative LDA* runtimes tend to decrease with increasing map size. Moreover, for a given map 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, solutions 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 subsection.

4.2. Performance tests

These tests are aimed to measure the path length and runtime performance of LDA*, compared to those of the reference algorithm, MOA*. In these tests, the same environmental parameters are used as in the previous subsection. There is no algorithm designed for path finding that considers multiple criteria for optimization other than the Multi-Objective A* (MOA). In order to compare the solution quality and the efficiency of our algorithm we compared our algorithm with MOA* which guarantees to find optimal solutions requiring substantially more time.

The tests were run on randomly generated maps, 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

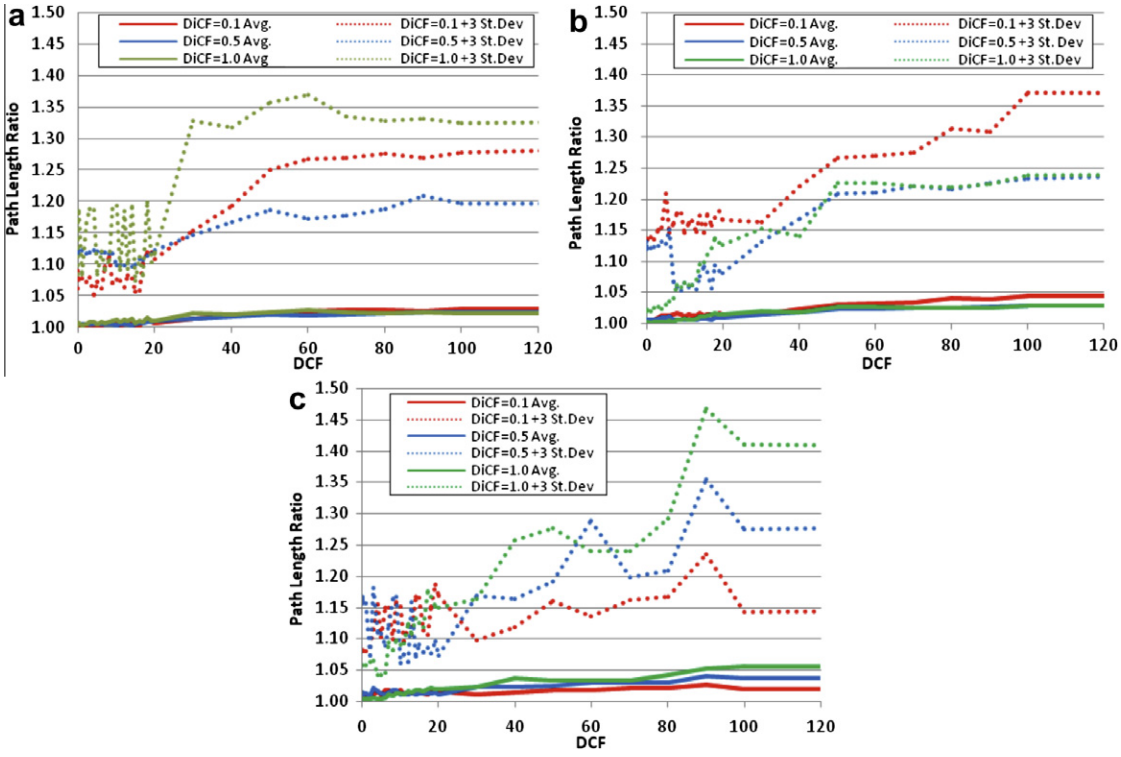

Fig. 7. Variation of path length ratio for 28 × 28 maps (a), 49 × 49 maps (b) and 98 × 98 maps (c).

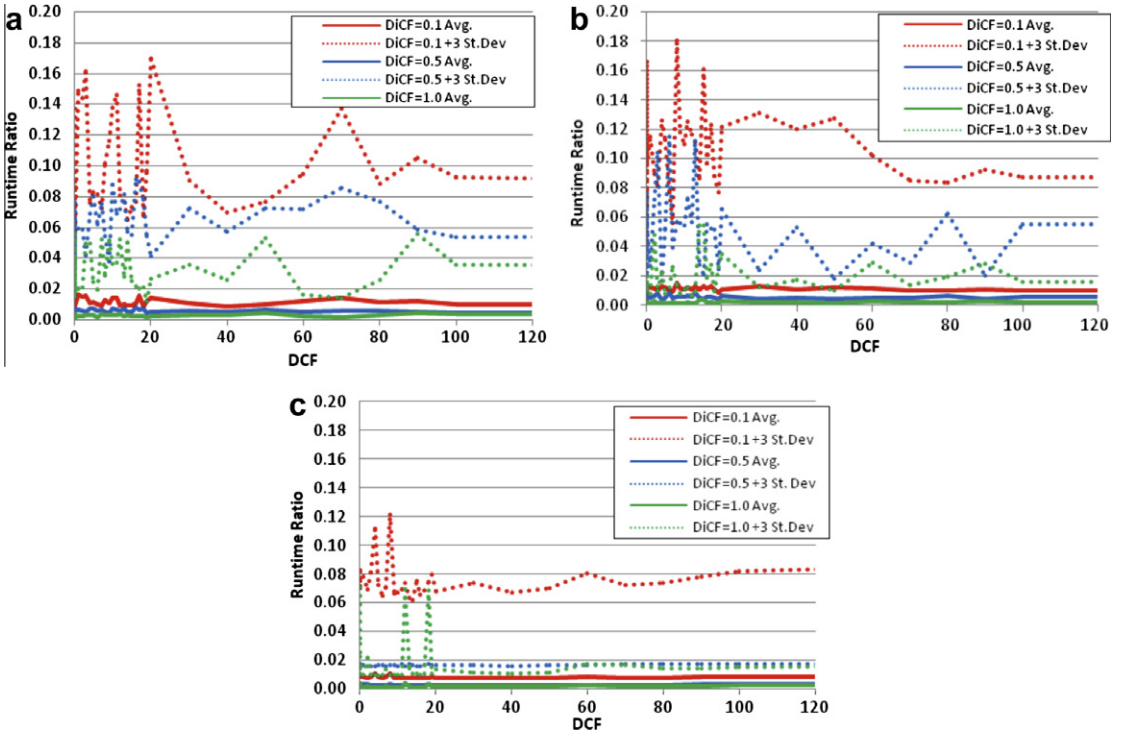

Fig. 8. Variation of runtime ratio of LDA* to MOA* with DCF for 28 × 28 maps (a), 49 × 49 maps (b) and 98 × 98 maps (c).

varied from zero to 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

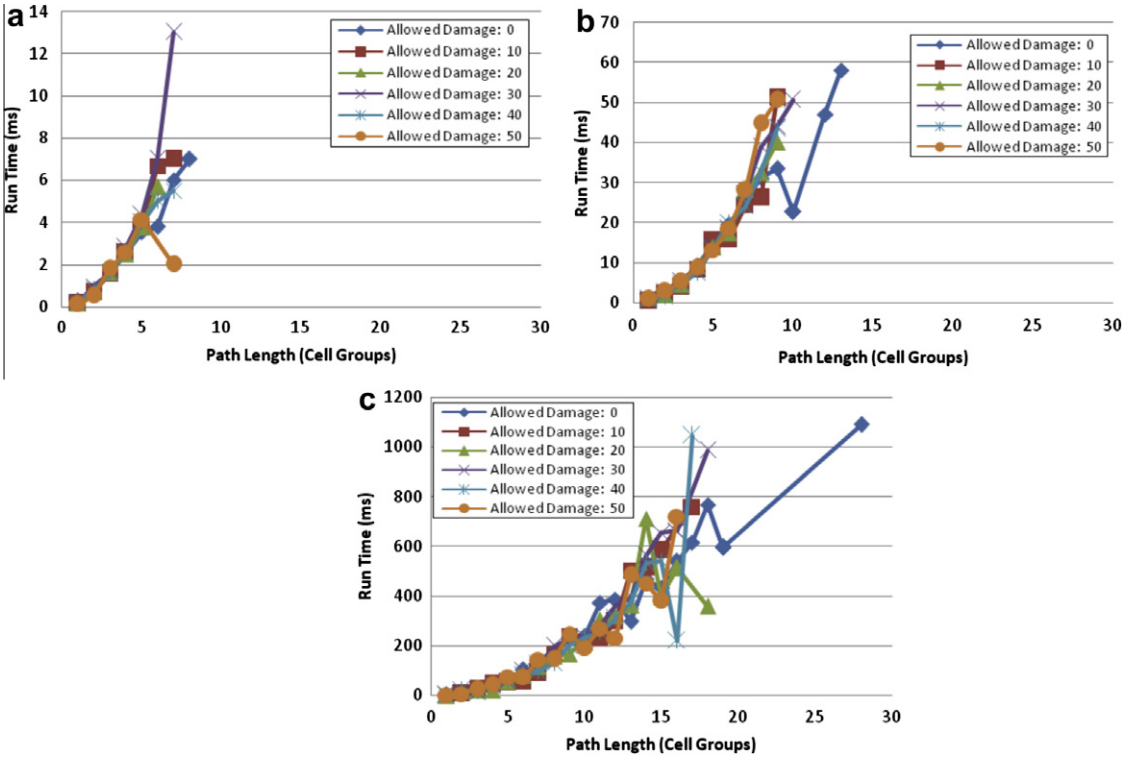

Fig. 9. Variation of runtime of LDA* with path length for 28 × 28 maps (a), 49 × 49 maps (b) and 98 × 98 maps (c).

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, ratio of LDA* to MOA*, and path length ratio of LDA* to MOA*.

Figs. 9a-c show the variation of LDA* runtime with increasing distance between start and goal cells for 28x28, 49x49 and 98x98 maps, 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 Figs. 9a-c 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 Figs. 10a-c. 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 Figs. 10b-c. 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 Figs. 11a-c. LDA* is observed to run much faster than MOA*, and time gained by utilizing LDA* in a similar problem instead of MOA* is found to be 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 values are even smaller 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 solutions path length found by LDA* to the shortest path found by MOA* is plotted in Figs. 12a-c. The results indicate that the paths found by LDA* are no more than 5% longer than the shortest path for small maps, about 8% for the medium-sized maps, and 10% for the large maps. Moreover, when

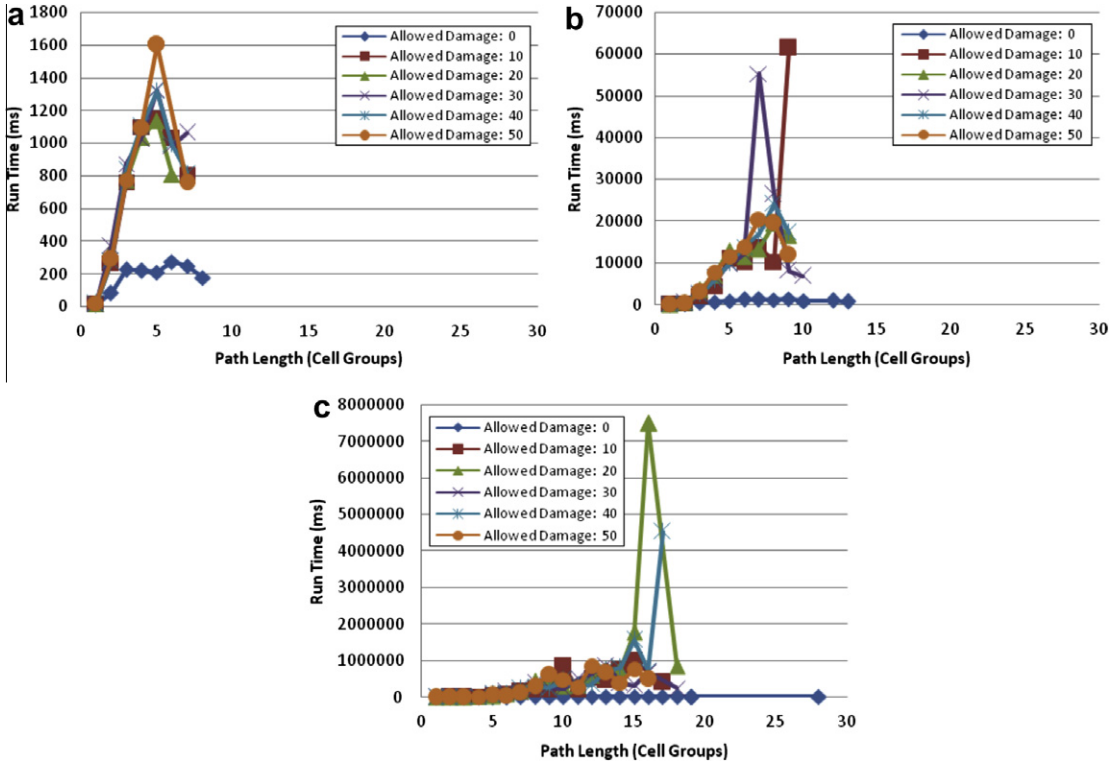

Fig. 10. Variation of runtime of MOA* with path length for 28 × 28 maps (a), 49 × 49 maps (b) and 98 × 98 maps (c).

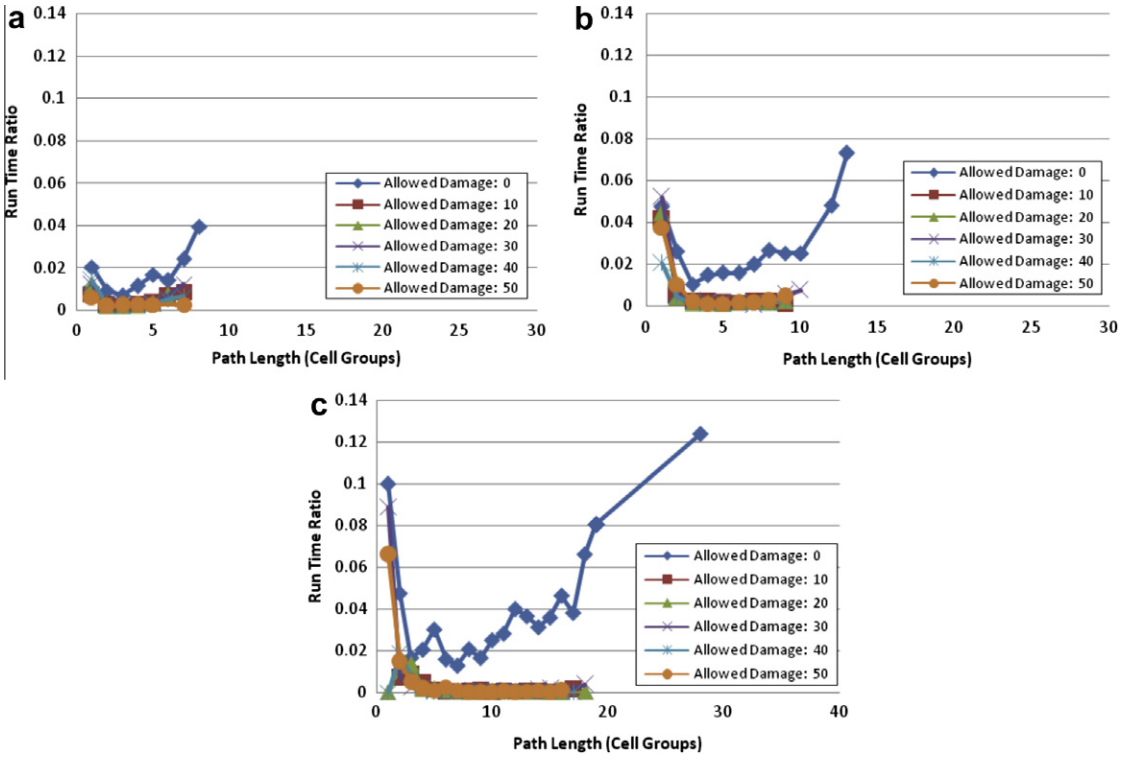

Fig. 11. Variation of runtime ratio with distance for 28 × 28 maps (a), 49 × 49 maps (b), and 98 × 98 maps (c).

allowed damage is 0.0, LDA* performs its best in terms of path length, and they are almost identical to the best paths 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

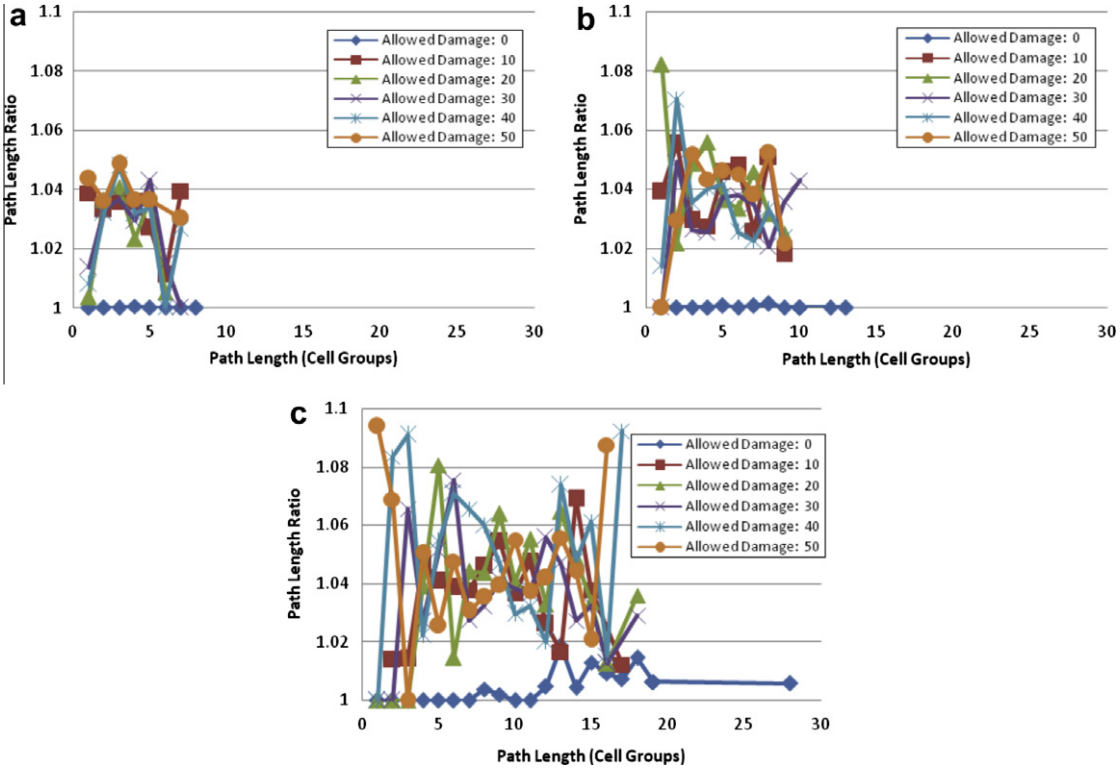

Fig. 12. Variation of path length ratio with distance for 28 × 28 maps (a), 49 × 49 maps (b), and 98 × 98 maps (c).

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.

4.3. Special cases

LDA* performed well on the randomly generated test maps for the environmental conditions defined in the previous sections. However, presence and distribution of special cases in the test data is unknown, and therefore, no 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 edges length of movement, but their threat zones may be greater than those used in the previous tests. In these setups, all cases other than the first one have a unique solution.

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 Fig. 13 was prepared and tested. The shortest path solution of MOA*, the solution of LDA* are also shown in the same figure. In this case, LDA* is able to find a path equivalent to the optimal, within 0.1% to 0.3% runtime of MOA*, and the maximum runtime for different DCF values is less than 8 ms.

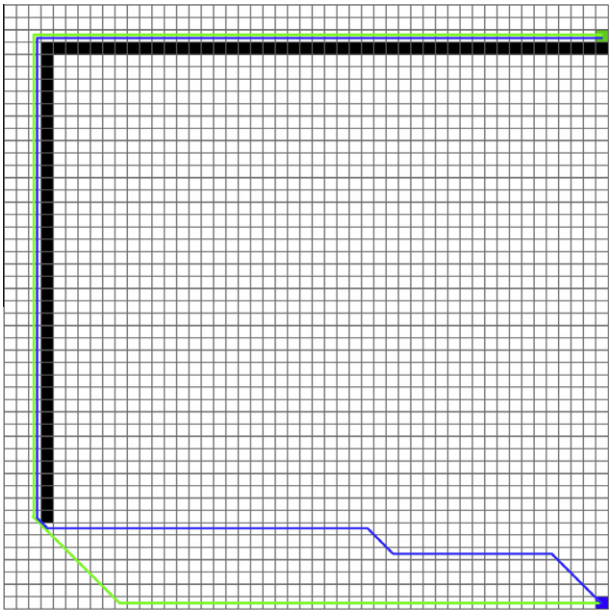

Fig. 13. Trap map for case 1 with the start cell at the bottom right corner, and the goal cell near the top right corner. Green path is the solution path of MOA* (shortest path), and the blue path is an equivalent solution by LDA*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

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 m (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 Fig. 14.

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. 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 Fig. 14. 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.

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 make single run of MOA* on the same setup. In addition, LDA* runs take very small amount of time as a maximum of 22 ms.

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 100 m (10 cells). The second threat source is placed towards the right side of the map, with an effective range radius of 103 m. 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 Fig. 15.

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 Fig. 15, 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 at most 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 ms for this setup.

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 m and higher damage of 3 HP. There is another threat source located below it, with a radius of 70 m, 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 Fig. 16.

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

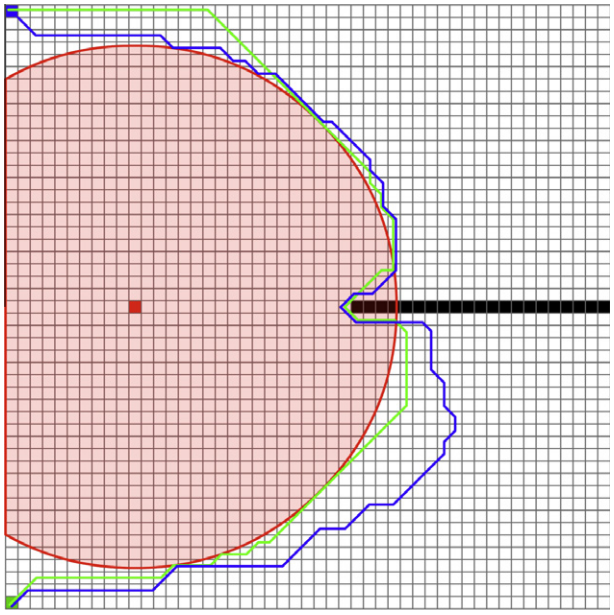

Fig. 14. Map for case 2 with the start cell at the top left corner (darker line), and the goal cell at the bottom left corner (lighter line). Green path is the solution path of MOA* (shortest path), and the blue path is the best solution found by LDA*, with DCF = 80. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

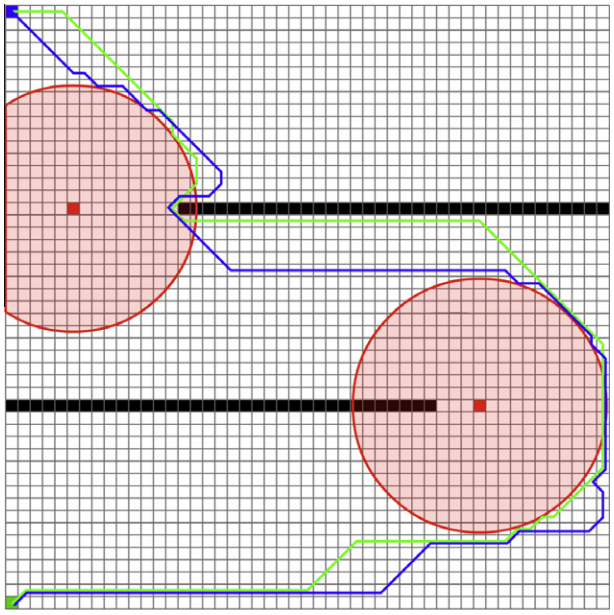

Fig. 15. Map for case 3 with the start cell at the top left corner, and the goal cell at the bottom left corner. Green path is the solution path of MOA* (shortest path), and the blue path is the best solution found by LDA*, with DCF = 1000, 10000 and 100000. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

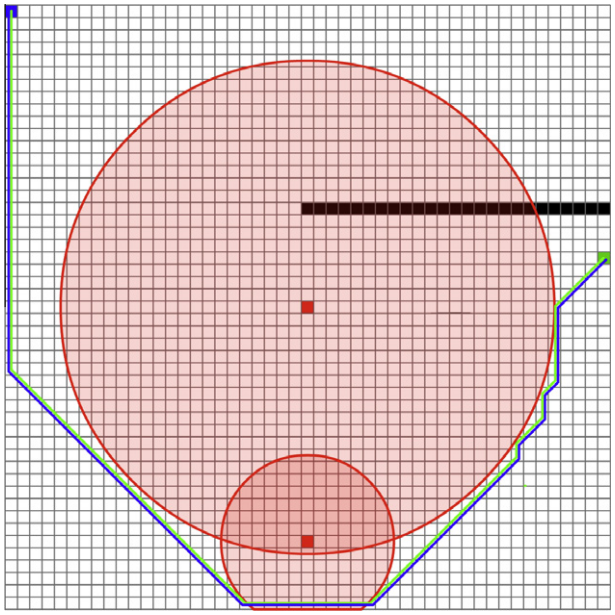

Fig. 16. Map for case 4 with the start cell at the top left corner, and the goal cell at the right edge. Green path is the shortest path obtained from MOA*, and the blue path is the best solution found by LDA*, with DCF = 0.0 through 0.1, and 2. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

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 ms.

5. Conclusion and future study

In this study, damage was used as a feasibility criterion in path search, along with path length as the optimality criterion. An A*-based heuristic path search algorithm was developed in order to be used in path search problems in computer games, where an agent is required to travel in a static and fully-observable maze-like grid environment, which involves non-moving turret-like structures that can inflict damage on the agent, if the agent passes through their areas of effect (threat zones, or ranges). Amount of damage and agents health were abstracted as numeric values.

This algorithm aimed to generate a suboptimal solution (a path from a starting position to a goal position), which would have a path length close (or equal) to the shortest path that would cause an agent traversing that path to suffer an amount of damage less than or equal to the predefined damage limit.

Considering hardware and software limitations, three different sizes of 28x28, 49x49 and 98x98 cells were selected as the map sizes of the environment, and three sets of tests were run. The first set was aimed to determine the best heuristic coefficients for the environmental properties. The second set aimed to test the performance of the algorithm against the reference algorithm - Multiobjective A* (MOA*). Finally the third set of tests examined performance of the algorithm in special cases.

Although it is neither a complete, nor an optimal algorithm, LDA* was found to perform satisfactorily in randomly generated maps, with a failure rate of 1 in 700 runs. Its execution times was found to be 4%, 8% and 13% less than those of MOA* runtimes for increasing map sizes. Paths generated by LDA* were found to be at most 5%, 8% and 10% longer than those of the optimal paths in small, medium and large maps, respectively.

LDA* runtimes exceed 10–20 ms for medium and large map sizes, and therefore the current implementation of the algorithm in Java lacks the pace required for real-time applications such as real-time computer games, only for large environments. However, better results can be obtained if it is implemented with a faster programming language, such as C or C++. Thus, it can possibly be used in larger maps with adequate performance.

The proposed method could be used in many areas like navigation of Semi-Automated Forces (SAF) [21] in military simulations and non-player characters (NPC) in computer games and real robots. In military simulations, there has been growing interest in modeling behaviors at both individual and unit level to simulate decision making and tactics used in real life for simulation based training, analysis and acquisition (OneSAF [22], TeBAT [23], ModSAF [24], SWARMM [25]). In that respect, navigation of individual soldiers/vehicles and that of a unit require efficient algorithms that minimizes length of the path (fuel, energy) and exposure to threat zones to be embedded in SAF.

For future study, researchers are encouraged to apply the basic principles of LDA* into incremental and/or hierarchical approaches [26] in order 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 ones is also highly encouraged, since such improvement are expected to shrink the search space, and contribute to the performance of the algorithm drastically.

References

[1] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on System Science and Cybernetics 4 (1968) 100–107.
[2] A. Stentz, Optimal and efficient path planning for unknown and dynamic environments, International Journal of Robotics and Automation 10 (3) (1995) 89–100.
[3] S.Koenig, M.Likhachev, D* lite, in: Proceedings of the AAAI Conference of Artificial Intelligence (AAAI), 2002, pp. 476–483.
[4] A. Stentz, The focussed *d** algorithm for real-time replanning, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI) 2, 1995, pp. 1652–1659.
[5] R. Korf, Real-time heuristic search, Artificial Intelligence 42 (2-3) (1990) 189–211.
[6] C. Undeger, F. Polat, Real-time edge follow: A real-time path search approach, IEEE Transaction on Systems, Man and Cybernetics, Part C 37 (5) (2007) 860–872.
[7] C. Undeger, F. Polat, Rttes: Real-time search in dynamic environments, Applied Intelligence 27 (2007) 113–129.
[8] D.J.H. Burden, Deploying embodied ai into virtual worlds, Knowledge-Based Systems 22 (2009) 540–544.
[9] B. Stewart, C. White, Multiobjective *a**, Journal of the Association for Computing Machinery 38 (4) (1991) 775–814.
[10] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik 1 (1959) 269–271.
[11] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed., Pearson, 2010.
[12] D.Demyen, M.Buro, Efficient triangulation-based pathfinding, in: Proceedings of the AAAI Conference, 2006, pp. 942–947.
[13] S. Koenig, M.Likhachev, Improved fast replanning for robot navigation in unknown terrain, in: Proceedings of the IEEE International Conference on Robotics and Automation 1, 2002, pp. 968–975.

[14] R. Korf, Depth-first iterative-deepening: An optimal admissible tree search, Artificial Intelligence 27 (1985) 97–109.
[15] P. Chakrabarti, S. Ghose, A. Acharya, S.S.C., Heuristic search in restricted memory, Artificial Intelligence 41 (1989) 197–221.
[16] R. Korf, Linear-space best-first search, Artificial Intelligence 62 (1) (1993) 41–78.
[17] Y.-H. Qu, Q.Pan, J.-G. Yan, Flight path planning of uav based on heuristically search and genetic algorithms, in: Annual Conference IEEE Industrial Electronics Society, (IECON+05), 2005, pp. 45–49.
[18] C. Hallam, K. Harrison, J. Ward, A multiobjective optimal path algorithm, Digital Signal Processing 11 (2001) 133–143.
[19] R. Metoyer, S. Stumpf, C. Neumann, J. Dodge, J. Cao, A. Schnabel, Explaining how to play real-time strategy games, Knowledge-Based Systems 23 (2010) 295–301.
[20] R. Nkambou, P. Fournier-Viger, E.M. Nguifo, Learning task models in ill-defined domain using an hybrid knowledge discovery framework, Knowledge-Based Systems 24 (2011) 176–185.
[21] D.A. Reece, Movement behavior for soldier agents on a virtual battlefield, Presence 12 (4) (2003) 387–410.
[22] D. Parsons, J. Surdu, B. Jordan, Onesaf: A next generation simulation modeling the contemporary operating environment, in: Proceedings of Euro-Simulation Interoperability Workshop, 2005.
[23] J. Vaughan, R.C.A. Lucas, R. Ronnquist, Towards complex team behavior in multi-agent systems using a commercial agent platform, Lecture Notes in Computer Science, Vol. 2564, Springer, 2003, pp. 175–185.
[24] J.Fugere, F.LaBoissonniere, Y.Liang, An approach to design autonomous agents within modsaf, in: Proceedings of IEEE SMC'99 Confeence on Systems, Man, and Cybernetics, Vol.2, Tokyo, Japan, 1999, pp. 534–539.
[25] D. McIlroy, B. Smith, C. Heinze, M. Turner, Air defence operational analysis using the swarmm model, in: Asia Pacific Operations Research Symposium, 1997.
[26] A. Autere, Hierarchical A^* based path planning - a case study, Knowledge-Based Systems 15 (1-2) (2002) 53–66.