Group Pathfinding Using Group Division

Mehdi Saeidianmanesh

A Thesis
In the Department
Of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

April 2015

© Mehdi Saeidianmanesh, 2015

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Mehdi Saeidianmanesh

Entitled: Group Pathfinding Using Group Division

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. L. Kosseim	Chair
Dr. A. Krzyzak	Examiner
Dr. D. Goswami	Examiner
Dr. T. Fevens and Dr. S.p. Mudur	Supervisor

Approved by ______________________
Chair of Department or Graduate Program Director

____
Dean of Faculty

Date ______________________

ABSTRACT

Group Pathfinding Using Group Division

Mehdi Saeidianmanesh

Pathfinding is an important problem in video games. Good pathfinding strategy, for both player characters and non-player characters is one of the key differences between a good game and a less successful one. Finding the shortest path for one character unit in a given environment is a very well-known problem for which there exist many solutions. When a group of units wants to pass through the map, the problem becomes more complicated. This thesis introduces the Reduced Wait Time (RWT) algorithm which is a multi-unit path planning algorithm where units can take several paths instead of just the shortest one to reduce the overall waiting time of the units in queues along the path and therefore reduces the total time needed to pass all units through the map.

The main goal of this thesis is propose the pathfinding algorithm RWT to reduce the total time for a group of units to pass a route. The simulation results shows that the RWT algorithm not only reduces the time to pass by decreasing the waiting time, but also can reduce the number of collisions between units which reduces the CPU usage which is another important consideration for games. Using the RWT algorithm also gives an opportunity to the level designers to be able to implement strategic pathfinding in games. Different routes have different strategic advantages and disadvantages over each other; being able to send units through different paths enables designers to consider strategic path-planning.

The RWT algorithm was implemented using the Unity game engine and tested on a number of randomly generated example maps. The experimental results were compared with the results from other related algorithms such as Local Repair A* and Maximum Flow to show that the RWT algorithm works better than other studied algorithms.

Acknowledgments

This thesis would not have been possible without the support of many people. Herein, I wish to convey my most sincere thanks to those who have helped me in various ways throughout the years.

First and foremost, I would like to express my sincerest appreciation to my supervisors, Dr. Thomas Fevens and Dr. Sudhir P. Mudur. Thank you for giving me the opportunity to participate in this research field. Thank you for your invaluable suggestions, guidance, inspiration and continuous encouragement throughout the course of my Masters project.

Without my friends in Canada, life would have been boring and dull. With all of you people, I enjoyed my life. The good times I spent with you, good times seemed like yesterday.

Finally, I would like to thank my dearest family with all my heart. Words are too weak to express all my gratitude to my parents and my younger sister for their endless love, care, support, encouragement and understanding. You were always there for me whenever I needed to hear your voice, which made me feel so calm and at peace.

Last, but not least, I would like to give very special thanks to my beloved wife, Fari: thank you for your tolerance, your care and love for me. I am willing to walk beside you and encourage each other throughout the difficult times, and to see wind, rain and rainbow together.

Table of Contents

List of Figures

List of Tables

Introduction

Pathfinding and path planning are important problems in computer games, traffic control systems, robotics, and military purposes. When we want to find an appropriate path, many aspects of the problem should be considered, because the goals of pathfinding in various applications are different. In some cases, minimum travel time is important, in other cases having minimum waiting time, in other cases traveling in a safe path or minimum length to pass and so on. Finding a path with minimum length to travel for one unit is a very well-known problem and many solutions are available (e.g., shortest path algorithms). The problem becomes more complicated when a group of units wants to pass through the map instead of just one unit. If we could assign a dedicated path to each unit, the problem would be easier to solve, but this may not be feasible in a map with narrow passages and with a large group of units who want to pass. Most of the time, the whole path or at least some parts of a path will need to be shared among different units. Sometimes the different sizes of units also need to be considered. This is because the capacity of a passage is related to the size of units, specifically that the capacity of a passage is the number of units that can pass side-by-side along the path.

The focus of this thesis work is on path finding in computer games and especially real time strategy games. Figure 0.1 shows an example of a real time strategy game. It is a battlefield with static and dynamic obstacles and armies from two opponent sides. There are different kinds of units in each side. Units can have different sizes and speeds and paths are shared among them.

Figure 0.1: screen shot from Generals: Command and Conquer, Electronic Arts studios game. The majority of strategy games, involve different kind of fighters that need to find their way through the terrain, while sharing some paths

In real time strategy (RTS) video games, there are usually groups of units that want to travel in the game world in order to reach their destinations. Each unit may need to follow a different path between its start and destination locations in the given environment. This environment can be two-dimensional maps, a huge terrain or a building with lots of rooms, etc. Each environment has its own limitations and specifications. We will use the term map to generally denote the representation of the environment through which the units have to travel from one point to another.

There are many solutions to find a path between two points. Dijkstra's algorithm and the well-known A* algorithm are two examples. These solutions try to find the shortest path between source and destination, and then unit takes that path to reach the destination.

These solutions can also be applied for a group of units. We can calculate the path for each unit individually; in this case, units are called agents. Alternatively, we can select a leader, find a path for the leader and then force other units to follow the leader. By adding some good steering behaviors to the units in the group, movement can appear more realistic. Crowd simulation and flocking system techniques are good examples of such solutions.

Having a good pathfinding strategy is very important in games. If the pathfinding strategy is not appropriate, the game can become very easy or hard to play. Units may do strange or silly behaviors and become easy to predict, and hence are vulnerable to defeat. There are many well-known games that have poor pathfinding techniques built-in.

The problem becomes more complex when there are multiple agents in the game. Units find their own paths and follow it. In the course of pathfinding, in case of bottlenecks in the map, that is an area becomes very dense, then units have to reroute and find an alternative path. This could happen multiple times. In such cases if the number of units is high and the passage is very narrow, the units never reach their destinations. They just rotate around each other and try to avoid collision with other units [17, 18, 19, 20].

1.1 Outline of the problem

Most pathfinding solutions have the same goal: finding the shortest path. However, with a large group of units, finding the shortest path and forcing all the units to pass through that one will not result in the minimum travel time and is also not strategically wise [11, 16]. Hence, alternative solutions need to be sought for travel by groups in an environment. In this work, we propose an alternative solution that splits the group, sends them via different paths, and sometimes regroups them along the path to obtain faster and safer paths.

In group pathfinding of RTS games, there are usually two problem situations to deal with:

1. Suppose there is a large group of tanks in a game and you want to send them to destroy your opponent's base. Your opponent's base has several entrances, but the one that has minimum distance to your tanks is a narrow passage. When you want to send your army into opponent base, for sure they will from a queue at the base entrance. At this moment the opponent can destroy all of your tanks easily when they are waiting in a queue.
2. If you send all the units from one path (shortest path), although it is the shortest one but each path has a limit on its width and therefore its capacity. Consider a large number of units wanting to pass through a passage with limited capacity. After a while, the shortest path will become full and there will be a queue at the entrance. In-line units should wait for the path to become free and it takes time (waiting time).

To address these problem situations, it would be better strategy to send some of the units through alternative paths, even if they are longer paths.

1.2 Research Methodology

This work is based on the requirement of pathfinding for a large group of units in computer games. Proposed algorithm (RWT) in the fourth section of this thesis can be used for both groups of player's characters or non-player characters (NPC) to find their routes from any given start point to destination in a game terrain. Previous studies about pathfinding, usually consider the length of paths as the main factor of pathfinding where, in many situation taking the shortest path will not result in having the minimum time to pass.

In a more general sense, the existing techniques of pathfinding are suitable for single units or small group of units. When it comes to large group of units or when the game world is a terrain with narrow passages, existing techniques are inefficient.

In this thesis, Reduced Waiting Time (RWT) algorithm has been proposed, which can handle pathfinding for both large and small number of units.

Further, simulations on multiple randomly generated maps have shown that RWT algorithm has some improvements over other studied algorithms. RWT algorithm reduces the time to pass, decreases the number of collisions and significantly reduces the waiting time of being in queue for units. Unity game engine has been used to test the RWT algorithm to validate its efficiency.

1.3 Contributions

- In this thesis, a new pathfinding algorithm is proposed to reduce the waiting time of units who want to move on a map in video games. By reducing the waiting time, we will have a lower time to pass for the group of characters. The main contribution of this thesis is as follows: Proposing a solution to reduce the needed time for a group of units to pass a route through non-trivial terrain by dividing a group of units and sending them via different routes.

- We also show that our algorithm also decreases the number of collisions between units that leads to a decrease of CPU and memory usage. For games to handle these collisions and finding proper reactions to those collisions increases the CPU and memory usage.
- We developed a random map generator that enables us to test different algorithms on automatically generated random maps to allow us to determine reliable and more general results from our experiments.
- We studied improved ways of dividing a shared passage's capacity between the paths that share that passage. Here, better way of dividing the shared passage, leads to lower time needed to pass units from that shared passage.

Most of algorithms that video games use for pathfinding just consider shortest paths and not the lower time for groups of characters that must share paths. For example, using A* in a game word with large group of units and narrow passages will result in lots of queues and therefore lots of waiting time of being in queue for units. The main contribution of this thesis is to decrease the time to pass for such a large group of units.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 Background and Related Works. In chapter two, a number of reported published pathfinding algorithms are reviewed to build essential background knowledge on methodologies for pathfinding in computer games. Navigation meshes and waypoints are also reviewed as two available infrastructures to implement pathfinding algorithms on.

Chapter 3 Group Pathfinding in Shared Passages with Different Capacities. A shared passage with different capacities is studied in chapter three. In this chapter, problem of having a passage that has two or more entrance and should share its capacity between those entrances is explained. To address the solution of how to dedicate shared passage's capacity to each entrance, a sample shared passage is considered and different results based and having static and dynamic ratios are explained. Chapter 3 concludes with experimental results of having a shared passage with different capacity ratios.

Chapter 4 Reduced Wait Time (RWT) algorithm. In chapter four, a new solution to handle group pathfinding with reduced waiting time is proposed. A random map generator is used to be able to create randomly generated test environments. RWT algorithm is introduced in this chapter, followed by testing the algorithm on a sample map.

Chapter 5 Simulations, Results and Discussion. In fifth chapter, RWT algorithm is compared with other well-known algorithms, which have been reviewed in chapter 2. The algorithms are compared in terms of time, traveled distance, waiting time and number of collisions. Advantages and disadvantages of each algorithm are discussed, also results are shown in charts and tables.

Chapter 6 Conclusions. In the final chapter we give our conclusions.

Background and Related Works

1.5 Path-planning methods

Path-planning solutions for a group of units in RTS games can be divided into two categories: Centralized and Decoupled methods [1].

1.5.1 Centralized methods

Centralized methods search to find paths for all units at a time. This method combines configuration spaces of individual units and creates one system for all. Road map method [2] is an example of this approach.

1.5.2 Decoupled method

Decoupled methods calculate the path for each individual unit separately and then try to solve potential collisions in the map. Cooperative path finding [4], which uses time-space table [3], is a good example of decoupled approach. In cooperative pathfinding, each unit knows the position and direction of all other units. Decoupled techniques assign priorities to each unit and based on those priorities the algorithm calculates and assigns paths [5].

1.6 Pathfinding algorithms

Dijkstra, Maximum Flow algorithm, A*, Local Repair A* and Cooperative A* are example solutions to handle pathfinding. We will discuss each in detail next.

1.6.1 Dijkstra

Dijkstra algorithm [6] is a graph search algorithm that finds single source shortest path in a graph with non-negative edge path cost. Computer scientist, Dr. Dijkstra, introduced the algorithm in 1959. The algorithm computes the distance from a start node to all other nodes.

The algorithm records a set S of nodes with distances to Start node. Initially S contains the Start node with distance equal to zero and all other nodes with the distance equal to infinity. Then source node is then placed in the open list. Then the following procedure is repeated until there are no more nodes in the open list:

- Pick a node n from the open list with lowest temporary distance, remove it from open list and put it on the closed, and change temporary distance to n to permanent.
- Compare the length of the path via n to all n's neighbors' temporary distances. For all the neighbors where the path to that neighbor via n is shorter then change the temporary length for the neighbor with new cost (path length via n) and put the neighbor on the open list.

At this point, the shortest paths and their lengths from node n to all other nodes are calculated.

1.6.2 Maximum Flow

Maximum Flow problem is to find feasible flows through a single source, single sink network. The Maximum Flow is the maximum feasible flow in network. Many methods are proposed to solve the Maximum flow problem with different time complexities, like Ford–Fulkerson algorithm or MPM [7] algorithm. Maximum Flow algorithm does not consider the "time to pass" factor. It just considers how to send the units through the map to reach a steady state of fully occupied paths. Goal of the solution is to reach a state in which maximum numbers of units are moving on the map at all times.

Consider a terrain with one source and one destination. Capacity of an edge represents the maximum number of units that can pass through an edge when they are moving in a row. Maximum Flow will divide the group into subgroups and try to send them through different paths instead of one shortest path.

1.6.3 A*

A* (pronounced a-star) [8] is the most common pathfinding algorithm in computer games. Peter Hart, Nils Nilsson and Bertram Raphael described the A* algorithm in 1968. A* is a greedy best-first search approach that uses heuristic to guide itself. A* takes into account, the distance which is already traveled from start point to current point, x and names it g(x). Another part is h(x) which is a future path-cost function. h(x) is constant time heuristic estimate of the distance between current point, x and the goal. Here is the A* equation [8]:

$$f(n) = g(n) + h(n)$$

A* does not blindly searches for paths. It finds the shortest path between two points with the appropriate use of heuristic and termination condition. Indeed, if the heuristic is a null heuristic (always equal to zero), A* is identical to Dijkstra's algorithm.

Algorithm starts with the initial node and maintain an open list, which is priority queue of nodes where for each node x, algorithm gives the higher priority to the node with lower f(x). Initially S contains the Start node with g(S) equal to zero and all other nodes x with the g(x) equal to infinity. Then, the node with lowest f(x) will be removed from the queue. For those neighbors n of x where the distance to the n via x is smaller than g(n), values of g(n) and f(n) will be updated and added into the open list. Until the smallest g(n) value on the open list is larger than the g(destination) value or the open list is empty, the algorithm continues above steps. When the algorithm terminates, the value of f(destination) is the length of the shortest path from start to destination node. At this point, the value of h(destination) should be zero and g(destination) is the length.

A* when applied as is to group path finding, will try to find the shortest path for each unit and will not be concerned with any bottlenecks or path congestion problems.

1.6.4 Local Repair A*

In Local Repair A* (LRA*) method, each unit uses A* algorithm and searches for a path to its destination. Units do not consider other units except neighbor units. Units begin to move in their assigned paths until a collision is about to happen. At this time (before collision actually happens), unit recalculates the path from current position to destination. Moving in cycle is a common problem of this solution. Consider a situation when a unit collides with another unit, then recalculates the path and finds a path in reverse direction and starts to move back in direction of new founded path. At this time, it collides with another unit and recalculates the path and again this one is in reverse direction. In crowded regions, the above-described scenario can occur a lot. It takes very long time to finish the path finding in such situations and sometimes it is even impossible (deadlock). Figure 0.1 is an example of a situation with possible cycles. In Figure 0.1 units A and B are moving in one direction (left to right) and C is moving on opposite direction. B will collide with C and recalculates its route, if the LRA* decides to change the path for unit B and the new assigned path is in reverse direction, a potential deadlock is unavoidable.

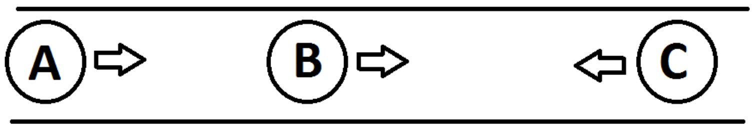

Figure 0.1: An example passage with capacity of one and three units. Arrows show unit's moving direction

1.6.5 Cooperative A*

Cooperative A* [4] is an algorithm for solving cooperative pathfinding problem. Cooperative pathfinding needs to have knowledge of all units and positions of obstacles at each moment. To know the exact status of each point in the map (it can be a point in waypoint maps or a tile in tiled maps) at each time, a third dimension is added into the map, time dimension. Waypoint maps are described further below. This space-time map is a three dimensional grid of cells. Divide the problem into single agent search, each agent performs the search in three dimensional space-time table and considers routes of other agents. After each unit's path is set in reservation table, that path will be unusable for other agent during the specified time. The reservation table represents the agents shared knowledge about each other's planned routes.

Consider a situation when a unit U1 is in cell(x , y) at the time t1. It can be represented as U1(x , y , t1). U1 wants to move to cell(x+1 , y). New status will be U1(x+1 , y , t2). This move could be done if and only if the cell(x+1 , y) is free at the time t2. Then if that cell is free, it will be marked as occupied at that time-space position.

- **Reservation table**

First unit chooses its path and other units will not occupy cells along that path at reserved time. For example, U1 wants to move from cell(x , y , t1) to cell(x+1 , y , t2) and then all neighbor cells are occupied at t2, so unit have to wait until t3 and then will move to cell(x+2 , y) at t4. These cells should be marked into a reservation table. A data structure with an entry for every cell of space-time map is required.

Figure 0.2 shows pathfinding for a unit which wants to move from its start to destination. Selected path is marked as occupied in the time-space reservation table.

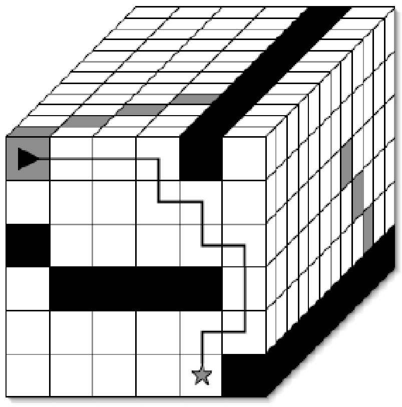

Figure 0.2: Three dimensional space-time maps proposed by D. Silver [4]. Dark black cells (■) are obstacle, which remain occupied all the time. Gray cells (▨) are the cells, which marked as reserved cells for the unit (▶)

1.7 Using Navigation Mesh vs. Using Waypoints

Walkable areas of the game terrain and possible paths between any two points in the game world should be represented and available to the pathfinding system. Then the pathfinding system uses suitable algorithms to find the best routes. Navigation Mesh (usually called NavMesh) and Waypoints are two representations of walkable areas in game world.

1.7.1 Navigation Mesh (NavMesh)

Navigation mesh is a set of convex polygons that describe the “walkable” surface of a 3D environment [9]. Navigation meshes are composed of convex polygons which when assembled; represent shape of the map analogous to a floor plan. The polygons in a mesh have to be convex, since this guarantees that the AI agent can move in a single straight line from any point in one polygon to another point in that polygon. Units can go from one area to another adjacent area when those two meshed-areas have shared points at their borders. Convex meshes are connected to their neighbors through shared adjacent nodes. Each of the convex polygons can then be used as nodes for a pathfinding algorithm. A NavMesh path consists of a list of adjacent nodes to travel along. Convexity guarantees that with a valid path the AI agent can simply walk in a straight line from one node to the next on the list [10].

Figure 0.3 shows a terrain with NavMesh areas.

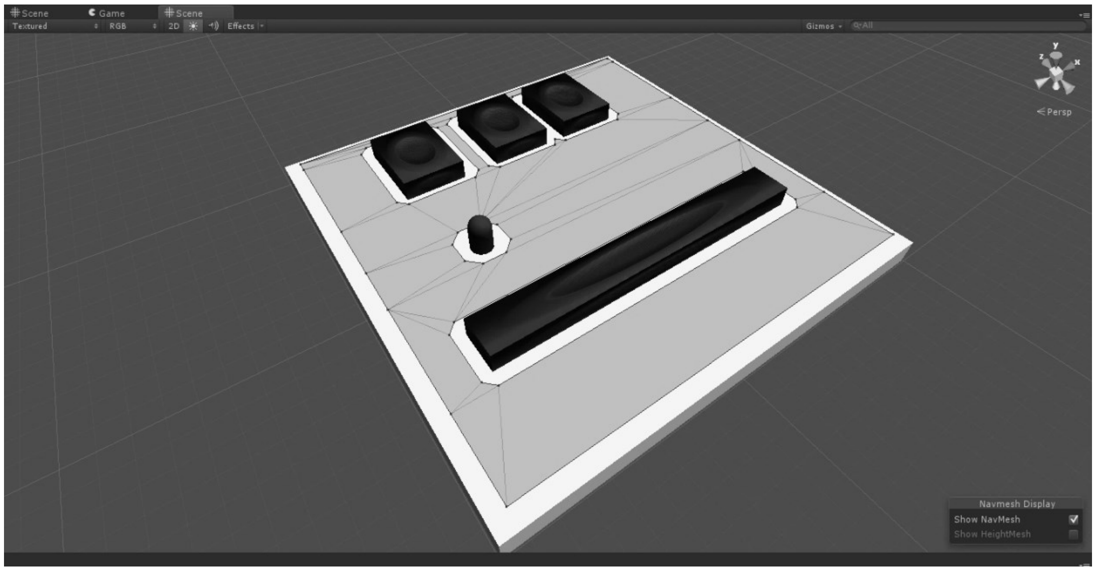

Figure 0.3: Terrain with NavMesh areas. Gray areas are meshed and therefore walkable areas and other areas (dark colored obstacles and white areas around them) are un-walkable areas. Screenshot from Unity game engine built-in NavMesh baker.

1.7.2 Waypoints

The waypoint system for navigation is a collection of nodes (points of visibility) with links between them [13]. Consider waypoints as nodes in a graph. Non-Player characters (NPCs) can navigate through the world using these points. Edges of the graph are connections between nodes. These edges are automatically generated in a preprocessing step. Level designers also can generate edges manually. Then pathfinding algorithms are used to find paths through the node-graph [12].

Travelling from one waypoint to another is a sub problem with a simple solution. By traveling along a number of waypoints, any two waypoints should be reachable in a map. If an AI agent wants to get from A to B, it walks to the closest waypoint seen from position A, then uses a pre-calculated route to walk to the waypoint closest to position B and then tries to find its path from there. This system has benefit of representing the map with the least amount of nodes for the pathfinder to deal with [21].

Figure 0.4 shows a terrain with waypoints. Each node is a waypoint and if there is a line between two waypoints, it means that they are connected and therefore there is a path between those two points.

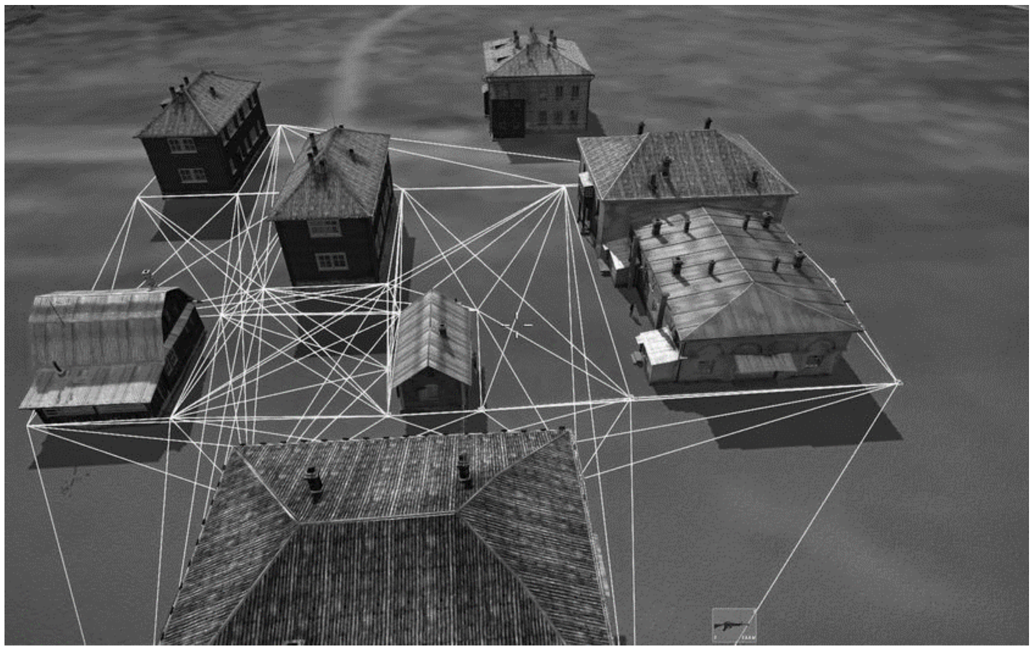

Figure 0.4: An example terrain with connected waypoints. Sample map from DAYZ, a multiplayer online game by Bohemia Interactive studios.

- **Strategic Points**

By pre-processing the waypoints relations, game designers can plan dynamic intelligent defense and attack strategies for Non-Player characters (NPCs). In this method, designers calculate and store strategic information about the relation between waypoints, and then NPCs use these strategic positions to act in more realistic and intelligent manner.

Positioning of the waypoints is done in level design, so game designers can calculate and store waypoints relation in level design step. [13].

By using waypoints, each point can get a number, which reflects its level of danger or in other words level of visibility. Waypoint with high level of visibility is a place in the map, which has high risk of being visible by opponent units. Points with high level of visibility are more vulnerable to enemy's line of fire. On the other hand, a point with lower risk is a good place to be hidden from enemy's fire. Figure 0.5 shows an example map with some high and low visibility level areas.

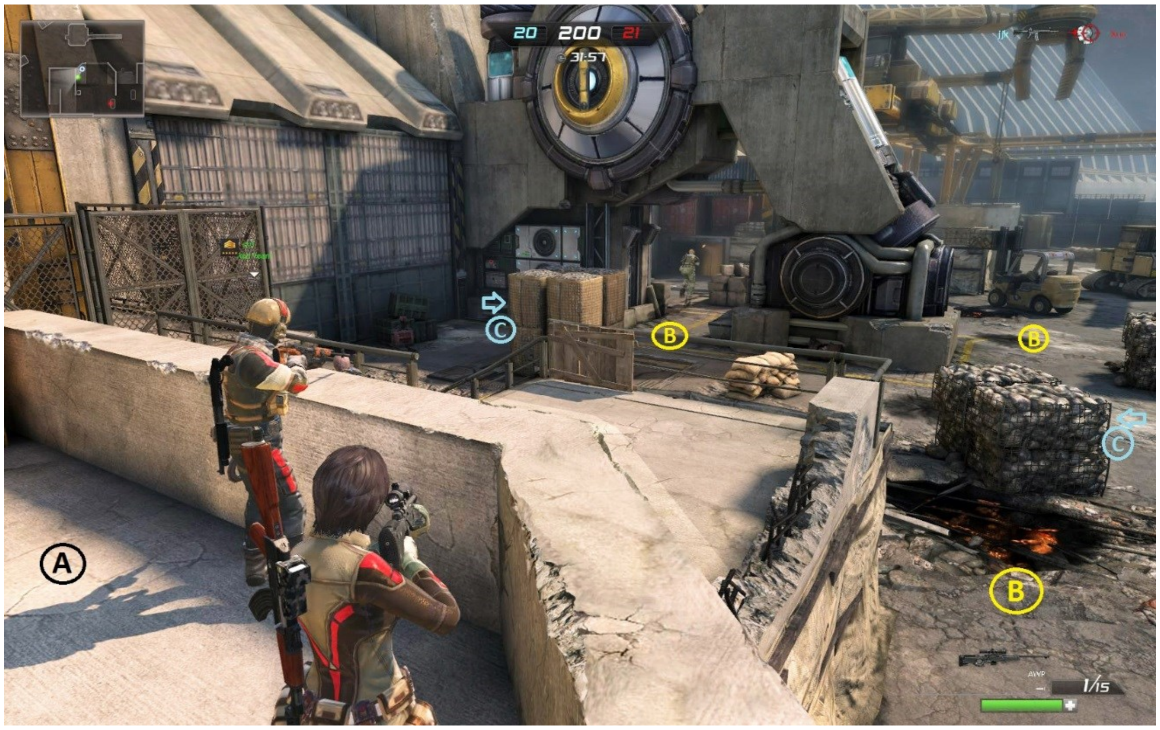

Figure 0.5: Screen shot with low-risk (C) and high-risk (B) areas for defending. Area marked as (A) is a good attacking position. Screenshot from "Mercenary Ops" game by Yingpei Games studios.

The same strategy can be used for attacking. A waypoint with higher attack level is a good point to start the attack. It can be a point with a better view on enemy's area or a point in a high place that gives a better opportunity to the shooter. Figure 0.5 shows such areas.

Strategic points are defined as pinch and ambush points. Game designers use the graph, to calculate pinch and ambush points.

Any area in the map that connects two other large areas will be considered as a pinch point. All the units, which want to travel from one side of pinch point to the other side, have to pass through this pinch point. Doorway into a room is an example of a pinch point.

Ambush points are strategic areas near the pinch points. NPCs, which stay in these ambush areas, are hidden from other units who are on the other side of pinch points [14]. Game developers can use ambush and pinch points to implement intelligent strategies for attack and defense.

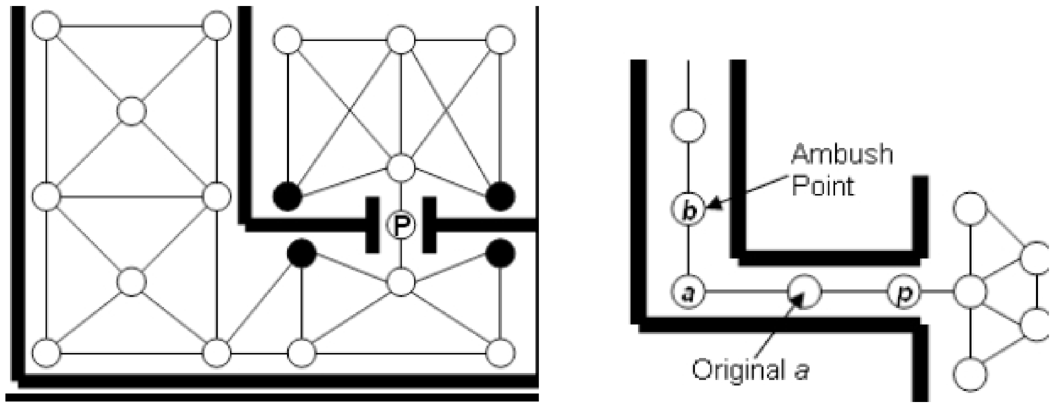

Figure 0.6: Example of pinch and ambush points in a map, proposed by D. White [14]

1.7.3 Comparing NavMesh and Waypoint

Both methods have some advantages and disadvantages. Based on the game, designers have to choose one method that fulfills game's requirements. Below are some recommendations for choosing between these two methods:

1. Games with large open areas need huge number of waypoints to implement; in this case, it is better to use NavMesh.
2. Characters in a game, which use waypoint, move in zigzag shape even when methods are added to smoothen the unit movement. In the other hand, characters, which use NavMesh for pathfinding, have smoother paths.
3. In a game with waypoints, it is harder to do path correction in dynamic maps (maps with dynamic obstacles).
4. It is hard to have one waypoint path for all kind of moving characters (think about a big Tank and a walking soldier).
5. Using NavMesh will give you more information about spots around a character.
6. Placing strategic points is an advantage of waypoint system. Although it is also possible to have different meshed areas in a map, it is easier to have pinch and ambush points when waypoint system is used.

1.8 Capacity of a path with different edges

Consider a path from start point S to destination D, in an example terrain. There will be some passages (edges) with different capacities during that path. Figure 0.7 shows such a path in a sample map.

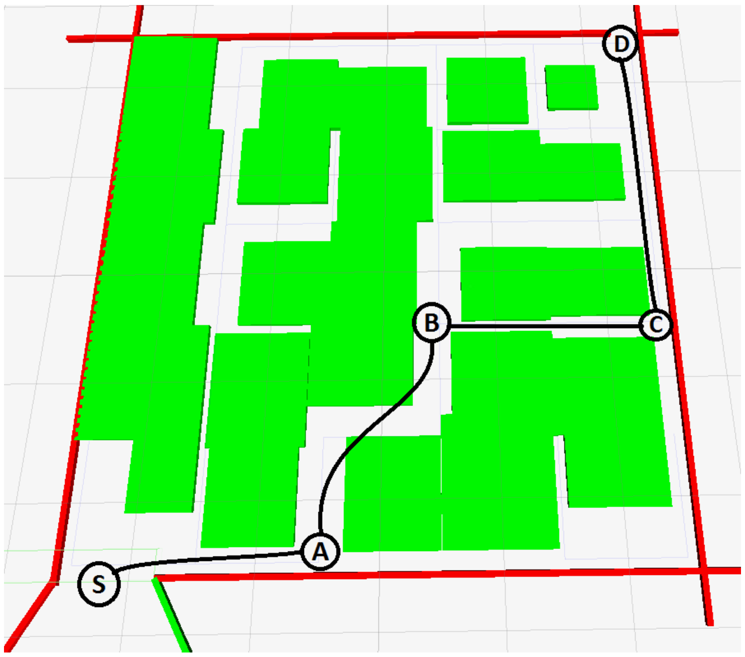

Figure 0.7: Paths with different edges and different capacities

In this example, edges Start to A, A to B, B to C and C to Destination, are in between the path from start to destination. Table 0.1 shows edges with capacities.

Table 0.1: Edges of the path with capacities

Edge	Capacity
Start to A	6
A to B	10
B to C	2
C to Destination	5

A group of units wants to go from Start to the Destination, as fast as possible. Capacity of the path is limited, so units cannot go through it at a time. Based on the capacity of the first edge (Start to A), that is six, six units can go through the path and other units have to wait for a while until the edge becomes empty and then go through it. Two different scenarios are possible:

(a) First scenario is sending units with maximum capacity of the edge from Start to point A.

(b) Second scenario is to send units with minimum capacity of the edges along the path that is edge from point B to point C with capacity of two.

Capacity is the number of units that can pass from a path while they are in a row. Capacity of Start to A is six and A to B is 10, so all the units coming from first edge can pass through edge A to B with the same speed and formation, without any delay. Consider that only 60% of available capacity of this edge is occupied. At the other end of edge, A to B there is edge B to C with capacity of two. Since capacity of edge B to C is less than A to B, units cannot go through this edge with the same speed and capacity as previous edge. It means there will be a queue of units at the end part of edge A to B. Meanwhile, other units are coming and the queue becomes bigger and bigger until a steady state is reached. Steady state is when the edges A to B and Start to A are fully occupied. At this state, only and only two more units can go into first edge, Start to A, at each time slot.

On the other hand, units can go through the path by the minimum capacity of the edges in that path, which is equal to two in this example. Units go through the edges, two in a row and continue their travel with a consistent speed and formation from start to the end. In this case, there is no delay or queue during the path, except at the entrance gate.

Required time, to pass all the units from Start to Destination is equal in both scenarios. It means that it does not matter to send the units with maximum capacity or by the minimum capacity of the edges in the path.

In situations where there are multiple sources and shared junctions in a map with, it is better to form the queue at the start point.

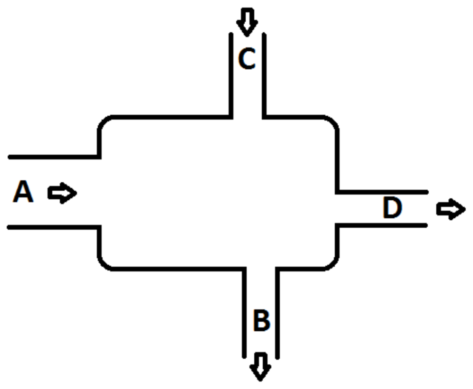

Figure 0.8: Shared junctions, area is shared between units coming from A, going to B and units coming from C, going to D

Figure 0.8 shows part of a map with a junction that two paths share. Suppose that there is a path from gate A to B and another one from C to D. Capacity of A is higher than B so, if units start to fill edge A with its maximum capacity, they cannot go into the B with the same speed and formation. Some units have to wait at the B entrance and by the time they will form a queue at the entrance and it gets bigger and bigger. After a while, units, which are waiting in the queue, will block the entrance of gate D and it means they are cutting the path from C to D. If there is a unit with a path from C to D, it cannot pass and will have to wait until almost all of the units of path from A to B pass.

In some cases, if the queues form at beginning points, the total time to path will be lower. By forming the queue at the start point of a path, we will give this opportunity to the units of other groups to use a partial capacity of shared junction and therefor the total time to pass will be less.

Conclusion: capacity of a path is equal to capacity of the edge with minimum capacity.

1.9 Summary

Pathfinding algorithms can be divided into two methods, centralized and decoupled. Centralized methods search to find paths for all units at a time where Decoupled methods calculate the path for each individual unit separately and then try to solve potential collisions in the map. These methods and their differences were reviewed in this chapter.

Several pathfinding algorithms were reviewed in detail in this chapter. Dijkstra, Maximum Flow algorithm, A*, Local Repair A* and Cooperative A* are some well-known pathfinding algorithms which were discussed. Dijkstra and A* try to find shortest path in a graph. Maximum flow algorithm just considers how to send the units through the map to reach a steady state of fully occupied paths regardless of the length of paths. Local repair A* works like A* unless when two units are near to collide, then algorithm will recalculate the pathfinding for those units. Cooperative A* uses A* for pathfinding but units have knowledge of other unit’s position at each time. To implement cooperative A* we need to have a time-space table.

In chapter two, we compared navigation meshes and waypoints, two different infrastructures for implementing pathfinding algorithms on. Characters which use

NavMesh for pathfinding, have smoother paths where placing strategic points is an advantage of waypoint system.

At the end of this chapter, capacity of a path with different edges was studied. It was mentioned that capacity of a path is equal to capacity of the edge with minimum capacity during that path.

In next chapter, we discuss about shared passages and how to divide it between paths which want to share the shared passage capacity.

Group Pathfinding in Shared Passages with Different Capacities

Some terrains have passages that lead to two or more paths (junction). Figure 0.1 is an example of such junctions.

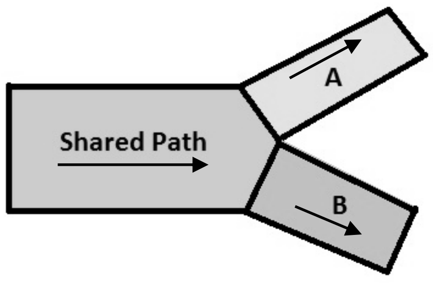

Figure 0.1: Paths A and B sharing one passage

The map in Figure 0.1 has a shared passage and two paths; it is like a 3way junction. All the units that want to move into those two paths, path A and path B, should pass through shared path. Shared path has a limited capacity. It is good to virtually divide the shared path into two separate subpaths (Figure 0.2). We can distribute the capacity between these two subpaths in some proportion. Giving different ratios to these subpaths will result in different total time for units at the end.

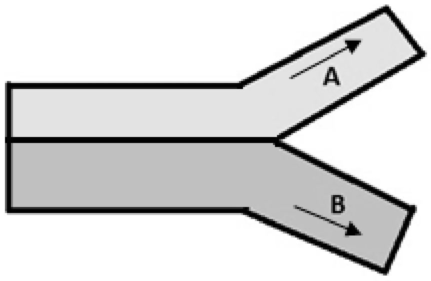

Figure 0.2: Shared path is divided into two different dedicated paths

Consider that all the units have the same speed and size, specified, say, by diameter. The "diameter" is used because, there is a cylinder shape bounding box collider around each unit to detect collisions. Each path has length, capacity and time-to-pass (TTP).

$$Time\ To\ Pass\ (TTP) = \frac{Path\ Length}{Unit\ Speed}$$

Suppose that, after running the pathfinding solution, each path will have a Path Portion. Path Portion is the number of units assigned to a path. Because of the limited capacity, in some points, queues will be formed and units will have to wait for a free spot to go through that path. Waiting time for each unit u is equal to:

$$Waiting\ Time\ for\ Unit\ u = (\text{number of units waiting in queue before u}) * \frac{Units\ Diameter}{Unit\ Speed}$$

Instead of a path with capacity of C, it is possible to consider that path has C different paths with capacity of one.

Now we have the exact number of units waiting on each line of paths with capacity of one. For the first unit in each line, there is no waiting time, so time to pass the path for that particular unit is equal to TTP, but for all other units time-to-pass is equal to TTP plus waiting time for that unit. The "Total waiting time" is equal to the number of units waiting in queue, except the first unit, multiplied by the time that is required to make a free room for the one unit:

$$Total\ waiting\ time = number\ of\ units\ assigned\ to\ that\ path(Path\ portion) * \frac{Unit\ Diameter}{Unit\ Speed}$$

Total time-to-pass for all of units is:

$$Total\ Tim\ to\ pass\ for\ all\ units = Waiting\ time\ for\ last\ unit + TTP$$

At this point, Path Portions for paths A and B are defined. Let us call them Path Portion A and Path Portion B. In addition, there is a ratio for shared path. Ratio for path A is Ratio A and ratio for path B is Ratio B. Units who want to pass through path A can use Ratio A of Capacity of shared path and it is Ratio B for units assigned to path B. Obviously, Ratio A plus Ratio B is equal to the Capacity of shared path.

If we divide Path Portion by ratio of each path, numbers of units that are waiting in each queue of path are calculated as below:

$$Number\ of\ units\ in\ each\ queue = \frac{Path\ Portion}{Ratio\ of\ Path}$$

Having the number of units in each queue, we can calculate total time to pass. Subsequently, different total time to paths can be calculated based on different ratios.

1.9.1 A Sample Shared Passage Problem

For this example, consider a group of 1000 units with the same diameter of 0.5 meter and speed of 10 meter/second, which wants to pass through the map. Let the waiting time for one unit to find a free spot in the path is 0.05second.

$$Waiting\ time\ for\ one\ unit = \frac{Diameter}{Speed} = \frac{0.5}{10}$$

As Figure 0.1 shows, there are two paths, A and B and also a shared passage. Therefore, there are two possible routes, one starts from shared path and goes through path A (R1), the other one starts again from shared path and goes through path B (R2). The word “Route” refers to two or more paths, which starts from start point to destination point.

Table 0.1: Description of the sample path

Route name	Route length	TTP	Path Portion
R1	40	4	400
R2	70	7	600

1.9.2 Static Ratios

Suppose that capacity of shared path is 10. If we want to send all 400 units of route R1 from the shared path and only use 20% of shared path's capacity, it will take 13.95 seconds. Doing the same calculation for different capacities (ratios) will show the results of Table 0.2. For example, calculations for 400 units of route R1, when it uses 40% of shared path, are computed in below. Similar calculations have been used for other capacities.

1) Divide Path Portion by ratio -> 400/4 = 100
2) Calculate waiting time for last unit in the line -> (100-1)*0.05 = 4.95
3) Sum waiting time for last unit and Time to pass -> 4.95 + 4 = 8.95

Table 0.2: Different ratios of shared path and time to pass for Routes R1 and R2

Capacity	R1	R2
1	23.95	36.95
2	13.95	21.95
3	10.65	16.95
4	8.95	14.45
5	7.95	12.95
6	7.3	11.95
7	6.85	11.25
8	6.45	10.7
9	6.2	10.3
10	5.95	9.95

Figure 0.3 shows different time to pass for different shared path ratios.

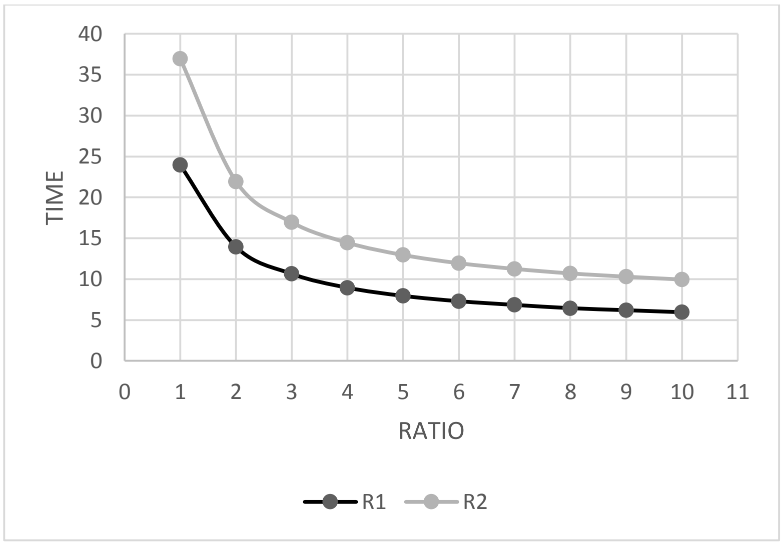

Figure 0.3: Different ratios of shared path and time to pass for routes R1 and R2

Consider a shared path with capacity of 10. We can give 1/10 to units who want to go to route R1 and 9/10 to units going to route R2. Call this ratio (1-9). There are 400 units for

R1 with capacity of one; it means that it takes 23.95 seconds for all the units to pass R1. In addition, there are 600 units for route R2 with capacity of nine. There will be 67 units in each line of route R2. It takes 10.3 seconds. By the ratio of (1-9), the total time to pass will be maximum of 23.95 and 10.3 that is 23.95 seconds. Doing the same calculations for other ratios will give result reflected in the Table 3.3. Figure 0.4 also shows results in a chart.

Table 0.3: Different ratios and time to passes

Ratio	
A - B	Time
0-10	15.9
1-9	23.95
2-8	13.95
3-7	11.25
4-6	11.95
5-5	12.95
6-4	14.45
7-3	16.95
8-2	21.95
9-1	36.95
10-0	15.9

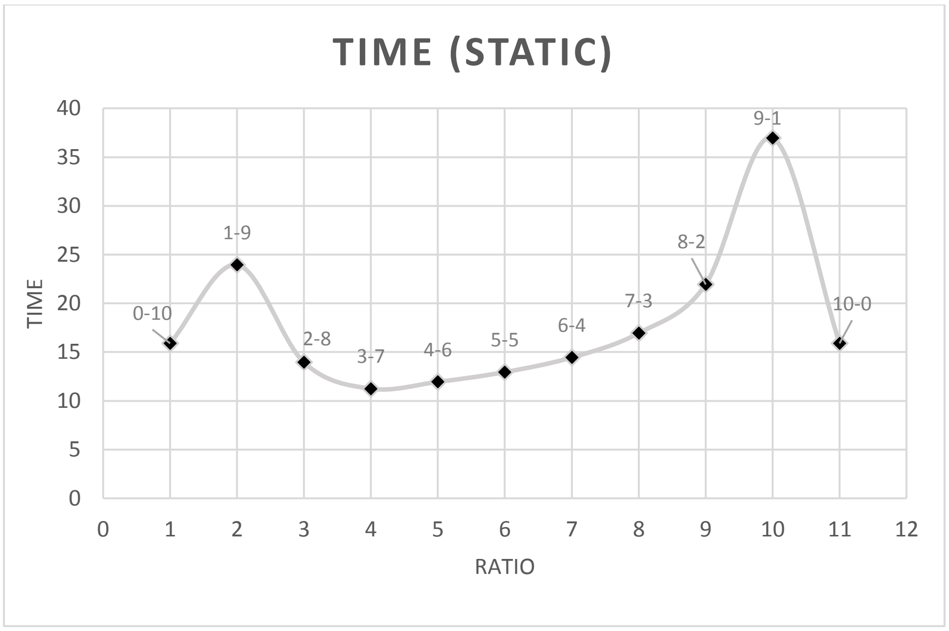

Figure 0.4: Different ratios and time to passes

Ratio of (0-10) means that all shared path's capacity is dedicate to units who want to go to path B and then when all of them are entered into shared path, 100 percent of the shared path is dedicated to units of path A.

Calculated time for each ratio is the maximum time to pass of units through R1 and R2 with that ratio. In this case, the best ratio is (3-7) with total time of 11.25 seconds.

1.9.3 Dynamic Ratios

In this case, again a portion of shared path is assigned to each path (A and B). From the beginning time, (t0) units start to go into shared path. After a while, in time (tx) all the units who want to go to one of the routes are placed in shared path. At this time, there are no more units for one of the routes and we can give 100% of the shared path to the other one. This method of sending units into the map is called dynamic path ratio. Total time to pass with dynamic path ratio is shown in Table 0.4.

Minimum total time to pass is 9 seconds and it is repeated in to ratios.

Table 0.4: Different ratios and time to pass with dynamic approach

Ratio	
R1-R2	Time (dynamic)
0-10	15.9
1-9	9.05
2-8	9
3-7	9.05
4-6	9
5-5	12.05
6-4	12.05
7-3	12
8-2	12.05
9-1	10
10-0	15.9

Results of assigning ratio dynamically are also shown in Figure 0.5.

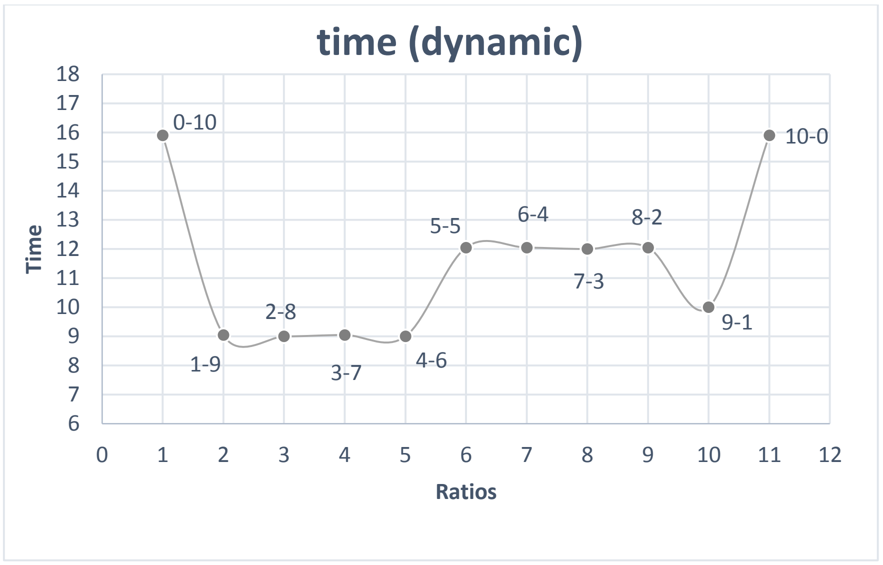

Figure 0.5: Different ratios and time to pass with dynamic approach.

1.9.4 Experimental results

All the experiments were run on an Intel Core i7 CPU with 16 GB RAMS. Unity game engine V4 professional edition is used to implement the experiment.

A group of 500 units with the speed of 5 meter per second and 0.5 meter diameter is used. There are two paths, P1 that has 45 meters length and P2 with 65 meters.

Shared path, path P1 and path P2 are shown in Figure 0.6. All the units start from S and D is their destination.

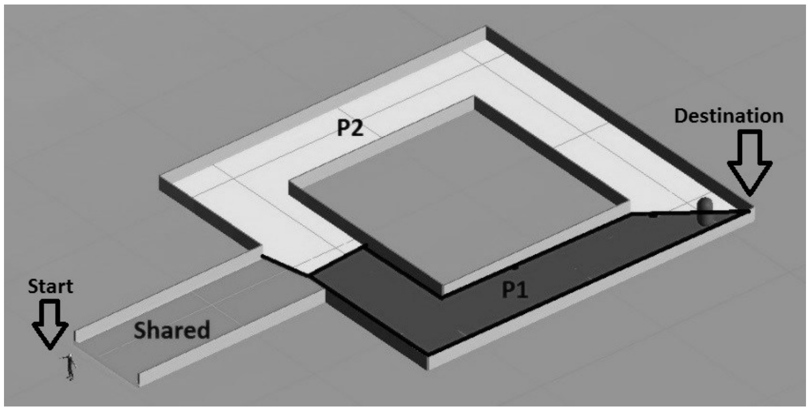

Figure 0.6: Two paths, P1 and P2 have to share one path

Table 0.5 and Figure 0.7 show different times for R1 when we use different ratios both for theoretical and experimental model.

Table 0.5: Route R1 using different capacities

Capacity	Theoretical	Experimental
1	38.9	44.0775
2	23.9	38.33
3	18.9	34.1825
4	16.4	31.255
5	14.9	30.5875
6	13.9	29.66
7	13.2	28.88
8	12.7	28.49
9	12.3	28.265
10	11.9	27.6525

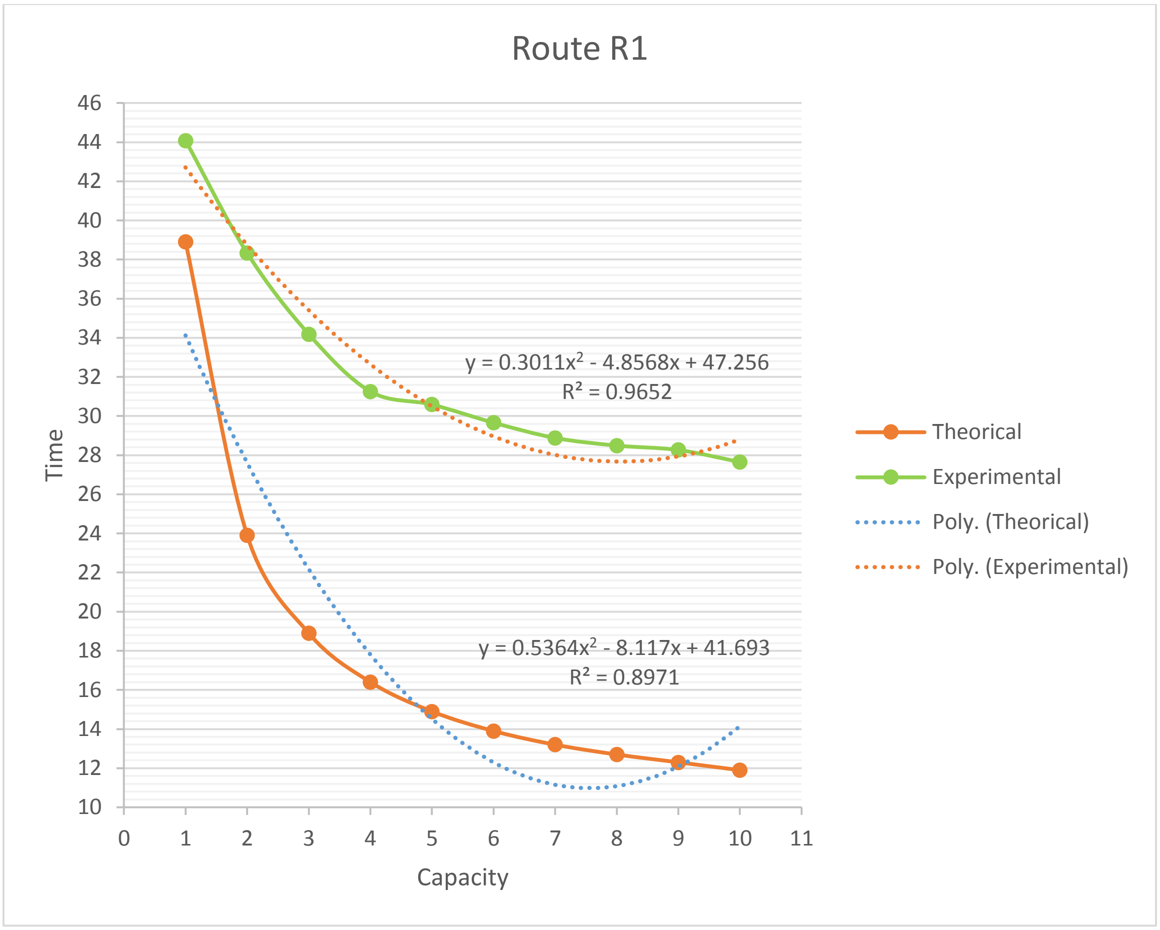

Figure 0.7: Route R1 using different capacities

Below table and chart show different times for R2 when we use different ratios both for theoretical and experimental model.

Table 0.6: Route R2 using different capacities

capacity	Theoretical	Experimental
1	32.9	42.12
2	22.9	38.8075
3	19.6	35.715
4	17.9	34.565
5	16.9	33.6
6	16.3	33.1675
7	15.8	32.4375
8	15.4	31.9125
9	15.2	31.725
10	14.9	31.6425

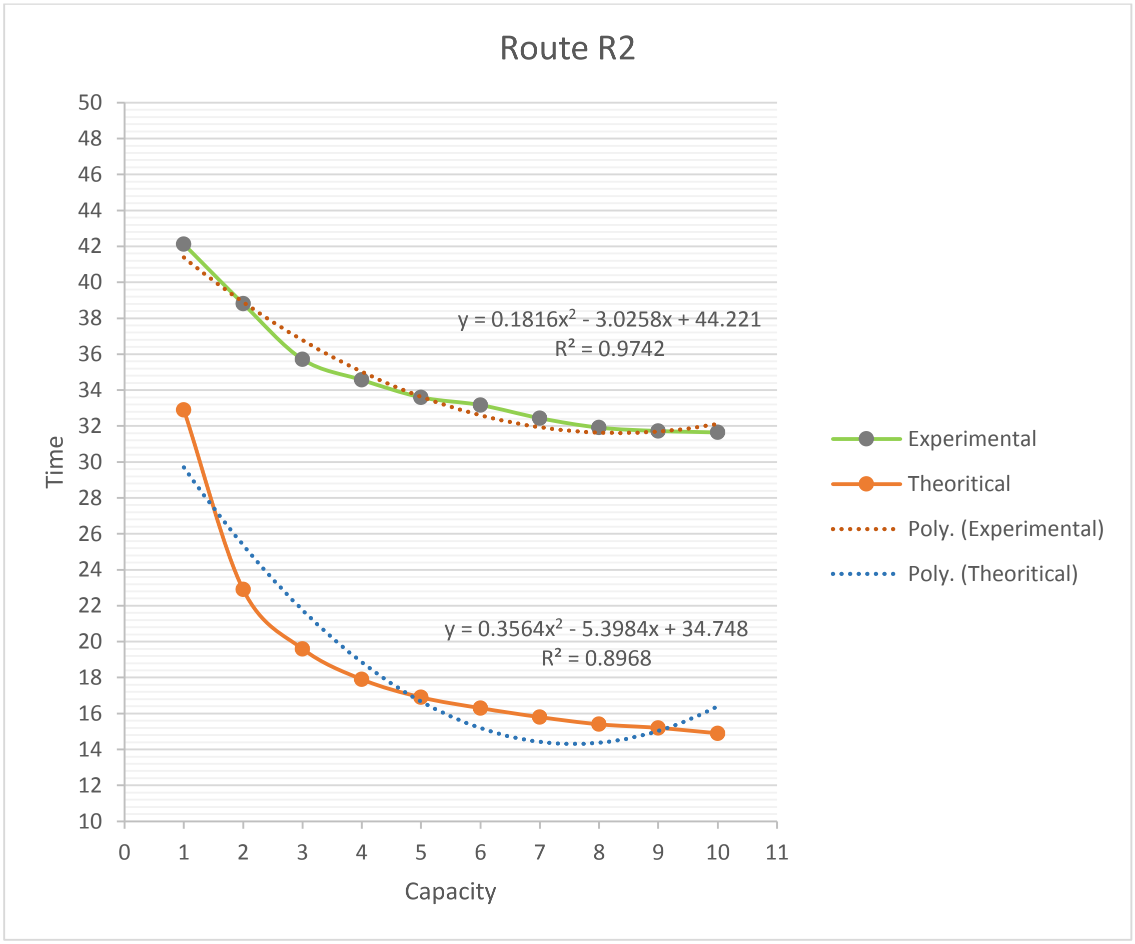

Figure 0.8: Route R2 using different capacities

1.9.5 Three routes share a passage

We performed the same experiment but this time with three routes. R1 is 40, R2 is 70 and R3 is 90. Unit speed is 10 and diameter is 0.5. 1300 units has been used and Path Portion for R1 is 400, R2 600 and R3 is 300 units.

Below we can see the tables and charts.

Table 0.7: Routes R1, R2, R3 using different capacities

	Time		
capacity	R1	R2	R3
1	23.95	36.95	23.95
2	13.95	21.95	16.45
3	10.65	16.95	13.95
4	8.95	14.45	12.7
5	7.95	12.95	11.95
6	7.3	11.95	11.45
7	6.85	11.25	11.1
8	6.45	10.7	10.85
9	6.2	10.3	10.65
10	5.95	9.95	10.45

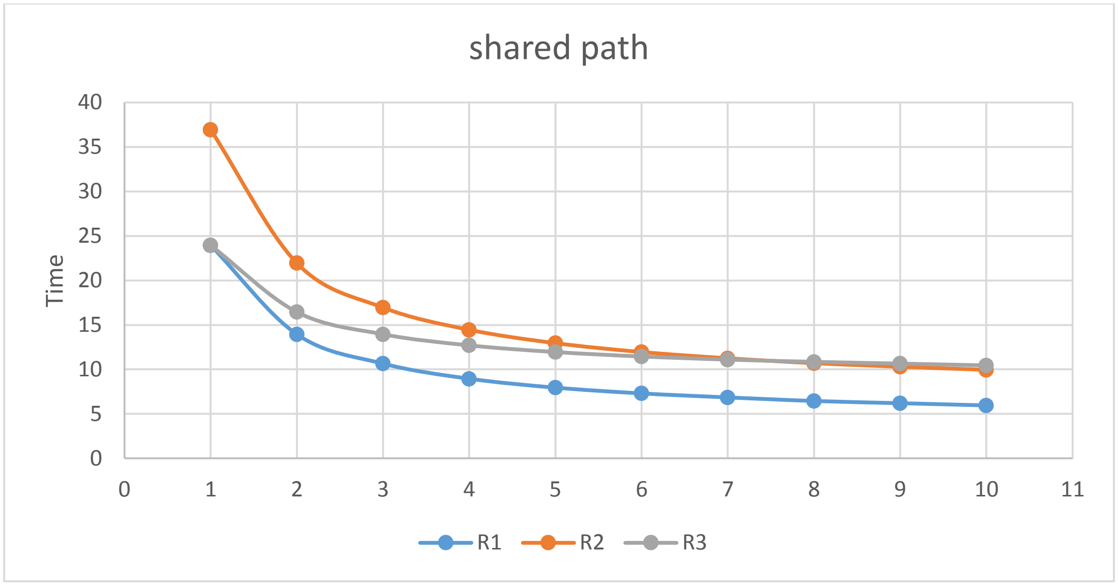

Figure 0.9: Routes R1, R2, R3 using different capacities

Here is the table if we send units through routes with fixed capacity.

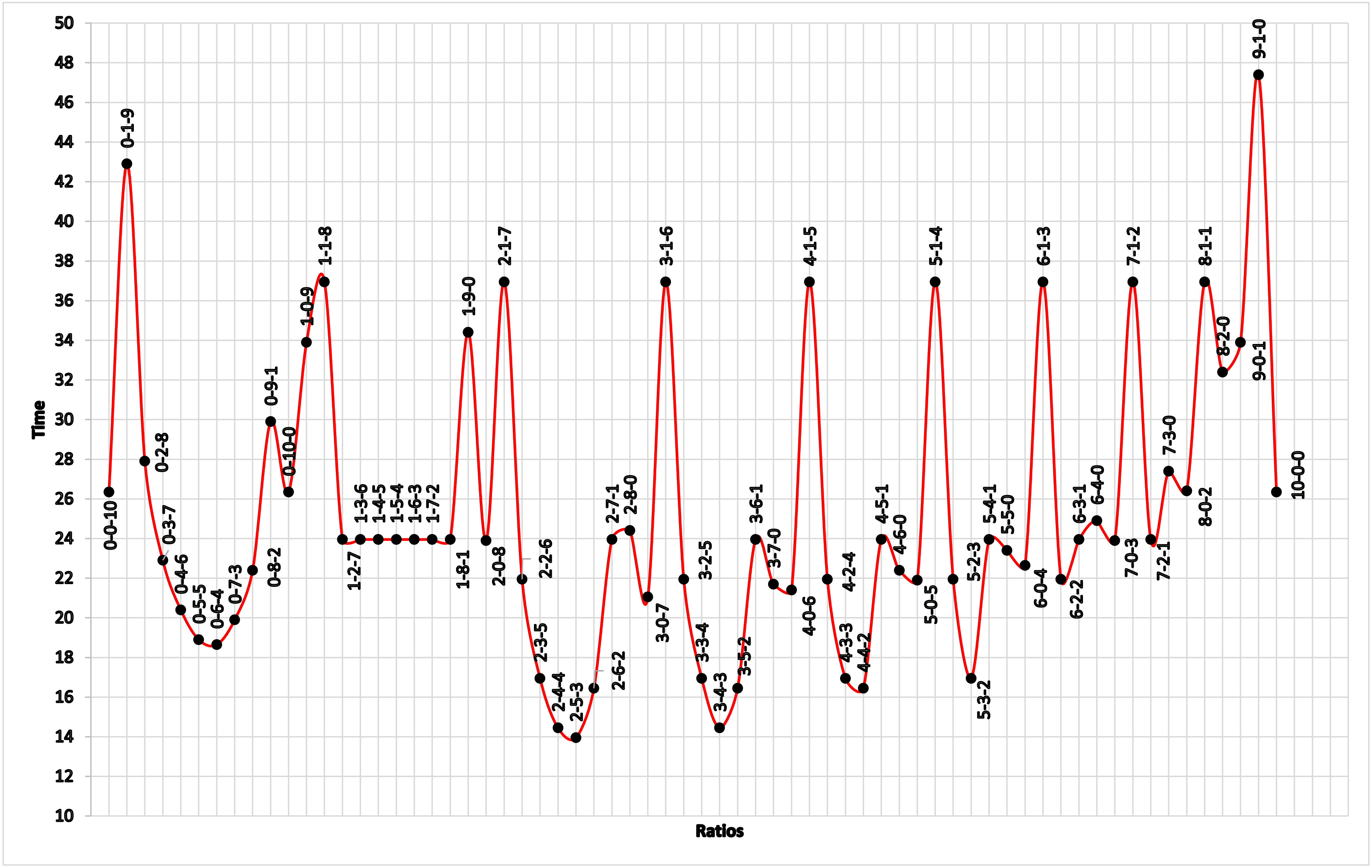

Figure 0.10: Units go through routes R1, R2, R3 using different ratios

Reduced Wait Time (RWT) algorithm

1.10 Map Generator

To compare different algorithms, maps with multiple paths and different capacities are needed. For sure, a terrain with a single path is not useful. In addition, it is good to test different algorithms on different map sizes with different statuses. Every terrain can be represented as a graph, where intersections are vertices and passages are edges. Graph of a terrain can be fully connected (complete graph) or semi-connected or even disconnected but at least there should be one path from start point to the end node of the graph, otherwise that maps is useless.

A random map generator is used to generate sample maps to test the algorithms and compare them. The map generator creates random maps in Unity 3D game engine. Map size, minimum number of available paths between start and destination point and chance of having more free passages are customizable. Every time that run the application it asks about customizable items of the terrain and builds a map based on your needs. It is possible to save the map to use the same one for other algorithms. Figure 0.1 shows a random map with different paths from start point to destination.

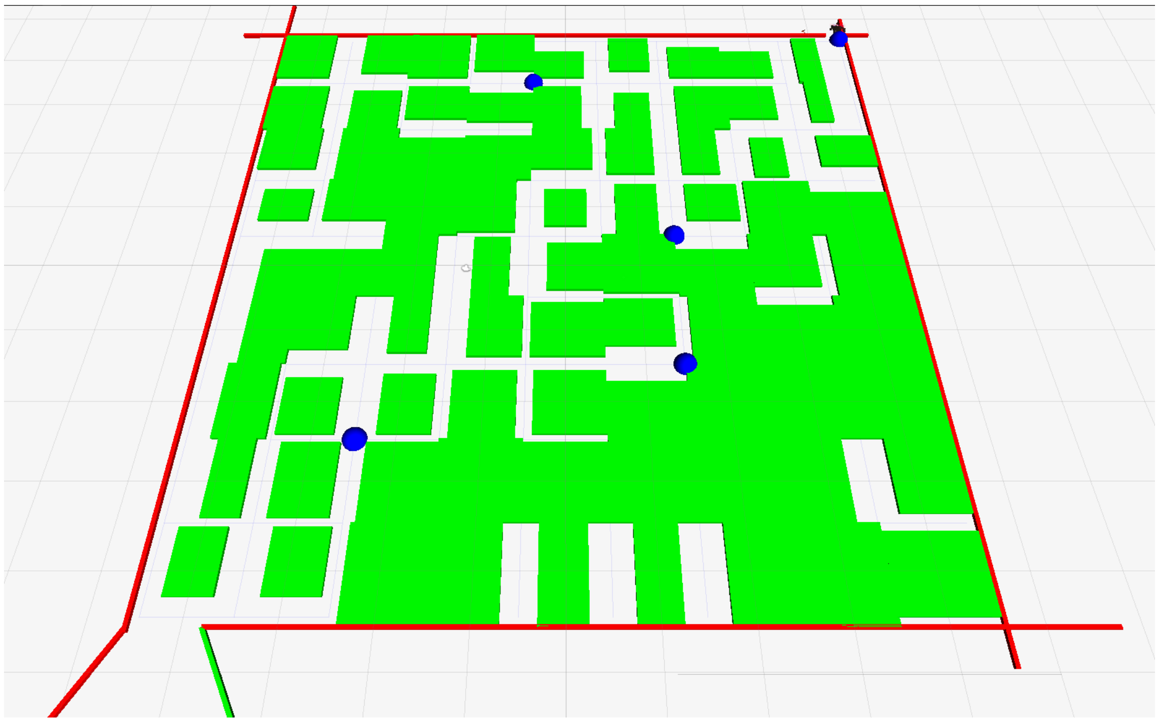

Figure 0.1: Randomly generated map. Green blocks are obstacles and white spaces are walkable areas

1.11 Map generator algorithm

First of all, map generator creates an empty terrain base on inserted map size. Then it marks a random path from start to destination and set the edges of that path as walkable (there is no obstacle there). To make the paths more randomized, map generator sets some random middle points, then it will find a path from start to middle point and then from middle point to destination point. Number of middle points is selectable by user but middle points will be placed randomly in the terrain. Finding a random path from start to destination begins from start point and ends in destination point. At each point (vertex of graph) in the map, there are three possible directions to take. Two of them will link you to closer points to the destination and one direction links to a point with more distance from destination. Choosing the direction to go is random but with different chances.

Chance of going to nodes that are closer to the destination should be more, otherwise we will have lots of going forward and backwards and result is a path with lots of loops in between. Figure 0.2 shows such a map with lots of loops.

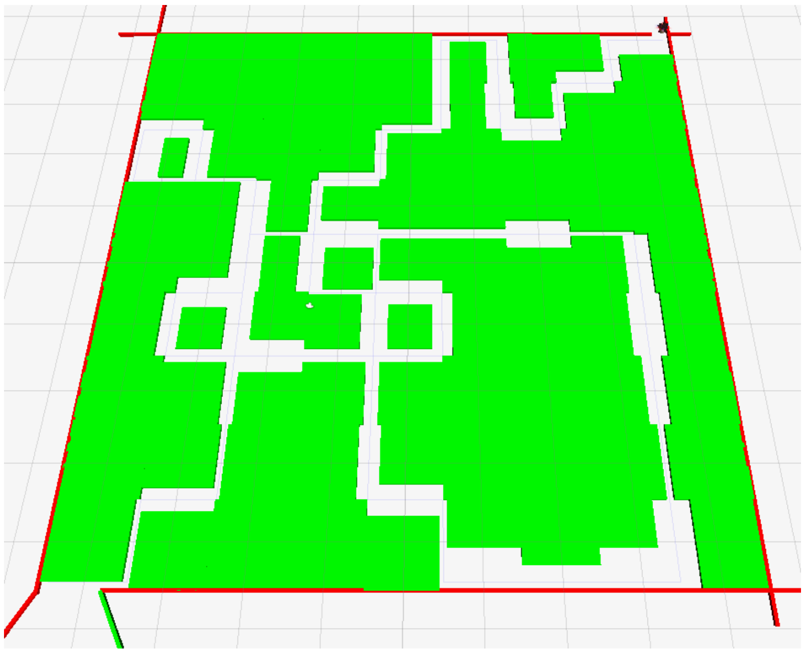

Figure 0.2: Randomly generated map with one path from start to end and many loops during the path.

At his point, we have some marked paths. Then, application starts to block the edges unless the edge is part of marked path. Also based on "chance of having more free passages" variable, that user sets at the beginning, some edges will remain free, considering the chance factor and random positions.

Now, it is time to assign a capacity to each edge of map that is walkable (not blocked). Each walkable edge will receive a random number as capacity. Assigned capacity will change the wideness of passages. Figure 0.3 shows a complete map with some units passing through it.

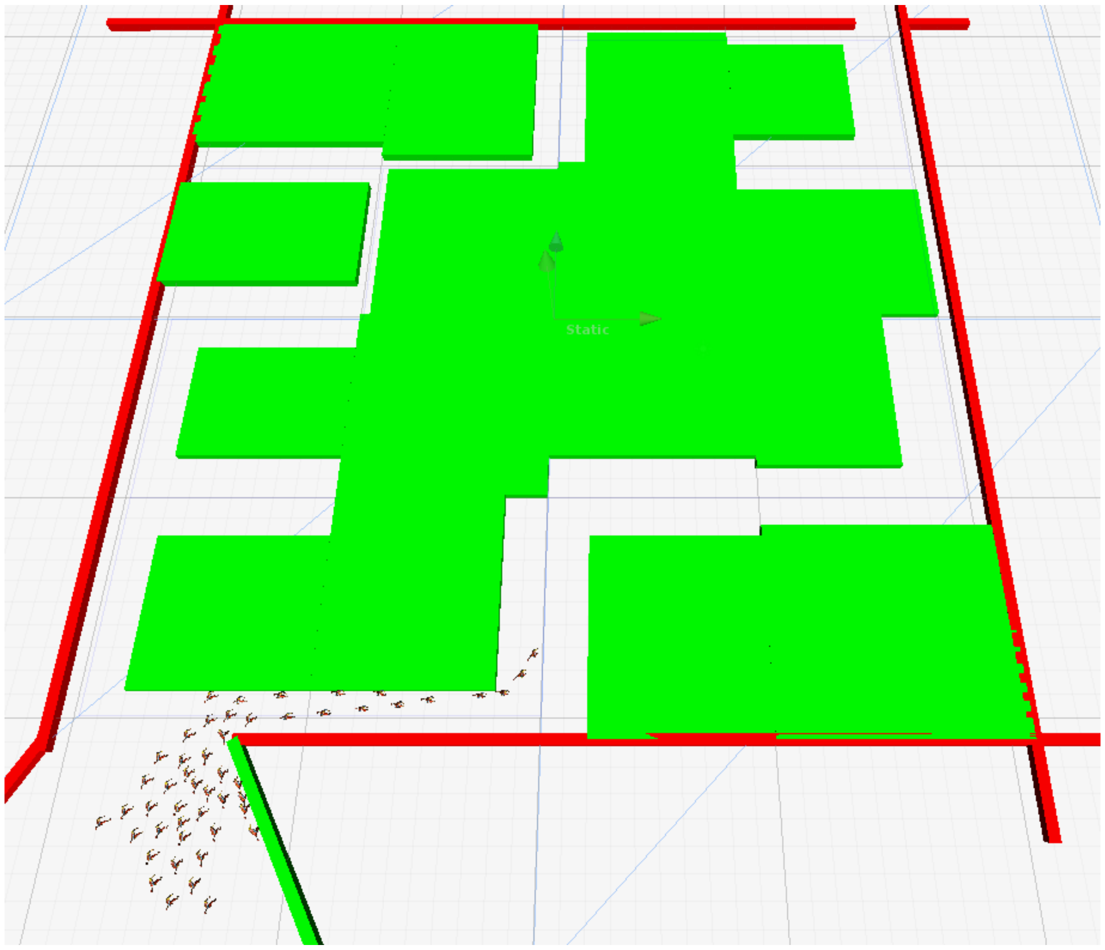

Figure 0.3: Random map with moving units

1.12 Reduced Wait Time (RWT) Algorithm for Group Path Finding

To minimize the time to pass factor, RWT algorithm divides the group of units into subgroups at some points during their routes in the terrain. Units can take different routes toward the destination to decrease wasted time of being in queue (waiting time).

Consider a terrain with many different paths. It can be considered as a graph with edges as paths and nodes as junctions with three (or more) paths meeting at the junction. Each edge has a capacity and length. There may be several different routes from start node to destination, each one with different capacity and length. The capacity of a route is the minimum capacity of all edges during that path. We can sort all the routes from source to destination by length. Instead of sending all the units from the shortest route, the RWT algorithm will divide the given group of units into smaller subgroups and send them through alternative routes (second shortest, third shortest and so on). This way it will decrease the waiting time of being in queue. Therefore, time of reaching the destination by the last unit becomes less. A group needs to be divided in such a way that the time to send a unit via an alternative route becomes less or equal to the time of sending that particular unit via the shortest route, after considering waiting time. That is, we have to take it to account the overhead of waiting in queue by adding it to the time to pass for a unit via the shortest route.

Each route has a length L. Unit's speed S is equal to maximum speed that the unit can move. Time to pass through a route TTP is equal to L/S.

$$Time\ to\ Pass\ (TTP) = \frac{Length\ (L)}{Unit's\ Speed\ (S)}$$

Units have a cylinder shape, bounding box to determine collisions, so each unit has a diameter D.

Because of the limited capacity, at some points, queues will be formed and units will have to wait for free room to go through that route. Waiting time (WT) for each unit u can be calculated as below:

$$Waiting\ Time\ for\ unit\ u = Number\ of\ units\ waiting\ in\ queue\ before\ u * \frac{Unit\ Diameter}{Unit\ Speed}$$

Time needed for a unit u to pass through a route r is equal to WT_u+TTP_r.

Suppose that there are only two routes, R1 and R2. TTP_{R1} is 10 seconds, TTP_{R2} is 20 seconds, and both routes have capacity of just one unit. If the waiting time for one unit is 1 second, it takes (2*1) +10 seconds to pass three units through R1. Therefore, the total time we need to pass three units through R1 is 12 seconds.

Sending just one unit through R2 takes 20 seconds. We can conclude that, if we have a small number of units there is no need to send them via R2 unless the waiting time for a unit who wants to go via R1 is equal or more than 10 seconds that means at least we need 11 units for R1. Therefore, there should be a certain minimum number of units after which, use of the second route becomes faster. We call this number **threshold** of taking second route. If the total number of units is greater than the threshold, it makes sense to use second shortest route.

1.12.1 The RWT algorithm

First of all, RWT algorithm needs to have information about all the routes from start to destination point. To obtain this information, Dijkstra, A* or any other pathfinding algorithm can be used. At this point, RWT algorithm takes shortest route and decreases the capacity of all the edges along that route by the route capacity (capacity of the edge with minimum capacity). The same will be done with other routes. It finds second shortest route, decreases the capacity of all the edges along that route by the route capacity and repeats this step until there is no more available routes from start to destination. After this, RWT has a table with a number of routes from start to destination. Routes are sorted based on length.

The next step in RWT starts with putting one unit in the route with maximum TTP (Time to pass). Total time will be equal to TTP_{max}. Now RWT algorithm can calculate thresholds for other routes then calculate Path Portions for all the routes.

Summation of all the Path Portions can be less than total number of units who want to pass the map. In this case, solution should assign the remaining units equally to all the routes. The reason why RWT divides all the remaining units among all the routes equally is that there is at least one unit in each route and if we add one more unit to any route total time to pass for that route will be previous time to pass plus waiting time for one unit.

1.12.2 Reduced Wait Time (RWT) Algorithm for sending N units in Pseudocode

Potentially there are n different routes from start node Start to destination node Destination in game world.

```
1.1  i=0
1.2  Until there is no path from Start to Destination Do
1.2.1    Apply A* algorithm to find the shortest path from Start to Destination.
1.2.2    i = i +1
1.2.3    Add this path i, its length $L_i$, and $TTP_i$ into the paths table.
```

1.2.4 Decrease the capacity of each individual edge along path i by the capacity of the edge with the minimum capacity of any edge along the original path.

1.3 n = i

1.4 At the end of Step 1.2, there will be a table of paths, which is sorted based on L_i for paths i = 1 to n and respectively TTP_i from minimum to maximum TTP.

1.5 Select the path n from the table (this path has also the maximum time to pass, TTP_n).

1.6 The number of units that are scheduled to pass along path i = 1 to n is

$$\left(\frac{(TTP_n - TPP_i)}{(\frac{D}{S})}\right) + 1$$

where D/S is the length of the path required for one unit. The above value is placed in the table for path i as Path $Portion_i$. This is the number of units to be sent initially along path i.

1.7 The total number of units that are scheduled to be sent before the solution reaches a max flow states is the sum of all the Path $Portion_1$ to Path $Portion_n$ together. After Path $Portion_i$ is send along path i, an equal number of units from the remaining units are scheduled for each path:

$$\frac{(N - \sum_{i=1}^{n} Path\ Portion_i)}{n}$$

This will be the final Path Portion for each route.

1.12.3 Using RWT Algorithm on an example map

In this example, 100 units with speed of 5 meter per second are positioned at start point and want to go to destination. Unit's diameter is 0.5. There are three routes. Capacity of route 1 is one, capacity of route 2 is one, and capacity of route 3 is two. Instead of considering capacity of more than one for a route, it can be assumed to be two routes with capacity of one.

Doing the first phase of pseudo-code will give the results which are reflected in the following table.

Table 0.1: Specifications of sample map

	Length	Capacity	Time to path
R1	30	1	6
R2	40	1	8
R3-1	50	1	10
R3-2	50	1	10

Sending the one unit via R3 takes 10 second for the unit to reach the destination versus sending a unit via R1 that takes only 6 seconds. If we send 41 units to R1, it takes 10 second for them to reach the destination. All the required calculations are as follow:

WT_{R1} = (number of units should in queue that is 40) * (D/S that is 0.1)

Total time-to-path for 41 units is $WT_{R1}+TTP_{R1}$ that is 4+6=10.

We can conclude that, if we send 41 units through R1 and at the same time send just one unit through R3 all of them will reach the destination at the same time. This time is less than sending all 42 units through shortest route that will be 10.1 seconds.

Doing the same, calculations will give us the result in below table for Path Portions.

Table 0.2: Path Portions

Path Name	Relative Path Portion
R1	41
R2	21
R3-1	1
R3-2	1

Path Portions of R1 + R2 + R3 are 64 units. Total number of units is 100. Therefore we have (100-64) 36 more units. We should divide this number by 4 and add it to each route. We need to add 9 more units to each route. Total number of units should pass through the routes is shown in Table 0.3: Unit portions of each path.

Table 0.3: Unit portions of each path

Path Name	Total Path Portion
R1	50
R2	30
R3-1	10
R3-2	10

If we send all 100 units with the portions mentioned above table, the total time to pass will be 10.9. This time is 5 seconds less than sending all the units form shortest path.

Simulations, Results and Discussion

In this section, RWT algorithm is compared with several other solutions in terms of time, traveled distance, number of collisions and waiting time.

RWT is compared with the other algorithms using a sample terrain, which was randomly generated with different paths and various capacities. As shown in Figure 0.1, the proposed map is maze shape terrain with single source and single sink, called S and E, respectively. S and E are considered as entrance and exit portals.

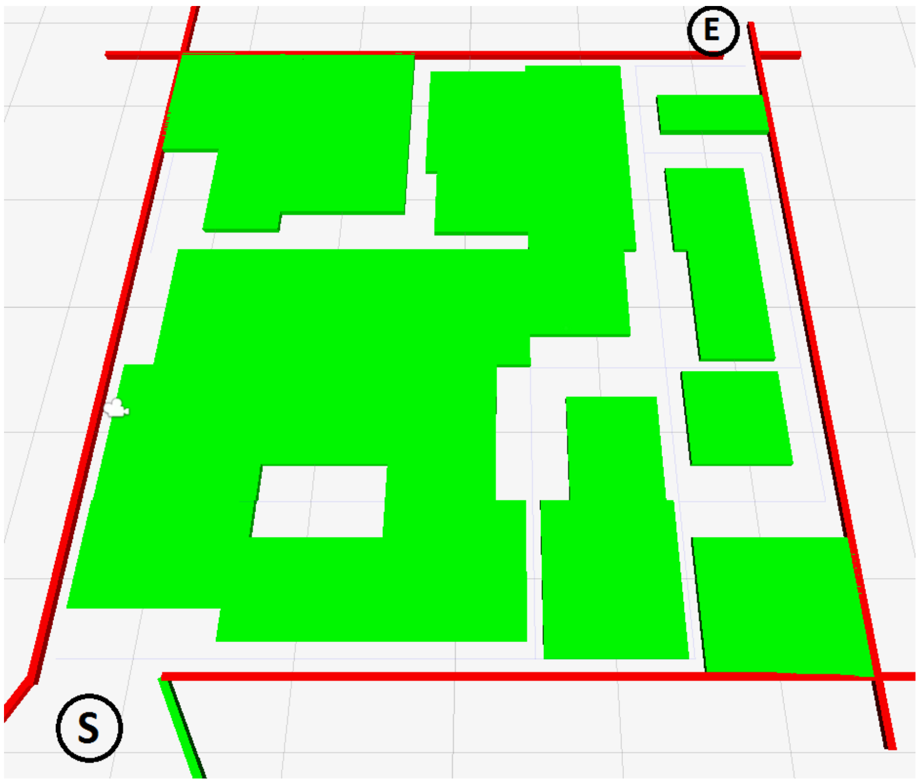

Figure 0.1: Sample terrain with one source point, S, and one destination point, E

The units are initially placed behind the entrance portal (S). These units were then sent through the map in order to pass the destination portal (E). Units will do the path finding and path following strategies based on the chosen algorithm.

A graph can be mapped to each terrain. Showing a map by graph is much easier to understand. Paths of the map are edges of the graph and intersections which connect 3 or more edges are vertices. Figure 0.2 shows a sample map and its related graph.

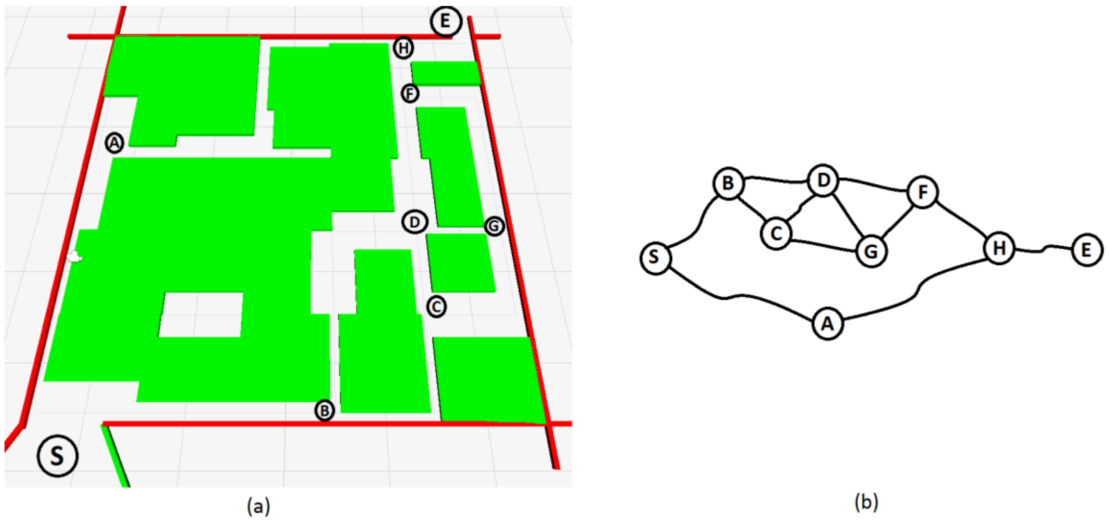

Figure 0.2: (a) Sample map and (b) its relative graph

1.13 Sample Map

A sample map is generated to test all the algorithms in a similar environment. Map is randomly generated with random values for lengths and capacities.

Table 0.1 shows all the edges and their relative length and capacity. Map has 7 vertices and 11 edges with different lengths. In the second row of the table, you can see the term

S > A which means there is a route from vertex S to A and its length is 10 meters with the capacity of 4. Capacity of 4 means, 4 units in a row can pass through this path.

Table 0.1: List of edges with their length and capacities

Route with start and End edges	Length	Capacity
S > A	10	4
S > D	15	4
A > B	12	1
A > C	10	1
B > E	10	3
C > B	13	2
C > E	17	1
C > F	12	1
D > C	15	2
D > F	10	2
F > E	15	2

Based on the graph there are eight different routes from start vertex, S to the end vertex E. Table 0.2 shows all the possible routes and their length, capacity and time to pass.

Table 0.2: All the paths from Start to Destination with their length and capacity

Route name	Length	Capacity
R1	32	1
R2	43	1
R3	37	1
R4	47	1
R5	53	2
R6	47	1
R7	57	1
R8	40	2

There is another property for each route called Time to Pass, TTP. Time to pass means how long it will take for a single unit to go from start to end using that route. Obviously, TTP is different from route to route. TTP is related to unit’s speed and route length. In this example, units are moving with the speed of 5 meter per second. A cylinder collider surrounds units. The cylinder’s diameter is 0.25 meter. Considering the unit’s speed, we add another column to the Table 0.2 that shows time to pass, TTP, for each route from S to E. Table 0.3 shows all the routes from S to E and needed TTP of those routes.

Table 0.3: All routes from S to E with their Time to Pass

Route name	Length	TTP
R1	32	6.4
R2	43	8.6
R3	37	7.4
R4	47	9.4
R5	53	10.6
R6	47	9.4
R7	57	11.4
R8	40	8

Time to pass for route R1 is 6.4 seconds; it means that if there is one unit positioned at S that wants to go to E using route R1, it takes 6.4 second.

Next for comparing RWT with other, we run different solutions on the sample map. Various parameters have been monitored in this experiment.

Compared solutions are:

- A*
- LRA*
- Maximum Flow
- RWT

Studied parameters are:

- Total time needed to pass all the units

- Distance traveled by the units
- Waiting time for all the units
- Total number of collisions between units

A* algorithm only use one route, the shortest one. Then units will be sent through that route. The shortest route in this example is R1 with 32 meters length with capacity of one. There are some edges in R1 with capacity of more than one but, as mentioned before, capacity of a route is equal to capacity of the edge with minimum capacity along that route. It takes 6.4 seconds for one unit to pass through R1 and reach the destination. Suppose that there is more than one unit to pass the map. Therefore, there will form a queue at the entrance portal. The first unit will pass without waiting in the line but for other units there will be a waiting time. Total time to pass for second unit is equal to time to traverse the route plus waiting time. As mentioned earlier in this section, waiting time for one unit is diameter divide by speed (D/S). In this example, diameter is 0.25 and speed is 5 so, waiting time is 0.05 second.

Maximum Flow uses R8, R6, R5, R2 and R1. Then it divides total units and sends them equally to these five different routes.

Local Repair A* works almost like A* but when two units are near to collision they re-rout and try to find another free route.

RWT uses R1, R3, R6 and R8. RWT will divide units into those 4 paths but unlike Maximum Flow, the portions are not equal. RWT and its division algorithm are well explained in previous chapter.

Table 0.4 compares these algorithms and relative routes.

Table 0.4: Specifications of different algorithms

Solution	Number of Routes	Path Portions	Total capacity of routes
A*	1	All the units	1
LRA*	N/A	N/A	Between 1 to 6
Maximum Flow	5	(Total number of units) / (total capacity of paths)	6
MySolution	4	Should calculate	5

1.14 Test results and discussions

1.14.1 Experimental Methodology

A randomly generated map has been used to test and compare all the algorithms. Each algorithm has been tested several times when the number of units was increasing from 150 to 1905 in increment of 150.

Each time, we select the number of units, position them at the start point and run the program for five times. The final result for each selected number of units is average of those five runs.

1. For each of four algorithms do:
2. For N from 1 to 13 do:
    2.1. Place (N * 150) units at start point and run the algorithm for 5 times.

    2.2. Calculate the final result as average of results of five tests in part (2.1)

    2.3. N = N + 150

1.14.2 Time

A group of 180 units wants to travel from Start point to End Point.

(a) A* will find the shortest route which is R1 and send them all through R1. Path Portion of R1 is 180. Total time to pass from R1 for 180 units using A* is 15.36.

$$TTP = ((Total\ \#\ of\ units * Capacity\ of\ shortest\ path) - 1) * (Waiting\ time\ for\ 1\ unit) + Time\ to\ pass\ R1$$

(b) LRA* finds the shortest route, sends all the units through it and waits for collision. Theoretically, LRA* do not let the collision to occur and will re-do the pathfinding for two units who are very near to have a collision. However, in practice some collisions are inevitable, especially when we have a dense area.

(c) Maximum flow divides all the units into 6 and sends them equally through the routes. Path Portion for each path is 30 units. Consider that there are only five routes for Maximum Flow but as it uses one route, R8 with capacity of 2, so total capacity is six. TTP from each route R_x is:

$$TTP_{Rx} = \left((Path\ portion - 1) * Waiting\ time\ for\ 1\ unit\ \right) + Time\ to\ pass\ of\ path\ R_x$$

Path Portion for each route is $\frac{180}{6} = 30$. Table 0.5 shows TTP for all routes.

Table 0.5: Route information when using Maximum Flow

Route name	Length	Capacity	TTP for 1 unit	TTP for all units of path portion
R1	32	1	6.4	7.85
R2	43	1	8.6	10.05
R5	53	1	10.6	12.05
R6	47	1	9.4	10.85

R8	40	2	8	9.45

Although capacity of R5 is two, when Maximum Flow algorithm is used, another route uses a part of capacity of R5 and at this time, there is only one available capacity for R5 to use. It should be mentioned again that routes share some edges with each other.

(d) RWT divides units in a different way (explained in chapter 4). After applying RWT there will be five different routes from S to E. Each route has its own length, capacity and Path Portion. TTP for each route R_x is equal to:

$$TTP_{Rx} = \big((Path\ portion - 1) * Waiting\ time\ for\ 1\ unit\,\big) + Time\ to\ pass\ of\ path\ R_x$$

Table 0.6 shows all routes with their relative Path Portion and time to pass using RWT.

Table 0.6: Route information when using RWT

Route name	Length	Capacity	TTP for 1 unit	TTP for all units of Path Portion
R1	32	1	6.4	9.6
R3	37	1	7.4	9.6
R6	47	1	9.4	9.6
R8	40	2	8	9.6

The results show that the Time to pass(TTP) increase with the increasing of the number of units, irrespective of the path finding strategy, as shown in Figure 0.3.

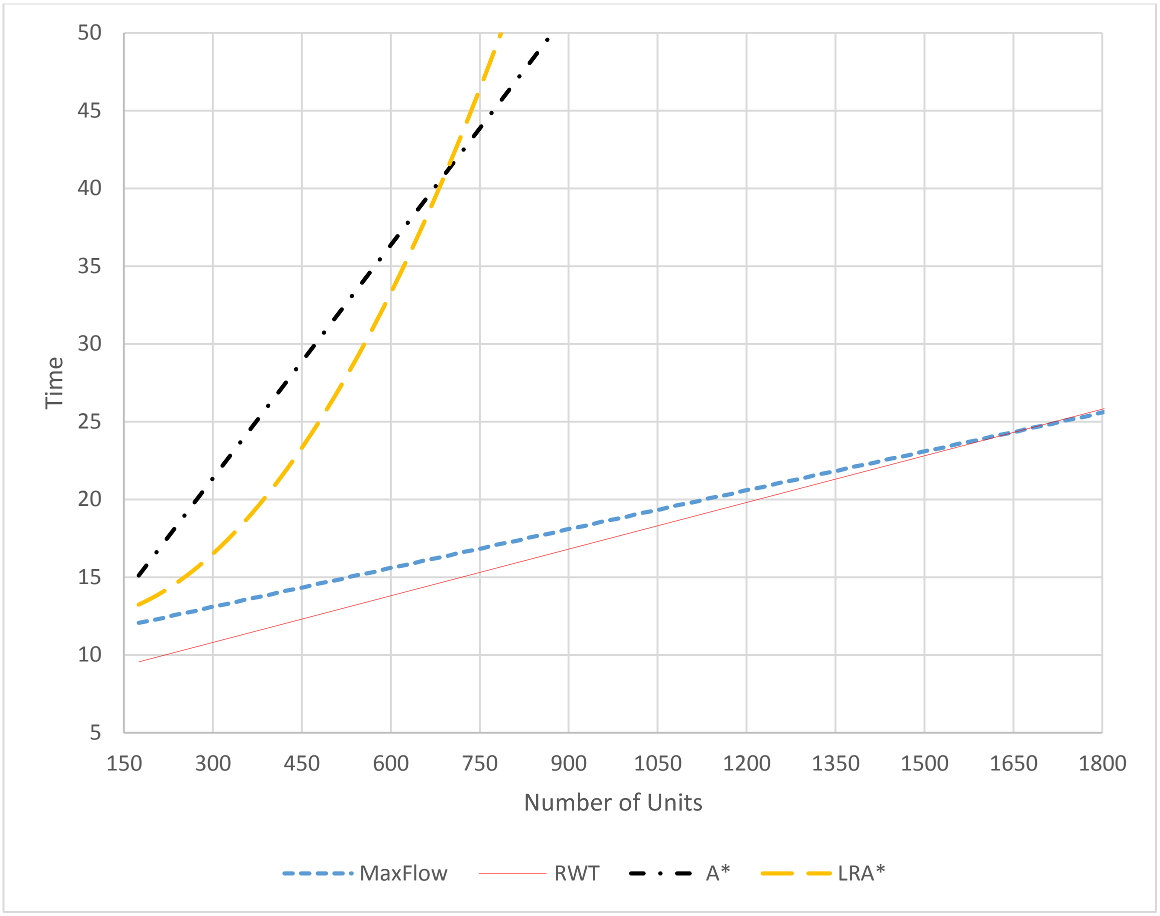

Figure 0.3: Time to pass for different number of units using different algorithms.

In terms of travelling time, A* takes lots of time to send units from S to E. A* finds the shortest route and force all the units to use that one route, therefore there will form a huge queue of waiting units at some parts of the map. Time to pass increases with number of units and for A* it is more than other solutions. It should be mentioned that for a small group of units traveling time with A* and RWT are the same and less than traveling time with Maximum Flow. As it is mentioned in chapter 4, RWT sorts all the routes by length and then selects among them based on number of units; if there is no need to use second shortest route it will not use it. In that case, RWT will use only one route, that is the shortest one, and consequently A* and RWT show the same behavior.

Figure 0.3 shows that it takes more time to complete movement process when Maximum Flow has been used in comparison to RWT, but up to a number of units. If the number of units is more than that number then RWT needs more time. That particular point is named border point. In this example, border point is 1655 units. 1655 units want to go from S to E, does not matter which solution is used, Maximum Flow or RWT; it takes 23.35 seconds. If number of units is less than border point it is better to use RWT, otherwise Maximum Flow works better.

1.14.3 Traveled Distance

Traveled distance is another parameter that has been studied to compare the solutions. Traveled distance is sum of the distance that each unit should travel to reach the destination.

$$Traveled\ Distance\ = \sum_{i=1}^{Total\ number\ of\ units} Distance\ Traveled\ by\ unit\ i$$

As it is mentioned before A* sends all the units from one route, that is shortest one, so A* should have the minimum traveling distance among these algorithms.

Maximum Flow and RWT will have more travelling distance. They divide units and send them through different routes. Obviously traveled distance for these two solutions should be more than A*. RWT has less traveling distance than Maximum Flow. RWT selects shortest route first and second shortest route next and so on. On the other hand, Maximum Flow does not consider route length. Maximum Flow just takes into account capacities and tries to maximize in-flow units.

Traveling distance in LRA* is also more than A*. LRA* has lots of turnings and cycles and re-routing that adds more traveling distance to each unit.

Figure 0.4 shows the comparison between solutions in term of traveled distance.

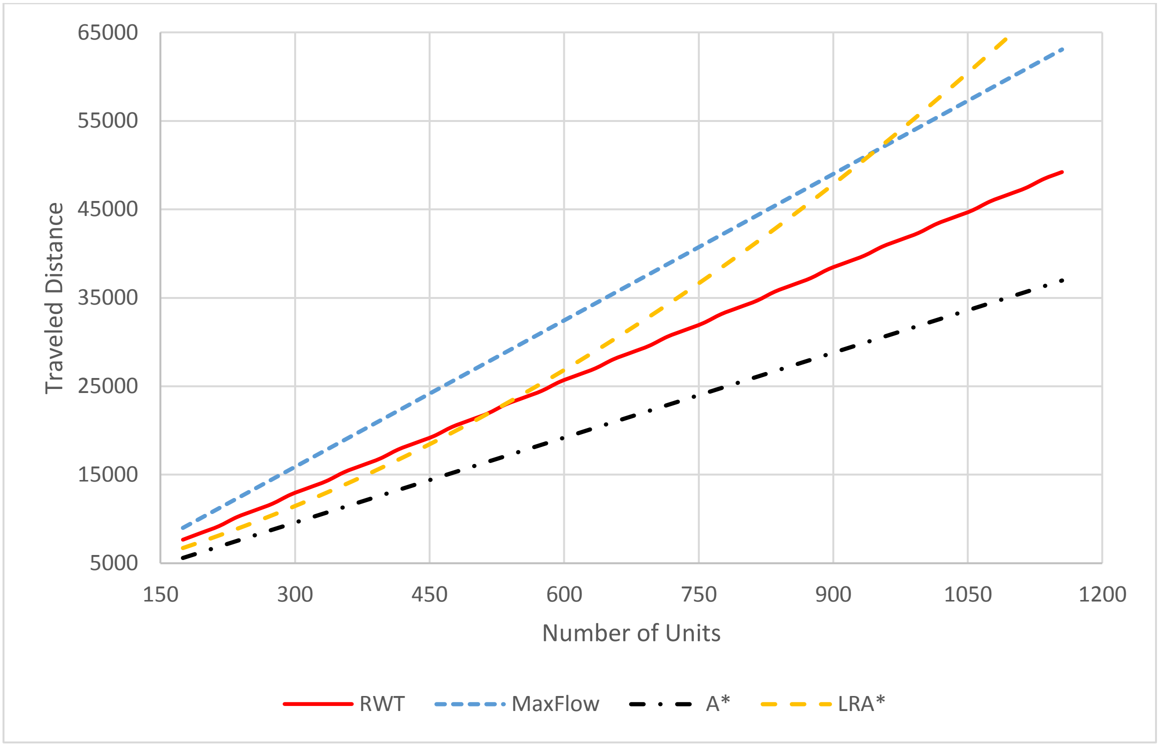

Figure 0.4: Traveled distance for different number of units

1.14.4 Waiting Time

The relative changes of waiting time are investigated under four different path-finding strategies, while the number of units is changing between 175 and 1000.

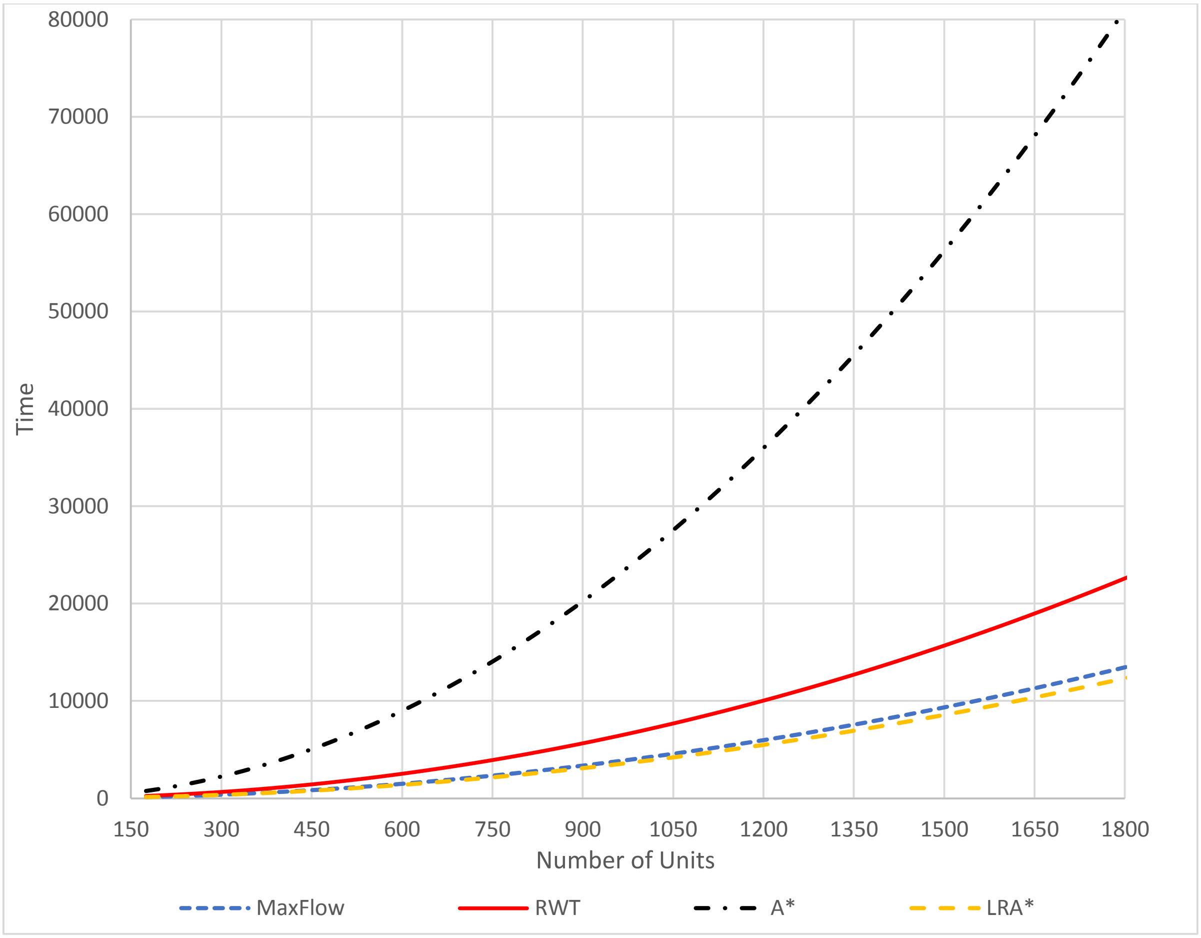

Figure 0.5: Waiting time

As Figure 0.5 shows, A* has more waiting time than other solutions. Calculated waiting is sum of waiting times of all the units. Those times, when a unit is moving on the terrain is not considered as waiting time. Waiting time is when a unit is not moving and waits in a queue for a free path to get on to.

$$Waiting\ Time\ = \sum_{i=1}^{Total\ number\ of\ units} Total\ time\ unit\ i\ spent\ in\ waiting\ state$$

Each route has a limited capacity, if the number of units, which want to pass through that route exceeds the route capacity they have to wait at the entrance portal of the route for a free spot. The additional units for a route results in the more waiting time for those units.

A* sends all the units from one route (shortest path) and because of limited capacity of that units have to wait for free spaces.

Maximum Flow and RWT divide units and send them through several routes. Consequently waiting time for these two solutions are less than A*.

Waiting time for Maximum Flow is less than RWT and it makes sense. The policy of Maximum Flow algorithm is to maximize the number of units, which are in flow.

As LRA* looks for potential collisions and executes re-routing for those potentially colliding units, it decreases chance of waiting in queues and therefore waiting time.

1.14.5 Number of Collisions

As was mentioned in Section 2.3 there are two common representations of walkable areas in video games, NavMesh and Waypoints. When waypoint system is used, units will move in lines from point to point and never collide with other neighbor units. When NavMesh system is used, units try to take the shortest possible path. When the route turns, units tend to take smallest turning radius. In this situation, there will be some collisions between neighbor units. Figure X shows the difference between two representations.

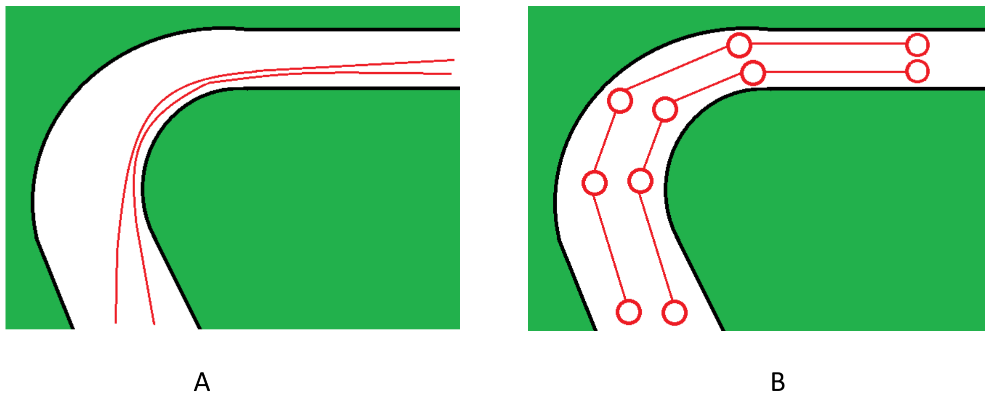

Figure 5.6: picture (A) shows turning path, which uses NavMesh, and picture (B) shows the same part of map with waypoints system.

As figure X shows, units in a game, which uses NavMesh, tend to take the smallest turning radiuses. Red lines in picture A are approximate paths in a route with capacity of two. Units in a game, which uses waypoint system, move in separate lines and will not collide with neighbor units. Path of picture B also has capacity of two.

Simulations of this study are implemented using NavMesh system. Although RWT algorithm is collision free, using NavMesh system results in some collisions especially at turning routs and at junctions.

The results show that increasing the number of units invariably yields higher number of collisions in all the algorithms. Number of collisions exponentially grows in dense areas where in-queue units try to find their paths and hit each other to find a free spot. Solutions with more waiting units should have more collisions. That is the reason behind high number of collisions in A* solution. Maximum Flow and RWT spread units and decrease chance of collision between units.

$$Total\ Number\ of\ Collisions = \frac{\sum_{i=1}^{Total\ number\ of\ units} Total\ Collisions\ of\ i}{2}$$

Every time two units collide, it considered as two collisions, so we divide number of collisions by two to have the correct number of collisions.

Figure 0.6 compares number of collisions for all algorithms when different number of units is placed on the terrain.

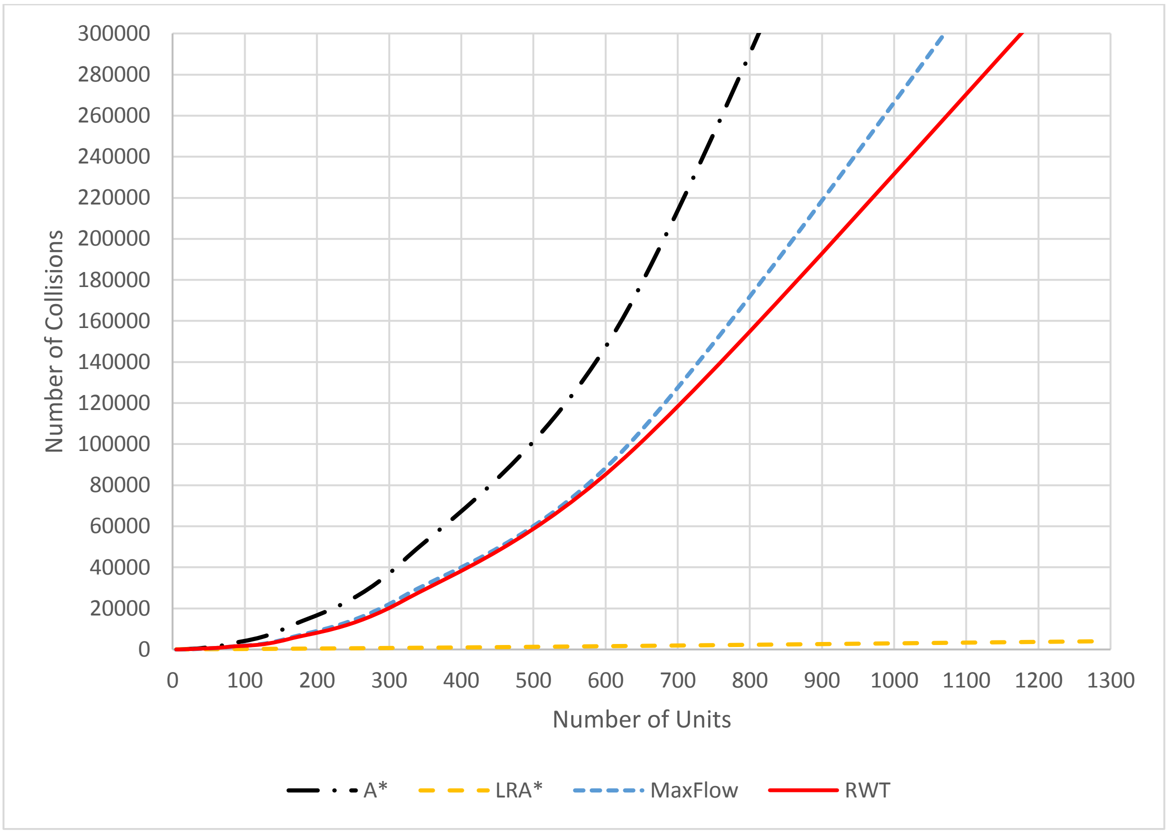

Figure 0.6: Number of collisions between units

Considering the fact that different steering behaviors affect all the algorithms in almost the same way [15], study of different steering behaviors is out of the scope of this study and was not considered in this study.

Conclusions and Future Works

Focus of this thesis is on single source - single destination group pathfinding problem. All units move with similar speed and have similar sizes. RWT algorithm is a good solution for group of units which start to navigate the map and try to find their routes to one destination in minimum possible time. There are other situations that can be considered for future works. For example, solution for other possibilities like:

- Multiple source - single destination
- Multiple source - multiple destination
- Single source - multiple destination
- Multiple source - multiple destination

These problems needs further research and perhaps enhanced algorithms.

To achieve a global solution, it is also necessary to consider group of units with different sizes and different speeds (infantry soldiers vs. armoured tanks).

References

1. J.C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, Boston, MA, 1991. ISBN 0-7923-9206-X.

2. P. Sveska and M. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), 1995.

3. M. Erdmann and T. Lozano-Perez. On multiple moving objects. Algorithmica, 2:477–521, 1987.

4. D. Silver. “Cooperative pathfinding,” In The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment, pages 23–28, 2005.

5. C. Warren. “Multiple robot path coordination using artificial potential fields,” In Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), pages 500–505, 1990.

6. E. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik vol. 1, pp. 269–271, 1959.

7. L. Ford and D. Fulkerson, “Constructing maximal dynamic flows from static flows,” Operation Research, vol. 6, pp. 419–433, 1958.

8. P. E. Hart, N. J.Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Transactions on Systems Science and Cybernetics 4(2), pp. 100--107, 1968.

9. R. Eberhart, ; S.Yuhui, “Computational Intelligence: Concepts to Implementations”. Morgan Kaufmann Publishers Inc. p. 118. ISBN 1558607595. 2007.

10. Wouter G. van Toll, Atlas F. Cook, and Roland Geraerts. 2012. A navigation mesh for dynamic environments. Comput. Animat. Virtual Worlds 23, 6 (November 2012), 535-546.

11. Haiqing Wang; Malik, O.N.; Nareyek, A., "Multi-unit tactical pathplanning," Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on , vol., no., pp.349,354, 7-10 Sept. 2009

12. Stout, Bryan, “The Basics of A* for Path Planning,” Game Programming Gems, pp. 254-263. Charles River Media, 2000.

13. LidÈn, Lars, "The Integration of Autonomous and Scripted Behavior through Task Management Artificial Intelligence and Interactive Entertainment" : Papers from the 2000 AAAI Spring Symposium, Technical Report SS-00-02, 51-55, 2000.

14. D. White. "Clarifications and extensions to tactical waypoint graph algorithms for video games," In Proceedings of the 45th annual southeast regional conference. ACM, New York, NY, USA, 316-320, 2007.

15. Wang, Ko-Hsin Cindy, and Adi Botea. "Fast and Memory-Efficient Multi-Agent Pathfinding." In ICAPS, pp. 380-387. 2008.

16. Danielsiek, H.; Stüer, R.; Thom, A.; Beume, N.; Naujoks, B.; Preuss, M., "Intelligent moving of groups in real-time strategy games," Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On , vol., no., pp.71,78, 15-18 Dec. 2008

17. Kamphuis, Arno, and Mark H. Overmars. "Finding paths for coherent groups using clearance." In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 19-28. Eurographics Association, 2004.

18. Sadek, Adel W., Brian L. Smith, and Michael J. Demetsky. "Dynamic traffic assignment: Genetic algorithms approach." Transportation Research Record: Journal of the Transportation Research Board 1588, no. 1 (1997): 95-103.

19. Vaniya, S., B. Solanki, and S. Gupte. "MULTI ROBOT PATH PLANNING ALGORITHMS: A SURVEY." International Journal of Computer Science and Engineering Research and Development, Volume 1, pp. 38-49, 2011.

20. Graham, Ross. "Realistic agent movement in dynamic game environments." Presentation in DiGRA Conference: Changing Views – Worlds in Play, 2005.

21. Lidén, Lars. "Strategic and tactical reasoning with waypoints." In AI Game Programming Wisdom, Charles River Media. Section five, pp. 211-220, 2002.