MARMARA ÜNİVERSİTESİ
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

ANY ANGLE PATH FINDING IN STOCHASTIC OBSTACLE SCENES

UFUK ASLAN

MASTER THESIS

Department of Computer Engineering

Thesis Supervisor

Assoc. Prof. Ali Fuat ALKAYA

ISTANBUL, 2019

MARMARA ÜNİVERSİTESİ
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

ANY ANGLE PATH FINDING IN STOCHASTIC OBSTACLE SCENES

UFUK ASLAN
(524104981)

MASTER THESIS
Department of Computer Engineering

Thesis Supervisor
Assoc. Prof. Ali Fuat ALKAYA

ISTANBUL, 2019

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Ufuk ASLAN, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled "**Any Angle Path Finding in Stochastic Obstacle Scenes**", on Sep 4, 2019 and has been found to be satisfactory by the jury members.

Jury Members

Assoc. Prof. Ali Fuat ALKAYA (Advisor)
Marmara University .... (SIGN) ....

Asst. Prof. Betül DEMİRÖZ BOZ (Jury Member)
Marmara University .... (SIGN) ....

Asst. Prof. Berna KİRAZ (Jury Member)
Fatih Sultan Mehmet Vakıf University .... (SIGN) ....

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Ufuk ASLAN be granted the degree of Master of Science in Department of Computer Engineering, Computer Engineering Program on Sep 18 2019. (Resolution no: 2019/19-02).

Director of the Institute
Prof. Dr. Bülent EKİCİ

TABLE OF CONTENTS

ABSTRACT

ANY ANGLE PATH FINDING IN STOCHASTIC OBSTACLE SCENES

Path planning with stochastic obstacles is a well-known research area. The Canadian traveler problem (CTP) is a challenging stochastic optimization problem of traversing in a given graph having blocked edges and the disambiguation status of these edges can be settled with predefined probabilities on the fly when the agent reaches to an adjacent vertex. Stochastic obstacle scene problem (SOSP) has obstacles in continuous environment which requires huge state spaces for the exact solution. Thus, discretized version of stochastic obstacle scene problem (D-SOSP) is most commonly used variant of CTP which has group of stochastic edges with probability dependency. States of these edges are assigned as ambiguous, traversable, or untraversable. The objective is to design a travel plan that would guarantee the shortest path including the obstacle disambiguation cost. Obstacle neutralization problem (ONP) is defined as obstacle neutralization capability, which is limited and predefined, of an agent on optimal path with an additional cost.

In this work, we present Any-angle (ANYA) path finding in discretized stochastic obstacle scenes using the exact algorithms AO* with caching (CAO*) and AO* with caching and neutralization (CAON*). The admissible upper bounds in the CAO* are found by making use of Dijkstra’s shortest path. However, ANYA algorithm, being recently proposed, is already shown to outperform those shortest path algorithms at the highest level of development on grid-based graphs. It looks for optimum length paths by investigating the interval sets which are established dynamically. Our methodology is clearly demonstrated through computational experiments using a data map called COBRA, a real example of US naval forces, and the use of several synthetically produced minefields.

ÖZET

OLASILIKSAL ENGELİ OLAN ALANLARDA HERHANGİ AÇI YÖNTEMİ İLE YOL BULMA

Stokastik engellerin olduğu yol planlaması, çok populer bir araştırma alanı olarak bilinir. Zorlayıcı bir stokastik optimizasyon problemi olan Kanadalı Gezgin Probleminde (CTP), bloke edilmiş yollar içeren bir haritada engele bitişik bir köşeye ulaşıldığında önceden tanımlanmış olasılıkla bu engel anlık olarak yok edilebilir. Stokastik engelli yol bulma probleminin (SOSP) gerçek çözümünde, sürekli bir ortamda büyük durum aralıkları gerektiren engeller bulunur. Bu nedenle, stokastik engel alanı probleminin ayrıklaştırılmış versiyonu (D-SOSP), olasılık bağımlılığı olan bir grup stokastik kenara sahip olduğu için CTP'nin en sık kullanılan çeşididir. Bu kenarların durumları belirsiz, geçilebilir veya değiştirilemez olarak atanır. Amaç, engeli yok etme maliyeti de dahil olmak üzere en kısa ayrık yolu garanti edecek bir seyahat planı tasarlamaktır. Engeli etkisizleştirme problemi (ONP), sınırlı ve önceden belirli ek maliyeti olan etkisizleştirme kabiliyetini barındırır.

Bu çalışmada; ayrıklaştırılmış stokastik engel alanlarında problemin tam çözümü için, önbellek kullanan AO* (CAO*) ve engeli etkisizleştiren AO* (CAON*) algoritmaları kullanarak herhangi açı (ANYA) yol bulma metodunun faydalarını sunuyoruz. Standart CAO*, kabul edilebilir üst sınırları bulurken Dijkstra'nın en kısa yol metodu kulanır. Bununla birlikte; yakın zamanda önerilen ANYA algoritmasının, ayrıklaştırılmış grafik üzerindeki üst düzey kısa yol algoritmaları arasında en iyi performans gösterdiği görülmüştür. ANYA, dinamik olarak kurulan aralık kümelerini inceleyerek optimum uzunluktaki yolları arar. Gerçek hayattan bir örnek olan ABD donanma kuvvetlerine ait mayın tarlası veri haritası COBRA ve rastgele oluşturulmuş çeşitli yapay haritalar üzerinde metodolojimizi çalıştırarak elde ettiğimiz hesaplama sonuçları, belirsizliği giderme ve engeli imha etme problemlerinin çözümünde belirgin bir iyileştirmeyi ortaya koymuştur.

SYMBOLS

K	**:** Disambiguation Count
L	**:** Neutralization Count
G	**:** Graph
v	**:** Vertex
e	**:** Edge
ρ	**:** Probability Function
t	**:** Target Vertex
s	**:** Starting Vertex
E′	**:** Stochastic Edge Set
E	**:** Deterministic Edge Set
V	**:** Vertex Set
y	**:** Vertical Axis
x	**:** Horizontal Axis
r	**:** Radius
c	**:** Center
P	: Path
n	: Node
f	**:** Function
g	**:** Graph Cost
h	**:** Heuristics

ABBREVIATIONS

CTP : Canadian Traveler Problem
SOSP : Stochastic Obstacle Scenes Problem
D-SOSP : Deterministic Stochastic Obstacle Scenes Problem
ONP : Obstacle Neutralization Problem
WCSPP : Weight Constrained Shortest Path Problem
CTPN : Canadian Traveler Problem with Neutralizations
SP : Shortest Path
DM : Decision Maker
ANYA : Any Angle
DA : Dijkstra's Algorithm
A* : A Star
AO* : AND OR Star
BAO* : AO* with Bounds
CAO* : BAO* with Caching
CAON* : CAO* with Neutralization
RD : Reset Disambiguation Policy
DT : Distance to Termination Algorithm
COBRA : Coastal Battlefield Reconnaissance & Analysis
API : Application Programming Interface
BFS : Bread First Search
2D : Two Dimensional

FIGURES

TABLES

1. INTRODUCTION

Any-angle path finding is the base navigation problem in artificial intelligence (AI) research are especially for robotics and computer game programming in order to find an optimum path length over unblocked edges quickly. The main purpose is to find a shortest path on a grid area such that blocked and unblocked cells represent the obstacles as constrained points. There are numerous exact methods which require huge operational time at the beginning of the process and use a function growing faster than linear. However, recently newer and most appropriate any angle finding algorithm called ANYA is proposed in [1]. While other works find approaching and close by paths by searching over each vertex from the grid, ANYA looks for optimum length paths by searching over interval sets which are identified to be done as one goes dynamically. ANYA picks up a single representative point from each interval such that this point is used to calculate acceptable and safe cost estimation for the whole cluster. If an optimal path exists, ANYA finds it in any way. In addition, it does not require any pre-computing or additional memory usage presentation. A set of experimental comparisons show that ANYA competes with a number of several recent pre-processing based and online methodologies and performs appreciably faster performance than the very common benchmark algorithm, A* as grid implementation [1,2].

The Canadian traveler problem (CTP) is a simple, but challenging, stochastic optimization problem to describe the very common experience in Canada as the final graph path becomes observable upon the blocked edges are explored. An agent travels on a given graph where some edges are blocked with certain probabilities and these edges update their disambiguation status dynamically when agent reaches an adjacent vertex [3, 4]. Suppose that a traveler wants to go from one site to another one by knowing all roads on the unreliable map [5]. Blockage of a road can be revealed just after reaching the target. The goal is to design a travel plan which would guarantee the shortest expected path starting from a given vertex to target vertex. The discrete stochastic obstacle scene problem (D-SOSP), which is also convenient path planning algorithm especially in minefields, is a special and relevant variant of CTP [6-9]. D-

SOSP is preferably applied on grid graphs with probabilistic dependency. This problem consists of discretization of grid graph on continuous space SOSP where an agent can navigate through a determined set of certain shaped obstacles truly in a scattered region. Initially, the agent can disambiguate the regions with the given probabilities when it is at the edge of true obstacle's boundary. The goal here is to decide on a disambiguation vertex among the intersection points on the shortest path route for minimizing the expected path length of the travelling time and distance. Recently, a new algorithm with the name CAO* is proposed in [6] that can solve the D-SOSP and CTP instances optimally and much faster than the state-of-the-art algorithms. CAO* has its own caching setup to prevent previously visited states to expand again and upper bounds are fair to be used at a node level to prune dynamically state space [3]. It does not complete in polynomial time, but significantly shortens the execution time of finding an exact solution when instances are relatively sized.

The Obstacle neutralization problem (ONP) is a well-known and studied path planning problem in telecommunication and aircraft industry [10, 11, and 12]. Limitations are neutralization count and agent's capability. Capability of neutralization refers to travelling through an obstacle by adding an extra cost to total traversal length. The challenging part is to find an optimal route. CTP and ONP seem to be closer when compared with characteristics inherently. They are even combined into single problem as Canadian Traveler Problem with Neutralizations (CTPN) in the literature [4].

The admissible upper bounds in the CAO* are found by exploiting some heuristics that make use of the shortest path algorithms like Dijkstra's[1]. It recurrently examines the available edges nearby starting from current node until reaching the goal. Scanning all closer nodes causes to examine the equal weights paths. Dijkstra with Best First Search[2] heuristic is fairly better alternative with its speed. It combines the heuristics of prioritizing the nodes which are also closer to target point. Building the AO tree requires too many shortest paths finding algorithm executions. Therefore, underlying data structure and methodology of the path finder make a difference.

---

[1] Scientist Edsger W. Dijkstra conceived this idea to produce shortest path tree.

[2] Peter Hart, Nils Nilsson and Bertram Raphael first published A* algorithm [13].

However, ANYA algorithm, being pretty recently proposed, is already shown to outperform cutting edge shortest path algorithms on grid graphs. Therefore, it is worth to embed ANYA in CAO* and CAON* and other heuristic algorithms developed for D-SOSP and CTP variants and assess the graph types that ANYA brings improvement in their performance. In this thesis, we propose the any angle path shortest path coordination in AO* based algorithms for the exact solution on the graph with obstacles. It is the combination counter information that path is not formed by octant edges in the graph whereas intersected edges are along edges on the grid graph. The main goal is to determine the graph environments where ANYA provides faster performance for the shortest traversal in AO* trees. It is always vital to solve the shortest path in a reasonable time when decision making process is required. An efficient implementation of ANYA results a challenging enhancement in CAO* searching for CTP with no additional memory overheads.

The rest of the chapters are arranged as follows: the path planning problem with variants and solution algorithm details in the literature are introduced in section 2. Section 3 defines the implementation and work done. In Section 4, results of our experiments and discussions are presented. Finally, concluding remarks and possible future work related to this study is given in section 5.

2. PATH PLANNING

2.1. Background

In this section, general background is given with explanatory figures to refresh the general terminology. Graph theory and Application Programming Interfaces (API) are preliminary topics to be explained before the problem and algorithm definitions.

2.1.1. Graph

A graph G is expressed with formula $G = (V, E)$ as the set of vertices where $V = \{v_1, v_2, v_3, \dots, v_n\}$ and edges $E = \{e_{1,} e_2, e_3, \dots, e_n\}$ in computer and math science. Vertices are also called nodes and construct the points of the graph. Edges are called as pair of vertices or links $e_k = (v_i, v_j\}$ where$e_k \in E, v_i \in V, v_j \in V$.

Graphs without any multiple edges between any vertices are called *simple graphs.* In this study, graph term is used for simple graph. Figure 2.1 illustrates simple *directed* and *connected* graph *G*. On connected graph, all vertices are reachable from each vertex over others. It is not *complete* graph, which means direct links do not exist between every pair of vertices. Vertices at the end of each link are called as *neighbors*.

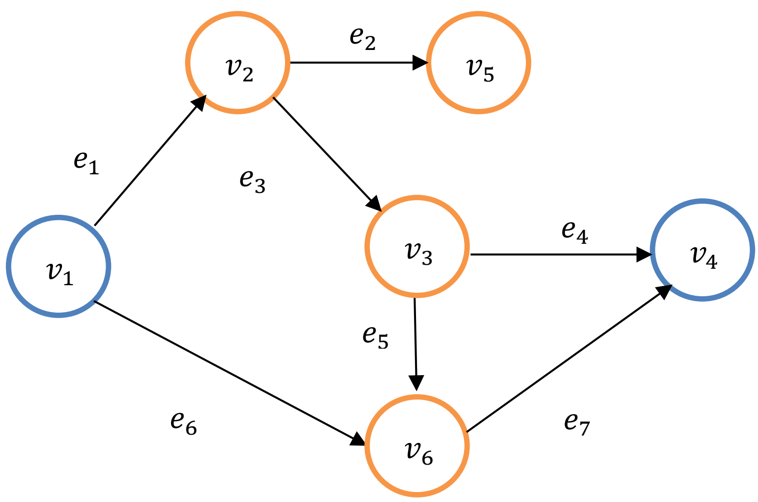

Figure 2.1 A sample directed graph

2.1.2. Path

Path is the fundamental concept in graph theory and path of length is derived from the unique vertices on the route. If there are repeated vertices in the path, it is called *cycle*.

In Figure 2.2 sample paths are given from the graph in Figure 2.1. Length of path is simple the sum of edge weights. Starting vertex $s = v_1$, target vertex, $t = v_4$ and routes $E_1 = \{v_1v_2, v_2v_3, v_3v_4\}$, $E_1 = \{v_1v_6, v_6v_4\}$ and $E_3 = \{v_1v_2, v_2v_3, v_3v_6, v_6v_4\}$

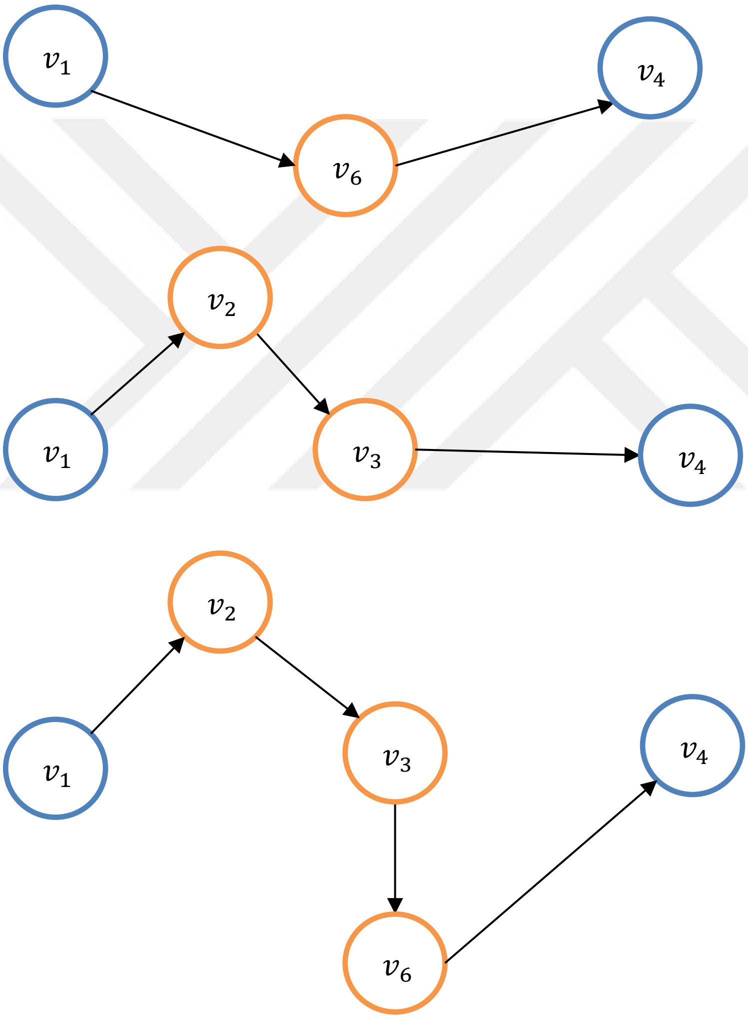

Figure 2.2 Sample paths with length 2, 3 and 4

Shortest path is simply the route having the smallest weight. Sometimes it is also called as cost or length. Graph path is the core component in this work.

2.1.2. APIs

JGraphT is an open source java library for the graph theory implementation with data structures, algorithms and visualization support [14]. It becomes the base of our development via flexible, powerful and efficient components.

Standard JavaGeom packages for two-dimensional (2D) geometry are used to define and perform operations in objects [15].

2.2. Path Planning Problems

Path planning is the key point of the navigation in game development and robotics. Problem starts with locating the initial and final points by given coordinates on a graph which can be either continuous or discretized graphs. It is an NP-Hard problem. Main objectives are length of the path and cost of the execution. Solutions can consist of avoiding predefined obstacles on the graph by disambiguating or neutralizing them if any path is available to reach the final destination. Therefore, finding the shortest path is applied several times within the solving methodology.

2.2.1. Stochastic Obstacle Scenes Problem

SOSP considers an agent travelling on a path in continuous space; wherein disk-shaped obstacles exist, between starting and target locations. At first, obstacles have the given probability at which agent can disambiguate it when reaches the adjacent vertex on the boundary. The main goal is reducing the travel length of expected path at the end. D-SOSP is discretized SOSP. It is formally defined on an undirected grid graph with specified starting and target vertices. Stochastic edges of this graph are the disk intersection edges and denoted by E' exist based on the remaining probability function $\rho: E' \rightarrow [0,1)$. The agent can traverse the deterministic edges E/E'. At every edge $e \in E'$, agent tries to disambiguate it to check whether e is traversable or not. Stochastic edges are at blocked status until they are exactly marked as traversable. Just after the disambiguation, the statuses of edges are temporarily changed until agent reaches to target vertex node. There is a fixed disambiguation cost and it is added to total traversal length.

In Figure 2.3, Disk arc represents the boundaries of an obstacle. Intersected edges are shown in bold and orange on a discretized scene.

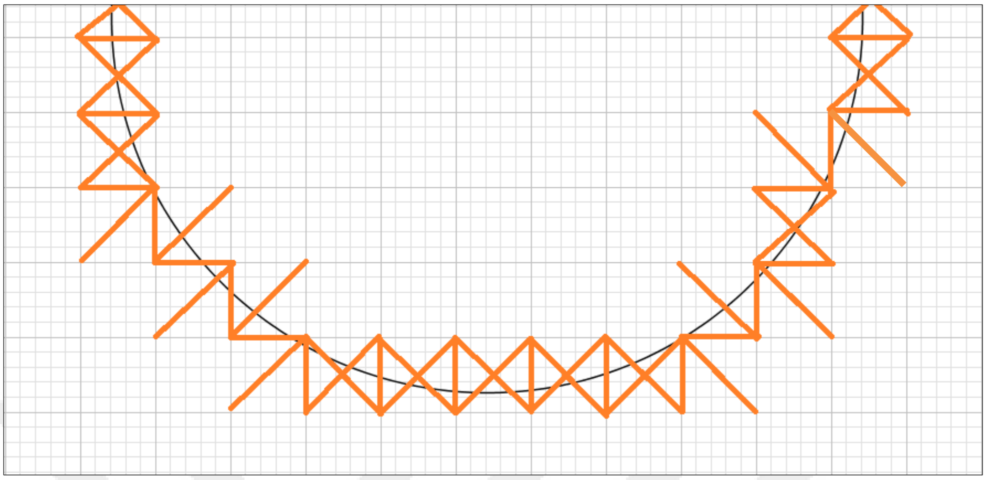

Figure 2.3 Intersected edges with half of a disk shaped obstacle on a discretized simple SOSP instance

The goal is finding the shortest expected walk from *s* to *t* in this graph by designating a disambiguation policy to decide the place and the timing. It is assumed that limit K denotes for the allowed disambiguation count [10].

2.2.2. Canadian Traveler Problem

CTP is the routing strategy with given uncertainty of road blockage. Snowfall blocks traveler passing from that road. When status of all blocked roads is same, devising a travel strategy is proven to be PSPACE complete. It is the generalized and graph theory related version of D-SOSP. Stochastic edges representing roads with independent probability exist in the problem definition. Blocked edges are only revealed just after the agent reaches any edge on the planned path. The most difficult part is devising an online strategy to find the optimal path. Although all blocked edges are known at the beginning, it is not clear that which edge is the next one on the expected shortest path.

Initial cost of the journey is unknown to the agent and cost of each edge is observed at the node. The main feature of this stochastic optimization is the sequential decision making and exploring the alternate edges.

Figure 2.4 and 2.5 illustrate the simple CTP graph. Solid lines reflect deterministic edges and dashed line is the stochastic edge. The agent starts to travel from node *t* to reach node *s*. It can directly pass to target or pass from node *w* with the probability 0.4.

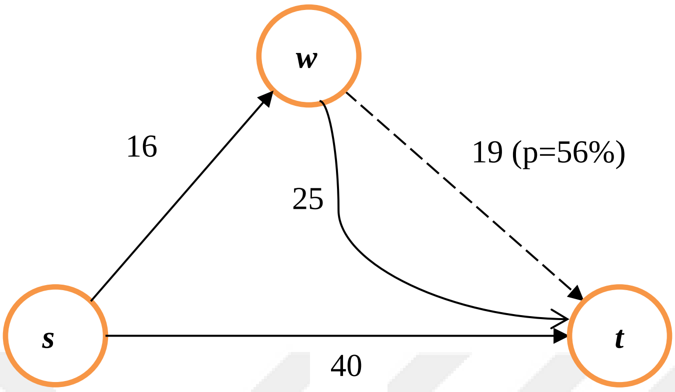

Figure 2.4 Deterministic and Stochastic edges on a sample of CTP

Total weight formula of the travel instead of zero risk path cost 40:

$$G = \{sw + (\rho * wt) + (1 - \rho) * (wst)\} = 16 + 19 * 0.56 + 25 * 0.44 = 37.64$$

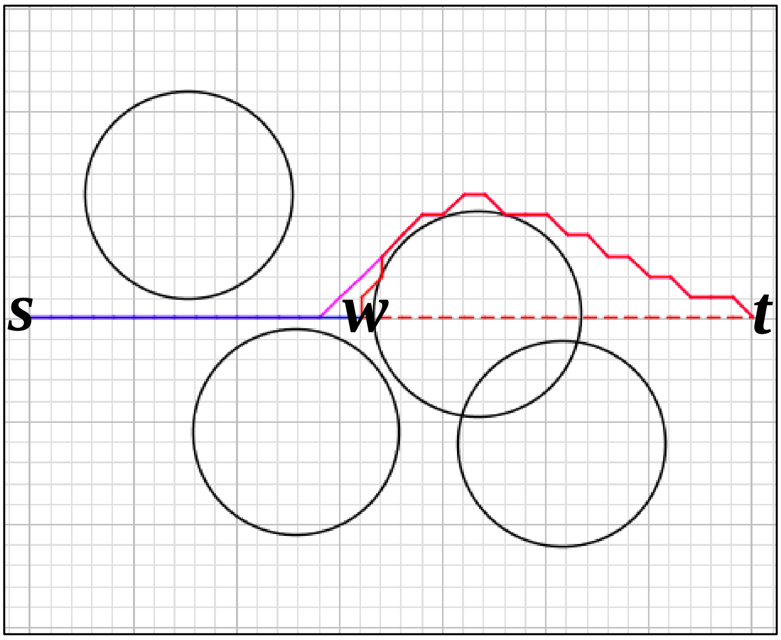

Figure 2.5 CTP representation on grid graph with 4 disk shaped obstacles

AO based solution tree is shown in Figure 2.6. Squares represent OR nodes and circles are used for AND nodes.

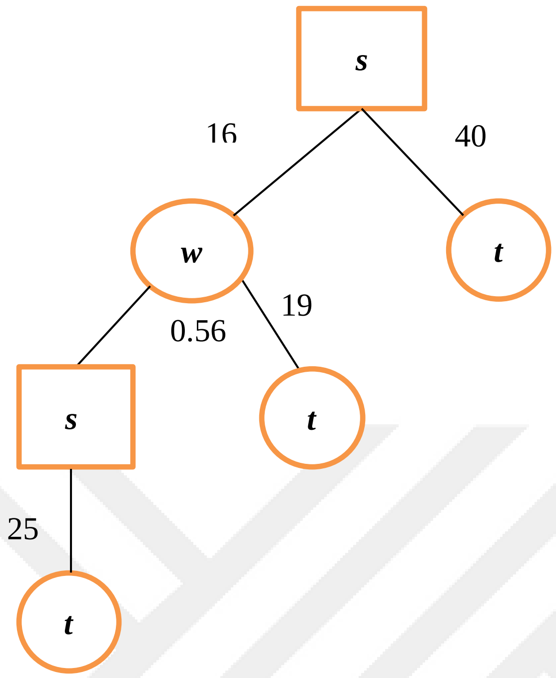

Figure 2.6 Complete AO* tree representation of CTP

2.2.3. Obstacle Neutralization Problem

Obstacle Neutralization Problem (ONP) is fundamentally a diverse version of Weighted Constrained Shortest Path Problem (WCSPP). It considers to finding minimum cost for the agent traversal between nodes. We come across many different weighted shortest path problem samples in real life. According to the value added we gain, these problems become vital ones with the obstacles respectively [16]. Some examples are finding a minimized way to:

- route aircrafts by taking fuel consumption and radars into account [17]
- transmit data packages through networks [18]

ONP is known as NP-Hard. It has both exact solutions and heuristics in the literature. It is simply solved in polynomial time, if following rules are applied:

- All edge weights are same

- Neutralization cost is constant
- Unlimited neutralization capability

In Figure 2.7, agent chooses to neutralize the node *y* to traverse from *s* to *t* instead of using the direct route. C stands for the neutralization cost and agent has the capability of passing through an obstacle. Node *w* represents the neutralization point of an obstacle.

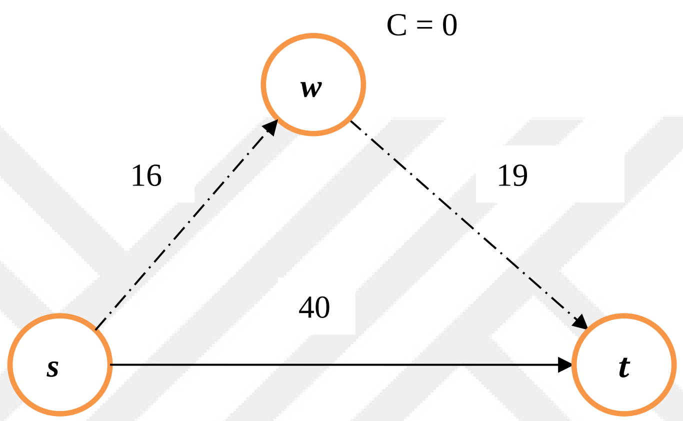

Figure 2.7 Graph with optimal path on ONP instance

Total weight formula of the travel:

$$G = \{(sw + C + wt)\} = 16 + 0 + 19 = 35$$

2.4. Shortest Path Algorithms

The shortest path is defined as the minimized total weight of the edges between two nodes in graph theory. Problem can be solved either undirected or directed graphs over adjacent points.

In Figure 2.8, graph has 6 marked interval points named from A to F, start node S and target node T on a grid area having dimensions (18x12). Solution path is illustrated as combination of 3 lines in red. Yellow line can be an alternative path which is not the shortest. Green lines represent non-visible edges on blocked cells. Nodes in the solution path are in format *V(x, y)* where *x* is horizontal and *y* is vertical coordinate.

$$P = \mathrm{S}\,(1.7) \rightarrow \mathrm{A}(6{,}12) \rightarrow \mathrm{B}\,(11{,}12) \rightarrow \mathrm{T}(16{,}7)$$

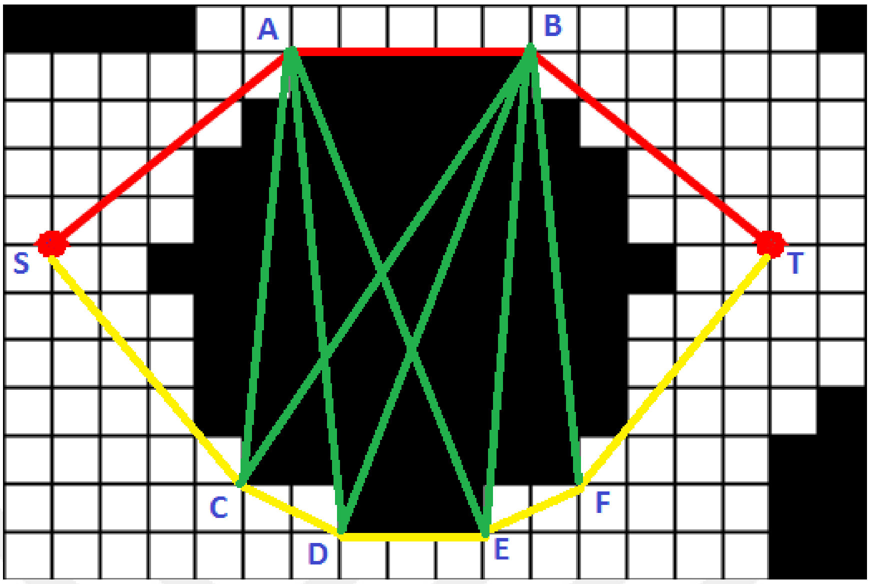

Figure 2.8 Shortest path from Starting to Target nodes over A and B.

2.4.1. Dijkstra's Shortest Path Algorithm

Dijkstra conceived his shortest path algorithm in 1955. Then it became the most commonly used algorithm as the main optimal subroutine, as long as non-negative wieghts are provided for the edges, in graph implementations and taught by every lecturer as the fundamental one. It was published 3 years later after firstly proposed. In fact, Dijkstra's algorithm was also firstly implemented as the solver for a known problem to show off new computers.

Simpliest way of setting edges weight to max value (∞) in grid graph implemetation. Directed graphs are also converted to undirecting graphs by defining opposite edge on adjacent nodes with same weight. It uses two maps to keep the *distance* and the visited nodes named *previous*. Base version uses the Bread First Search (BFS) approach to traverse the graph and cause to visit many vertices pointlessly. We can also call Dijkstra as a greedy algorihtm due to making optimal choice at each stage locally to find the optimum weight in global. It is implementes as one function with complexity $O(|V|^2)$ to calculate the cost of the path.

Pseudocode of the algroihtm is given in Figure 2.9. It stops when target vertex is reached or no vertex left in the queue. It continuously updates distance map with the alternatives when shorter one encountered during graph traversal.

```
function Dijkstra(Graph, source):
    dist[source] ← 0                           // Initialization

    create vertex priority queue Q

    for each vertex v in Graph:
        if v ≠ source
            dist[v] ← INFINITY                 // Unknown distance from source to v
        prev[v] ← UNDEFINED                    // Predecessor of v

        Q.add_with_priority(v, dist[v])


    while Q is not empty:                      // The main loop
        u ← Q.extract_min()                    // Remove and return best vertex
        for each neighbor v of u:              // only v that are still in Q
            alt ← dist[u] + length(u, v)
            if alt < dist[v]
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist, prev
```

Figure 2.9 Dijkstra’s shortest path pseudo code from Wikipedia [19]

2.4.2. A* Shortest Path Algorithm

A* (pronounced as Astar) is defined as the extended version of Dijkstra’s. It is widely preferred due to increase in performance. Function *h(n)* in the formula, represents the heuristics used as the search guide.

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

This concept was invented by group of researchers 10 years after Dijkstra [13]. It prioritizes the vertices in the queue by estimating the cost instead of expanding the nodes without any control .

Pseudocode is given in Figure 2.10. It uses two sets named *openSet*, which is the priority queue to be explored by lowest score, and *closedSet*, which stores the visited nodes to skip at next visit.

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path

// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from node n.
function A_Star(start, goal, h)

    // The set of discovered nodes that need to be (re-)expanded.
    // Initially, only the start node is known.
    openSet := {start}

    // For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from start to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the cost of the cheapest path from start to n currently known.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n).
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        closedSet.Add(current)
        for each neighbor of current
            if neighbor in closedSet
                continue
            // d(current,neighbor) is the weight of the edge from current to neighbor
            // tentative_gScore is the distance from start to the neighbor through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if neighbor not in openSet
                openSet.add(neighbor)
            else if tentative_gScore < gScore[neighbor]
                // This path to neighbor is better than any previous one. Record it!
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := gScore[neighbor] + h(neighbor)

    // Open set is empty but goal was never reached
    return failure
```

Figure 2.10 A* shortest path pseudo code from Wikipedia [19]

2.4.3. Any Angle Shortest Path Algorithm

Any angle path finding is the root algorithm for routing through shortest path in AI research area, especially robotics and computer games. Problem is solved in grid area where blocked edges are taken into account in more efficiently in memory and execution time. ANYA shortest path is proposed an exact method and defines an optimal path if available. An intermediate point is traversable if it is next to another traversable cell. If not, it is unreachable. Intersection is a common point shared by entirely four neighboring cells. The intersection point where three cells are traversable and one cell is not is called as corner. Visibility alternatives of cells are explicitly shown in Figure 2.11.

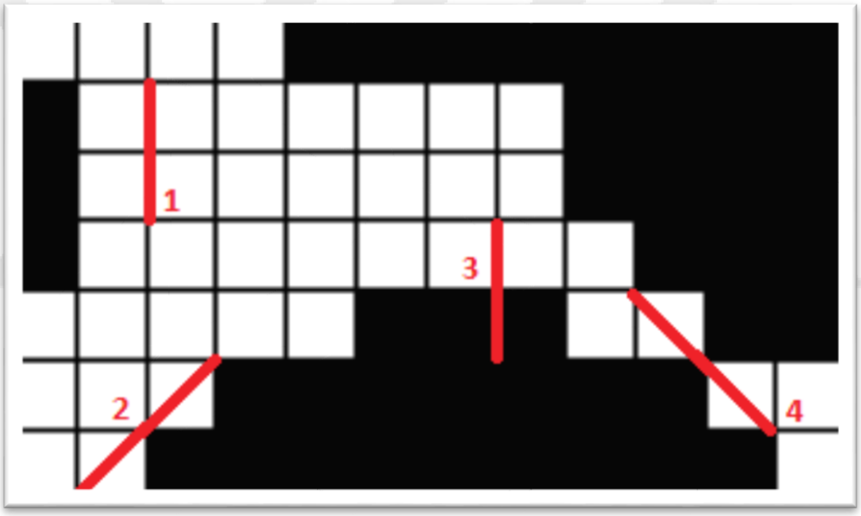

Figure 2.11 Example of visible (no:1,2) and non-visible (no:3,4) ANYA cell joints

ANYA search node evaluation is simply formulated as the function of calculating the shortest path starting from point *s* to target point *t* via distance function d intervals between points r and p. All visible between points are continuous straight lines.

$$f(s,r,p,t) = g(r) + d(r,p) + h(p)$$

ANYA algorithm has fundamentals to define cells are connected through the points. Two points are defined as visible when a straight-line path can connect them without:

- Passing through any cell which is not traversable.

- Passing between two diagonally adjacent cells which are not traversable.

Evaluation criteria properties of ANYA path finding algorithm are as follows:

- Effective and Productive (runtime and path length),
- Simple and basic (simple to implement and understand),
- General (graph implementation) [20].

Figures 2.12 represents disk shaped obstacle fields in discrete grid graph with zero disambiguation cost.

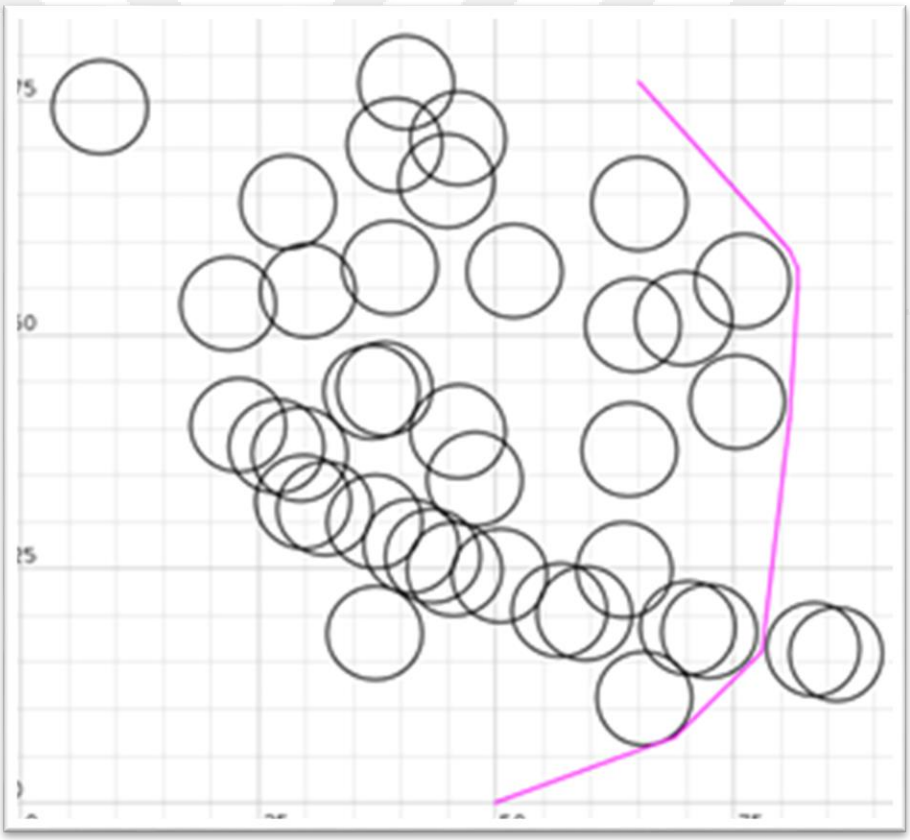

Figure 2.12 Disk shaped obstacles representation on Simple obstacle field of discretized graph with zero disambiguation count

Obstacles are sketched again on ANYA grid as blocked cells in Figure 2.13 for the same path. Therefore, there are always corners to traverse cells through visible cells in both figures.

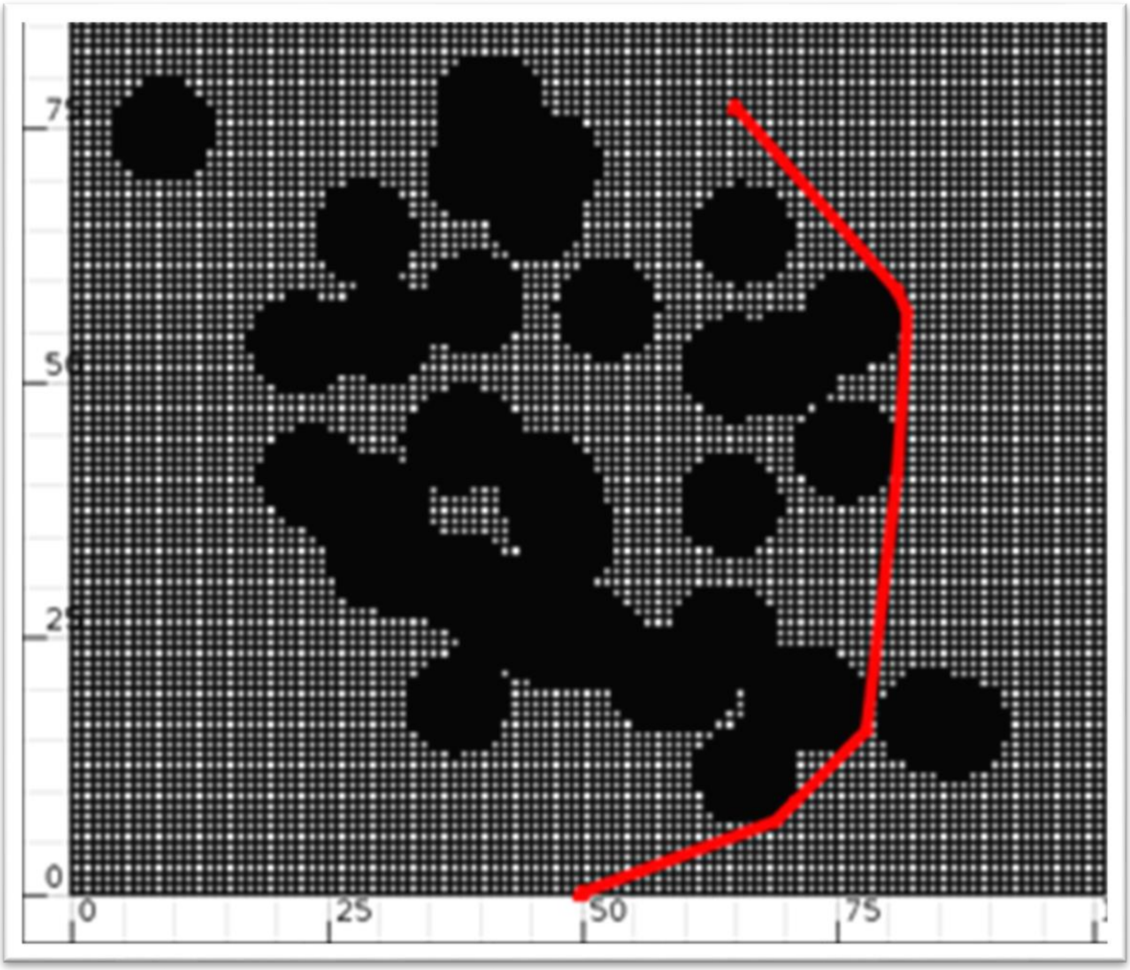

Figure 2.13 AnyaGrid representation with blocked and unblocked cells on simple obstacle field of discretized graph with zero disambiguation count

Search procedure of ANYA presented in Figure 2.14. It continuously expands the successors of current node until target node is not found at current node expand.

```
function performanya (Grid, s,  t)
//source location s
//target location t
//Start node with root r0 located off the grid
open := f(I = [s], r0 )
while open is not empty do;
        (I, r) :=  pop(open)
        if t in I then
                return path_to(I)
        end if
        for all (I0, r0) in successors(I, r) do;
        //Successor pruning
                if prune(I0; r0) = true then
                        open.Add( f(I0, r0) )
                end if
         end for
end while
return null
```

Figure 2.14 ANYA search procedure Pseudocode from Optimal Any-Angle Path Finding In Practice [1]

2.5. Constrained Shortest Path Planning Algorithms

Constrained shortest path planning problems turn out to be NP-Complete, in general. Therefore, they are solved via both exact algorithms and heuristic implementations. Path finding algorithms require the shortest path during the graph traversal in common. AO* based algorithms have the principal of traversing the paths regardless of the fact that it reaches to terminating node or not. Total cost is calculated by adding up cost from current to target node. Caching the visited paths and pruning the tree methods make great difference. In problem definition, disambiguation or neutralization of an obstacle or region is an option for the agent, which is also called as Decision Maker (DM) [21]. The optimal solution should have shortened execution time and minimum travel cost.

2.5.1. AO* Based Methods

Root of AO tree is defined with two main functions to assign length to every node and probability of each one. Tree of AO is implemented as AND/OR graph which is commonly used by researchers. Each node has the function named as true label. The optimal value is true label of root node of the graph.

2.5.2. BAO*

BAO* is based on AO* search algorithm. It extremely prunes AO tree to increase the overall performance by utilizing lower and upper bounds on the path lengths [21]. There is always and upper bound of zero risk expected length. It only examines small portion of the solution tree by means of efficient problem structure. When compared with AO*, less resources are required to find the exact solution. It provides new feature of expanding OR nodes. Successor AND nodes are not generated when regular normal zero risk path length is less than traversal path length. It also removes the nodes when regular path length is exceeded.

2.5.3. CAO*

CAO* is an exact method and AO* based search algorithm in the area of D-SOSP and CTP. It builds an AO solution tree progressively by alternating expansion and propagation steps to calculate the optimal policy. It is also AO* based as BAO*. Obstacles on the agent's path are composed from a state (true, false or ambiguous) and disambiguation capabilities. OR nodes represent the expanded nodes. AND nodes, corresponding to true/false consequences of the disambiguation, represent the children of them. CAO* gives a true label value to each node and keeps the cost of reaching to target. This methodology is called as cost to go value in AO* searching terminology. Admissible upper and lower bounds do not underestimate the true label of the node. Untraversable stochastic edges are excluded to get the deterministic shortest path, which has the lower bound value. To update true label of root node, alternative expansions and propagations improve the tree. Nodes are expanded until non-terminal node with lower bound is searched at the expansion step. Lower bound values are always calculated before adding children of this node to the solution tree. The expansion node's lower bound is calculated again and again using lower bound values of its children at the propagation step.

Solution tree is populated with new lower bound information until it comes across with a node with unaffected lower bound [4, 10]. Statuses of nodes are refreshed updated at step of cost propagation. A node is expanded until it is marked as the terminal node.

CAO* inherits two important characteristics of D-SOSP: admissible upper bounds and state overlaps in the AO tree. Expanding only OR nodes is the basis feature of CAO*. Only if sum of distance to parent OR node and lower bound of an AND node is less than admissible upper bound value, it is added to child node list. Thus, AND node stands for direct path to target node. In particular, whenever the tree has a newly added AND node, the upper and lower bound values are recalculated by CAO*. During the step of propagation in CAO*, these bounds are used for pruning bad AND children of OR parents. If sum of lower bound and distance to parent of a node is higher than the highest upper bound at that level, it is called as a bad child. In that way, size of solution tree is reduced by pruning process. AO* with bounds (BAO)* is introduced as the discretized version of AO* search algorithm [21]. It has upper and lower bound utilization and improved pruning techniques to reduce run time of exact solution runtime. When no more disambiguation remains after current node is expanded, CAO* updates true label and the status flag of the last node as termination. This reduces the high cost of taking the children of this node into account individually to expand in the future again. CAO* searches for the AND node in the cache before adding it into tree. If same node has been added before, this expanded OR node is linked to the cached AND node as another parent and AND node is added to the children list of OR node. Significant improvements are recorded with the help of the caching mechanism in terms of memory usage and execution time.

2.5.4. CAON*

CAON* algorithm has extended from CAO* to solve obstacle neutralization problem on grid graph. Therefore, it is also an exact algorithm with features of caching and pruning [10]. In formal definition, solution tuple is represented as (s, t, C, L) where L indicates the neutralization capability and C is the neutralization cost.

In Figure 2.15, an example path is illustrated where neutralization count is 1 and 2 respectively. Straight blue line represents the path from starting node to obstacle. Red circles are the neutralized obstacles and orange is used for the color of path inside the obstacle. Dashed blue line is the path from obstacle to target node.

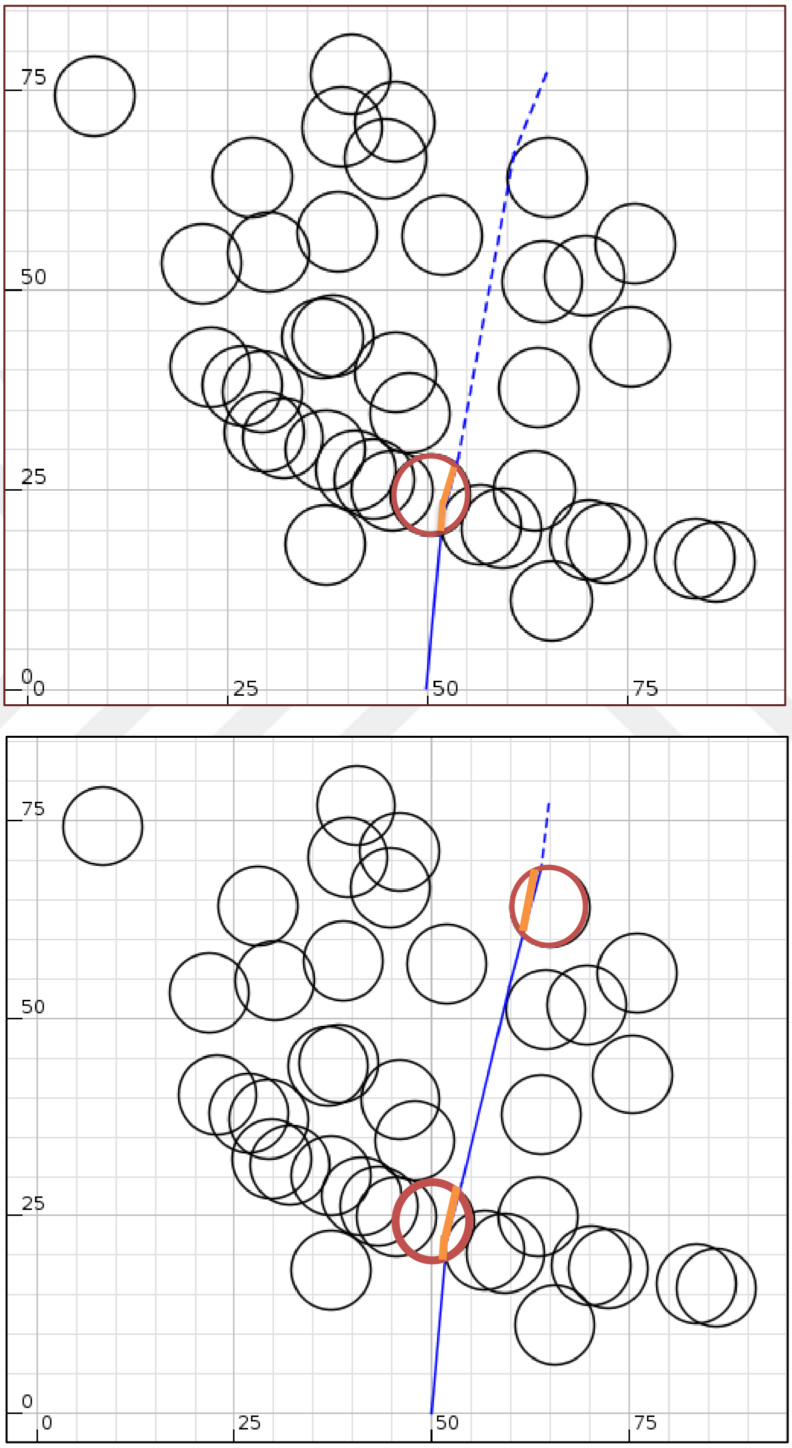

Figure 2.15 Shortest Paths where obstacle neutralization count is 1 and 2

3. IMPLEMENTATIONS AND WORK DONE

In this section, implementation methodology, design, datasets, algorithms and general structure of this work is given in detail.

The main idea is to adapt ANYA shortest path finder algorithm for the exact solution to solve path planning problem on discretized map with stochastic obstacles. Researchers mostly concentrate on improving the algorithm performance via heuristics or data structure enhancements. Dijkstra's algorithm is mostly preferred shortest path implementation spite of the drawback of processing time. In fact, it performs blind search even implemented as greedy. Especially, during an AO* tree is built; spanning paths cover almost all roads in the map at the end. The experiment results proved that shortest paths are on similar direction. But ANYA always routes through as if fine tuning is being applied for the perfect result.

Algorithm presentations in this work involve discretized grid graph and disk-shaped obstacles. Edges inside the disks are not accessible. Edges at the boundaries of these disk-shaped obstacles are candidate for disambiguation points.

ANYA only fails to pass joint cells on the grid graph. When obstacles are so close, adjacent cells become blocked. DA (Dijkstra's Algorithm) can pass this cell diagonally.

CAO* with ANYA implementation has following characteristics:

- Shape of obstacles is constructed as disks having parameters: center coordinates and radius.
- A cell is marked as traversable if interval from this cell to center of any passable obstacles is less than the radius.
- An obstacle becomes passable when it is disambiguated.
- When disambiguation count is non zero, total path length is summation of false and true state path lengths multiplied with the probability.

3.1. Material and Methodology

An object-oriented mechanism is designed for this project. Polymorphism and inheritance properties of Java programming language were applied.

Figure 3.1 shows a flow diagram to explain framework of algorithm executions. Disambiguation count and cost parameters are input to the solver class.

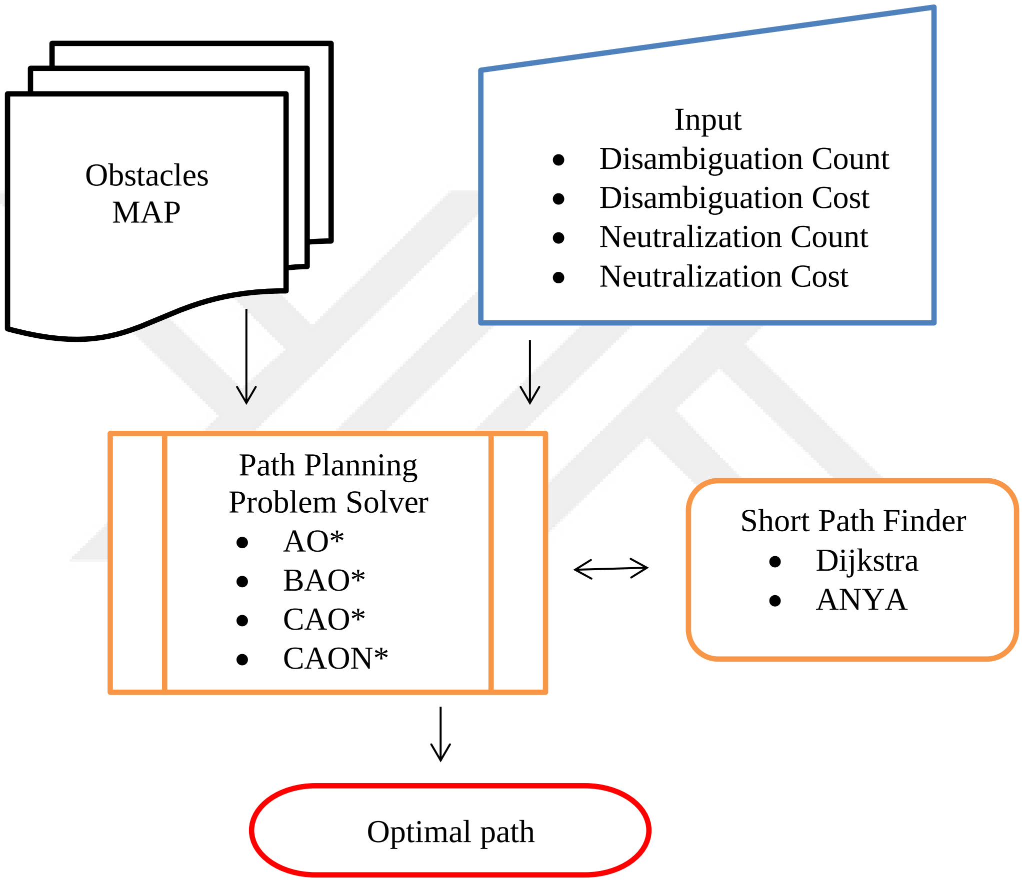

Figure 3.1 Path Planning Solver Framework

3.2. Dataset

Obstacles are loaded from text files. Figure 3.2 shows disk shaped obstacle file content. First 10 lines are obstacles. Last two lines are starting and end points. Graphical representation is shown in Figure 3.3. Starting and end point is marked as small blue dots. Obstacles having with *no=1, 4* and *10* are explicitly shown with *x* and *y* coordinates.

```
64.0092 64.1639 5 0.419896 1
85.4478 65.2474 5 0.659262 1
46.0695 50.8057 5 0.419539 1
83.1574 8.79742 5 0.935489 1
72.7933 73.9377 5 0.712397 1
45.8865 76.0289 5 0.952563 1
59.992 79.3296 5 0.915369 1
68.3216 40.0733 5 0.182677 1
26.7789 37.471 5 0.668525 1
46.4133 35.4161 5 0.0965257 1
50 100 0 0 0
50 1 0 0 0
```

Figure 3.3 Obstacle file content

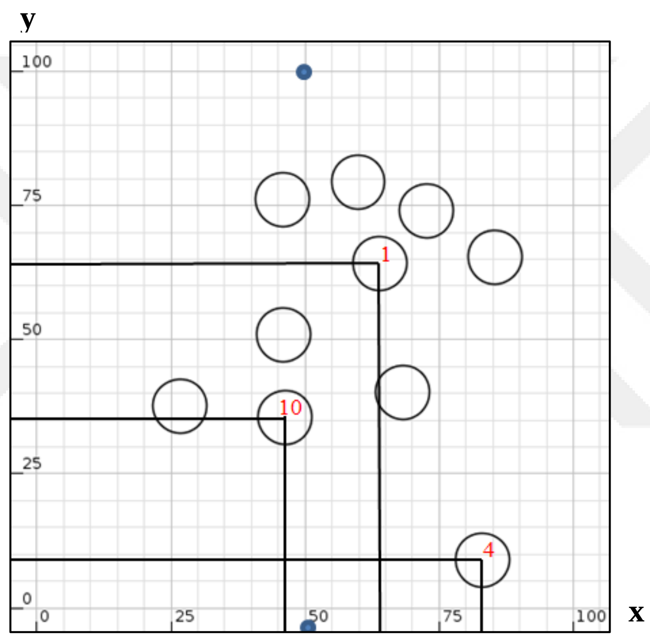

Figure 3.2 Disk Shaped Obstacles on Grid

Following attributes are used to construct the obstacle:

- x coordinates of center
- y coordinate of center
- Radius
- Disambiguation probability (0<p<1)
- Actual Existence of obstacle (true/false)

3.3. Solution Design

Figure 3.4 includes the class diagram for Shortest Path Finder and AO* implementations with disambiguation and neutralization.

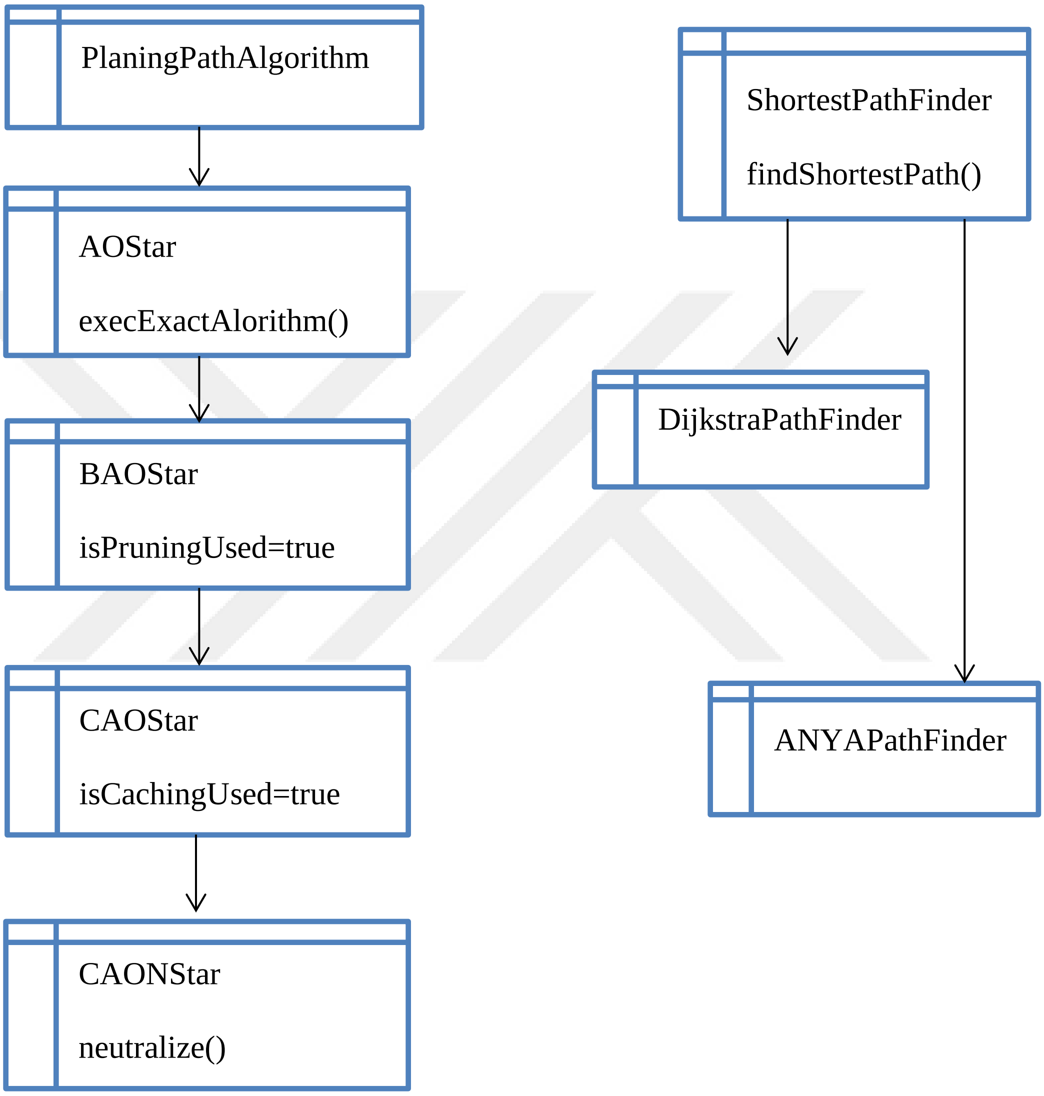

Figure 3.4 Class Diagram for the Problem and Shortest Path Finder

In Figure 3.5 includes class diagram for the problem definitions and grid graphs inheritances.

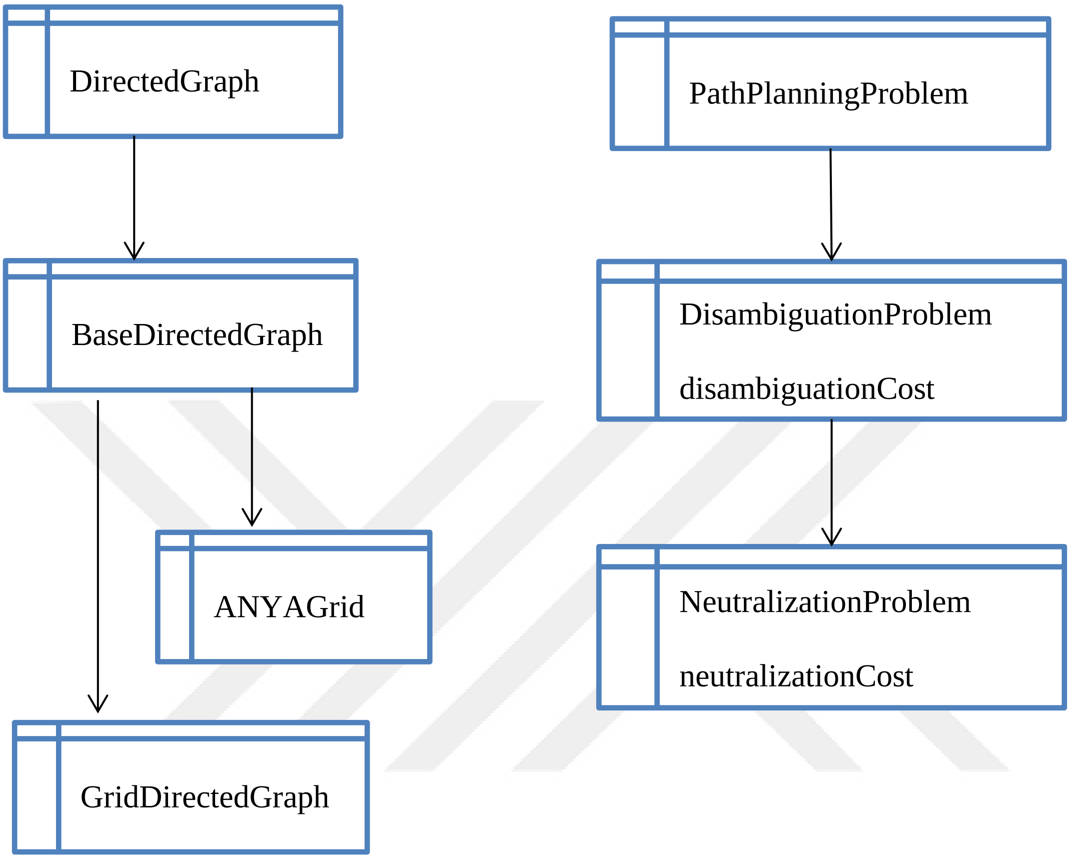

Figure 3.5 Class Diagram for Graphs and Exact Path Planning Algorithm

4. RESULTS AND DISCUSSIONS

We run all experiments on a computer having an Intel Core i7 8 cores CPU running at 2.20 Ghz using 16 GB of RAM, running Linux OS Ubuntu 18 x64. Main development and testing tool are Eclipse IDE [22] for Enterprise Java Developers (Version 2018-12) on Java 11 [15] runtime.

We also used java APIs with names Log4j [23] (logging services), JavaGeom (mathematics and two-dimensional geometry) and JgraphT [14] (java graph theory). Before deciding parametrization and datasets to use for algorithms, different test cases are conducted as the preliminary preparation. Combination of each parameter set is applied at least 5 times and their average is recorded computation results.

In this section, we introduce the experimental results, which were computed on both real and synthetic maps, and visualization of them via charts, graphs and tables. Path lengths, including disambiguation and neutralization costs, and execution times were recorded for AO* based path planning algorithms to compare the ANYA with the conventional Dijkstra's shortest path.

4.1. Computational Experiments with Disambiguation on a Real Instance

This section presents experiments and computations output to compare the performance and cost of path length with obstacle disambiguation capability. We used the minefield map (100x100 grid) belongs to U.S. Navy forces and this domain workspace is named COBRA data [24].

Regarding the map COBRA, shortest path algorithm calls almost 5000 times when disambiguation count equals to 1 whereas it reaches to more than 200000 for disambiguation count is 2. Number of cached nodes is 1000 and number of pruned nodes is 5500 nodes after 62 expansions when 2 obstacles are disambiguated.

In Table 4.1, solution quality of COBRA map is compared with different disambiguation count and cost combinations in detail. Optimal path length is same for all AO* based methods, which are differentiated by caching and pruning process.

Algorithm improvement reflects the execution time positively.

Table 4.1 Solution Quality Comparison Using COBRA

Disambiguation Count	**Cost**	CAO* Dijkstra	CAO* ANYA	BAO* Dijkstra	BAO* ANYA	AO* Dijkstra	AO* ANYA	CAO* Diff (%)
0	**0**	104.33	100.52	104.33	100.52	104.33	100.52	3.65
1	**0**	87.66	83.12	87.66	83.12	87.66	83.12	5.17
	2	89.66	85.12	89.66	85.12	87.66	85.12	5.06
2	**0**	85.80	82.83	85.80	82.83	**	**	3.46
	2	89.80	86.83	89.80	86.83	**	**[3]	3.30

In Figure 4.1, Solution quality comparison for the solution quality of CAO* and BAO* in terms of path length is visualized as line chart. Vertical axis shows disambiguation count and horizontal axis shows path lengths.

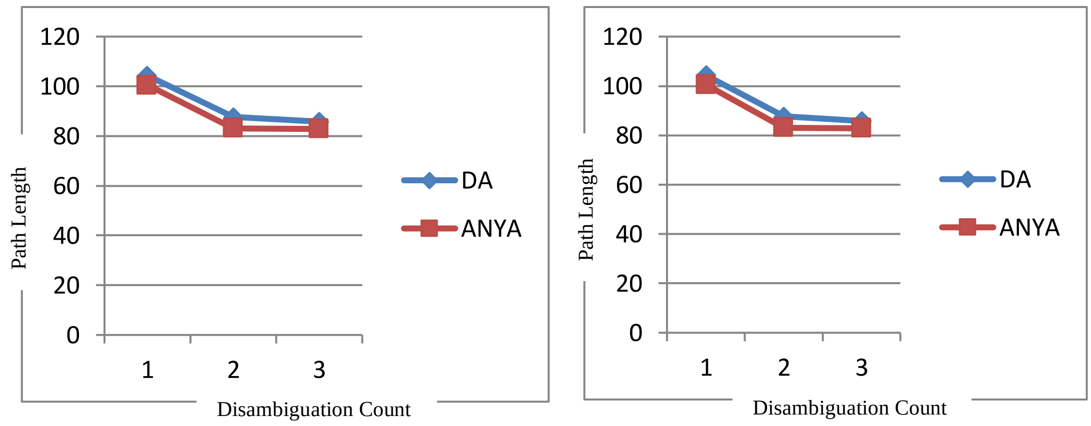

Figure 4.1 Line chart representation of Solution Quality comparison CAO* and BAO* with disambiguation count using COBRA

In addition to optimal shortest path length, comparison of path finder algorithm execution times is given in Table 4.2. Experiments proved that when number of

[3] Experiment was not completed in reasonable time with more than 200 expansions.

disambiguation count increases, there is also positive impact in runtime. We explain this behavior with the direct proportion to the number of shortest paths calls in codes. When disambiguation cost is 0, resulting path is names as zero risk distance and execution time is very close to 0. When pruning and caching features are disabled as AO*, experiment was not completed in reasonable time with more than 200 expansions by using for neither Dijkstra nor ANYA.

Table 4.2 Path Finder Execution Time Comparison Using COBRA

Disambiguation Count	CAO* (seconds)		BAO* (seconds)		AO* (seconds)		CAO* Diff (%)
	Dijkstra	ANYA	Dijkstra	ANYA	Dijkstra	ANYA	
0	0	0	0	0	0	0	-
1	3.4	3	3.5	3.1	3.2	3	13
2	149	88	191	116	**	**	69

At the end of CAO Test executions with ANYA, solution quality of the path is increased 5.6% in average. Runtime of shortest path algorithm also speeds up 12% in average when compared with Dijkstra. Execution of CAO* needs ANYA calls around 9000 times when disambiguation count is 2. When disambiguation count is 0, it was not worthwhile to run the algorithm to record timing.

Figure 4.2 shows this dataset (39 truly circular obstacles) on which ANYA shortest path with disambiguation count is 1. Figure 4.3 is also an exciting representation of the network which is formed by shortest paths drawn all together on same graph.

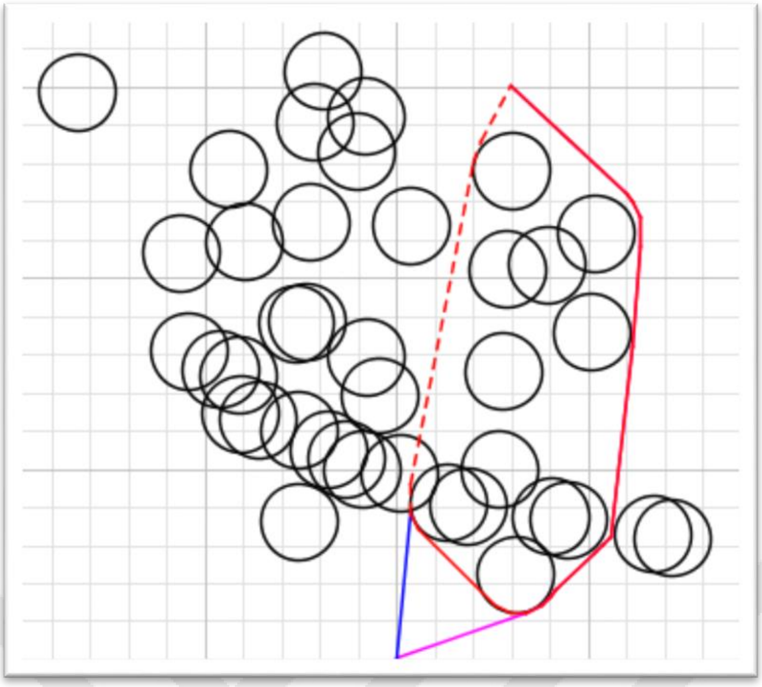

Figure 4.3 Optimal solution from ANYA on COBRA naval minefield data set with disambiguation count is 1

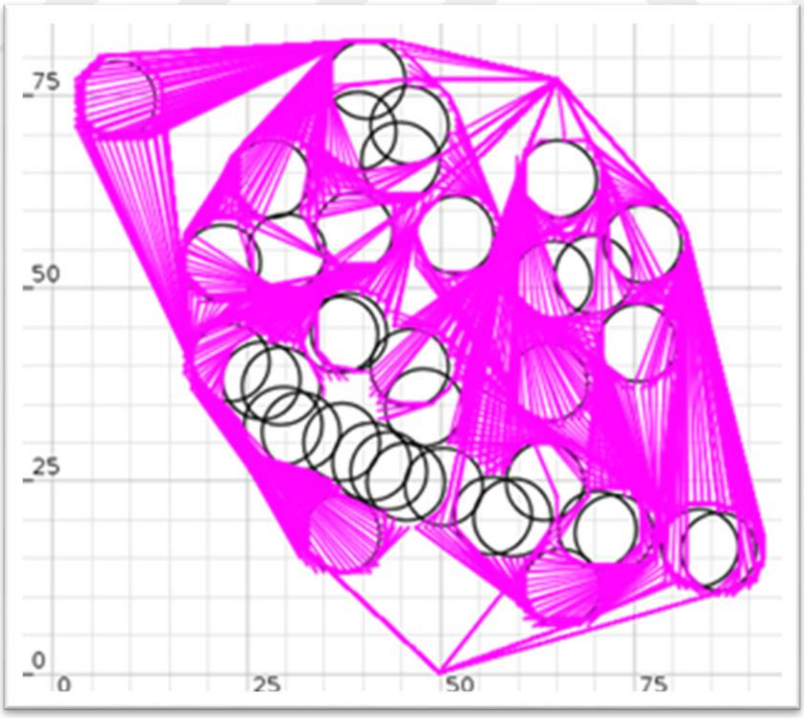

Figure 4.2 All CAO* shortest path network using the COBRA naval minefield data set with ANYA

4.2. Computational Experiments with Disambiguation on Synthetic Instances

Besides the COBRA navy map, a set of sample random maps are also used for the experimental comparison. More than 100 map files are generated and 10 of them are selected as the inputs to the CAO Tester class.

Common characteristics of these generated maps:

- Number of obstacles is 100
- Obstacles are disc shaped and have the radius length 5
- Each obstacle has random coordinates (x, y) for the center and the disambiguation probability ($0<p<1$)
- Starting vertex is (50, 0) and Target vertex is (50, 100)
- Grid map is represented as 100x100 matrix
- Disambiguation cost is 0

Due to the number of obstacles it is more illustrative to explain how ANYA beats Dijkstra's algorithm. There are plenty of zigzags in Dijkstra's shortest path result while ANYA draws smoother edges as shown on the right in Figure 4.4. Dijkstra shows an optimal solution but not an ideal one. ANYA does not only provide proper visual quality on the graph but also the optimal path length.

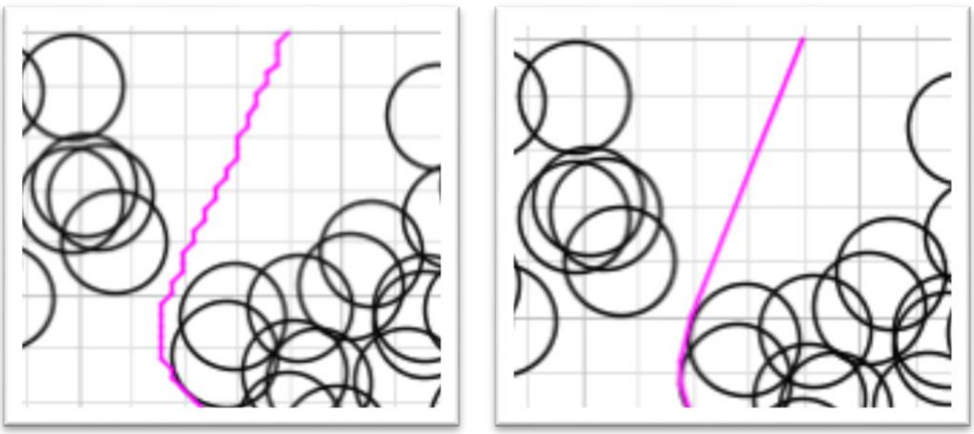

Figure 4.4 Shortest path comparison of Dijkstra vs. ANYA by using a random graph

Solution quality details for random maps are shown in Table 4.3 by comparing with different number of obstacle disambiguation where disambiguation cost is 0 and 1.

Table 4.3 Solution Quality Comparison using Different Random Graph Samples

Random Graph Sample Name	**Disambiguation Count**	CAO* (path length)			Shor-test path calls
		ANYA	Dijkstra	Diff (%)	
r2	**0**	110.14	113.94	3.45	1
	1	105.98	109.75	3.56	8712
r10	**0**	148.77	157.58	5.92	1
	1	139.10	144.29	3.73	8972
r17	**0**	112.95	120.71	6.87	1
	1	111.41	117.80	5.74	8207
r22	**0**	157.52	165.19	4.87	1
	1	156.39	164.00	4.87	8297
r31	**0**	121.65	131.29	7.92	1
	1	121.25	130.64	7.74	8987
r57	**0**	100.77	104.14	3.34	1
	1	100.76	103.89	3.11	9452
r76	**0**	110.06	117.39	6.66	1
	1	107.36	114.38	6.54	9292
r85	**0**	111.74	119.88	7.28	1
	1	110.21	116.63	5.68	8872
r97	**0**	110.62	118.22	6.87	1
	1	105.84	113.39	7.13	8452
Avg.		119.03	125.73	5.63	8804
Std. Dev.		18.44	19.38	1.63	428

Execution time comparison details for random maps are also given in Table 4.4 where disambiguation cost is 0 and 1.

Table 4.4 Path Finder Algorithm Execution Time Comparison using Different Random Graph Samples

Random Graph Sample Name	**Disambiguation Count**	CAO* (seconds)			Shor-test path calls
		ANYA	Dijkstra	Diff (%)	
r2	**0**	0.02	0.02	-	1
	1	5.35	12.72	137.76	8712
r10	**0**	0.02	0.02	-	1
	1	5.92	16.07	171.45	8972
r17	**0**	0.02	0.02	-	1
	1	5.00	12.55	151.00	8207
r22	**0**	0.02	0.02	-	1
	1	5.23	14.27	172.85	8297
r31	**0**	0.02	0.02	-	1
	1	5.25	11.30	115.24	8987
r57	**0**	0.02	0.02	-	1
	1	6.11	11.26	84.29	9452
r76	**0**	0.02	0.02	-	1
	1	6.13	13.03	112.56	9292
r85	**0**	0.02	0.02	-	1
	1	6.05	11.07	82.98	8872
r97	**0**	0.02	0.02	-	1
	1	6.15	11.70	90.24	8452
Avg.		2.85	6.34	124.26	8804
Std. Dev.		2.93	6.60	35.64	428

In Figure 4.5, CAO* shortest path network almost covers the full graph. This span also gives an idea about the edges of path found via Dijkstra's algorithm on same map.

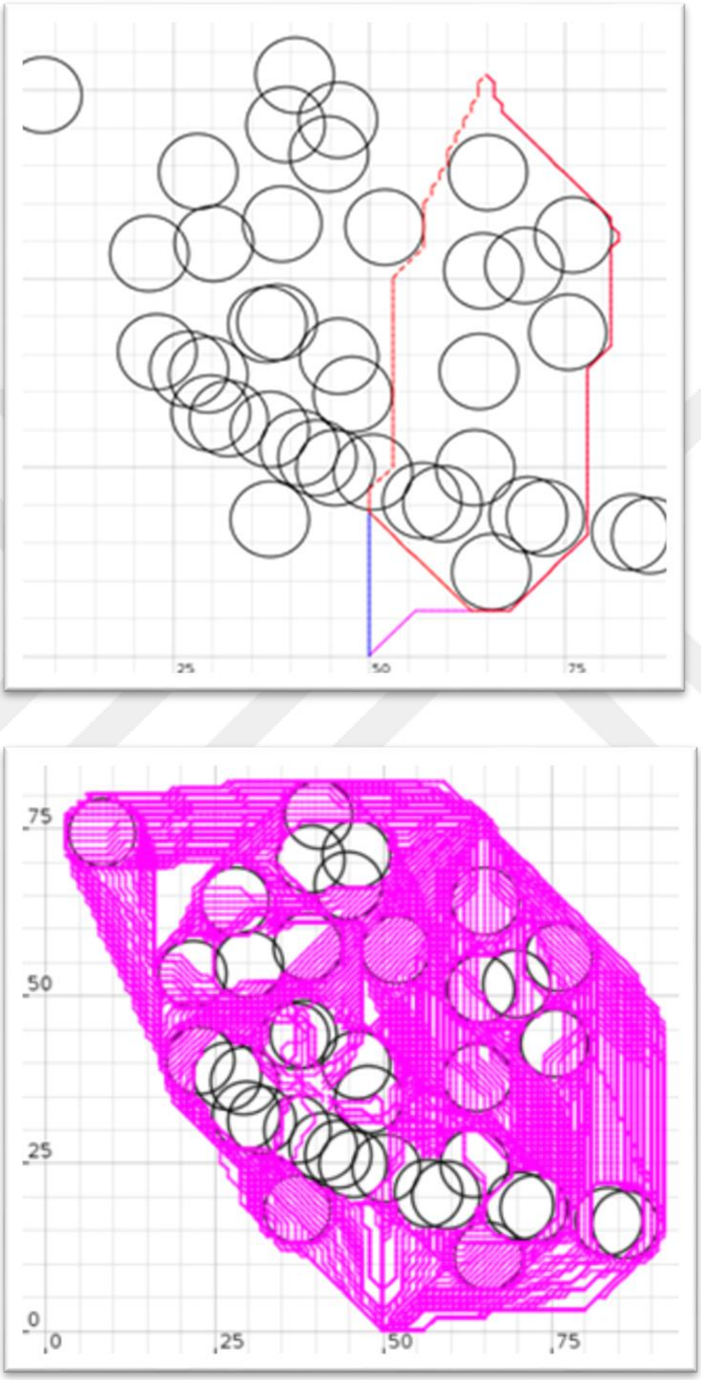

Figure 4.5 All CAO* shortest path network using the COBRA naval minefield data set with Dijkstra

4.3. Computational Experiments with Neutralization on a Real Instance

This section presents experiments and computations output to compare the performance and cost of path length for the obstacle neutralization capability. We used random graphs and COBRA data [24].

Regarding the map COBRA, shortest path algorithm calls almost 500 times when neutralization count equals to 1 whereas it reaches to more than 2000 for neutralization count is 2. Number of cached nodes is 100 and number of pruned nodes is 200 nodes after 7 expansions when 2 obstacles are neutralized.

In Figure 4.6 shows an example with neutralization of 3 obstacles on COBRA map. Optimal path passes thorough neutralized obstacles, filled with color blue, with predefined additional cost.

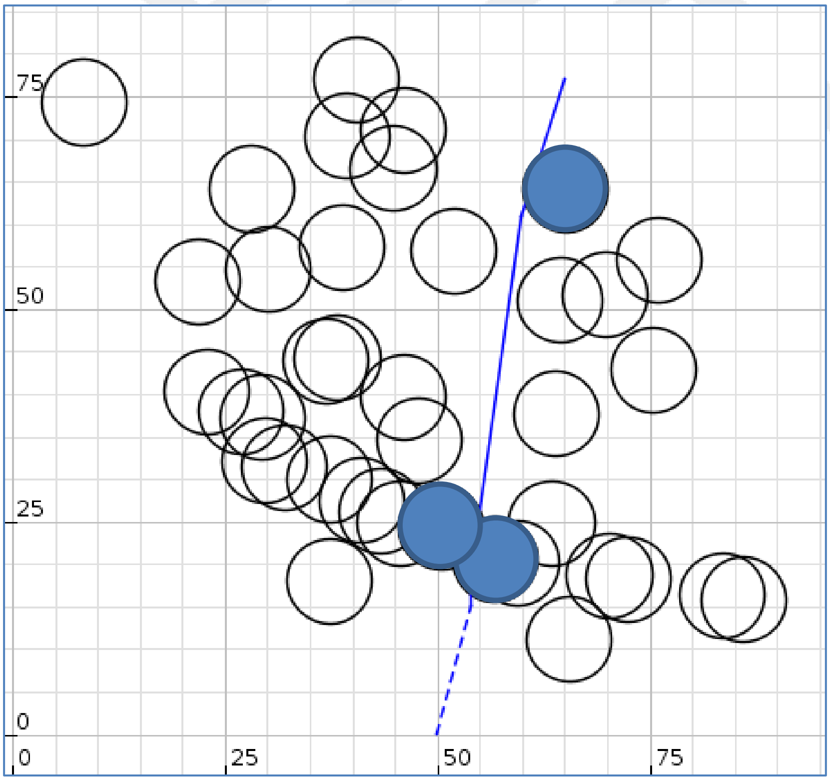

Figure 4.6 CAON* shortest path using the COBRA naval minefield data set with ANYA when neutralization count is 3

In Table 4.5, solution quality of COBRA map is compared with different neutralization count and cost combinations in detail. When neutralization count is 0, shortest path algorithm finds exactly same path length and route with the disambiguation experiments. When neutralization count is 3, CAON* really neutralizes all 3 obstacles after 19 expansions on the path determined via ANYA. Whereas, only 2 obstacles after 10 expansions are neutralized by Dijkstra's although more obstacle neutralization options are available.

Table 4.5 Solution Quality Comparison Using COBRA with Neutralization Capability

Neutralization		CAON*		CAON* Diff (%)
Count	**Cost**	ANYA	Dijkstra	
0	**0**	100.52	104.32	3.78
1	**0**	79.39	84.04	4.65
	2	81.39	86.04	4.65
2	**0**	78.80	83.22	4.42
	2	80.80	85.22	4.42
3	**0**	78.71	83.22	4.51
	2	84.71	87.22	2.51
4	**0**	78.71	83.22	4.51
	2	84.71	87.22	2.51

Comparison of path finder algorithm execution times is given in Table 4.6. Experiments proved that number of path finder algorithm call linearly affects the execution time. When neutralization cost is 0, there is also positive impact to decide to neutralize to solve the path planning problem. After $L >= 4$, number of expansions reaches to 10K and working with COBRA map takes longer execution times and not more than 3 obstacles are neutralized even with zero cost. Therefore, CAON* with ANYA may not report better worse performance in such cases.

Table 4.6 Path Finder Execution Time Comparison Using COBRA with Neutralization Capability

Neutralization Count	CAON* (seconds)		CAON* Diff (%)
	ANYA	Dijkstra	
0	0	0	-
1	0.42	0.74	76
2	1.33	2.44	83
3	2.39	3.99	67
4	5.41	5.30	-2

In Figure 4.7, comparison for the solution quality of CAON* in terms of path length and execution time is visualized as line chart. Vertical axis shows neutralization count. Horizontal axis shows path lengths and execution time in seconds respectively.

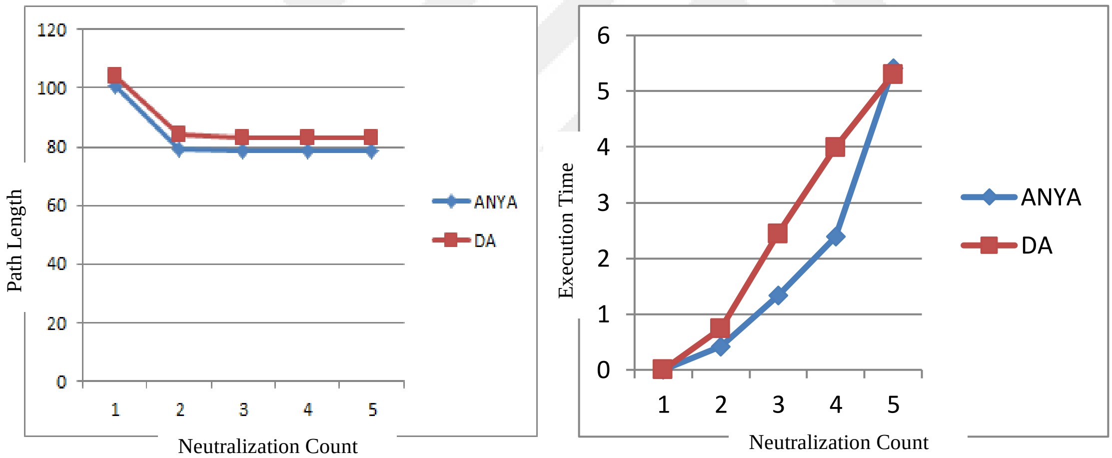

Figure 4.7 Line chart representation of Solution Quality comparison for CAON* with neutralization count using COBRA

Figure 4.8 and Figure 4.9 illustrates that using ANYA and Dijkstra's Shortest Path Finder algorithm may results with different neutralization points on different obstacles. Dijkstra's method can pass through the red line to neutralize green disk on the left. This proposition comes from following root causes:

- Representation of disk-shaped obstacle on grid area

- Non-visible points of ANYA

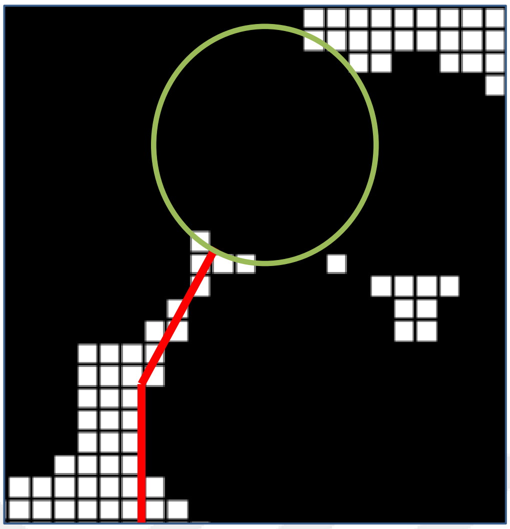

Figure 4.8 CAON* with ANYA neutralization drawback due to non-visible points on a random grid map

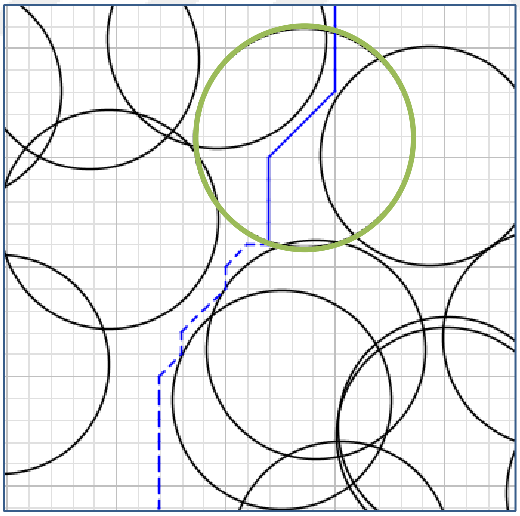

Figure 4.9 CAON* with Dijkstra's SP neutralization points on a random grid map

Comparison of path finder algorithm solution quality with both disambiguation and neutralization capability is given in Table 4.7.

Table 4.7 Solution Quality Comparison Using COBRA with both Disambiguation and Neutralization Capability

Neutralization Count	**Disambiguation Cost**	CAON*		CAON*Diff (%)
		Dijkstra	ANYA	
1	**1**	83.93	79.33	5.48
	2	84.29	79.39	5.81
2	**1**	83.93	79.24	5.58
	2	84.15	78.80	6.35

Figure 4.10 is the illustration of a path planning with disambiguation and neutralization counts are 1 on COBRA map. There are 3 paths in the graph: dashed red line is the first neutralization point, dashed blue line is the route to disambiguation point, and straight blue lines are latest part of paths after neutralization of these obstacles.

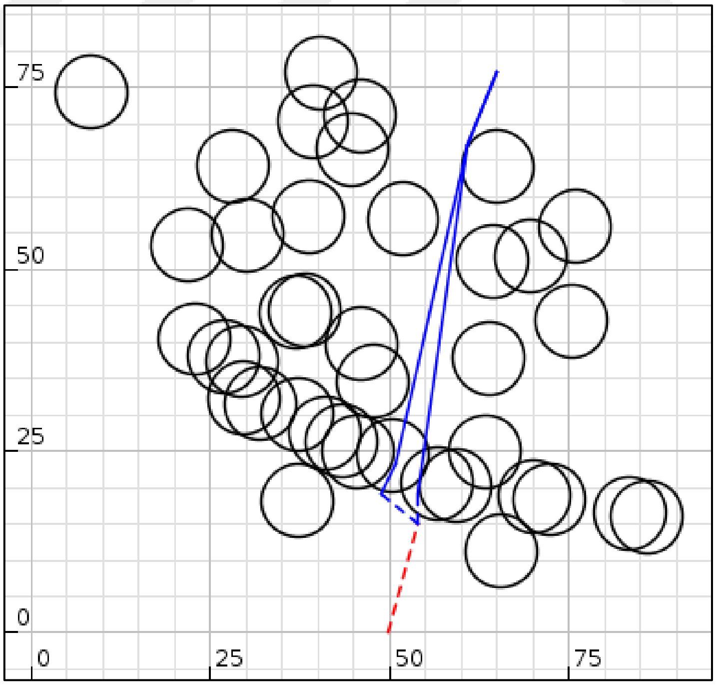

Figure 4.10 CAON* path planning using the COBRA naval minefield data set with ANYA when both disambiguation and neutralization counts are 1

Table 4.8 Path Finder Execution Time Comparison Using COBRA with both Disambiguation and Neutralization Capability

Neutralization Count	**Disambiguation Cost**	CAON*(seconds)		CAON* Diff (%)
		Dijkstra	ANYA	
1	**1**	7.09	1.91	73
	2	21.10	10.65	49
2	**1**	17.02	11.10	34
	2	33.42	23.05	31

4.3. Computational Experiments with Neutralization on Synthetic Instances

Solution quality details for random maps are shown in Table 4.9 by comparing with different number of obstacle neutralizations where neutralization cost is 0.

Table 4.9 Solution Quality Comparison using Different Random Graph Samples with Neutralization Capability

Random Graph Sample Name	**Neutralizati on Count**	CAO* (path length)			Shortest path calls
		ANYA	0Dijkstra	Diff (%)	
r2	**1**	106.36	110.62	3.85	975
	2	103.64	107.45	3.55	45392
r10	**1**	134.61	116.32	-15.72	1011
		108.26	113.01	4.20	26778
r17	**1**	105.68	113.25	6.68	918
	2	102.93	108.04	4.73	6999
r22	**1**	154.88	162.36	4.61	978
	2	133.06	138.56	3.97	24669
r31	**1**	108.29	114.08	5.08	999
	2	106.79	113.25	5.70	21512
	1	100.08	101.65	1.54	1034

r57	**2**	100.08	101.65	1.54	8152
r76	**1**	102.52	109.11	6.04	1015
	2	101.82	107.45	5.24	14846
r85	**1**	107.44	114.08	5.82	1025
	2	103.99	109.11	4.69	28340
r97	**1**	106.07	109.94	3.52	898
	2	103.02	108.28	4.86	11451
Avg.		110.52	114.34	3.32	10944
Std. Dev.		14.77	14.31	4.94	13221

Execution time comparison details with different number of obstacle neutralizations for random maps are also given in Table 4.10 where neutralization cost is 0.

Table 4.10 Path Finder Algorithm Execution Time Comparison using Different Random Graph Samples with Neutralization Capability

Random Graph Sample Name	**Neutralization Count**	CAO* (seconds)			Shortest path calls
		ANYA	Dijkstra	Diff (%)	
r2	**1**	0.96	1.76	45.45	975
	2	31.51	69.95	54.95	45392
r10	**1**	0.92	2.51	63.35	1011
	2	20.22	55.01	63.24	26778
r17	**1**	0.74	1.97	62.44	918
	2	4.84	8.25	41.33	6999
r22	**1**	0.89	3.83	76.76	978
	2	19.18	48.48	60.44	24669
r31	**1**	0.81	1.71	52.63	999
	2	15.82	33.70	53.06	21512
r57	**1**	0.81	2.94	72.45	1034
	2	5.14	15.19	66.16	8152
r76	**1**	0.81	1.78	54.49	1015

	2	9.72	34.70	71.99	14846
r85	**1**	0.80	2.78	71.22	1025
	2	18.65	44.03	57.64	28340
r97	**1**	0.72	2.66	72.93	898
	2	7.58	33.82	77.59	11451
Avg.		7.78	20.28	62.11	10944
Std. Dev.		9.34	22.54	10.54	13221

5. CONCLUSIONS

Game Programmers and AI researchers are interested in optimal path if available and ANYA is confirmative and evident solution with its adaptations in exact path finding algorithms.

5.1. Summary

Finding the shortest path is considered one of the most investigated graph problems. It is also one of the oldest as it has been researching since 19th century. Later finding the routes between physical locations became a common research area with variations and under certain conditions. In CTP, a traveler makes a sequence of decisions to choose a route from a source to a target through a determined set of certain shaped obstacles in a scattered region. In ONP, a traveler has neutralization capability for the blocked routes. All variant of path planning problems meets the common point: an underlying adapted shortest path finder. ANYA is the most recent alternative of the path finding algorithms. It is online, optimal and efficient technique to use as the shortest path finder in D-SOSP.

In this work, we consider the D-SOSP as the most challenging optimization part and the CAO* algorithm results with an exact solution for the AO* search-based problem. Time complexity of CAO* is not polynomial but it requires fewer hardware resources when compared to basic AO*. It is a special structure of D-SOSP to fulfill it. Especially, CAO* has the cache to and does not need expanding visited states before and needs less space. It dynamically prunes state space at the node level via acceptable upper and lower bounds for faster search. CAON* has also the neutralization methods additional to expansion, caching and pruning features of CAO*. Conventionally, it considers finding upper bound shortest paths from all applicable disambiguation points. This results an expected increase in execution time due to the need for shortest path algorithm execution. We introduce the ANYA, which do not need the lattice graph edges, embedded to CAO* and CAON*. Anya does not depend on any preliminary calculations, does not add any memory expenses to find an optimal path length. We use the COBRA map and other randomly generated synthetic data domain in our tests.

We indicate with our experiments that path finding without pre-defined edges obviously shorten the expected shortest path with disambiguation. Our conclusion is that path planning is the central position of all travelling problems. Data structure and speed is the vital properties to navigate from source to target. Since large graphs can require long execution time and high memory, the process can be exited before planning is completed. Traditional shortest path algorithms need to discretize a continuous environment. Since less memory is required for the graphs having small number of vertices and edges in a characteristic manner-choosing ANYA gains much more advantages in memory usage.

5.2. Future Work

Due to the easiness and performance constraints, we preferred to work in grid area. Edges on the path can be considered as continuous lines but ANYA path contains corners on grid map and does not draw arcs on the obstacle boundaries. Working on fully continuous graph would be also challenging since our work showed that ANYA adaptation contributes to solution quality in path length and execution time.

Heuristics are alternative solvers of path planning problems to provide near optimal paths. Reset Disambiguation (RD) Policy is proved to find optimal path for a special variant of SOSP [25]. It is generally less than optimal but efficient in computation time. Term of resetting is to change the status of visited edge from traversable to ambiguous after disambiguation. Determination of optimal reset policy prevents from disambiguating same edge after it is found to be not traversable. Basis is to use optimal edge weights of reset variant or optimal edges for parallel graphs in original version without reset. Distance to Termination (DT) Algorithm is introduced as the navigation guidance between current and terminating nodes by considering both disambiguation cost and penalty function.

Another possible problem adaptation is to work with new shapes of the obstacle which are not geometric, irregular obstacles like S-shape, Z-shape, T-shape or L-shape in Tetris.

REFERENCES

1. Harabor D, Grastien A, Oz D, Aksakalli V (2016) Optimal Any-Angle Pathfinding In Practice Journal of Artificial Intelligence Research 56: 89-118.
2. Harabor D, Grastien A (2013) An Optimal Any-Angle Pathfinding Algorithm. In Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 201
3. Aksakalli V, Sahin O F, Ari I (2016) An AO* Based Exact Algorithm for the Canadian Traveler Problem. INFORMS Journal on Computing 28(1): 96-111.
4. Yildirim S, Aksakalli V, Alkaya A F (2019) Canadian Traveler Problem with Neutralizations, Expert Systems with Applications, vol. 132, pp. 151-165.
5. Liao C S, Y. Huang Y, (2014) The Covering Canadian Traveler Problem, Theoretical Computer Science 530: 80–88
6. Aksakalli V, Ari I (2014) Penalty-based algorithms for the stochastic obstacle scene problem. INFORMS J. Comput. 26(2): 370–384.
7. Ye X, Priebe C E (2010) A graph-search based navigation algorithm for traversing a potentially hazardous area with disambiguation. Internat. J. Oper. Res. Inform. Sys.1: 14–27
8. Aksakalli V, Fishkind DE, Priebe C E, Ye X (2011) The reset disambiguation policy for navigating stochastic obstacle fields. Naval Res. Logist. 58(4):389–399.
9. Aksakalli V, Ceyhan E (2012) Optimal Obstacle Placement with disambiguations. Ann. Appl. Stat. 6(4): 1730–1774.
10. Yildirim S, Aksakalli V, Alkaya A F (2016), Disambiguation points sampling heuristic for the stochastic obstacle scene problem, International Conference on IEOM, Kuala Lumpur, Malaysia, 8-10 March 2016, pp. 1-10
11. Alkaya A F, Aksakallı V, Priebe C E (2015) A penalty search algorithm for the obstacle neutralization problem. Computers and Operations Research, 53, pp. 165-175.
12. Alkaya A F, Oz D (2017) An optimal algorithm for the obstacle neutralization problem. Journal of Industrial & Management Optimization, 13 (2): 835-856
13. Hart P E, Nilsson N J, Raphael B (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths, in IEEE Transactions on Systems

Science and Cybernetics, vol. 4, no. 2, pp. 100-107

14. “JGraphT Homepage” [Online] Available: https://jgrapht.org
15. “Java Homepage” [Online] Available: https://www.java.com
16. Dumitrescu I, Boland N, (2001) Algorithms for the weight constrained shortest path problem. International Transactions in Operational Research 8 15–29.
17. Zabarankin M, Uryasev S, and Pardalos P (2002) Optimal risk path algorithms. In Cooperative control and optimization, pages 273–298. Springer
18. Kuipers F, Korkmaz T, Krunz M, and Van Mieghem P (2004) Performance evaluation of constraint-based path selection algorithms. IEEE Network, 18(5):16–23
19. “Wikipedia Homepage” [Online] Available: https://en.wikipedia.org
20. Nash A, Koenig S (2013) Any-Angle Path Planning. AI Magazine, 34 (4), 9.
21. Aksakalli V (2007) The BAO* algorithm for stochastic shortest path problems with dynamic learning, 46th IEEE Conference on Decision and Control, pp. 6003-6008.
22. “Eclipse Homepage” [Online] Available: https://www.eclipse.org
23. “Apahe Log4j Homepage” [Online] Available: https://logging.apache.org/log4j
24. “COBRA” Costal Battlefied Reconnaissance and Analysis [Online] Available: http://www.navy.mil/navydata/fact_display.asp?cid=2100&tid=1237&ct=2
25. Aksakalli V, Fishkind D E, Priebe C E and Ye X. (2011) The reset disambiguation policy for navigating stochastic obstacle fields. Naval Res. Logist. 58 389–399.

BIOGRAPHY

Ufuk ASLAN

Marmara University, Goztepe Campus
Engineering Faculty, Computer Science and Engineering Dept.
PK:34722, Kadikoy, Istanbul, Turkiye

e-mail: uaslan@gmail.com

PERSONAL INFO

Place of Bith: Istanbul, Turkey

Date of Birth: 1981

EDUCATION

B.S., Computer Science and Engineering Marmara University, Faculty of Engineering

Istanbul, Turkey, (1999-2004) GPA:3.47 (rank 3rd in dept.)

WORK EXPERIENCE

(2013, present) Globit Global Bilgi Teknolojileri A.S., Istanbul, Turkey

- Technical Expert and Team Leader, Professional Services and Support

(2008, 2013) 32Bit Bilgisayar Hizmetleri Ltd. Sti., Kocaeli, Turkey

- Senior Software Developer, Fintech Implementations

(2004, 2008) 32Bit Bilgisayar Hizmetleri Ltd. Sti., Istanbul, Turkey

- Junior Software Developer, Infotainment Systems, Bank Treasury Applications

---

FOREIGN LANGUAGE

English (Upper Intermediate)

RESEARCH INTERESTS

Shortest Path, Path Planning and Task Scheduling