5-7-2018

Multi-Objective Pathfinding in Dynamic Environments

Halen Whiston
University of Windsor

Multi-Objective Pathfinding in Dynamic Environments

By

Halen D. Whiston

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2018

MULTI-OBJECTIVE PATHFINDING IN DYNAMIC ENVIRONMENTS

BY

HALEN D. WHISTON

APPROVED BY:

M. Hlynka

Department of Mathematics and Statistics

A. Jaekel

School of Computer Science

S. Goodwin, Advisor

School of Computer Science

April 26, 2018

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix. I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Traditional pathfinding techniques are known for calculating the shortest path from a given start point to a designated target point on a directed graph. These techniques, however, are inapplicable to pathfinding problems where the shortest path may prove to be hazardous for traversal, or where multiple costs of differing unit-types lie along the same path. Moreover, the shortest path may not be optimal if it requires forfeiting a valuable resource. While strategic methods have been proposed in the past to completely avoid paths determined to be dangerous, these methods lack the functionality to provide agents the ability to decide which resources are more valuable for conservation, and which resources possess the greatest risk at being lost.

For environments where risk varies dynamically across edges, we propose a solution that can determine a path of least expected weight based on multiple properties of edges. With this Multi-Objective Pathfinding technique, agents can make decisions influenced by highest priority objectives and their preferences to trading off some resources for others. The solution is based on traditional pathfinding techniques, extending their usability to cover strategic and dynamic scenarios where additional properties contained within the search map could render them useless. Nevertheless, our solution is compatible with problems where the goal is to simply find the least weighted path, otherwise known as the objectively resource-conservative path among a set of vertices in a graph.

Dedication

To my beloved family:

Grandparents: Theresa (Lain) & Phillip Yoong, Helen & Dennis Whiston.

Parents: Priscilla & Jason.

Siblings: Shaelyn, Keagan, and Quintan.

Acknowledgements

I would like to thank Dr. Scott Goodwin for working as my advisor and providing guidance throughout the course of my graduate term at the University of Windsor. I am honoured to have had this opportunity to work with him and could not have asked for a better supervisor, speaking as a relatively new member in the field of scientific research.

I would like to thank my internal reader, Dr. Arunita Jaekel, for showing her support and having an interest in my research area. I would also like to thank my external reader, Dr. Myron Hlynka, for providing his support for my defense.

I would like to give a special thanks to the graduate secretary, Mrs. Karen Bourdeau, for helping me whenever I had difficulties understanding the logistics of the program, and also for answering questions I had pertaining to the curriculum in general. Her assistance was greatly appreciated.

Finally, I would like to thank my family for supporting me over the course of my academic career. Although you already know it, I just want to say that I love you all so very much.

Table of Contents

List of Figures

List of Algorithms

Chapter 1: Introduction

1.1 Problem Domain

Determining the shortest path from a start position to a destination is a necessary goal for travelers, as the shortest possible route will always lead to the least amount of resources being spent. [1] This process is commonly known as pathfinding, and it is widely used within the game development industry, typically in the form of computer-controlled agents that rely on artificial intelligence for their path planning in order to provide an engaging experience for the player. Outside of entertainment software, pathfinding may also see uses in optimizing resource space in warehouse management and geographical exploration.

Traditionally, pathfinding has been used to determine a minimum cost path between nodes through the use of several *best-first search* algorithms that rely on resource-related or distance-related costs. While this field has been widely studied by numerous contributors in the past, the situation changes when a probabilistic factor is included in the problem that attributes to modifications in edge costs. A prime example of this would be in a battlefield setting, where certain areas on the graph may be treacherous for the agent to traverse due to strategically placed hazards (landmines, sentries, etc.). If the probability of triggering one of these hazards is too great, it may be more beneficial for our agent to find an alternative route around these hazards, even if the optimal path requires the agent to take a risk. Conversely, if the risk of danger is not great, then perhaps traversing the hazards would lead to an optimal solution after all. Nevertheless, the agent must be able to evaluate how much risk it is willing to take at the cost of

not being able to find the shortest path. It is in cases like these where the shortest distance path no longer becomes our optimal path, as choosing to follow such a path could forfeit valuable resources (such as the agent's vitality in the example). This type of scenario adds another layer to traditional pathfinding problems in the form of additional properties contained within the search map.

1.2 Pathfinding

The objective in pathfinding (or path planning) is to find an optimal path that can be routed between two predetermined nodes, referred to as the *start node* and the *goal node* respectively. If we map this out on a graph *G*, we can consider two existing vertices on that graph, *s* and *g* and the various paths that may exist between them in order to determine an optimal solution. This is the type of problem travelers are presented with when they have to find a path from their current location to their destination. We refer to the travelers utilizing pathfinding strategies as *agents*. An agent can be best described as, "a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future." [2] In the simplest of cases, the agent represents the actor or character utilizing the pathfinding solution in the environment.

1.2.1 Graph Representation

Pathfinding environments are often represented as maps, typically in one of three classic representations: Grids, Waypoint Graphs, and Navigation Meshes (abbreviated Navmesh). Grids are made up of a series of tiles representative of terrain and contain the node data for each available vertex on the map. They can be generated relatively quickly and are also easy to

implement in typical pathfinding frameworks due to their block-like structure. In a grid, each tile represents a node in the map, and each node has its own set of edge-costs that must be paid in order to allow traversal to or from respective neighbouring nodes.

Waypoint graphs (Figure 1) are another method of map representation. They are composed of a series of nodes placed at different locations in a map to represent positions an agent can either stay on or pass through. Lines representing the edges conjoin vertices in a waypoint graph and denote the edges in which traversal is possible. Waypoint graphs have the added benefit of containing fewer vertices than a grid layout of the same map, making them ideal for representing the environment of an enclosed area, or an area that is partitioned. The disadvantage, however, is that as vertices are added to the waypoint graph, the complexity of both rendering and navigating these vertices increases, which is why these types of layouts tend to be used in environments where a minimal amount of nodes is expected.

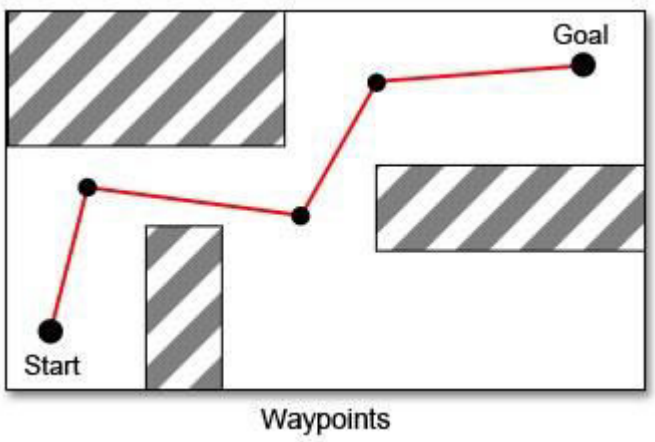


Figure 1: Search map depicting waypoint navigation. An agent may travel along a nearby edge to reach a neighbouring node. [16]

The final type of graph representation known as a navmesh (Figure 2) is effective at representing a real-world layout in a graph form. Navmeshes are made up of interconnected polygons; borrowing attributes from both grids and waypoint graphs to illustrate a traversable area. The

nodes themselves are within each polygon of a navigation mesh, and agents that are situated in each polygon have free reign of traversal within that polygon without having to travel to a neighbouring vertex. This makes navmeshes ideal for representing complex or polygonal terrain, as the meshes can be altered to fit within the desired environment.

In this thesis, grids will be used to represent the search map for all experiments pertaining to the proposed solution.

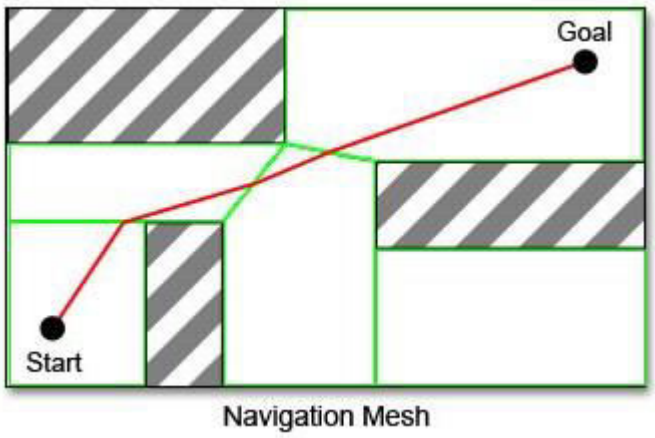


Figure 2: Search map depicting navigation using navmeshes. An agent may freely move within an area of a navmesh if a node it is stationed on occupies it. [16]

1.2.2 Search Algorithms

For solving pathfinding problems, search algorithms are used to find an optimal path between any two vertices on a graph. Uninformed search algorithms such as *Depth-First Search* and *Breadth-First Search* exhaust every possible node relative to the starting position until the goal is reached. They are a type of brute force approach, as no knowledge of the destination's location is kept in memory. Conversely, informed searches can more accurately discern the location of the goal by utilizing known attributes of edge-weights. *Dijkstra's Algorithm*, for example, finds and compares the shortest possible path from the current node to all of its nearest neighbours. By

only choosing to follow the edges with the smallest weights, the method can effectively route the shortest path without having to visit nodes that may never need to be traversed. It is more efficient than typical uninformed searches, since there is a direct correlation between the speed of a solution and the number of nodes that require expansion, as described by Nilsson et al. where optimal algorithms "examine the smallest number of nodes necessary to guarantee a minimum cost selection." [4]

1.2.3 Admissibility

The goal of pathfinding is to generate a subgraph representative of an optimal path from any given node *s* to its respective goal node *g*. We refer to these algorithms as *admissible* if they can guarantee to find an optimal path if such a path exists in the graph. [4] Likewise, when using heuristics for informed searches, it is imperative that the estimated cost never exceeds that of the actual cost to reach a respective node. Moreover, *consistent* heuristics must satisfy the triangle inequality:

$$\square(n) = 0$$

$$h(n) \leq c(n, n') + h(n'); for\ all\ vertices\ n \in N\ and\ n' \in succ(n)\ where\ n \neq n_{goal}$$

Where *h(n)* represents the heuristic function of the current node, n. The triangle inequality ensures that the heuristic value will consistently be less than or equal to the actual cost of the current node to its neighbour plus the estimated distance of the neighbour to the goal, resulting in equal estimation at worst. It is this property that defines *heuristic consistency*, meaning that the sum of two sides of a triangle must be longer than each individual side. The *A* Search* algorithm is a prime example of a pathfinding solution that relies on admissible heuristics, as not only is the method guaranteed to find an optimal path (provided one exists), but its cost estimates are also

never overestimated relative to the actual costs between nodes. Admissible heuristics will always be optimistic with regards to cost estimation. [1]

In general, the *informedness* of a heuristic is comparable to other potential heuristics under the following theorem:

$$\forall n \mid h_1(n) \leq h_2(n) \leq h_x(n) \rightarrow h_2 \; is \; more \; informed \; than \; h_1$$

A heuristic that is more informed than the alternative will also find the optimal path by expanding fewer nodes in the same scenario.

1.2.4 Heuristics

Informed searches rely on knowing information about the problem domain, and are able to do so through the use of heuristics. *Heuristic functions* provide a means of estimation, further reducing the number of expanded nodes as opposed to uninformed searches. In traditional pathfinding solutions, estimations typically include calculating a minimum distance from neighbouring nodes to the goal, which is an objectively more informed method than greedily expanding the closest neighbour relative to the current position. As such, the subgraph of expanded nodes generated tends to grow towards the direction of the goal.

Traditionally, two forms of heuristic functions are used to calculate cost estimations:

1. *Manhattan Distance*
2. *Euclidean Distance*

The Manhattan Distance is calculated based on the distance between two points along the axes of grid lines. The term is inspired by the grid-like structure of road networks in Manhattan, New York. The sum of the grid lines forming the distances between the two points becomes a representation of the distance, and can be determined using the following formula:

$$h(n) = \left(|x_i - x_j|\right) + \left(|y_i - y_j|\right)$$

Where p_1 and p_2 are points on a grid, and $p_1 = (x_i, y_i)$ and $p_2 = (x_j y_j)$, respectively.

The Euclidean Distance is somewhat different from the Manhattan Distance (Figure 3), as instead of the distance being calculated along the lines of the grid, it is calculated as the length of a straight line formed between two points. The formula can be described as follows using the Pythagorean Theorem:

$$h(n) = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$$

Where, again, p_1 and p_2 are points on a grid, and $p_1 = (x_i, y_i)$ and $p_2 = (x_j, y_j)$ respectively.

It should be noted that the approach proposed in this thesis utilizes Euclidean Distance for any heuristic calculations.

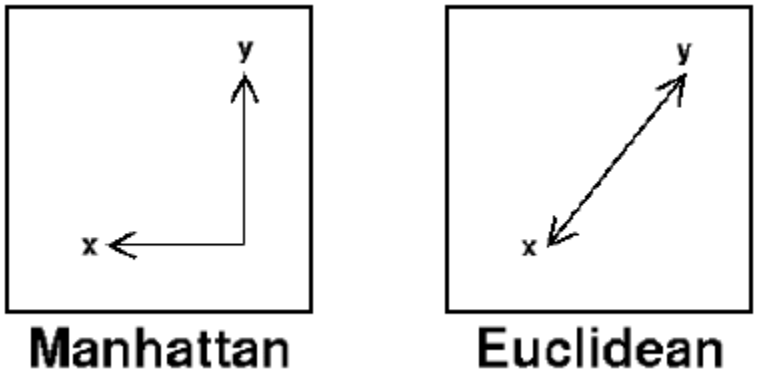


Figure 3: Depiction of Manhattan Distance versus Euclidean Distance. [17]

1.3 Optimal Path

In traditional pathfinding problems, the optimal path between two nodes is always the shortest, however, depending on the context of the problem the optimal path has different meanings. For one such problem as described by Aljubayrin et al., the optimal path may be the *shortest and*

safest path, provided that danger and survival are accounted for in the scenario. [7] We must note that the optimal solution will not always be the same across similar pathfinding problems, depending on the context and how we choose to define what our optimal path should be.

1.4 Thesis Contribution

This research has several contributions to the study of pathfinding. First and foremost, have extrapolated the concept of traditional pathfinding problems to applications where the loss of multiple resources is to be minimized instead of just one. In essence, we made some modifications to traditional pathfinding solution approaches in order to solve a specific type of scenario that would otherwise go unsolved. Additionally, our solution has been developed to include risks (more specifically, weights that have a probability tied to their values) that may arise in environments, inspired by the works of other authors and their strategic solutions. These modifications extend the usability of traditional pathfinding solutions to handle problems they normally could not handle before. Flexibility in resource prioritization is also included in our solution, and also ensures compatibility with existing systems due to the nature of the solution being an extension of traditional pathfinding solutions. Whatever the case, our research aims to be applicable to all types of environments. Furthermore, with a relationship to previous algorithms such as A* Search established, we also attempt to identify an accurate heuristic, much like that which is used in traditional A* Search. An accurate heuristic that can be applied to Multi-Objective Pathfinding could also potentially lead to additional related solutions in the future. Finally, this work includes not only an algorithmic solution to the problem but also a testing bed in the form of a pathfinding framework for comparing and experimenting with various solutions.

1.5 Thesis Outline

This thesis is divided into seven chapters, the bulk of which will be in Chapter 3. Chapter 1 covers the introduction to pathfinding and familiarizes the reader with concepts typically seen in research related to pathfinding. In Chapter 2, we will be analyzing various other methods used in pathfinding, including their uses, advantages, drawbacks, and applications. Chapter 3 will cover our proposed approach to Multi-Objective Pathfinding, and will also discuss several scenarios to the theory behind its intended applications. A total of twelve scenarios will be covered, each one building upon the previous before a general solution to cover all cases will be presented. In Chapter 4, we describe our experimental setup, limitations, assumptions, and system structure related to any experiments. Chapter 5 will show the results of our experiments and comparisons between our approach and existing solutions, including Dijkstra's Algorithm and A* Search. Chapter 6 marks our concluding remarks on the subject material, and Chapter 7 provides an insight or related work that we wish to include in the future.

Chapter 2: Background

In this section, we review various types of pathfinding algorithms and their uses in the field. While the work presented in this document is related, the basis of the solution borrows from approaches that have previously proven to be successful. Traditionally, the goal of pathfinding has been to find the least weighted path through a series of interconnected vertices, and many of the algorithms that will be mentioned here are based on this principle. For the purposes of this section and all other areas mentioned within this document, the notion of a *traditional pathfinding algorithm* is used to describe solutions that can be applied to the classical problem of searching a graph for the shortest possible path between two vertices.

2.1 Dijkstra's Algorithm

The idea behind a *best-first search* algorithm is as follows: Given a set of vertices on a graph and the traversal cost of each node relative to our starting position, we choose to explore the nodes whose cost is the minimum and expand its children from there. [14] Comparisons can be made between the current path and potentially shorter paths in order to find a true optimal path. By only choosing to follow the edges we estimate to be the least expensive, we can eventually obtain the shortest path. By doing so, we derive a greedy solution that also guarantees an optimal path (assuming there is at least one possible path toward the goal in the first place). Dijkstra's Algorithm is such an example of best-first search algorithms, which is based on the concept of navigating directed graphs while minimizing the "distance travelled" at the same time. The problem is determining a path in which the number of resources spent can be minimized, as compared to problems such as the *Travelling Salesman Problem* referenced in Dijkstra's original

work. If the distance traversed can be minimized, then naturally the amount of time and resources spent reaching a given destination will also be minimal. Dijkstra uses the idea that each node in a graph can be broken down into different sets in order to define their states as the following:

1. Visited
2. Next to be visited
3. Not yet visited

The starting node becomes the initial visited node in the set and from there the algorithm branches out to neighboring nodes, comparing multiple paths from the current node and choosing the shortest path between them. From the paths extending the newly visited paths, the next smallest path to the neighbouring nodes can be determined, accumulating from the weight of the paths previously traversed. Following this idea of only choosing the shortest paths based on the current node being visited, the minimal path branching from the origin node to every other node (and therefore a chosen node) can be determined.

Algorithm 1: Dijkstra's Algorithm Pseudo-code

```
Input: Node start, Node goal
1: Main()
2:   openList := Ø;              //priority queue based on g
3:   closedList := Ø;
4:   g(start) := 0;
5:   openList.Insert(start);
6:   while openList != Ø do
7:     current: = openList.Pop();
8:     if current == goal then
9:       return 'path found';
10:    closedList.Insert(current);
11:
12:    for each neighbour Є current.Neighbours do
13:      if neighbour in closedList then
14:        continue;
15:      if not neighbour in open then
16:        openList.Insert(neighbour);
17:      if g(current)+ c(current,neighbour) < g(neighbour) then
18:        g(neighbour) := g(current) + c(current, neighbour);
19:        parent(neighbour) := current;
20:    return 'path not found';
21: end;
```

2.2 A* Search

Using Dijkstra's Algorithm as a base, the A* Search algorithm attempts to obtain a more informed search as opposed to greedily choosing the nearest neighbour without any knowledge of where the goal is located. In order for this algorithm to function, all nodes in the graph are required to track their own secondary cost that represents the estimated distance relative to the goal position. Each node is aware of both its cost of traversal from the starting node and its estimated distance from the goal. Using these key values (denoted by a $g(x)$ and $h(x)$ function respectively), a more informed search than that of Dijkstra's Algorithm can take place and will allow the agent to expand nodes that are much closer to its preferred path.

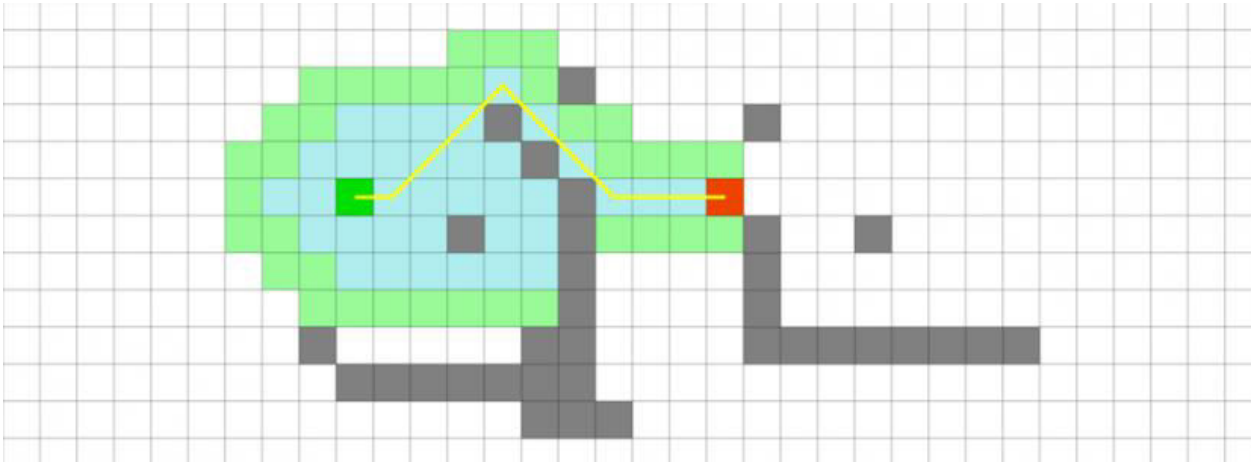


Figure 4: Evaluated path using A* Search [18].

At a high level, A* Search can be used to find the shortest path within a directed graph from a starting point *s* and an ending point *g*. The algorithm uses the knowledge based on the surrounding nodes from the current point and calculates the shortest possible route to the next node, while also taking into consideration the actual distance from the current node to the destination node [15]. The function is described as:

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

where *g(n)* calculates the distance from the current node to the next node, and *h(n)* calculates the estimated distance from the current node to the destination node. The minimum distance returned from this function over a series of nodes is the minimized route and thus the shortest path. It follows a breadth-first search ideology, analyzing the open list, which is a list of available points to traverse from the current node. The *g(n)* value is calculated as the cost to go from one node to the next, whereas the *h(n)* value represents the heuristic or the actual distance from a given node to the goal (ignoring obstacles and collisions). Any node that is put into the closed list becomes

the next node to recalculate *g(n)* for and the cycle continues until the goal node is reached, at which point a path can be drawn from the goal back to the starting node [15].

Algorithm 2: A* Search Pseudo-code

```
Input: Node start, Node goal
1: Main()
2:  openList := Ø;            //priority queue based on f
3:  closedList := Ø;
4:  g(start) := 0;
5:  f(start) := g(start) + h(start);
6:  openList.Insert(start);
7:  while openList != Ø do
8:    current: = openList.Pop();
9:    if current == goal then
10:     return 'path found';
11:   closedList.Insert(current);
12:
13:   for each neighbour Є current.Neighbours do
14:     if neighbour in closedList then
15:       continue;
16:     if not neighbour in open then
17:       openList.Insert(neighbour);
18:     if g(current)+ c(current,neighbour) < g(neighbour) then
19:       g(neighbour) := g(current) + c(current, neighbour);
20:       f(neighbour) := g(neighbour) + h(neighbour);
21:       parent(neighbour) := current;
22: return 'path not found';
23:end;
```

Two lists are maintained during processing, appropriately labelled the *open* and *closed* lists in typical implementations. Upon initialization, the starting node is added to the open list and is subsequently "expanded" in order for successor nodes to be realized. These successors are then added to the open list, while the predecessor enters the closed list (or alternatively is labelled as *visited*). From here, the *f-scores* of successors are compared to the predecessor, and the minimum value is selected for expansion on the next iteration. This process continues until all elements in

the open list have been exhausted, or the designated goal node has been expanded. The final procedure involves tracing the path all the way back from the goal node to the start node, using each node's predecessor to denote the path.

Similarly to Dijkstra's Algorithm, A* Search need only rely on two properties that exist within the nodes, which can be used to represent a geographical map realistically. In a realistic setting, however, this information may not lead to a preferred path if danger should lie within the optimal path itself. In this way, it suffers from the same drawbacks as Dijkstra's Algorithm in that this solution will find the absolute shortest path regardless of how dangerous or unpreferred a path may be. It will surely find a solution within a reasonable time, but it's applications to a problem beyond the scope of the shortest pathfinding problem are somewhat limited. That being said, A* Search is one of the most commonly used pathfinding algorithms in the industry and has been easily replicated in pathfinding solutions and video game entertainment software.

2.3 Lifelong Planning A*

The majority of the solutions mentioned in this thesis are of the heuristic or uniformed types, meaning that they either rely on a heuristic for estimated closest paths, or they cascade through the nearest neighbours as in Depth-First Searches or Breadth-First Searches. The works of Koenig et al. propose Lifelong Planning A*, which is an incremental solution inspired by the A* Search approach, which utilizes a heuristic. The incremental nature of the algorithm lends itself to be similar to A* Search on its first iteration. However, subsequent iterations are capable of finding alternate paths in an environment where small changes occur near the optimal route. [8] This is because the solution "reuses information from previous searches," essentially reducing processing time by determining which costs have not changed in previous iterations, thereby

ignoring any recomputation required to those specific cells. A secondary value in addition to the primary cost value of each node is maintained, known as the *rhs-value* which is derived from a DynamicSWSF-FP algorithm. These rhs-values are one-step lookahead values that track the minimum cost over all neighbours of the current node traversing over to the next neighbour, and are thus "potentially better informed than the g-values." By ensuring that the nodes are locally consistent (in other words, the g-value of each node equals its respective rhs-value), there is no need to check node costs relative to the starting node if a change in the environment is made. If a node's rhs-value is not equal to its respective g-value, then an inconsistency in the graph has occurred in one of the current node's neighbours. These nodes that are locally inconsistent therefore become the first nodes to be replanned, which ultimately remain a fraction of the total population of nodes provided only a few edges or vertices were altered in the search map. In the average case, it is concluded that Lifelong Planning A* can be more efficient than A* Search and similar A* Search refinement solutions when "the path-planning problems change only slightly, and the changes are close to the goal."

2.4 Extended A* with Survivability

Kim et al. introduce an enhanced version of A* Search that attempts to solve the problem of finding a *shortest*, *safest* path in a known environment. As the name implies, the approach is based on the traditional A* Search algorithm, in that it utilizes costs on the map to evaluate the shortest path from a given starting point to a specified destination. [13] They provide a real-life example of an automated combat vehicle that can navigate terrain in a dangerous environment, typically a war-torn battlefield or similar area where some zone could be treacherous. Since the A* Search algorithm always prioritizes finding the path of least cost, an additional cost function

is appended to the existing function. They devise a *Probability Hit* function used to determine the likeliness of being targeted by an enemy or the chances of encountering a dangerous threat at any given node. The *Probability Hit* function represents an added weight onto the cost function of A* Search, where a survivability rate is included for when nodes are being expanded. Hence, this algorithm is considered an "extended" version of A* Search because it determines not only the shortest route but also the safest route based on the likeliness of survivability. [13] Additionally, an agent could very well traverse through a dangerous node if that same node just so happens to be on the shortest path to the destination. At a high level, the algorithm can effectively minimize not only the distance travelled but also the risk of being threatened. It should be noted, however, that, like its predecessor, the Extended A* Search with Survivability approach does not handle dynamic environments, and is much better suited to situations where the risk in an area can be predictable or simply evaluated. It also does not cover situations in which the agent may wish to prioritize speed over safety, thereby taking a risk in itself in order to reach its goal within a quicker time frame.

Consider the following example, where our agent represents a soldier and the map comprised of a network of nodes represents a battlefield.

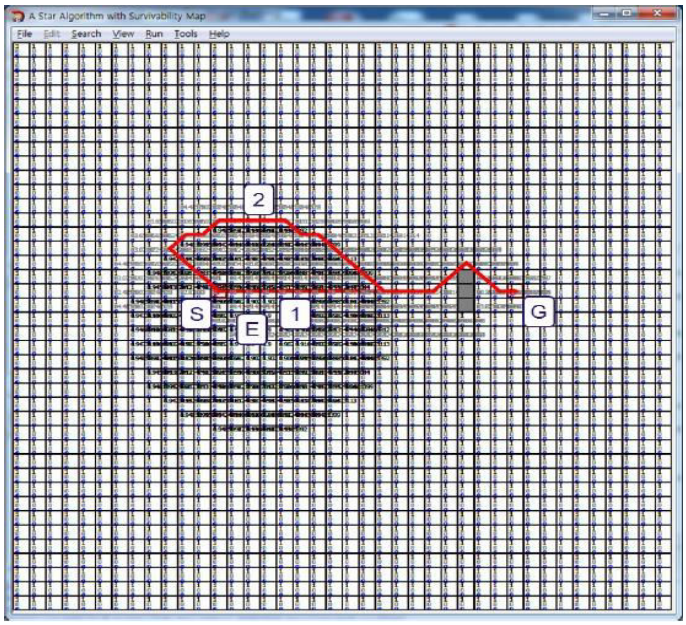


Figure 5: Study of New Path Planning Algorithm Using Extended A* Algorithm with Survivability.

In the demonstration, a soldier may wish to reach their destination in the middle of the battlefield, but using a regular A* Search would force them to traverse unsafe terrain, which is represented by the gray circular area on the map. A* would yield *path 1* as labelled in the graph. Following such a path requires heavy traversal through risky terrain, and thus becomes hazardous to the agent using the algorithm. Imagining that the circular area could represent a minefield, it would make sense for the agent to reach their destination in the shortest amount of time possible while also minimizing the risk of being in such an area at any given point in time. Thus, the Extended A* with Survivability algorithm reroutes the path, giving us the example as shown in *path 2*. This path intentionally avoids hazardous areas, navigating on the very outskirts of the danger zone to find the closest node to the goal. With no other option but to advance further, the algorithm draws a straight line to the goal from the outskirts of the hazardous area, thereby evaluating what we determine to be the safest, shortest, possible path. Thus, in a

situation where risk minimization is to be prioritized over the time it takes to reach the destination itself, a traditional A* Search approach would not be sufficient.

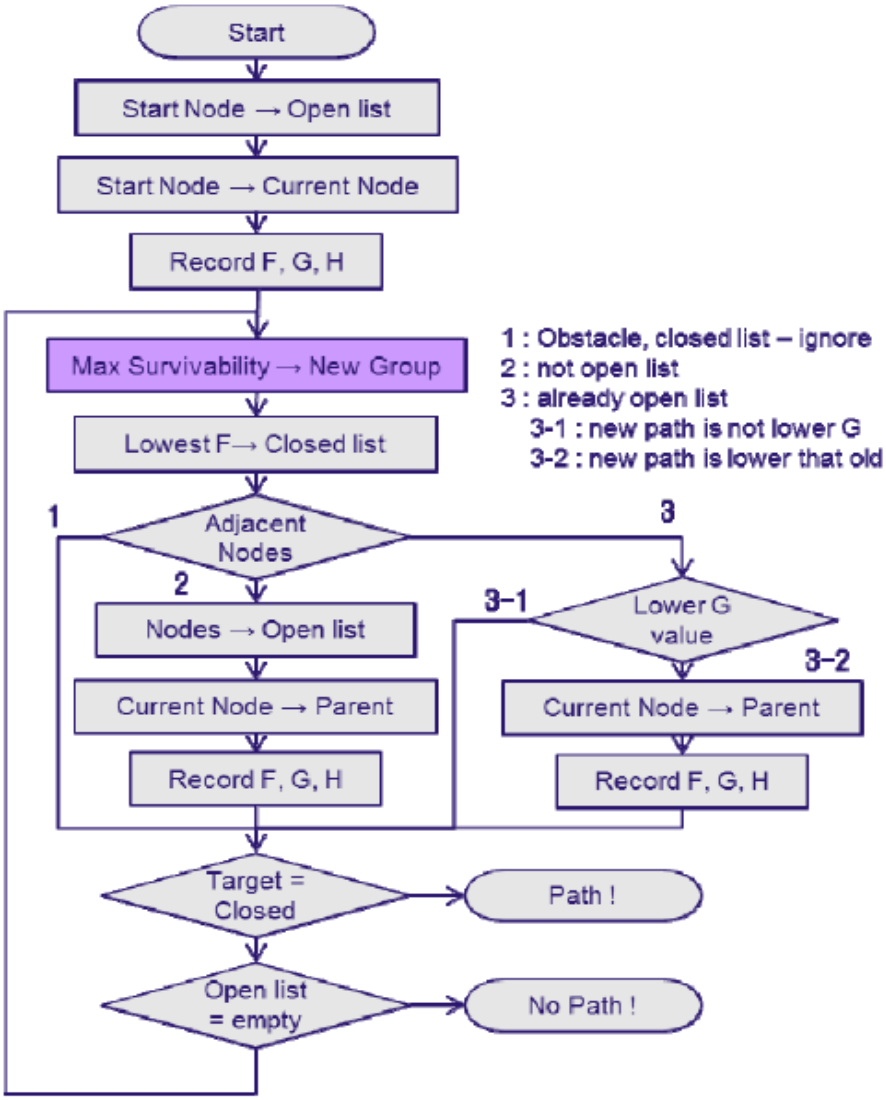


Figure 6: Workflow of Extended A* with Survivability solution. Determining max survivability takes place before the A* Search algorithm is executed.

The drawback to this algorithm is that it will always prioritize safety over minimizing the distance travelled, and other solutions that rely on maintaining safety over distance also suffer from this. [9] [11] [12] Algorithms that prioritize safety in such a manner will expand nodes all

along the danger zone until it finds that singular node is leading to a path of safety. This becomes more of a problem when the destination is not that far relative to an agent's position but is perceived to be far if the hazardous area encompasses a large portion of the map. Hazardous areas are essentially treated as obstacles that the agent must either outright avoid or travel close to in order to reach their designated goal. In a realistic scenario, this type of solution may not be feasible because, by the time the agent does make it to their goal, they could have tremendously saved time by simply cutting across the hazardous area and taking a risk as if it were a traditional pathfinding problem.

2.5 Strategic Pathfinding

Singh and Holder's work presents a solution to the problem of pathfinding related to a measurable risk versus navigation time. Their goal was to have the agent navigate an environment without predetermined waypoints and strategically make decisions based on the estimated likelihood of encountering an enemy, which they depicted in a combat simulator. Calculations pertaining to the probability of being struck by an attack at each point in the map also factors into the pathfinding strategy. This provides more flexibility in the pathfinding decision-making that previously mentioned solutions do not cover. Unlike these methods that prioritize one resource over another, the Strategic Pathfinding method accounts for risk, in addition to the time it would take for an agent to reach its goal. A strategic path is defined to be, "a trade-off between the time of traversal and the risk along the path." [7]

Two levels of abstraction are required to utilize a strategic pathfinding method. The *Area level* defines a set of areas that an agent may walk over, and is abstracted from the search map in order to estimate a rough layout of obstacles and non-obstacles. The *Grid level* is one level lower and

represents the set of actual traversable points that can be navigated through across each area. Having multiple levels plays into the movement options of the agent, such as knowing when to walk, jump, or sprint, which can affect the time of traversal in certain areas. The risks themselves are determined from the *Meta-Weight* function, a cost that is a modification of the standard edge costs used by traditional pathfinding algorithms. This *Meta-Weight*, while coinciding with the presence of potential risks in an area, alter the paths of agents that otherwise would go unaffected. The risk itself is calculated based on a function specific to their testbed, essentially representing the *hit probability* of being struck by an enemy at a specific node or area. More detail will be given on this in later sections, as the hit probability essentially represents the risk, which is the factor that will be affecting our approach to strategic pathfinding. Technically speaking, the risk can be attributed to the probability of losing a valuable resource, whether that resource is the vitality of the agent or some other property that holds just as much value depending on the scenario. The hit probability (or probability of resource loss) can be represented as a percentage of danger distributed over each area, and may be calculated as given:

$$HP_{total} = HP_1 + HP_2(1 - HP_1) + \cdots + HP_n(1 - HP_{n-1})(1 - HP_{n-2}) \ldots (1 - HP_1)$$

where HP_{total} represents the probability of danger along a given path.

In short, risk along the length of a path can be determined by calculating the total risk with each node in sequence, multiplied by the chances of not being in danger within each node previously passed. The result becomes a total amount of risk that can be associated with a path, or any specific node along that path. Nevertheless, by having an accurate representation of how much risk is in an area, the path processing can choose to allow the agent to avoid such hazardous paths or take the risk of going through them altogether depending on the preference of the agent

itself. Naturally, this preference is determinable by the type of implementation at hand, and can make the difference between an agent that will run through a zone of enemy fire, or taking the safer way out by fleeing instead. Choosing to follow a safer route will result in having to expand more nodes compared to shorter routes if the hazardous nodes just so happen to be blocking the otherwise shortest path.

Algorithm 3: Strategic Pathfinding Pseudo-code

```
Input: Node start, Node goal, Float p
1: Main()
2:  openList := Ø;            //priority queue based on g
3:  closedList := Ø;
4:  g(start) := 0;
5:  openList.Insert(start);
6:  while openList != Ø do
7:    current: = openList.Pop();
8:    if current == goal then
9:      return 'path found';
10:   closedList.Insert(current);
11:
12:   for each neighbour Є current.Neighbours do
13:     if neighbour in closedList then
14:       continue;
15:     if not neighbour in open then
16:       openList.Insert(neighbour);
17:       risk := CalcRisk(current);
18:     if g(current)+ c(current,neighbour) * (risk * p + 1) <
g(neighbour) then
19:       g(neighbour) := g(current) + c(current, neighbour);
20:       parent(neighbour) := current;
21:   return 'path not found';
22:end;
23:CalcRisk(c)
24:  current := c;
25:  risk := risk(current);
26:  while parent(current) != null do
27:    risk := risk * (1-risk(parent(current)));
28:
29:  if parent(current) == null
30:    return risk(current);
31:  else
```

```
32:    return risk + CalcRisk(parent(current));
33:end;
```

For evaluating risk over a path, Singh and Holder utilize the *Meta-Weight* formula, which is a function that represents edge costs as a factor of weight, risk, and the agent's preference. [7] They describe this function as follows:

$$MetaWeight = (HP * RiskVsTime + 1) * Euclidean\ Distance$$

Where *HP* designates the hit probability, and *RiskVsTime* designates the preference factor of the agent. In this case, the hit probability can be equated to the probability of forfeiting a resource (or risk). As for *RiskVsTime*, this variable can be any value greater than 0, as the higher the value, the more intensely the agent will pursue a safer path. By referring to a *safer path*, we are referring to the path that yields the least amount of risk. In either case, one resource is planned to be minimized along the subgraph, which is done through the use of Dijkstra's Algorithm after some cost manipulation with the preference factor and risk. The urgency of objectives may be calculated separately from the actual pathfinding, and from this, it can be determined how important certain objectives may be to the agent relative to the specific resource (typically risk versus distance) to be minimized along a perceived path.

This ability to minimize costs while also providing some form of strategy has been utilized in studies before. [10] Drawbacks of this solution typically imply a specific type of map setup for it to be usable, thus making it impractical for broad applications. Furthermore, the Strategic Pathfinding method comes with the disadvantage of weights being required to be within a certain threshold for the preference factor to have a large enough bearing on the pathfinding process. There is no scalability between these properties, as one factor can overpower another, inevitably

skewing the results. In the following section, we will look at ways to mitigate this disadvantage, as well as a means to generalize the cost function so that strategic methods can be more accessible to implementations via multi-objective optimization.

Chapter 3: Proposed Approach

3.1 Motivation

Path planning research has typically been derived from the notion of attempting to find the shortest existing path in a limited search space. In all cases, the path of least cost will always yield the most efficient traversal time, but realistically this path may not be the most feasible depending on the resources that are at stake. For example, an auto-piloted supply truck would not want to follow the shortest path to its destination if such a path required it to travel through a minefield, else risking the possibility of losing supplies or the vehicle itself. Similarly, commuters who frequently crossroads on their way to work may find themselves with the dilemma of crossing the road prematurely or heading to the crosswalk where travel would be safer for pedestrians. Logically, the crosswalk is the safest point at which to start crossing from, but if a commuter should be behind schedule, then they are more likely to act impulsively in order to reach their destination on time. Thus, taking the risk would be more beneficial by saving travel time. At what point, however, would it be optimal for a person to cross the road prematurely as opposed to waiting patiently for the hazards to clear? The solution can be described as a tradeoff between two resources (in this case, the distance to be covered and the person's own vitality), both of which the agent may wish to minimize, but can only prioritize one over the other, or potentially a little bit of both. This question became the basis for Multi-Objective Pathfinding.

3.2 Notation

In graph theory, a graph defined G is a set $\{v_i\}$ of vertices (nodes) and a set $\{e_{ij}\}$ of interconnected edges (arcs) (see Nilsson et al.). Any edge e_{pq} that exists on the graph implies a connection between vertices v_p and v_q, and v_q is a successor of v_p. [4] Let $V = \{v_i\}$ as a set of vertices, and $E = \{e_j\}$ as a set of edges. Each edge has with it an associated weight (or cost) that is defined as $W = \{w_e\}$, where e denotes the current edge. We can broadly define a *directed weighted graph* $G = (V, E)$ where:

$$V = \{v_1, v_2, v_3, \ldots, v_{|V|}\} \text{ is the set of vertices,}$$

$$E = \{e_1, e_2, e_3, \ldots, e_{|E|}\} \text{ is the set of edges,}$$

$$W(E) = \{w_1, w_2, w_3, \ldots, w_{|E|}\} \text{ is the set of edge-weights.}$$

Each edge is an ordered pair (v_i, v_j) which denotes the existence of a directed edge from vertex v_i to vertex v_j. The number of vertices in the graph is $|V|$, the number of edges in the graph is $|E|$, and the number of edge-weights $|W(E)| = |E|$as each edge has an associated weight.

If there is an edge e_{ij} from vertex v_i to vertex v_j, v_j is said to be *adjacent* to v_i. Adjacencies in a graph are typically represented with an $O(|V|^2)$ adjacency matrix or an $O(|V| + |E|)$ adjacency list. A *path* is a sequence of connected vertices $v_i, v_{i+1}, \ldots, v_n$ (or, equivalently, a sequence of edges $e_i, e_{i+1}, e_{i+2}, \ldots, e_{n-1}, e_n$). The weight of a path is the sum of the weights of the edges of the path.

A comparison can be made to traditional graphs in pathfinding problems, where an edge only has one resource type associated with it (such as distance). In *pathfinding*, the vertices of the graph typically represent locations, and the edges represent traversable connections between

locations. Weights typically represent distances, and the objective is to find the shortest (least weighted) pathway through the graph from a starting vertex v_s to a goal vertex v_g.

In this chapter, we will describe several instances of the multi-objective pathfinding problem through the use of scenarios. Each scenario depicts an example of a very simple search map (for ease of explanation) and how its properties require different solutions for processing the data. Specific keywords are used to define each scenario individually and outline the various differences between them. There are six keywords in total, which are defined as follows:

- Single (Resource): Each edge in the graph contains one value that makes up the total weight. This is the attribute that traditional pathfinding solutions work with.
- Multiple (Resource): Each edge in the graph contains at least two values that make up the total weight.
- Deterministic: Costs do not utilize random variables in their calculations.
- Stochastic: Costs utilize random variables in their calculations, and so expected values of events must be evaluated. Stochastic scenarios come with two subclasses:
    - Independent: The result of future events does not rely on previous events.
    - Dependent: The result of future events relies on previous events.
- Static: Properties of the graph do not change over the course of the pathfinding process, such as in traditional pathfinding problems.
- Dynamic: Properties of the graph will change over the course of the pathfinding process. Scenarios including this factor will have time-dependent evaluations.

Scenario	Single/Multiple	Deterministic/Stochastic-Independent/Stochastic-Dependent	Static/Dynamic
1	Single	Deterministic	Static
2	Multiple	Deterministic	Static
3	Single	Stochastic-Independent	Static
4	Multiple	Stochastic-Independent	Static
5	Single	Stochastic-Dependent	Static
6	Multiple	Stochastic-Dependent	Static
7	Single	Deterministic	Dynamic
8	Multiple	Deterministic	Dynamic
9	Single	Stochastic-Independent	Dynamic
10	Multiple	Stochastic-Independent	Dynamic
11	Single	Stochastic-Dependent	Dynamic
12	Multiple	Stochastic-Dependent	Dynamic

3.2.1 Scenario 1: Static Single Resource

Applications for Scenario 1 include those commonly seen in traditional pathfinding problems and are exemplified by Dijkstra's Algorithm and A* Search. A single resource (typically representing the remaining distance an agent has to travel between each node) is situated on each edge and is a cost to be minimized. Characters in a game environment that are only concerned with minimizing the distance of travel can greatly benefit from this solution, hence why this scenario is the most commonly encountered.

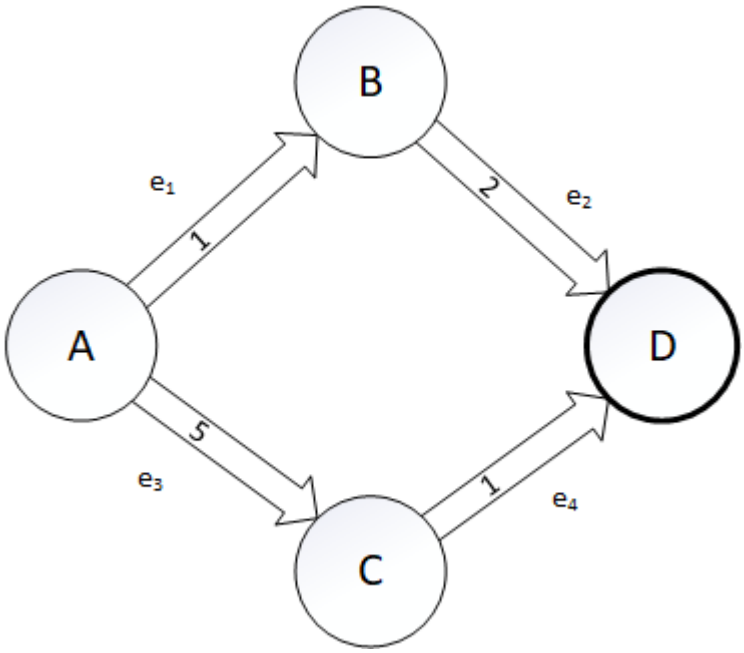


Figure 7: Problem Graph featuring a static single resource per edge.

We define our sets like so:

$V = \{A, B, C, D\}$

$E = \{e_1, e_2, e_3, e_4\}$

$W = \{w_{e_1}, w_{e_2}, w_{e_3}, w_{e_4}\} = \{1, 2, 5, 1\}$

The expanded search tree used by A* is illustrated as such:

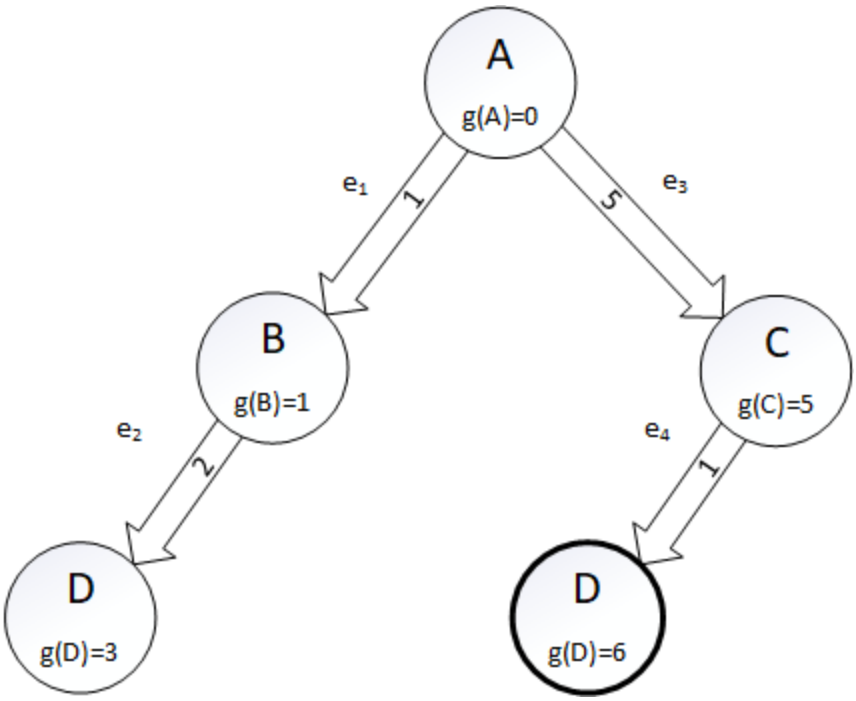


Figure 8: Expanded search tree featuring a static single resource using the aforementioned Problem Graph (Figure 7). Two paths are realized with one path yielding less cost than the other.

In *multi-objective pathfinding*, the weights represent a combination of values of multiple resources. For example, an edge weight might be used to represent a combination of the distance travelled and fuel consumption. The objective is to find a path of least cost where weight is defined as a function of the resources.

The resources in multi-objective pathfinding can be expressed as a $r \times |E|$ matrix R where there is a row for each resource and a column for each edge. $R[i, j]$ represents the amount of resource i expended when traversing edge j. To determine the weight of an edge in a multi-objective graph, a $1 \times r$ conversion vector $C = \{c_1, c_2, \ldots, c_r\}$ is defined, where each conversion factor represents the value of the associated resource in terms of some standard units. For example, if we could travel a longer path, yet save fuel, the conversion factor between distance and fuel would be based on how much extra distance we would be willing to travel to save a given amount of fuel. This value functions similarly to the preference factor used in Holder's Strategic Pathfinding

algorithm in that it measures the intensity of a particular resource relative to another as opposed to the intensity of wanting to take a risk. Moreover, this value converts the units of the resource, normalizing the formula so that like terms can be collected.

Under this scheme, $W(E) = CR$. Using the conversion vector to convert the resource matrix to the edge-weight vector provides the ability to solve multi-objective pathfinding problems using the same A* framework utilized in shortest path problems.

3.2.2 Scenario 2: Static Multiple Resources

While A* Search remains to be one of the most commonly used solutions in the industry, it is not without its drawbacks. The ability to deduce the path of least cost may not be best suited in some scenarios. If, for example, an agent wishes to travel the shortest amount of distance to reach its destination, the environment could contain hazards, making some nodes dangerous to cross. In the most extreme cases, nodes that are hazardous can be treated as obstacles to be avoided. There is, however, an alternative in the form of appending multiple costs to the same edge and giving them scaling properties relative to one another.

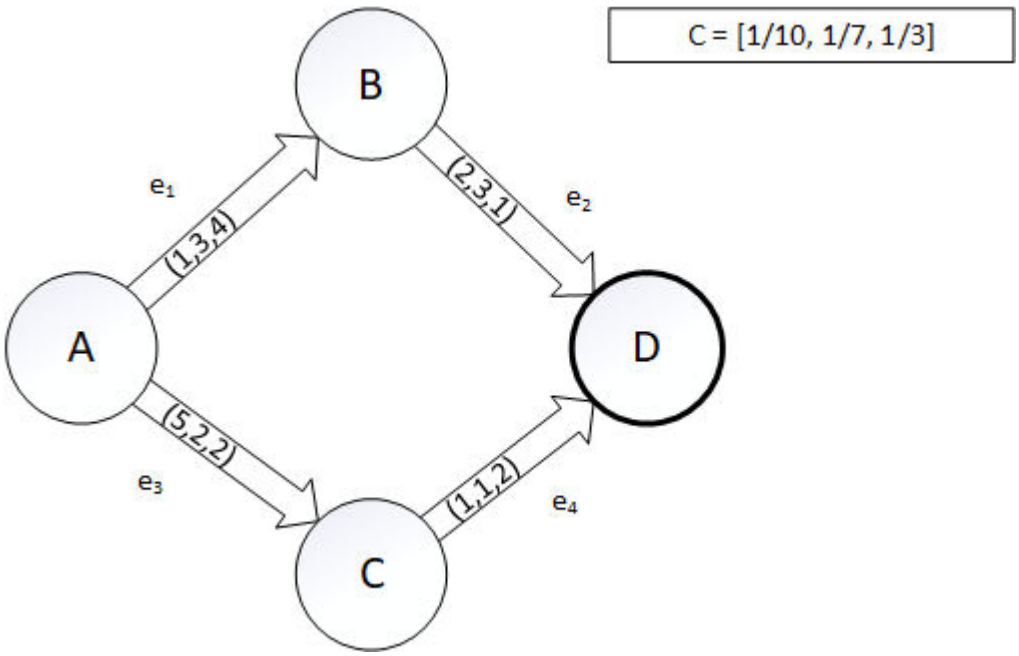


Figure 9: Problem Graph featuring multiple resources. Notice that each edge contains multiple values to denote weight.

Given the above example, we define our sets like so:

$V = \{A, B, C, D\}$

$E = \{e_1, e_2, e_3, e_4\}$

$W = \{w(e_1), w(e_2), w(e_3), w(e_4)\}$

$$R = \begin{bmatrix} 1 & 2 & 5 & 1 \\ 3 & 3 & 2 & 1 \\ 4 & 1 & 2 & 2 \end{bmatrix}$$

$$C = \left[\frac{1}{10}, \frac{1}{7}, \frac{1}{3}\right]$$

Now, by using the formula $W(E) = CR$:

$$W(E) = \left\{\begin{matrix} \left[\frac{1}{10} * 1 + \frac{1}{7} * 3 + \frac{1}{3} * 4\right], \left[\frac{1}{10} * 2 + \frac{1}{7} * 3 + \frac{1}{3} * 1\right], \\ \left[\frac{1}{10} * 5 + \frac{1}{7} * 2 + \frac{1}{3} * 2\right], \left[\frac{1}{10} * 1 + \frac{1}{7} * 1 + \frac{1}{3} * 2\right] \end{matrix}\right\}$$

Implying:

$$w(e_1) = \left[\frac{1}{10} + \frac{3}{7} + \frac{4}{3}\right], w(e_2) = \left[\frac{1}{5} + \frac{3}{7} + \frac{1}{3}\right], w(e_3) = \left[\frac{1}{2} + \frac{2}{7} + \frac{2}{3}\right], w(e_4) = \left[\frac{1}{10} + \frac{1}{7} + \frac{2}{3}\right]$$

$W \approx \{1.86, 0.96, 1.45, 0.91\}$

The expanded search tree used by A* is illustrated as such:

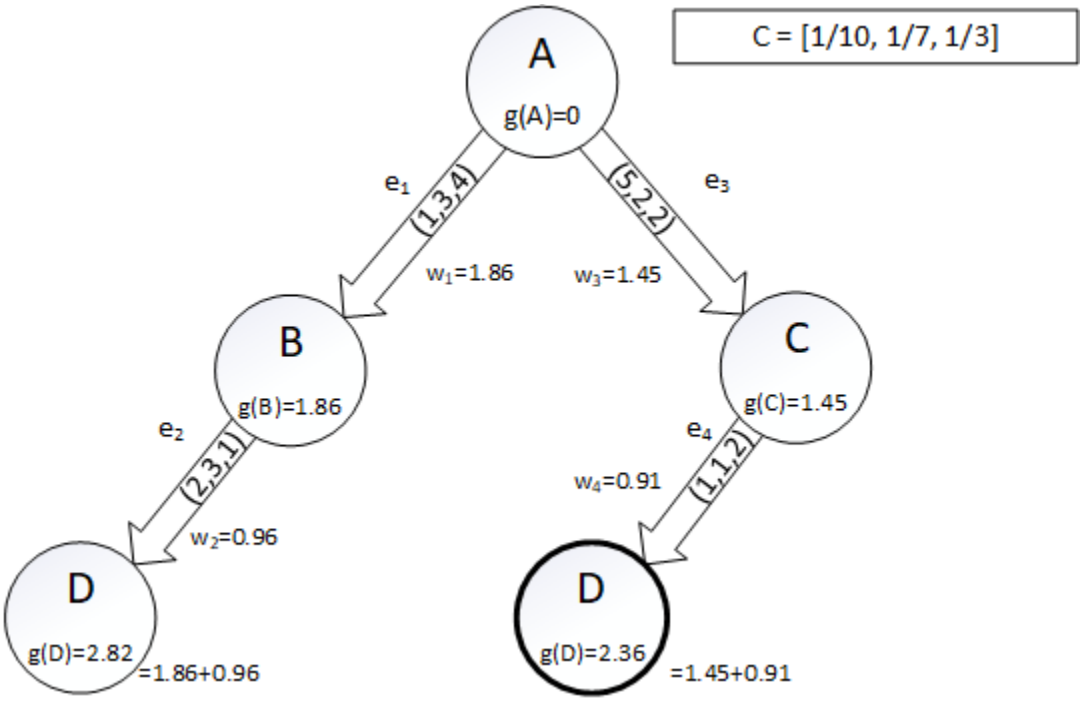


Figure 10: Expanded search tree featuring multiple resources using the aforementioned Problem Graph (Figure 9). Two paths are realized with one path yielding less cost than the other.

Applications of this solution are found in situations where an agent has to pay multiple costs in order to traverse an edge. Say, for example, a car has to pass over different types of terrain such as dirt, sand, and ice and each of these terrain types litter the search map in patches. Driving over sand causes more rotations of the tires than dirt, and even more for driving over ice. The distance to the destination will remain the same, but resources such as fuel are also added as costs for each terrain type. Thus, a decision must be made on whether the distance to minimize should take priority over the usage of fuel, and the conversion rate must be modified to accommodate this decision.

Edges may also contain risks, which act as further alterations to existing resource costs. With risks involved, resources no longer have a set flat cost, but a series of possible costs each with their own probabilities of occurrence. In other words, each resource becomes an event, with an expected value that can be calculated using a familiar formula.

We can derive an *expected value* of a random variable X, denoted $E'(X)$, using the following function inspired by the notation cited by R. Hamming:

$$E'(X) = \sum_i P_i * x_i$$

Where P_i is the probability of an event i, and x_i is a discrete random variable representing a single outcome of event i. [3] Let X be a discrete random variable with a set of possible outcomes Ω. Let x_i be one element of Ω and P_i be the probability of the outcome being x_i (i.e. $P(X = x_i) = P_i$). Using a six-sided die example, each side of the die is given a value from 1 to 6. The possible outcomes of X are the elements of set Ω:

$$x_i \in \Omega = \{1, 2, 3, 4, 5, 6\}$$

Applying the formula to this problem knowing that the odds of rolling any one particular side are ⅙, the outcome versus probability distribution can be described in the following table:

Outcome	**Probability**
1	$\frac{1}{6}$
2	$\frac{1}{6}$
3	$\frac{1}{6}$
4	$\frac{1}{6}$
5	$\frac{1}{6}$

6	$\frac{1}{6}$

And the expected value can be calculated like so:

$$E'(X) = \frac{1}{6}(1) + \frac{1}{6}(2) + \frac{1}{6}(3) + \frac{1}{6}(4) + \frac{1}{6}(5) + \frac{1}{6}(6) = 3.5$$

Given a resource x_{ij}, an event e can be associated to it containing a set of outcomes and probabilities. The expected value taken from this event becomes the updated value of the resource. We can extend the resource matrix of the above scenario to include risks.

3.2.3 Scenario 3: Static Single Resource with Expected Values over Independent Events

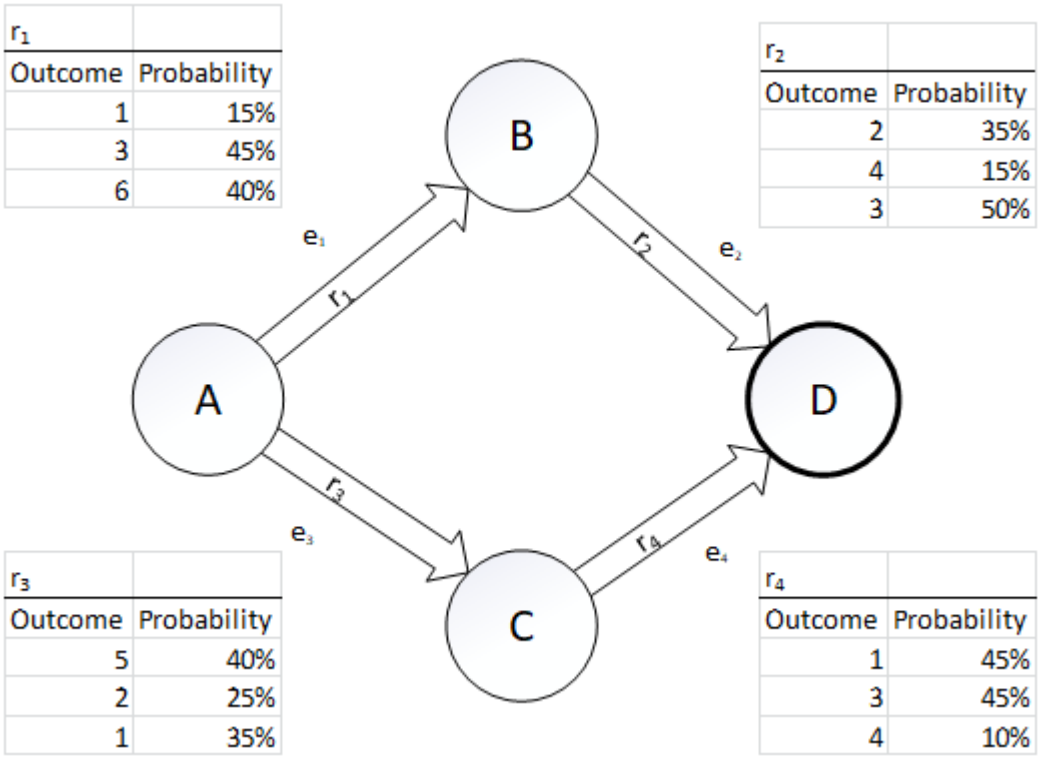


Figure 11: Problem Graph featuring a single resource per edge with expected values. Each outcome comes with a probability of occurrence, and thus an expected value can be determined for each resource.

Resources are now composed of an event, and each event contains a series of pairs (or tuples) p_i representing an outcome and an associated probability. The notation used for events can be described as:

$$Ev(r_x) = \{p_{r_x1}, p_{r_x2}, \dots, p_{r_xi}\}$$

Thus, the expected value of the resources using the R matrix becomes:

$$E'(R) = [p_{r_11} + p_{r_12} + p_{r_13} \quad p_{r_21} + p_{r_22} + p_{r_23} \quad p_{r_31} + p_{r_32} + p_{r_33} \quad p_{r_41} + p_{r_42} + p_{r_43}]$$

$$= [1 * .15 + 3 * .45 + 6 * .4 \quad 2 * .35 + 4 * .15 + 3 * .5 \quad 5 * .4 + 2 * .25 + 1 * .35 \quad 1 * .45 + 3 * .45 + 4 * .1]$$

$$E'(R) = [3.9 \quad 2.8 \quad 2.85 \quad 2.2]$$

Implying:

$$w(e_1) = [3.9], w(e_2) = [2.8], w(e_3) = [2.85], \ w(e_4) = [2.2]$$

$$W \approx \{3.9, 2.8, 2.85, 2.2\}$$

The expanded search tree used by A* is illustrated as such:

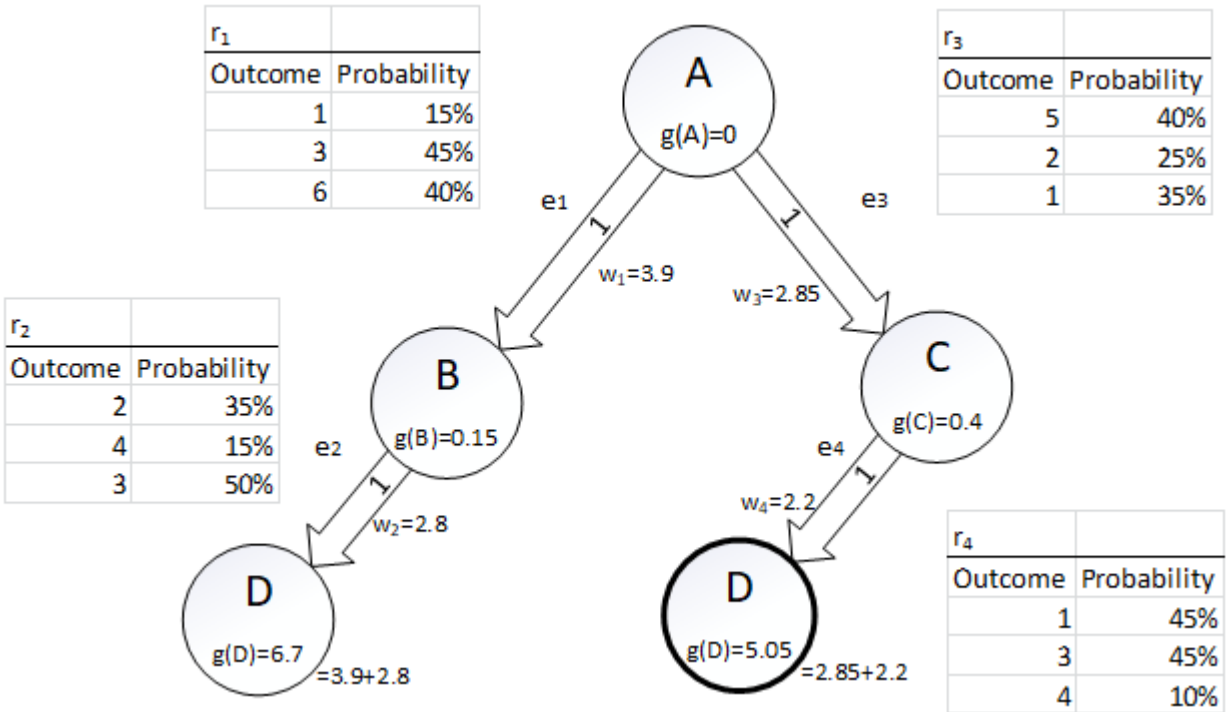
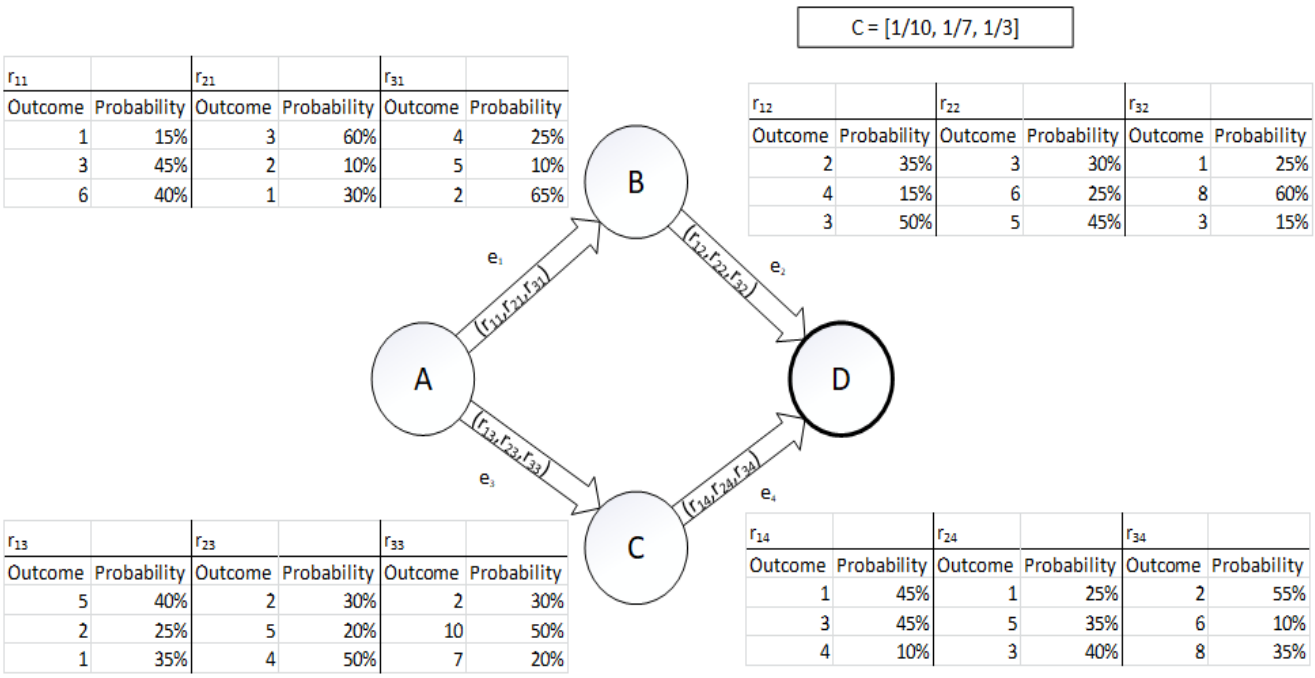


Figure 12: Expanded search tree featuring a single resource with expected values over independent events using the aforementioned Problem Graph (Figure 11).

Applications of this solution may be used in situations where there is some uncertainty in the costs, such as the examples used in Strategic Pathfinding [7]. In Strategic Pathfinding, agents attempt to reach their designated goal while also reducing their chances of being executed by hazards along the path. These hazards come in the form of risks, which overall have an effect on the edge-costs between nodes. It's ideal that agents will try to travel the path of least cost in terms of both distance and risk, thus making this solution applicable to that kind of problems.

3.2.4 Scenario 4: Static Multiple Resources with Expected Values over Independent Events

For scenarios that utilize expected values, it is assumed there is no dependence between the probability distribution of resources. For example, if the probability distribution changes in one resource, it does not affect the probability distribution of another resource on the same edge. This assumption is more applicable to scenarios with known temporal values. However, it should also be noted for the static cases. Developing a solution that supports joint probability distribution scenarios is not covered in this work, but is left for future work.



r_{11}		r_{21}		r_{31}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	15%	3	60%	4	25%
3	45%	2	10%	5	10%
6	40%	1	30%	2	65%

r_{12}		r_{22}		r_{32}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
2	35%	3	30%	1	25%
4	15%	6	25%	8	60%
3	50%	5	45%	3	15%

r_{13}		r_{23}		r_{33}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
5	40%	2	30%	2	30%
2	25%	5	20%	10	50%
1	35%	4	50%	7	20%

r_{14}		r_{24}		r_{34}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	45%	1	25%	2	55%
3	45%	5	35%	6	10%
4	10%	3	40%	8	35%

Figure 13: Problem Graph featuring multiple resources with expected values. Each outcome comes with a probability of occurrence, and thus an expected value can be determined for each resource.

The expected value of the resources using the R matrix becomes:

$$E'(R)$$

$$= \begin{bmatrix} 1*.15+3*.45+6*.4 & 2*.35+4*.15+3*.5 & 5*.4+2*.25+1*.35 & 1*.45+3*.45+4*.1 \\ 3*.6+2*.1+1*.3 & 3*.3+6*.25+5*.45 & 2*.3+5*.2+4*.5 & 1*.25+5*.35+3*.4 \\ 4*.25+5*.1+2*.65 & 1*.25+8*.6+3*.15 & 2*.3+10*.5+7*.2 & 2*.55+6*.1+8*.35 \end{bmatrix}$$

$$E'(R) = \begin{bmatrix} 3.9 & 2.8 & 2.85 & 2.2 \\ 2.3 & 4.65 & 3.6 & 3.2 \\ 2.8 & 5.5 & 7 & 4.5 \end{bmatrix}$$

$$C = \left[\frac{1}{10}, \frac{1}{7}, \frac{1}{3}\right]$$

Now, by using the formula $W(E) = CE^{I}(R)$:

$$W(E) = \left\{ \begin{matrix} \left[\frac{1}{10}*3.9+\frac{1}{7}*2.3+\frac{1}{3}*2.8\right], \left[\frac{1}{10}*2.8+\frac{1}{7}*4.65+\frac{1}{3}*5.5\right], \\ \left[\frac{1}{10}*2.85+\frac{1}{7}*3.6+\frac{1}{3}*7\right], \left[\frac{1}{10}2.2+\frac{1}{7}*3.2+\frac{1}{3}*4.5\right] \end{matrix} \right\}$$

Implying:

$$w(e_1) = \left[\frac{3.9}{10} + \frac{2.3}{7} + \frac{2.8}{3}\right], w(e_2) = \left[\frac{2.8}{10} + \frac{4.65}{7} + \frac{5.5}{3}\right], w(e_3) = \left[\frac{2.85}{10} + \frac{3.6}{7} + \frac{7}{3}\right],$$

$$w(e_4) = \left[\frac{2.2}{10} + \frac{3.2}{7} + \frac{4.5}{3}\right]$$

$$W \approx \{1.65, 2.78, 3.13, 2.18\}$$

The expanded search tree used by A* is illustrated as such:

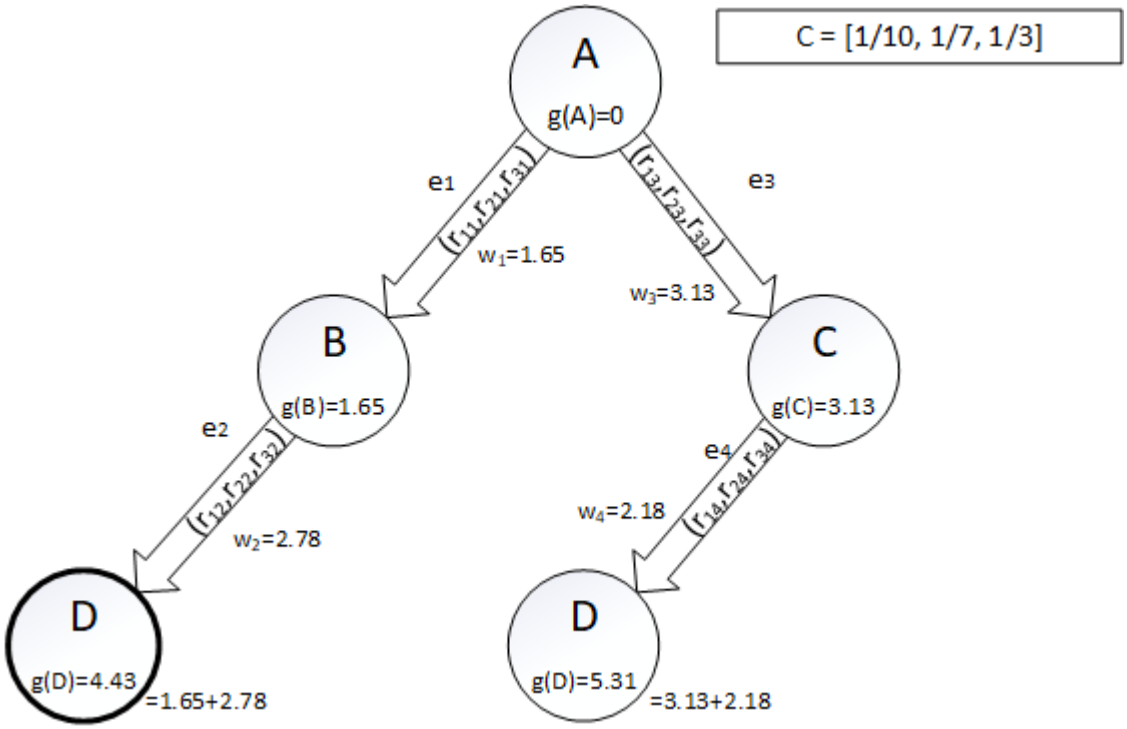


Figure 14: Expanded search tree featuring multiple resources with expected values using the aforementioned Problem Graph (Figure 13). Two possible paths are realized with one path yielding less cost than the other.

3.2.5 Scenario 5: Static Single Resource with Expected Values over Dependent Events

In some scenarios, risks may be denoted as part of dependent events, meaning that the probability of future events occurring changes with each successive outcome in the previous calculation. Expected values may also be derived using a modified formula:

$$E'(X) = \sum_i P_i * \left[\prod_{j=1}^{i-1}(1 - P_j)\right] * x_i$$

Where P_i is represented as the probability of the only event i occurring and previous events j failing to occur.

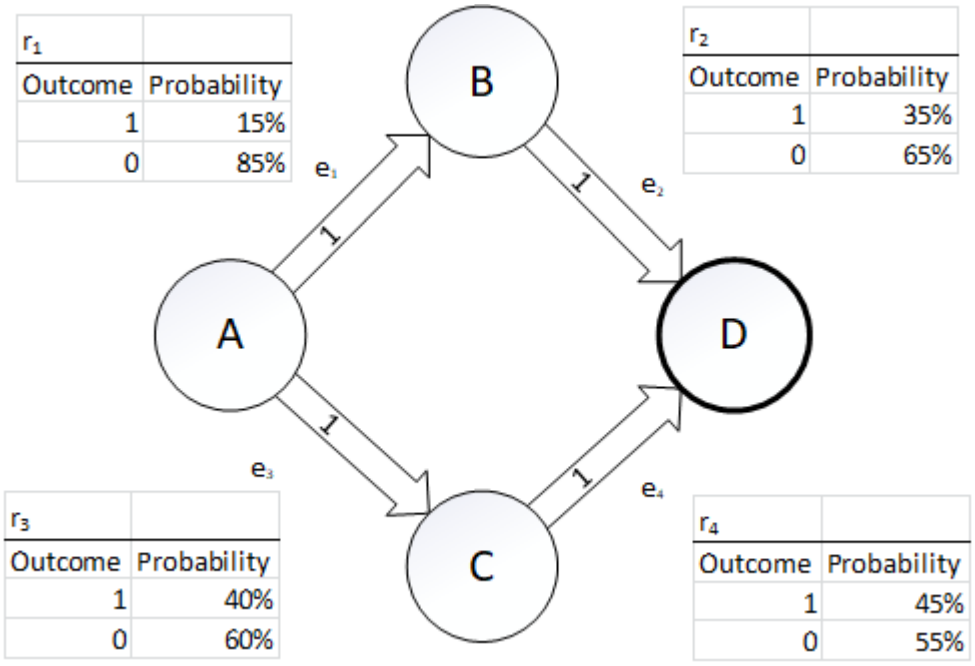


Figure 15: Problem Graph featuring a single resource per edge with expected values.

The expected value of the resources using the R matrix becomes:

$$E'(R) = [1 * .15 \quad 1 * .35(1 - .15) \quad 1 * .4 \quad 1 * .45(1 - .4)]$$

Implying:

$$w(e_1) = [0.15],\ w(e_2) = [0.35(1 - 0.15)],\ w(e_3) = [0.4],\ w(e_4) = [0.45(1 - 0.4)]$$

$$W \approx \{0.15, 0.3, 0.4, 0.27\}$$

The expanded search tree used by A* is illustrated as such:

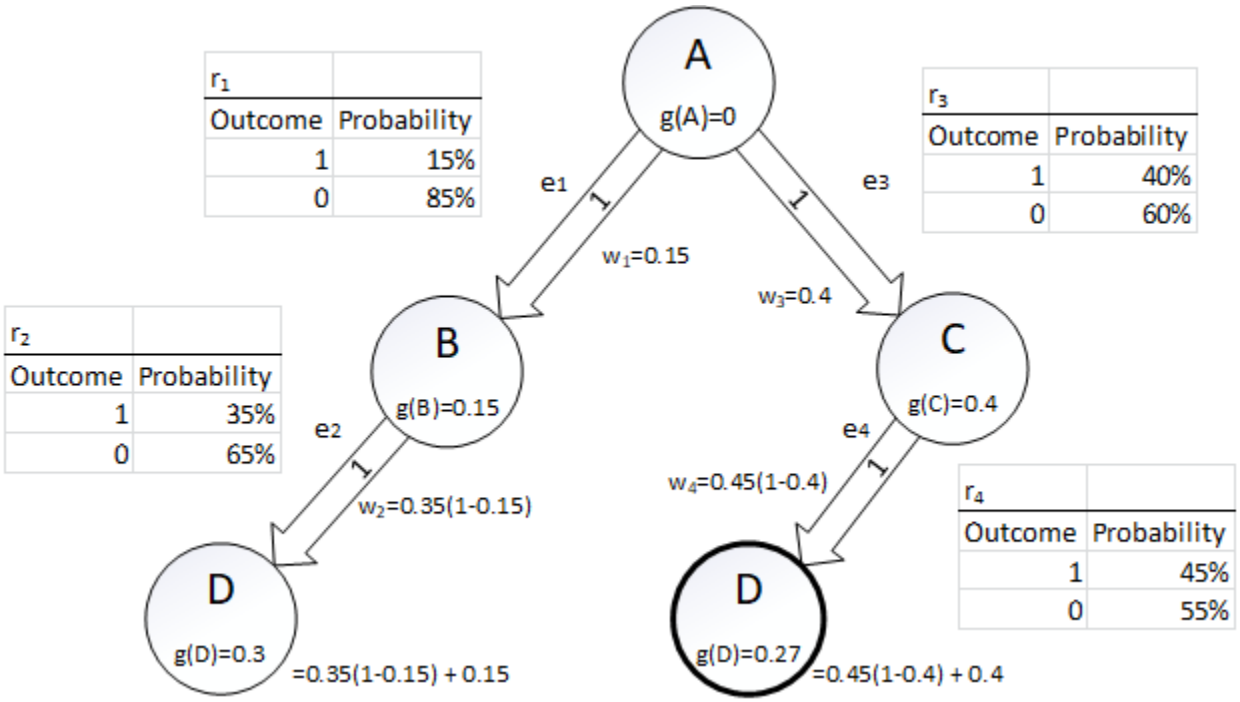
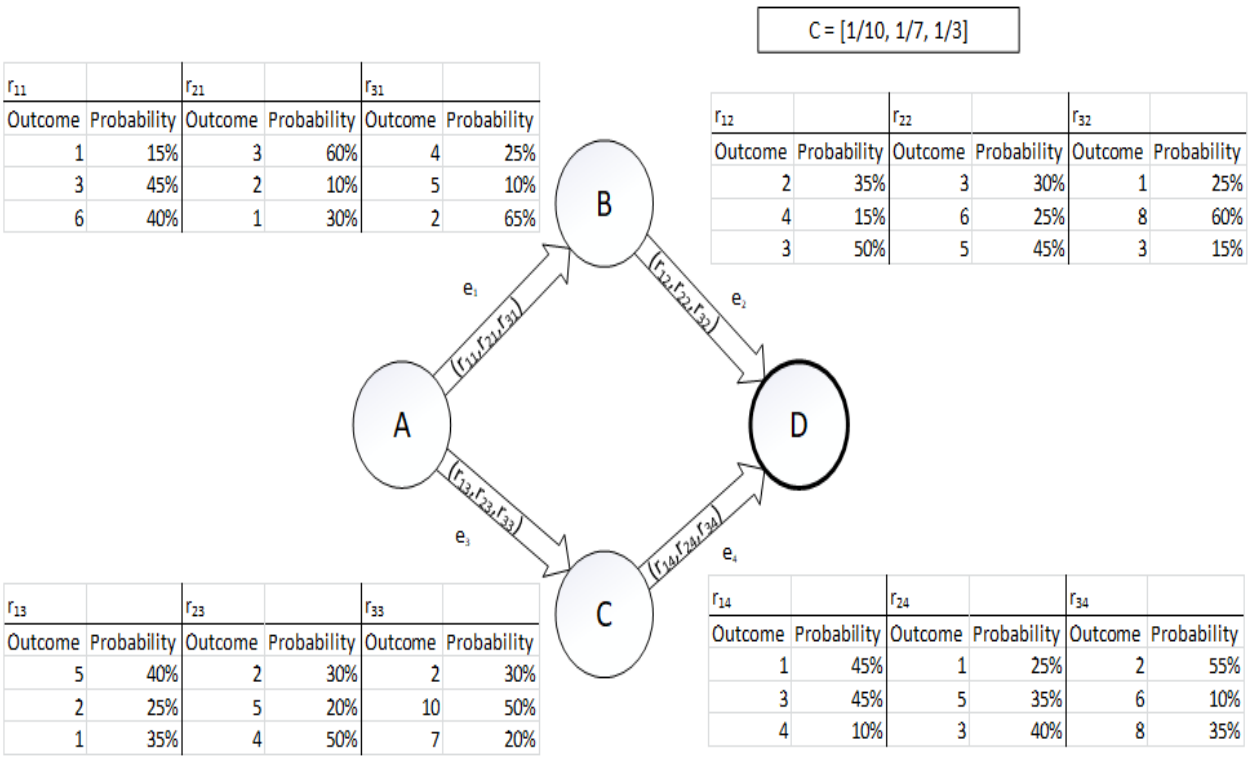


Figure 16: Expanded search tree featuring a single resource with expected values over dependent events using the aforementioned Problem Graph (Figure 15).

3.2.6 Scenario 6: Static Multiple Resources with Expected Values over Dependent Events

Recall that for Scenario 3, the formula $\mathrm{E}'(X) = \sum_i P_i * x_i$, is used for determining the expected values of costs with *independent events*. With *dependent events*, however, we must use the modified formula as demonstrated in Scenario 4, as the success of subsequent events depends on the success of preceding events.



r_{11}		r_{21}		r_{31}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	15%	3	60%	4	25%
3	45%	2	10%	5	10%
6	40%	1	30%	2	65%

r_{12}		r_{22}		r_{32}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
2	35%	3	30%	1	25%
4	15%	6	25%	8	60%
3	50%	5	45%	3	15%

r_{13}		r_{23}		r_{33}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
5	40%	2	30%	2	30%
2	25%	5	20%	10	50%
1	35%	4	50%	7	20%

r_{14}		r_{24}		r_{34}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	45%	1	25%	2	55%
3	45%	5	35%	6	10%
4	10%	3	40%	8	35%

Figure 17: Problem Graph featuring multiple resources with expected values. Each outcome comes with a probability of occurrence, and thus an expected value can be determined for each resource. For successive outcomes, the probability of previous edge-weights needs to be taken into consideration, hence the dependencies.

The example depicted in Figure 13 can be reused in this scenario (see Figure 17), and we can denote the expected values using the R matrix as follows:

$$E'(R) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \end{bmatrix}$$

Where:

Row 1:

$r_{11} = 1 * .15 + 3 * .45 + 6 * .4,$

$r_{12} = (2 * .35) * (1 - .15) + (4 * .15) * (1 - .45) + (3 * .5) * (1 - .4),$

$r_{13} = 5 * .4 + 2 * .25 + 1 * .35,$

$r_{14} = (1 * .45) * (1 - .4) + (3 * .45) * (1 - .25) + (4 * .1) * (1 - .35),$

Row 2:

$r_{21} = 3 * .6 + 2 * .1 + 1 * .3,$

$r_{22} = (3 * .3) * (1 - .6) + (6 * .25) * (1 - .1) + (5 * .45) * (1 - .3),$

$r_{23} = 2 * .3 + 5 * .2 + 4 * .5,$

$r_{24} = (1 * .25) * (1 - .3) + (5 * .35) * (1 - .2) + (3 * .4) * (1 - .5),$

Row 3:

$r_{31} = 4 * .25 + 5 * .1 + 2 * .65,$

$r_{32} = (1 * .25) * (1 - .25) + (8 * .6) * (1 - .1) + (3 * .15) * (1 - .64),$

$r_{33} = 2 * .3 + 10 * .5 + 7 * .2,$

$r_{34} = (2 * .55) * (1 - .3) + (6 * .1) * (1 - .5) + (8 * .35) * (1 - .2)$

Thus:

$$E'(R) = \begin{bmatrix} 3.9 & 1.83 & 2.85 & 1.54 \\ 2.3 & 3.29 & 3.6 & 2.18 \\ 2.8 & 4.67 & 7 & 3.31 \end{bmatrix}$$

$$C = \left[\frac{1}{10}, \frac{1}{7}, \frac{1}{3}\right]$$

Now, by using the formula $W(E) = CE'(R)$:

$$W(E) = \begin{Bmatrix} \left[\frac{1}{10} * 3.9 + \frac{1}{7} * 2.3 + \frac{1}{3} * 2.8\right], \left[\frac{1}{10} * 1.83 + \frac{1}{7} * 3.29 + \frac{1}{3} * 4.67\right], \\ \left[\frac{1}{10} * 2.85 + \frac{1}{7} * 3.6 + \frac{1}{3} * 7\right], \left[\frac{1}{10} 1.54 + \frac{1}{7} * 2.18 + \frac{1}{3} * 3.31\right] \end{Bmatrix}$$

Implying:

$$w(e_1) = \left[\frac{3.9}{10} + \frac{2.3}{7} + \frac{2.8}{3}\right], w(e_2) = \left[\frac{1.83}{10} + \frac{3.29}{7} + \frac{4.67}{3}\right], w(e_3) = \left[\frac{2.85}{10} + \frac{3.6}{7} + \frac{7}{3}\right],$$

$$w(e_4) = \left[\frac{1.54}{10} + \frac{2.18}{7} + \frac{3.31}{3}\right]$$

$W \approx \{1.65, 2.21, 3.13, 1.56\}$

The expanded search tree used by A* is illustrated as such:

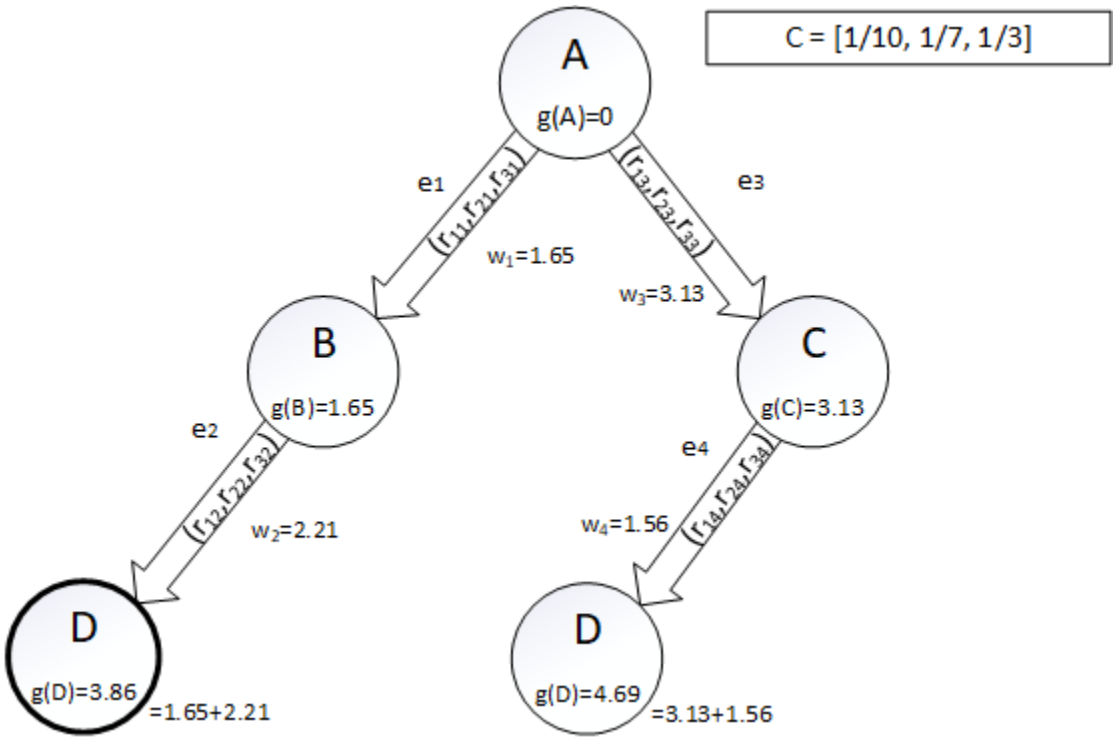


Figure 18: Expanded search tree featuring multiple resources with expected values over dependent events using the aforementioned Problem Graph (Figure 17).

3.2.7 Scenario 7: Dynamic Single Resource with Known Temporal Values

In the previous scenarios, maps were described as static, meaning that the properties of the graph do not change over the course of the pathfinding process. For scenarios 7 and 8, we prepare a solution catered to edge-weights that may change during processing, and whose values are a predetermined function of time. We refer to these values as *known temporal values*. It should be noted that they may not always evaluate to the same result at any given point during the pathfinding process.

A new variable t is introduced, representing the current step in time. An extended weight function that accounts for time is defined as:

$$W(t) = R(t)$$

Where, similar to Scenario 1, Scenario 3 and Scenario 5, *R* is a one-dimensional matrix of the set containing the resources on each edge. During path processing, the value of a resource r_n in *R* is calculated for its value at time *t*. Unlike previous scenarios, however, which do not impose changes onto the A* Search algorithm itself, one small modification to the cost formula in A* Search must be made for the dynamic cases:

$$f(n) = w(g(n), t)$$

Where the total cost leading up to node *n* accepts the current cost and the current step in time as parameters. The total cost can now vary depending on the value of *t* when node *n* is expanded. This change is necessary and is stored in each node at the time it is expanded much like in the previously described A* Search algorithm. This modification to the cost function also holds true for all remaining scenarios following this point.

The changes may be applied to the existing Dijkstra's Algorithm on the following lines:

```
17:  if g(current)+ c(current,neighbour,t) < g(neighbour) then
18:    g(neighbour) := g(current) + c(current, neighbour,t);
```

It should be noted that while it is possible for finite loops to occur during the path planning process, the solution is still guaranteed to find an optimal path so long as one exists. This is not exclusive to dynamic cases, but static cases as well. Although alternative paths may result in looping during processing, when the g-value of a dead-end path that is also an alternative exceeds that of the optimal path, planning will still follow the optimal path as is true for Dijkstra's Algorithm.

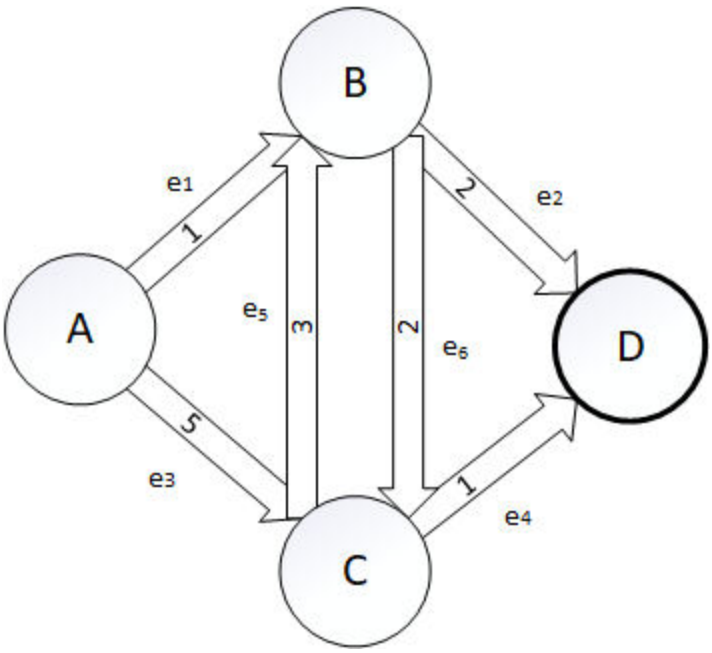


Figure 19: Problem Graph featuring a single resource per edge with known temporal values. Not that it looks very similar to scenario 1, however, each edge now also has an underlying weight function associated with it. Resources on the edges are shown at time 0.

Due to the previously defined *dynamic* property of the graph, edge-weights will change at different times. Suppose these values evaluate to the following:

t	w_{e_1}	w_{e_2}	w_{e_3}	w_{e_4}	w_{e_5}	w_{e_6}
0	1	2	5	1	3	2
1	3	8	6	3	2	1
2	3	2	5	3	2	6
3	2	4	7	3	2	1
4	2	4	2	3	6	4

Based on this *known* data, we can evaluate what the given edge-weights will be at any given point in time. Moreover, the costs to be determined can be calculated based on the current time-step.

Like previous scenarios, we define our sets as follows, however, this time we base the weight in terms of time *t*:

$V = \{A, B, C, D\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

$W = \{w(e_1), w(e_2), w(e_3), w(e_4), w(e_5), w(e_6)\}$

At time $t = 0$, our set will be:

$$W(0) = \{(w_{e_1}, 0), (w_{e_2}, 0), (w_{e_3}, 0), (w_{e_4}, 0), (w_{e_5}, 0), (w_{e_6}, 0)\} = \{1, 2, 5, 1, 3, 2\}$$

At time $t = 1$:

$$W(1) = \{(w_{e_1}, 1), (w_{e_2}, 1), (w_{e_3}, 1), (w_{e_4}, 1), (w_{e_5}, 1), (w_{e_6}, 1)\} = \{3, 8, 6, 3, 2, 1\}$$

And subsequently:

$$W(2) = \{(w_{e_1}, 2), (w_{e_2}, 2), (w_{e_3}, 2), (w_{e_4}, 2), (w_{e_5}, 2), (w_{e_6}, 2)\} = \{3, 2, 5, 3, 2, 6\}$$

$$W(3) = \{(w_{e_1}, 3), (w_{e_2}, 3), (w_{e_3}, 3), (w_{e_4}, 3), (w_{e_5}, 3), (w_{e_6}, 3)\} = \{2, 4, 7, 3, 2, 1\}$$

$$W(4) = \{(w_{e_1}, 4), (w_{e_2}, 4), (w_{e_3}, 4), (w_{e_4}, 4), (w_{e_5}, 4), (w_{e_6}, 4)\} = \{2, 4, 2, 3, 6, 4\}$$

Thus, the values contained in set W are time-dependent. It is also assumed that the time to traverse an edge is fixed and known. Note that in some instances, the time of traversal is equal to the amount of resource used, although this may not always be the case.

The expanded search tree used by A* is illustrated as such:

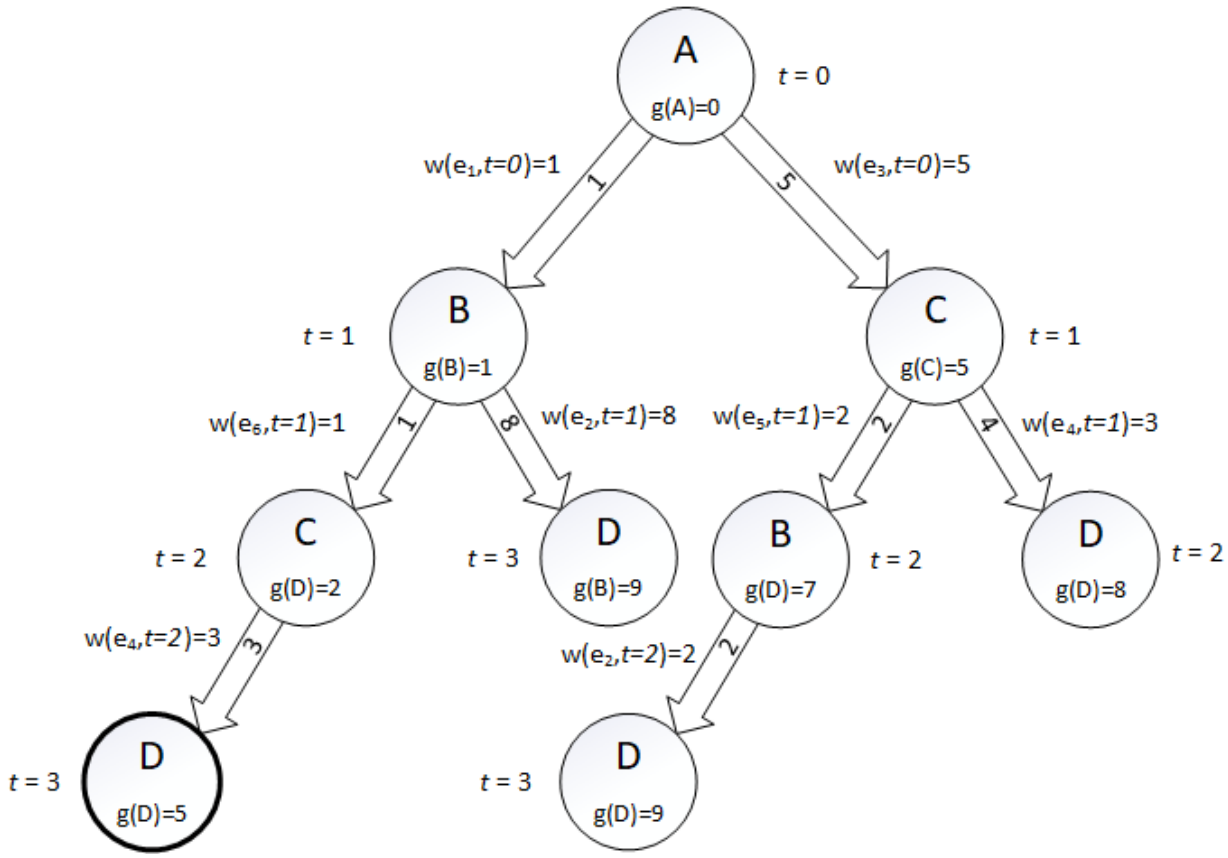


Figure 20: Expanded search tree featuring single resource with known temporal values based on the aforementioned Problem Graph (Figure 19). The chart displaying what each edge evaluates to at time *t* is described per edge.

3.2.8 Scenario 8: Dynamic Multiple Resources with Known Temporal Values

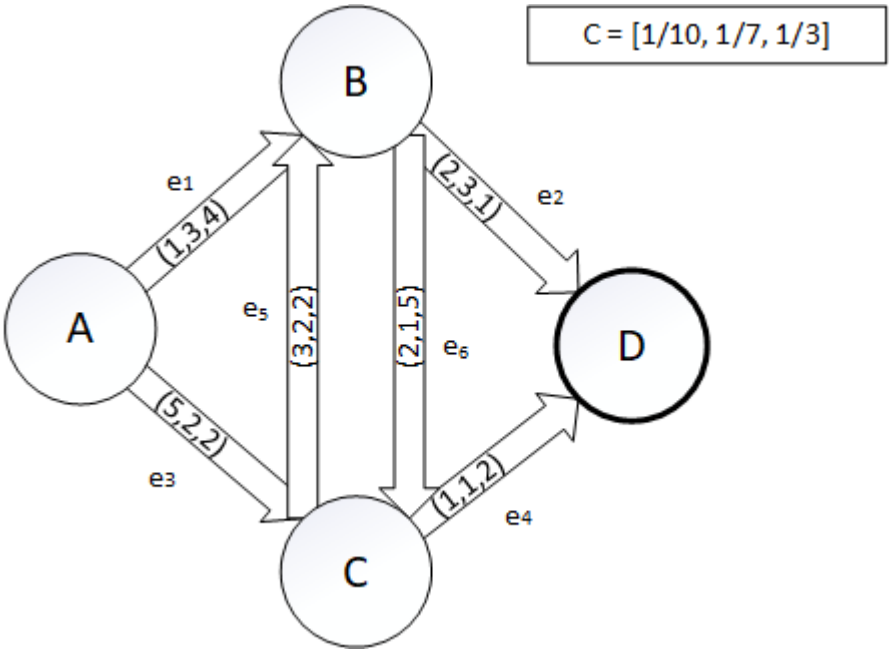


Figure 21: Problem Graph featuring multiple resources per edge with known temporal values. Note that it looks very similar to scenario 2, however, each edge now also has an underlying weight function associated with it, and each resource has a corresponding value at a given time *t*. Resources on the edges are shown at time 0.

Suppose these values evaluate to the following for each set of resources:

r_1:

t	$w_{e_1r_1}$	$w_{e_2r_1}$	$w_{e_{3r_1}}$	$w_{e_4r_1}$	$w_{e_5r_1}$	$w_{e_6r_1}$
0	1	2	5	1	3	2
1	3	8	6	3	2	1
2	3	2	5	3	2	6
3	2	4	7	3	2	1

r_2:

t	$w_{e_1r_2}$	$w_{e_2r_2}$	$w_{e_{3r_2}}$	$w_{e_4r_2}$	$w_{e_5r_2}$	$w_{e_6r_2}$
0	3	3	2	1	2	1
1	2	1	6	7	4	3
2	6	0	2	1	1	2
3	4	3	2	1	7	2

r_3:

t	$w_{e_1r_3}$	$w_{e_2r_3}$	$w_{e_{3r_3}}$	$w_{e_4r_3}$	$w_{e_5r_3}$	$w_{e_6r_3}$
0	4	1	2	2	2	5
1	2	4	3	2	2	1

2	1	2	2	8	5	4
3	1	3	2	5	5	2

Similar to scenario 2, we define a matrix for storing the attributes of the problem graph. In this case, however, we define a 3-dimensional matrix $r \times |E| \times t$ matrix R where there is a row for each resource, a column for each edge, and a depth for each step in t. $R[i, j, k]$ represents the amount of resource i expended when traversing edge j at specified time k.

We define our sets as follows:

$V = \{A, B, C, D\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

$W = \{w(e_1), w(e_2), w(e_3), w(e_4), w(e_5), w(e_6)\}$

For r_1, our sets will be:

$W_{r_1}(0) = \{(w_{e_1r_1}, 0), (w_{e_2r_1}, 0), (w_{e_3r_1}, 0), (w_{e_4r_1}, 0), (w_{e_5r_1}, 0), (w_{e_6r_1}, 0)\} = \{1,2,5,1,3,2\}$

$W_{r_1}(1) = \{(w_{e_1r_1}, 1), (w_{e_2r_1}, 1), (w_{e_3r_1}, 1), (w_{e_4r_1}, 1), (w_{e_5r_1}, 1), (w_{e_6r_1}, 1)\} = \{3,8,6,3,2,1\}$

$W_{r_1}(2) = \{(w_{e_1r_1}, 2), (w_{e_2r_1}, 2), (w_{e_3r_1}, 2), (w_{e_4r_1}, 2), (w_{e_5r_1}, 2), (w_{e_6r_1}, 2)\} = \{3,2,5,3,2,6\}$

$W_{r_1}(3) = \{(w_{e_1r_1}, 3), (w_{e_2r_1}, 3), (w_{e_3r_1}, 3), (w_{e_4r_1}, 3), (w_{e_5r_1}, 3), (w_{e_6r_1}, 3)\} = \{2,4,7,3,2,1\}$

For r_2, our sets will be:

$W_{r_2}(0) = \{(w_{e_1r_2}, 0), (w_{e_2r_2}, 0), (w_{e_3r_2}, 0), (w_{e_4r_2}, 0), (w_{e_5r_2}, 0), (w_{e_6r_2}, 0)\} = \{3,3,2,1,2,1\}$

$W_{r_2}(1) = \{(w_{e_1r_2}, 1), (w_{e_2r_2}, 1), (w_{e_3r_2}, 1), (w_{e_4r_2}, 1), (w_{e_5r_2}, 1), (w_{e_6r_2}, 1)\} = \{2,1,6,7,4,3\}$

$W_{r_2}(2) = \{(w_{e_1r_2}, 2), (w_{e_2r_2}, 2), (w_{e_3r_2}, 2), (w_{e_4r_2}, 2), (w_{e_5r_2}, 2), (w_{e_6r_2}, 2)\} = \{6,0,2,1,1,2\}$

$W_{r_2}(3) = \{(w_{e_1r_2}, 3), (w_{e_2r_2}, 3), (w_{e_3r_2}, 3), (w_{e_4r_2}, 3), (w_{e_5r_2}, 3), (w_{e_6r_2}, 3)\} = \{4,3,2,1,7,2\}$

For r_3, our sets will be:

$W_{r_3}(0) = \{(w_{e_1r_3}, 0), (w_{e_2r_3}, 0), (w_{e_3r_3}, 0), (w_{e_4r_3}, 0), (w_{e_5r_3}, 0), (w_{e_6r_3}, 0)\} = \{4,1,2,2,2,5\}$

$$W_{r_3}(1) = \{(w_{e_1r_3}, 1), (w_{e_2r_3}, 1), (w_{e_3r_3}, 1), (w_{e_4r_3}, 1), (w_{e_5r_3}, 1), (w_{e_6r_3}, 1)\} = \{2,4,3,2,2,1\}$$

$$W_{r_3}(2) = \{(w_{e_1r_3}, 2), (w_{e_2r_3}, 2), (w_{e_3r_3}, 2), (w_{e_4r_3}, 2), (w_{e_5r_3}, 2), (w_{e_6r_3}, 2)\} = \{1,2,2,8,5,4\}$$

$$W_{r_3}(3) = \{(w_{e_1r_3}, 3), (w_{e_2r_3}, 3), (w_{e_3r_3}, 3), (w_{e_4r_3}, 3), (w_{e_5r_3}, 3), (w_{e_6r_3}, 3)\} = \{1,3,2,5,5,2\}$$

The finalized weights after conversion will look as such:

$$W(E,t)$$

$$= \begin{Bmatrix} \left[\frac{1}{10} * W_{r_1}(t) + \frac{1}{7} * W_{r_2}(t) + \frac{1}{3} * W_{r_3}(t)\right], \left[\frac{1}{10} * W_{r_1}(t) + \frac{1}{7} * W_{r_2}(t) + \frac{1}{3} * W_{r_3}(t)\right], \\ \left[\frac{1}{10} * W_{r_1}(t) + \frac{1}{7} * W_{r_2}(t) + \frac{1}{3} * W_{r_3}(t)\right], \left[\frac{1}{10} * W_{r_1}(t) + \frac{1}{7} * W_{r_2}(t) + \frac{1}{3} * W_{r_3}(t)\right], \\ \left[\frac{1}{10} * W_{r_1}(t) + \frac{1}{7} * W_{r_2}(t) + \frac{1}{3} * W_{r_3}(t)\right], \left[\frac{1}{10} * W_{r_1}(t) + \frac{1}{7} * W_{r_2}(t) + \frac{1}{3} * W_{r_3}(t)\right] \end{Bmatrix}$$

The expanded search tree used by A* is illustrated as such:

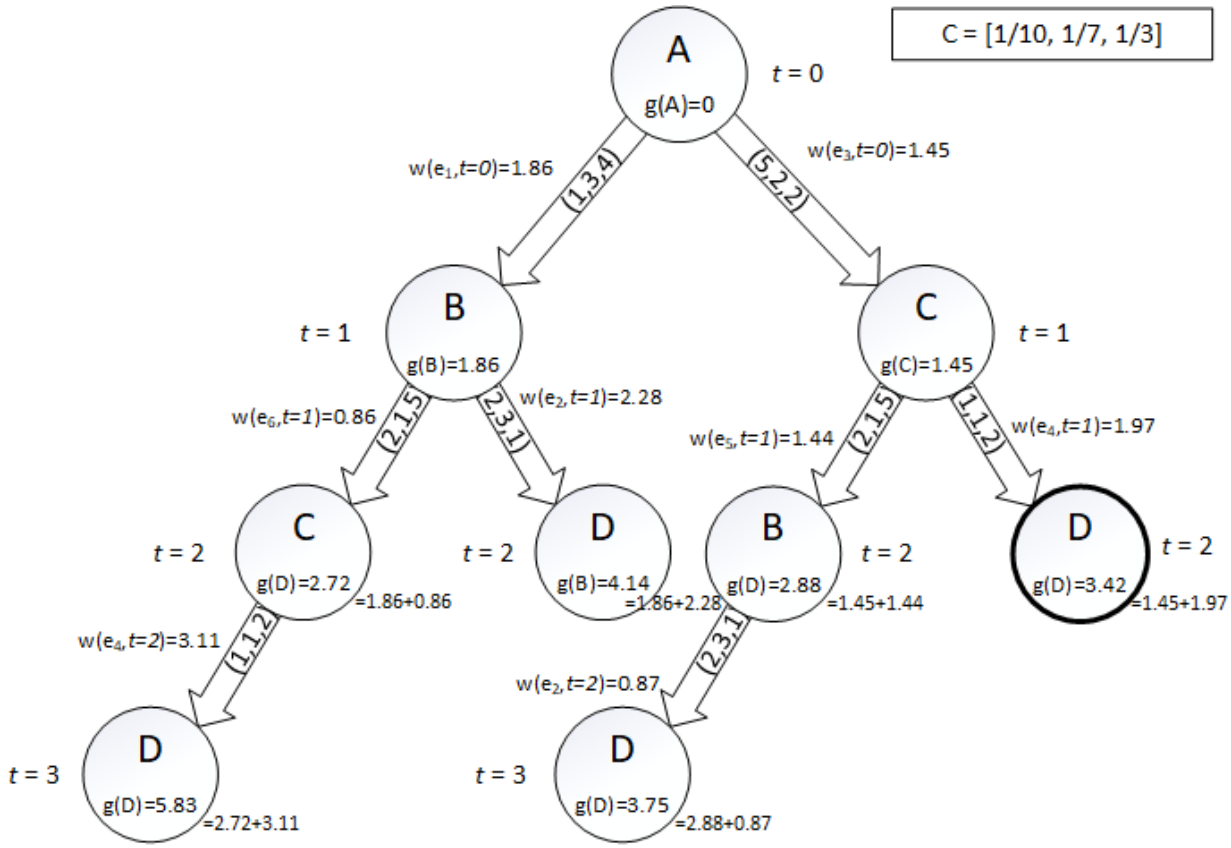


Figure 22: Expanded search tree featuring multiple resources per edge with known temporal values based on the aforementioned Problem Graph (Figure 21).

The edge-weights are determined based on the following calculations:

$$w_{r_1}(e_1, t=0) = 1, w_{r_2}(e_1, t=0) = 3, w_{r_3}(e_1, t=0) = 4$$

$$\rightarrow w(e_1, 0) = \left(1 * \frac{1}{10}\right) + \left(3 * \frac{1}{7}\right) + \left(4 * \frac{1}{3}\right) = 1.86$$

$$w_{r_1}(e_3, t=0) = 5, w_{r_2}(e_3, t=0) = 2, w_{r_3}(e_3, t=0) = 2$$

$$\rightarrow w(e_3, 0) = \left(5 * \frac{1}{10}\right) + \left(2 * \frac{1}{7}\right) + \left(2 * \frac{1}{3}\right) = 1.45$$

$$w_{r_1}(e_4, t=1) = 3, w_{r_2}(e_4, t=1) = 7, w_{r_3}(e_4, t=1) = 2$$

$$\rightarrow w(e_4, 1) = \left(3 * \frac{1}{10}\right) + \left(7 * \frac{1}{7}\right) + \left(2 * \frac{1}{3}\right) = 1.97$$

$$w_{r_1}(e_2, t=1) = 8, w_{r_2}(e_2, t=1) = 1, w_{r_3}(e_2, t=1) = 4$$

$$\rightarrow w(e_2, 1) = \left(8 * \frac{1}{10}\right) + \left(1 * \frac{1}{7}\right) + \left(4 * \frac{1}{3}\right) = 2.28$$

$$w_{r_1}(e_5, t=1) = 2, w_{r_2}(e_5, t=1) = 4, w_{r_3}(e_5, t=1) = 2$$

$$\rightarrow w(e_5, 1) = \left(2 * \frac{1}{10}\right) + \left(4 * \frac{1}{7}\right) + \left(2 * \frac{1}{3}\right) = 1.44$$

$$w_{r_1}(e_6, t=1) = 1, w_{r_2}(e_6, t=1) = 3, w_{r_3}(e_6, t=1) = 1$$

$$\rightarrow w(e_6, 1) = \left(1 * \frac{1}{10}\right) + \left(3 * \frac{1}{7}\right) + \left(1 * \frac{1}{3}\right) = 0.86$$

$$w_{r_1}(e_4, t=2) = 3, w_{r_2}(e_4, t=2) = 1, w_{r_3}(e_4, t=2) = 8$$

$$\rightarrow w(e_4, 1) = \left(3 * \frac{1}{10}\right) + \left(1 * \frac{1}{7}\right) + \left(8 * \frac{1}{3}\right) = 3.11$$

$$w_{r_1}(e_2, t=2) = 2, w_{r_2}(e_2, t=2) = 0, w_{r_3}(e_2, t=2) = 2$$

$$\rightarrow w(e_2, 1) = \left(2 * \frac{1}{10}\right) + \left(0 * \frac{1}{7}\right) + \left(2 * \frac{1}{3}\right) = 0.87$$

3.2.9 Scenario 9: Dynamic Single Resource with Expected Values over Independent Events and Known Temporal Values

In this scenario, known temporal values can be applied to expected values. Not only does the outcome of resources have an expected value, but the probability at which these outcomes occur may also be time-dependent. Thus, a weight function must be applied to both of these components.

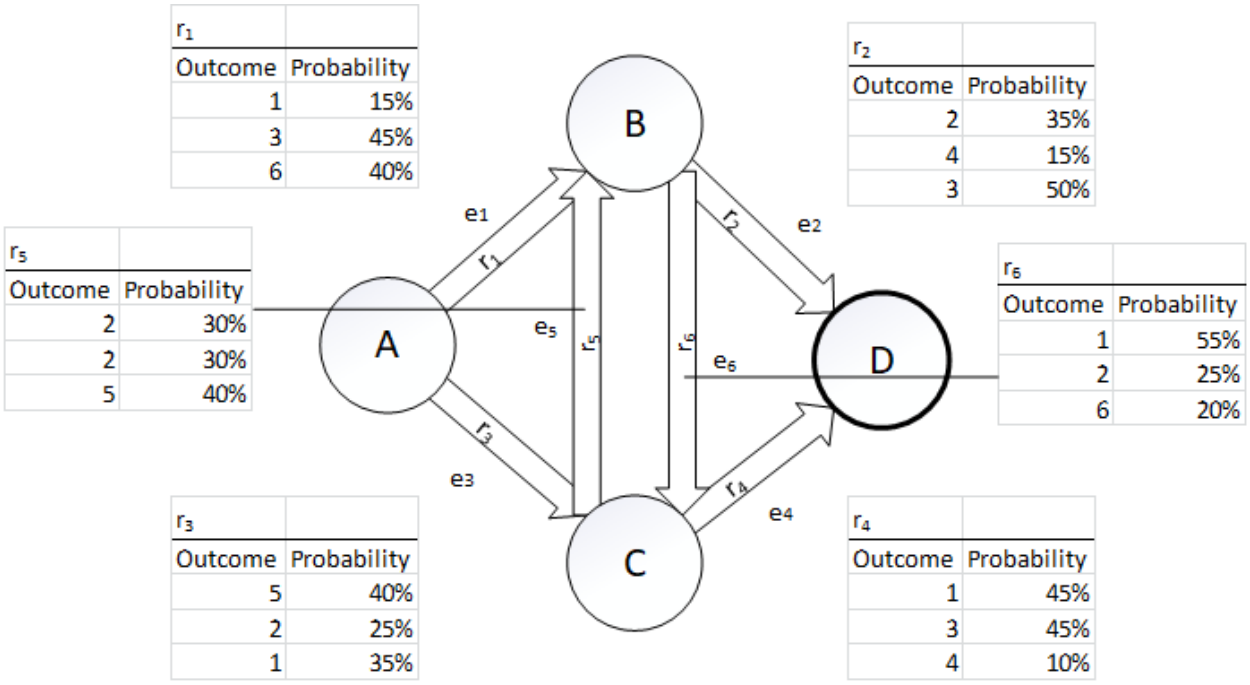


Figure 23: Problem Graph featuring a single resource per edge with independent events and known temporal values. Each edge has an underlying weight function associated with it, and each resource has a corresponding expected value at a given time *t*. Resources on the edges are shown at time 0.

Like previous scenarios, we define our sets as follows:

$V = \{A, B, C, D\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

$W = \{w(e_1), w(e_2), w(e_3), w(e_4), w(e_5), w(e_6)\}$

Due to the values being temporal, both the outcomes and the associated probabilities (paired) per event may change over time. Suppose these events evaluate to the following:

Ev_{r_1}:

t	$p_{r_1 1}$	$p_{r_1 2}$	$p_{r_1 3}$
0	(1,0.15)	(3,0.45)	(6,0.4)
1	(4,0.3)	(2,0.3)	(2,0.4)
2	(2,0.1)	(5,0.2)	(0,0.7)
3	(5,0.15)	(0,0.4)	(1,0.45)

Ev_{r_2}:

t	$p_{r_2 1}$	$p_{r_2 2}$	$p_{r_2 3}$
0	(2,0.35)	(4,0.15)	(3,0.5)
1	(3,0.3)	(6,0.3)	(1,0.4)
2	(1,0.1)	(2,0.2)	(1,0.7)
3	(7,0.15)	(1,0.4)	(5,0.45)

Ev_{r_3}:

t	$p_{r_3 1}$	$p_{r_3 2}$	$p_{r_3 3}$
0	(5,0.4)	(2,0.25)	(1,0.35)
1	(3,0.3)	(3,0.3)	(1,0.4)
2	(1,0.1)	(2,0.2)	(4,0.7)
3	(0,0.15)	(7,0.4)	(2,0.45)

Ev_{r_4}:

t	$p_{r_4 1}$	$p_{r_4 2}$	$p_{r_4 3}$
0	(1,0.45)	(3,0.45)	(4,0.1)
1	(3,0.3)	(2,0.3)	(6,0.4)
2	(1,0.1)	(1,0.2)	(4,0.7)
3	(6,0.15)	(2,0.4)	(3,0.45)

Ev_{r_5}:

t	$p_{r_5 1}$	$p_{r_5 2}$	$p_{r_5 3}$
0	(2,0.3)	(2,0.3)	(5,0.4)
1	(5,0.3)	(2,0.3)	(1,0.4)
2	(4,0.1)	(2,0.2)	(5,0.7)
3	(4,0.15)	(3,0.4)	(6,0.45)

Ev_{r_6}:

t	$p_{r_6 1}$	$p_{r_6 2}$	$p_{r_6 3}$
0	(1,0.55)	(2,0.25)	(6,0.2)
1	(1,0.3)	(6,0.3)	(4,0.4)
2	(1,0.1)	(5,0.2)	(5,0.7)
3	(1,0.15)	(1,0.4)	(3,0.45)

The expected values for each edge-weight will be:

$$W(0) = \{(w_{E'(e_1)}, 0), (w_{E'(e_2)}, 0), (w_{E'(e_3)}, 0), (w_{E'(e_4)}, 0), (w_{E'(e_5)}, 0), (w_{E'(e_6)}, 0)\}$$

$$= \begin{Bmatrix} 1*.15+3*.45+6*.4, 2*.35+4*.15+3*.5, 5*.4+2*.25+1*.35, \\ 1*.45+3*.45+4*.1, 2*.3+2*.3+5*.4, 1*.55+2*.25+6*.2 \end{Bmatrix}$$

$$= \{3.9, 2.8, 2.85, 2.2, 3.2, 2.25\}$$

$$W(1) = \{(w_{E'(e_1)}, 1), (w_{E'(e_2)}, 1), (w_{E'(e_3)}, 1), (w_{E'(e_4)}, 1), (w_{E'(e_5)}, 1), (w_{E'(e_6)}, 1)\}$$

$$= \begin{Bmatrix} 4*.3+2*.3+2*.4, 3*.3+6*.3+1*.4, 3*.3+3*.3+1*.4, \\ 3*.3+2*.3+6*.4, 5*.3+2*.3+1*.4, 1*.3+6*.3+4*.4 \end{Bmatrix}$$

$$= \{2.6, 3.1, 2.2, 3.9, 2.5, 3.7\}$$

$$W(2) = \{(w_{E'(e_1)}, 2), (w_{E'(e_2)}, 2), (w_{E'(e_3)}, 2), (w_{E'(e_4)}, 2), (w_{E'(e_5)}, 2), (w_{E'(e_6)}, 2)\}$$

$$= \begin{Bmatrix} 2*.1+5*.2+0*.7, 1*.1+2*.2+1*.7, 1*.1+2*.2+4*.7, \\ 1*.1+1*.2+4*.7, 4*.1+2*.2+5*.7, 1*.1+5*.2+5*.7 \end{Bmatrix}$$

$$= \{1.2, 1.2, 3.3, 3.1, 4.3, 4.6\}$$

$$W(3) = \{(w_{E'(e_1)}, 3), (w_{E'(e_2)}, 3), (w_{E'(e_3)}, 3), (w_{E'(e_4)}, 3), (w_{E'(e_5)}, 3), (w_{E'(e_6)}, 3)\}$$

$$= \begin{Bmatrix} 5*.15+0*.4+1*.45, 7*.15+1*.4+5*.45, 0*.15+7*.4+2*.45, \\ 6*.15+2*.4+3*.45, 4*.12+3*.4+6*.45, 1*.15+1*.4+3*.45 \end{Bmatrix}$$

$$= \{1.2, 3.7, 3.7, 3.05, 4.38, 1.9\}$$

The expanded search tree used by A* is illustrated as such:

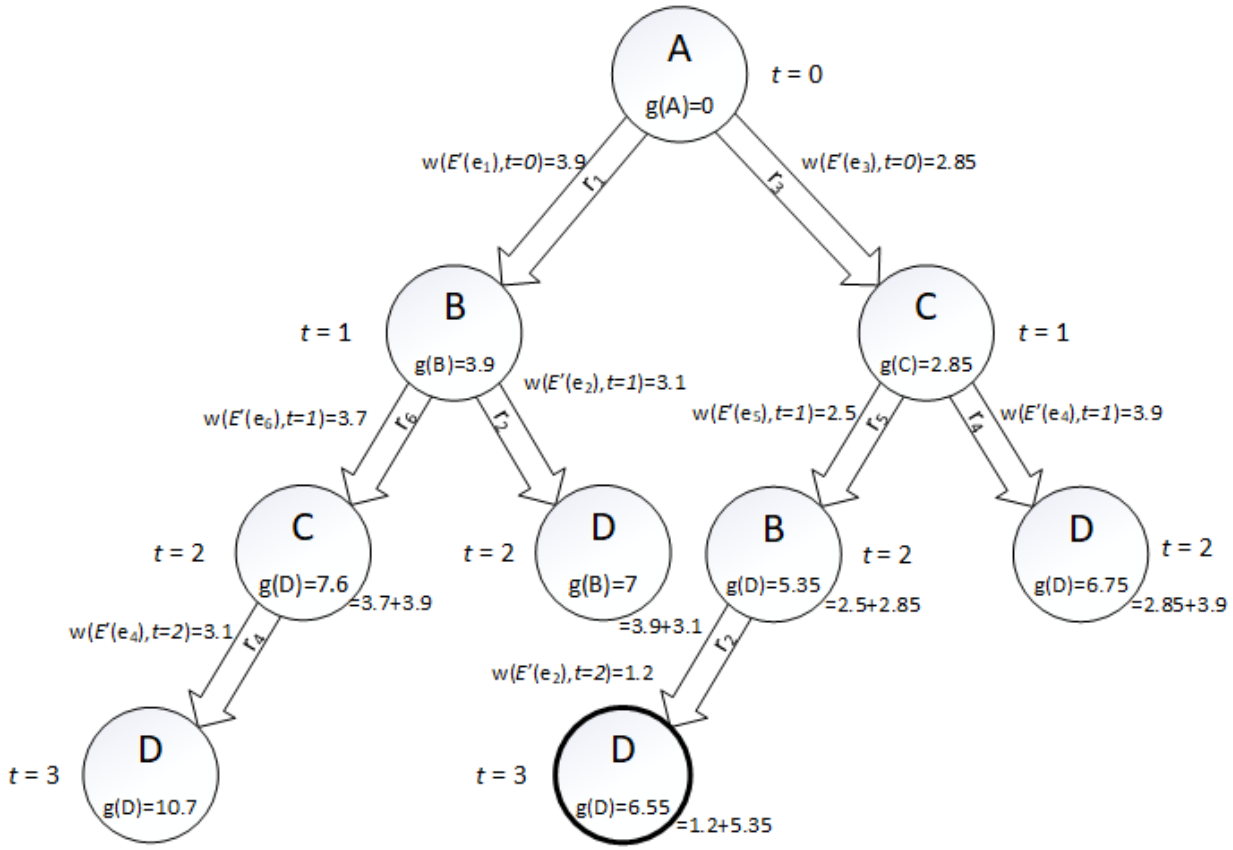
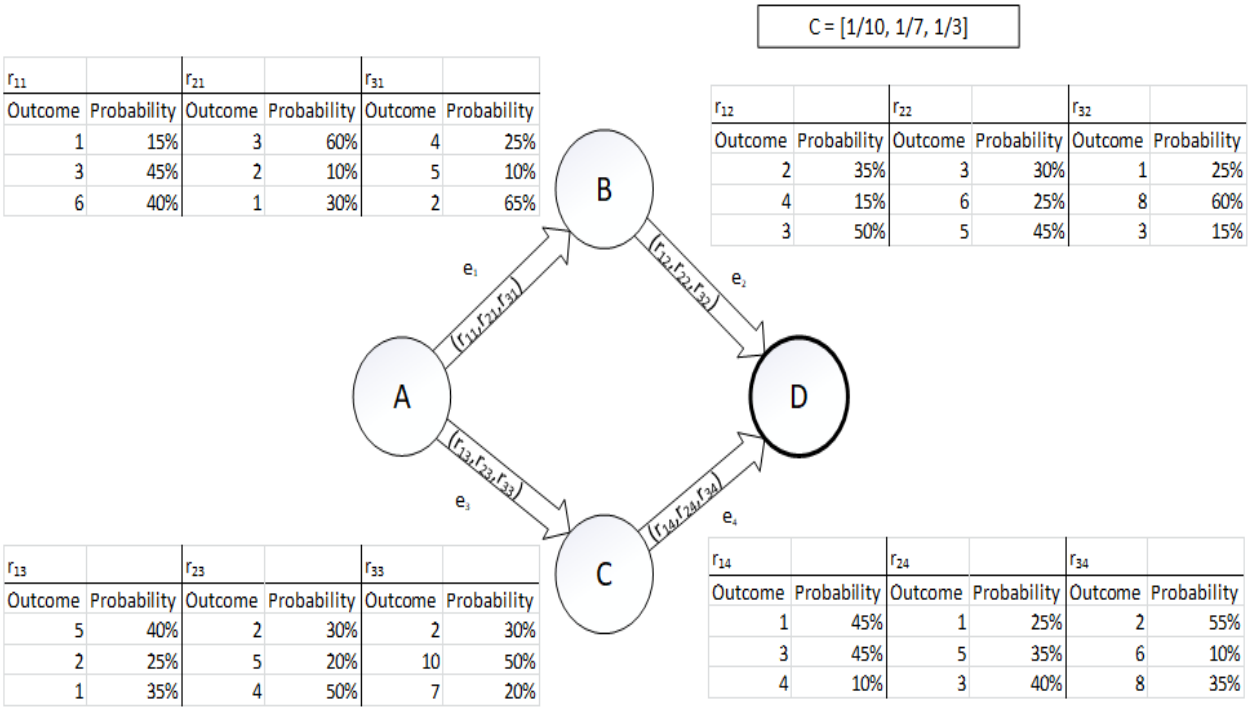


Figure 24: Expanded search tree is featuring a single resource per edge over independent events with known temporal values based on the aforementioned Problem Graph (Figure 23).

3.2.10 Scenario 10: Dynamic Multiple Resources with Expected Values over Independent Events and Known Temporal Values



r_{11}		r_{21}		r_{31}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	15%	3	60%	4	25%
3	45%	2	10%	5	10%
6	40%	1	30%	2	65%

r_{12}		r_{22}		r_{32}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
2	35%	3	30%	1	25%
4	15%	6	25%	8	60%
3	50%	5	45%	3	15%

r_{13}		r_{23}		r_{33}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
5	40%	2	30%	2	30%
2	25%	5	20%	10	50%
1	35%	4	50%	7	20%

r_{14}		r_{24}		r_{34}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	45%	1	25%	2	55%
3	45%	5	35%	6	10%
4	10%	3	40%	8	35%

Figure 25: Problem Graph featuring multiple resources per edge with independent events and known temporal values. Each edge has an underlying weight function associated with it, and each resource has a corresponding expected value at a given time *t*. The conversion rate matrix is applied to resources for cost calculation. Resources on the edges are shown at time *t* =0.

Suppose the temporal expected values evaluate to the following:

Ev_{e_1}:

t	$p_{e_1r_11}$	$p_{e_1r_12}$	$p_{e_1r_13}$	$p_{e_1r_21}$	$p_{e_1r_22}$	$p_{e_1r_23}$	$p_{e_1r_11}$	$p_{e_1r_32}$	$p_{e_1r_33}$
0	(1,0.15)	(3,0.45)	(6,0.4)	(3,0.6)	(2,0.1)	(1,0.3)	(4,0.25)	(5,0.1)	(2.0.65)
1	(4,0.25)	(5,0.1)	(2.0.65)	(3,0.6)	(2,0.1)	(1,0.3)	(1,0.15)	(3,0.45)	(6,0.4)
2	(3,0.6)	(2,0.1)	(1,0.3)	(4,0.25)	(5,0.1)	(2.0.65)	(3,0.6)	(2,0.1)	(1,0.3)
3	(4,0.25)	(5,0.1)	(2.0.65)	(1,0.15)	(3,0.45)	(6,0.4)	(4,0.25)	(5,0.1)	(2.0.65)

Ev_{e_2}:

t	$p_{e_2r_11}$	$p_{e_2r_12}$	$p_{e_2r_13}$	$p_{e_2r_21}$	$p_{e_2r_22}$	$p_{e_2r_23}$	$p_{e_2r_11}$	$p_{e_2r_32}$	$p_{e_2r_33}$
0	(2,0.35)	(4,0.15)	(3,0.5)	(3,0.3)	(6,0.25)	(5,0.45)	(1,0.25)	(8,0.6)	(3.0.15)
1	(1,0.25)	(8,0.6)	(3.0.15)	(1,0.25)	(8,0.6)	(3.0.15)	(2,0.35)	(4,0.15)	(3,0.5)
2	(3,0.3)	(6,0.25)	(5,0.45)	(2,0.35)	(4,0.15)	(3,0.5)	(3,0.3)	(6,0.25)	(5,0.45)
3	(1,0.25)	(8,0.6)	(3.0.15)	(2,0.35)	(4,0.15)	(3,0.5)	(2,0.35)	(4,0.15)	(3,0.5)

Ev_{e_3}:

t	$p_{e_3r_11}$	$p_{e_3r_12}$	$p_{e_3r_13}$	$p_{e_3r_21}$	$p_{e_3r_22}$	$p_{e_3r_23}$	$p_{e_3r_11}$	$p_{e_3r_32}$	$p_{e_3r_33}$
0	(5,0.4)	(2,0.25)	(1,0.35)	(2,0.3)	(5,0.2)	(4,0.5)	(2,0.3)	(10,0.5)	(7.0.2)
1	(2,0.3)	(5,0.2)	(4,0.5)	(2,0.3)	(10,0.5)	(7.0.2)	(5,0.4)	(2,0.25)	(1,0.35)
2	(2,0.3)	(10,0.5)	(7.0.2)	(5,0.4)	(2,0.25)	(1,0.35)	(2,0.3)	(5,0.2)	(4,0.5)
3	(2,0.3)	(10,0.5)	(7.0.2)	(5,0.4)	(2,0.25)	(1,0.35)	(2,0.3)	(5,0.2)	(4,0.5)

Ev_{e_4}:

t	$p_{e_4r_11}$	$p_{e_4r_12}$	$p_{e_4r_13}$	$p_{e_4r_21}$	$p_{e_4r_22}$	$p_{e_4r_23}$	$p_{e_4r_11}$	$p_{e_4r_32}$	$p_{e_4r_33}$
0	(1,0.45)	(3,0.45)	(4,0.1)	(1,0.25)	(5,0.35)	(3,0.4)	(2,0.55)	(6,0.1)	(8.0.35)
1	(2,0.55)	(6,0.1)	(8.0.35)	(2,0.55)	(6,0.1)	(8.0.35)	(1,0.45)	(3,0.45)	(4,0.1)
2	(2,0.55)	(6,0.1)	(8.0.35)	(1,0.45)	(3,0.45)	(4,0.1)	(2,0.55)	(6,0.1)	(8.0.35)
3	(2,0.55)	(6,0.1)	(8.0.35)	(1,0.45)	(3,0.45)	(4,0.1)	(2,0.55)	(6,0.1)	(8.0.35)

The expected value of the resources at each iteration of t becomes:

$$E'(W(0)) = \begin{bmatrix} \left(w_{E'(p_1e_1)}, 0\right) & \left(w_{E'(p_1e_2)}, 0\right) & \left(w_{E'(p_1e_3)}, 0\right) & \left(w_{E'(p_1e_4)}, 0\right) \\ \left(w_{E'(p_2e_1)}, 0\right) & \left(w_{E'(p_2e_2)}, 0\right) & \left(w_{E'(p_2e_3)}, 0\right) & \left(w_{E'(p_2e_4)}, 0\right) \\ \left(w_{E'(p_3e_1)}, 0\right) & \left(w_{E'(p_3e_2)}, 0\right) & \left(w_{E'(p_3e_3)}, 0\right) & \left(w_{E'(p_3e_4)}, 0\right) \end{bmatrix}$$

$E'(W(0))$

$$= \begin{bmatrix} 1*.15+3*.45+6*.4 & 2*.35+4*.15+3*.5 & 5*.4+2*.25+1*.35 & 1*.45+3*.45+4*.1 \\ 3*.6+2*.1+1*.3 & 3*.3+6*.25+5*.45 & 2*.3+5*.2+4*.5 & 1*.25+5*.35+3*.4 \\ 4*.25+5*.1+2*.65 & 1*.25+8*.6+3*.15 & 2*.3+10*.5+7*.2 & 2*.55+6*.1+8*.35 \end{bmatrix}$$

$$= \begin{bmatrix} 3.9 & 2.8 & 2.85 & 2.2 \\ 2.3 & 4.65 & 3.6 & 3.2 \\ 2.8 & 5.5 & 7 & 4.5 \end{bmatrix}$$

$$E'(W(1)) = \begin{bmatrix} \left(w_{E'(p_1e_1)},1\right) & \left(w_{E'(p_1e_2)},1\right) & \left(w_{E'(p_1e_3)},1\right) & \left(w_{E'(p_1e_4)},1\right) \\ \left(w_{E'(p_2e_1)},1\right) & \left(w_{E'(p_2e_2)},1\right) & \left(w_{E'(p_2e_3)},1\right) & \left(w_{E'(p_2e_4)},1\right) \\ \left(w_{E'(p_3e_1)},1\right) & \left(w_{E'(p_3e_2)},1\right) & \left(w_{E'(p_3e_3)},1\right) & \left(w_{E'(p_3e_4)},1\right) \end{bmatrix}$$

$E'(W(1))$

$$= \begin{bmatrix} 4*.25+5*.1+2*.65 & 1*.25+8*.6+3*.15 & 2*.3+5*.2+4*.5 & 2*.55+6*.1+8*.35 \\ 3*.6+2*.1+1*.3 & 1*.25+8*.6+3*.15 & 2*.3+10*.5+7*.2 & 2*.55+6*.1+8*.35 \\ 1*.15+3*.45+6*.4 & 2*.35+4*.15+3*.5 & 5*.4+2*.25+1*.35 & 1*.45+3*.45+4*.1 \end{bmatrix}$$

$$= \begin{bmatrix} 2.8 & 5.5 & 3.6 & 4.5 \\ 2.3 & 5.5 & 7 & 4.5 \\ 3.9 & 2.8 & 2.85 & 2.2 \end{bmatrix}$$

$$E'(W(2)) = \begin{bmatrix} \left(w_{E'(p_1e_1)},2\right) & \left(w_{E'(p_1e_2)},2\right) & \left(w_{E'(p_1e_3)},2\right) & \left(w_{E'(p_1e_4)},2\right) \\ \left(w_{E'(p_2e_1)},2\right) & \left(w_{E'(p_2e_2)},2\right) & \left(w_{E'(p_2e_3)},2\right) & \left(w_{E'(p_2e_4)},2\right) \\ \left(w_{E'(p_3e_1)},2\right) & \left(w_{E'(p_3e_2)},2\right) & \left(w_{E'(p_3e_3)},2\right) & \left(w_{E'(p_3e_4)},2\right) \end{bmatrix}$$

$E'(W(2))$

$$= \begin{bmatrix} 3*.6+2*.1+1*.3 & 3*.3+6*.25+5*.45 & 2*.3+10*.5+7*.2 & 2*.55+6*.1+8*.35 \\ 4*.25+5*.1+2*.65 & 2*.35+4*.15+3*.5 & 5*.4+2*.25+1*.35 & 1*.45+3*.45+4*.1 \\ 3*.6+2*.1+1*.3 & 3*.3+6*.25+5*.45 & 2*.3+5*.2+4*.5 & 2*.55+6*.1+8*.35 \end{bmatrix}$$

$$= \begin{bmatrix} 2.3 & 4.65 & 7 & 4.5 \\ 2.8 & 2.8 & 2.85 & 2.2 \\ 2.3 & 4.65 & 3.6 & 4.5 \end{bmatrix}$$

$$E'(W(3)) = \begin{bmatrix} \left(w_{E'(p_1e_1)}, 3\right) & \left(w_{E'(p_1e_2)}, 3\right) & \left(w_{E'(p_1e_3)}, 3\right) & \left(w_{E'(p_1e_4)}, 3\right) \\ \left(w_{E'(p_2e_1)}, 3\right) & \left(w_{E'(p_2e_2)}, 3\right) & \left(w_{E'(p_2e_3)}, 3\right) & \left(w_{E'(p_2e_4)}, 3\right) \\ \left(w_{E'(p_3e_1)}, 3\right) & \left(w_{E'(p_3e_2)}, 3\right) & \left(w_{E'(p_3e_3)}, 3\right) & \left(w_{E'(p_3e_4)}, 3\right) \end{bmatrix}$$

$E'(W(3))$

$$= \begin{bmatrix} 4*.25+5*.1+2*.65 & 1*.25+8*.6+3*.15 & 2*.3+10*.5+7*.2 & 2*.55+6*.1+8*.35 \\ 1*.15+3*.45+6*.4 & 2*.35+4*.15+3*.5 & 5*.4+2*.25+1*.35 & 1*.45+3*.45+4*.1 \\ 4*.25+5*.1+2*.65 & 2*.35+4*.15+3*.5 & 2*.3+5*.2+4*.5 & 2*.55+6*.1+8*.35 \end{bmatrix}$$

$$= \begin{bmatrix} 2.8 & 5.5 & 7 & 4.5 \\ 3.9 & 2.8 & 2.85 & 2.2 \\ 2.8 & 2.8 & 3.6 & 4.5 \end{bmatrix}$$

We also have our conversion set:

$$C = \left[\frac{1}{10}, \frac{1}{7}, \frac{1}{3}\right]$$

Now, by using the formula $W(E) = CE'(R)$:

$$W(E,0) = \left\{ \begin{matrix} \left[\frac{1}{10}*3.9+\frac{1}{7}*2.3+\frac{1}{3}*2.8\right], \left[\frac{1}{10}*2.8+\frac{1}{7}*4.65+\frac{1}{3}*5.5\right], \\ \left[\frac{1}{10}*2.85+\frac{1}{7}*3.6+\frac{1}{3}*7\right], \left[\frac{1}{10}2.2+\frac{1}{7}*3.2+\frac{1}{3}*4.5\right] \end{matrix} \right\}$$

$$W(E,1) = \left\{ \begin{matrix} \left[\frac{1}{10}*2.8+\frac{1}{7}*2.3+\frac{1}{3}*3.9\right], \left[\frac{1}{10}*5.5+\frac{1}{7}*5.5+\frac{1}{3}*2.8\right], \\ \left[\frac{1}{10}*3.6+\frac{1}{7}*7+\frac{1}{3}*2.85\right], \left[\frac{1}{10}4.5+\frac{1}{7}*4.5+\frac{1}{3}*2.2\right] \end{matrix} \right\}$$

$$W(E,2) = \left\{ \begin{matrix} \left[\frac{1}{10}*2.3+\frac{1}{7}*2.8+\frac{1}{3}*2.3\right], \left[\frac{1}{10}*4.65+\frac{1}{7}*2.8+\frac{1}{3}*4.65\right], \\ \left[\frac{1}{10}*7+\frac{1}{7}*2.85+\frac{1}{3}*3.6\right], \left[\frac{1}{10}4.5+\frac{1}{7}*2.2+\frac{1}{3}*4.5\right] \end{matrix} \right\}$$

$$W(E,3) = \left\{ \begin{matrix} \left[\frac{1}{10}*2.8+\frac{1}{7}*3.9+\frac{1}{3}*2.8\right], \left[\frac{1}{10}*5.5+\frac{1}{7}*2.8+\frac{1}{3}*2.8\right], \\ \left[\frac{1}{10}*7+\frac{1}{7}*2.85+\frac{1}{3}*3.6\right], \left[\frac{1}{10}4.5+\frac{1}{7}*2.2+\frac{1}{3}*4.5\right] \end{matrix} \right\}$$

Implying:

$$w(e_1,0)=\left[\frac{3.9}{10}+\frac{2.3}{7}+\frac{2.8}{3}\right], w(e_2,0)=\left[\frac{2.8}{10}+\frac{4.65}{7}+\frac{5.5}{3}\right],$$

$$w(e_3,0)=\left[\frac{2.85}{10}+\frac{3.6}{7}+\frac{7}{3}\right], w(e_4,0)=\left[\frac{2.2}{10}+\frac{3.2}{7}+\frac{4.5}{3}\right]$$

$$W(0)\approx\{1.65,2.78,3.13,2.18\}$$

$$w(e_1,1)=\left[\frac{2.8}{10}+\frac{2.3}{7}+\frac{3.9}{3}\right], w(e_2,1)=\left[\frac{5.5}{10}+\frac{5.5}{7}+\frac{2.8}{3}\right],$$

$$w(e_3,1)=\left[\frac{3.6}{10}+\frac{7}{7}+\frac{2.85}{3}\right], w(e_4,1)=\left[\frac{4.5}{10}+\frac{4.5}{7}+\frac{2.2}{3}\right]$$

$$W(1)\approx\{1.91,2.27,2.31,1.83\}$$

$$w(e_1,2)=\left[\frac{2.3}{10}+\frac{2.8}{7}+\frac{2.3}{3}\right], w(e_2,2)=\left[\frac{4.65}{10}+\frac{2.8}{7}+\frac{4.65}{3}\right],$$

$$w(e_3,2)=\left[\frac{7}{10}+\frac{2.85}{7}+\frac{3.6}{3}\right], w(e_4,2)=\left[\frac{4.5}{10}+\frac{2.2}{7}+\frac{4.5}{3}\right]$$

$$W(2)\approx\{1.4,2.42,2.31,2.26\}$$

$$w(e_1,3)=\left[\frac{2.8}{10}+\frac{3.9}{7}+\frac{2.8}{3}\right], w(e_2,3)=\left[\frac{5.5}{10}+\frac{2.8}{7}+\frac{2.8}{3}\right],$$

$$w(e_3,3)=\left[\frac{7}{10}+\frac{2.85}{7}+\frac{3.6}{3}\right], w(e_4,3)=\left[\frac{4.5}{10}+\frac{2.2}{7}+\frac{4.5}{3}\right]$$

$$W(3)\approx\{1.77,1.88,2.31,2.26\}$$

The expanded search tree used by A* is illustrated as such:

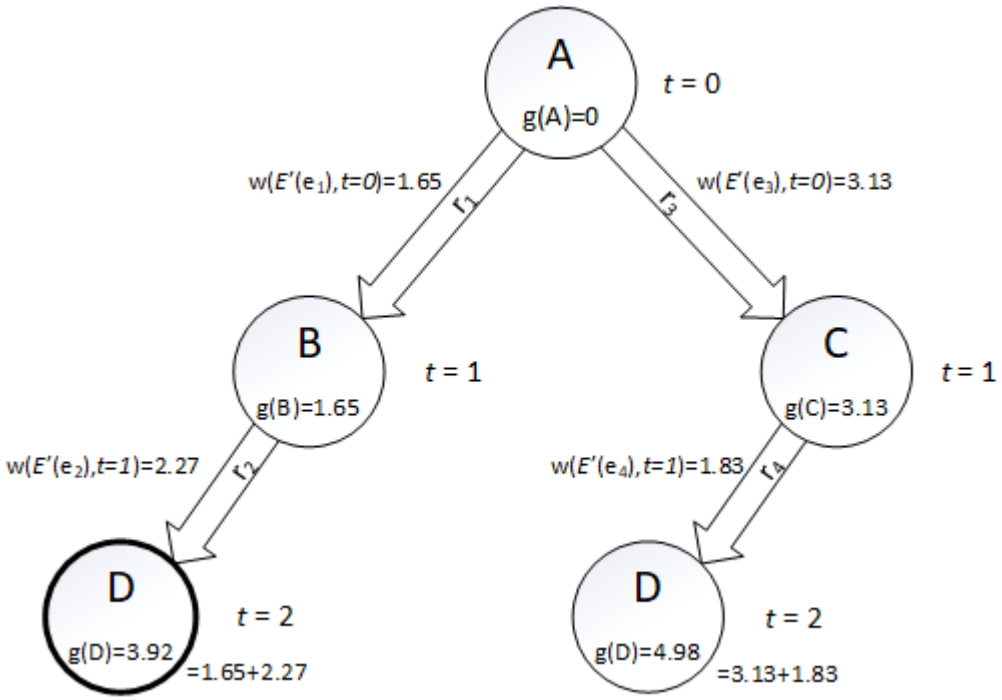


Figure 26: Expanded search tree featuring multiple resources per edge over independent events with known temporal values based on the aforementioned Problem Graph (Figure 25).

3.2.11 Scenario 11: Dynamic Single Resource with Expected Values over Dependent Events and Known Temporal Values

Similar to Scenario 5, events that take place concurrently may have dependencies between them even though they may also have their values altered over time.

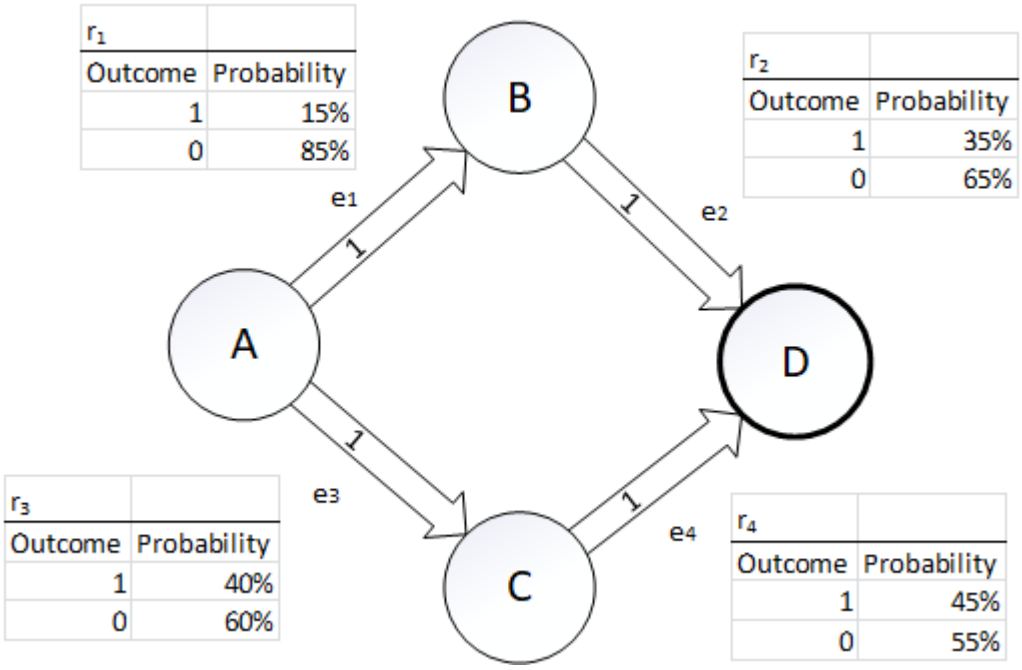


Figure 27: Problem Graph featuring a single resource per edge with dependent events and known temporal values. Each edge has an underlying weight function associated with it, and each resource has a corresponding expected value at a given time *t*. Probabilities are also dependent on previous events that take place along the same path. Resources on the edges are shown at time *t*=0.

We define our sets as follows:

$V = \{A, B, C, D\}$

$E = \{e_1, e_2, e_3, e_4\}$

$W = \{w(e_1), w(e_2), w(e_3), w(e_4)\}$

Suppose the temporal expected values evaluate to the following:

Ev_{r_1}:

t	$p_{r_1 1}$	$p_{r_1 2}$
0	(1,0.15)	(0,0.85)
1	(1,0.5)	(0,0.5)
2	(1,0.7)	(0,0.3)
3	(1,0.45)	(0,0.55)

Ev_{r_2}:

t	$p_{r_2 1}$	$p_{r_2 2}$
0	(1,0.35)	(0,0.65)
1	(1,0.3)	(0,0.7)
2	(1,0.8)	(0,0.2)
3	(1,0.6)	(0,0.4)

Ev_{r_3}:

t	$p_{r_3 1}$	$p_{r_3 2}$
0	(1,0.4)	(0,0.6)
1	(1,0.4)	(0,0.6)
2	(1,0.8)	(0,0.2)
3	(1,0.15)	(0,0.85)

Ev_{r_4}:

t	$p_{r_4 1}$	$p_{r_4 2}$
0	(1,0.45)	(0,0.55)
1	(1,0.5)	(0,0.5)
2	(1,0.6)	(0,0.4)
3	(1,0.3)	(0,0.7)

For dependent events, expected values are dependent on the success of previous events.

The expected values for each edge-weight will be:

$$W(0) = \{(w_{E'(e_1)}, 0), (w_{E'(e_2)}, 0), (w_{E'(e_3)}, 0), (w_{E'(e_4)}, 0)\}$$

$$= \{1 * .15, 1 * .35(1 - .15), 1 * .4, 1 * .45(1 - .4)\} = \{0.15, 0.3, 0.4, 0.27\}$$

$$W(1) = \{(w_{E'(e_1)}, 1), (w_{E'(e_2)}, 1), (w_{E'(e_3)}, 1), (w_{E'(e_4)}, 1)\}$$

$$= \{1 * .5, 1 * .3(1 - .5), 1 * .4, 1 * .5(1 - .4)\} = \{0.5, 0.15, 0.4, 0.3\}$$

$$W(2) = \{(w_{E'(e_1)}, 2), (w_{E'(e_2)}, 2), (w_{E'(e_3)}, 2), (w_{E'(e_4)}, 2)\}$$

$$= \{1 * .7, 1 * .8(1 - .7), 1 * .4, 1 * .6(1 - .4)\} = \{0.7, 0.24, 0.4, 0.36\}$$

$$W(3) = \{(w_{E'(e_1)}, 3), (w_{E'(e_2)}, 3), (w_{E'(e_3)}, 3), (w_{E'(e_4)}, 3)\}$$

$$= \{1 * .45, 1 * .6(1 - .45), 1 * .4, 1 * .3(1 - .4)\} = \{0.45, 0.33, 0.4, 0.18\}$$

The expanded search tree used by A* is illustrated as such:

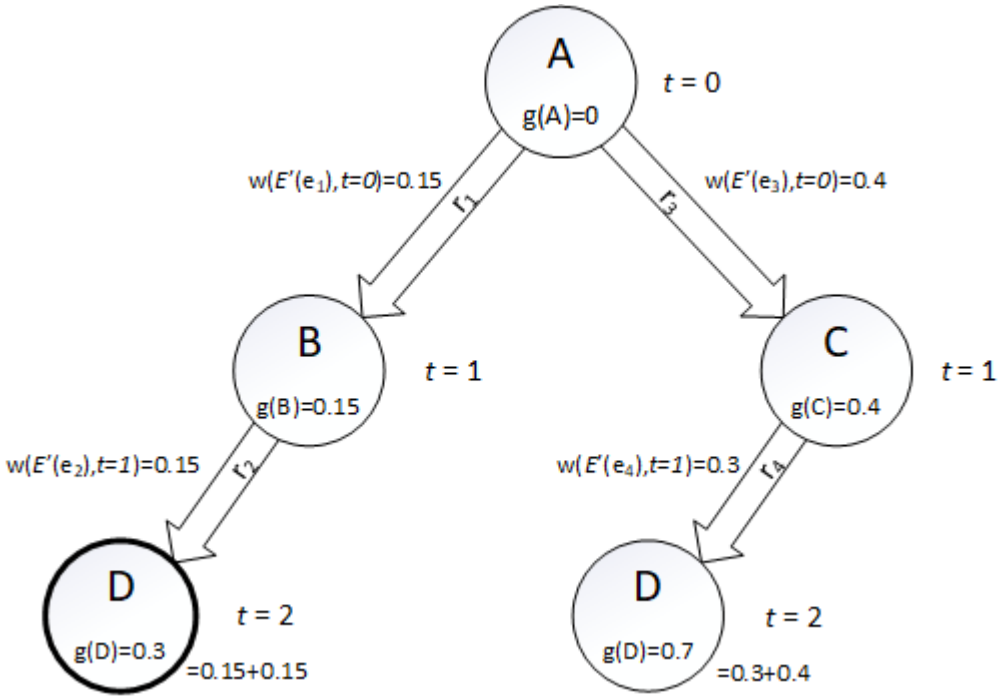
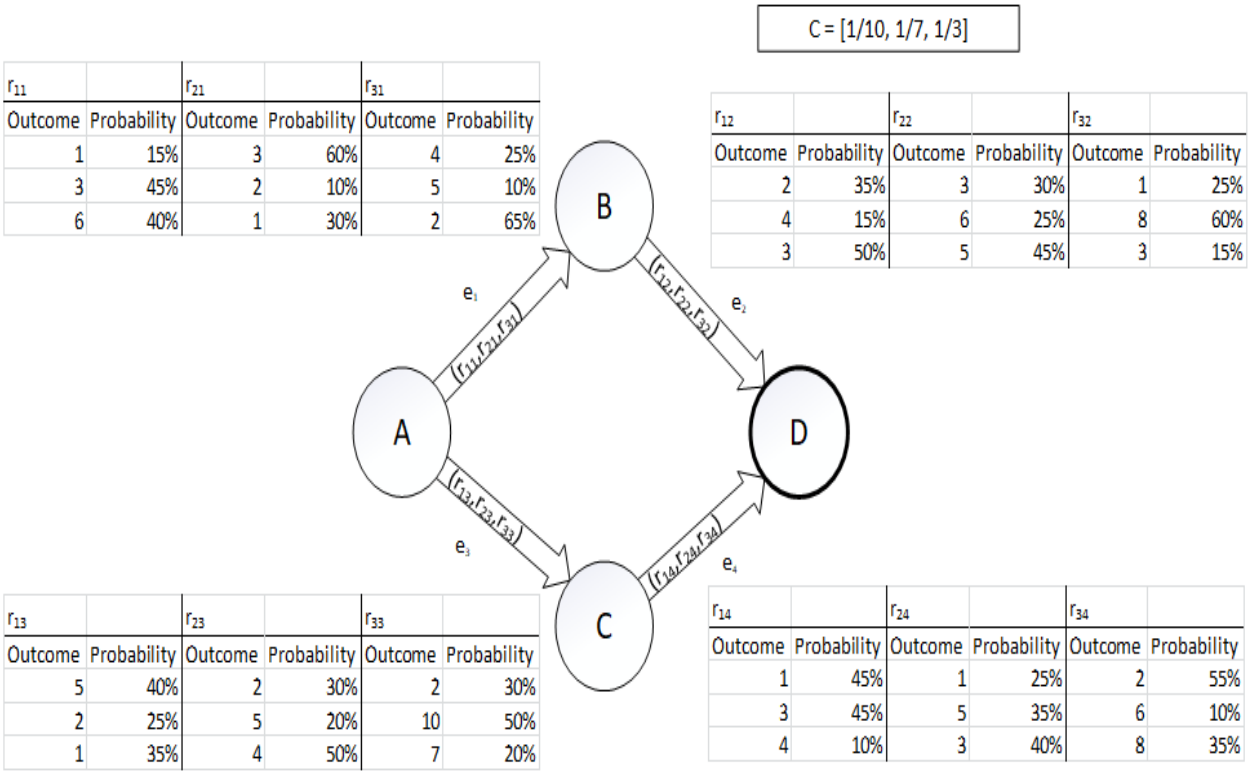


Figure 28: Expanded search tree featuring a single resource per edge over dependents events with known temporal values based on the aforementioned Problem Graph (Figure 27).

3.2.12 Scenario 12: Dynamic Multiple Resources with Expected Values over Dependent Events and Known Temporal Values

For this solution, the same approach utilizing dependent events in previous scenarios is used.



r_{11}		r_{21}		r_{31}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	15%	3	60%	4	25%
3	45%	2	10%	5	10%
6	40%	1	30%	2	65%

r_{12}		r_{22}		r_{32}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
2	35%	3	30%	1	25%
4	15%	6	25%	8	60%
3	50%	5	45%	3	15%

r_{13}		r_{23}		r_{33}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
5	40%	2	30%	2	30%
2	25%	5	20%	10	50%
1	35%	4	50%	7	20%

r_{14}		r_{24}		r_{34}	
Outcome	Probability	Outcome	Probability	Outcome	Probability
1	45%	1	25%	2	55%
3	45%	5	35%	6	10%
4	10%	3	40%	8	35%

Figure 29: Problem Graph featuring multiple resources per edge with dependent events and known temporal values. Each edge has an underlying weight function associated with it, and each resource has a corresponding expected value at a given time *t*. Probabilities are also dependent on previous events that take place along the same path. The conversion rate matrix is applied to resources for cost calculation. Resources on the edges are shown at time *t* =0.

Suppose the temporal expected values evaluate to the following:

Ev_{e_1}:

t	$p_{e_1r_11}$	$p_{e_1r_12}$	$p_{e_1r_13}$	$p_{e_1r_21}$	$p_{e_1r_22}$	$p_{e_1r_23}$	$p_{e_1r_11}$	$p_{e_1r_32}$	$p_{e_1r_33}$
0	(1,0.15)	(3,0.45)	(6,0.4)	(3,0.6)	(2,0.1)	(1,0.3)	(4,0.25)	(5,0.1)	(2.0.65)
1	(4,0.25)	(5,0.1)	(2.0.65)	(3,0.6)	(2,0.1)	(1,0.3)	(1,0.15)	(3,0.45)	(6,0.4)
2	(3,0.6)	(2,0.1)	(1,0.3)	(4,0.25)	(5,0.1)	(2.0.65)	(3,0.6)	(2,0.1)	(1,0.3)
3	(4,0.25)	(5,0.1)	(2.0.65)	(1,0.15)	(3,0.45)	(6,0.4)	(4,0.25)	(5,0.1)	(2.0.65)

Ev_{e_2}:

t	$p_{e_2r_11}$	$p_{e_2r_12}$	$p_{e_2r_13}$	$p_{e_2r_21}$	$p_{e_2r_22}$	$p_{e_2r_23}$	$p_{e_2r_11}$	$p_{e_2r_32}$	$p_{e_2r_33}$
0	(2,0.35)	(4,0.15)	(3,0.5)	(3,0.3)	(6,0.25)	(5,0.45)	(1,0.25)	(8,0.6)	(3.0.15)
1	(1,0.25)	(8,0.6)	(3.0.15)	(1,0.25)	(8,0.6)	(3.0.15)	(2,0.35)	(4,0.15)	(3,0.5)
2	(3,0.3)	(6,0.25)	(5,0.45)	(2,0.35)	(4,0.15)	(3,0.5)	(3,0.3)	(6,0.25)	(5,0.45)
3	(1,0.25)	(8,0.6)	(3.0.15)	(2,0.35)	(4,0.15)	(3,0.5)	(2,0.35)	(4,0.15)	(3,0.5)

Ev_{e_3}:

t	$p_{e_3r_11}$	$p_{e_3r_12}$	$p_{e_3r_13}$	$p_{e_3r_21}$	$p_{e_3r_22}$	$p_{e_3r_23}$	$p_{e_3r_11}$	$p_{e_3r_32}$	$p_{e_3r_33}$
0	(5,0.4)	(2,0.25)	(1,0.35)	(2,0.3)	(5,0.2)	(4,0.5)	(2,0.3)	(10,0.5)	(7.0.2)
1	(2,0.3)	(5,0.2)	(4,0.5)	(2,0.3)	(10,0.5)	(7.0.2)	(5,0.4)	(2,0.25)	(1,0.35)
2	(2,0.3)	(10,0.5)	(7.0.2)	(5,0.4)	(2,0.25)	(1,0.35)	(2,0.3)	(5,0.2)	(4,0.5)
3	(2,0.3)	(10,0.5)	(7.0.2)	(5,0.4)	(2,0.25)	(1,0.35)	(2,0.3)	(5,0.2)	(4,0.5)

Ev_{e_4}:

t	$p_{e_4r_11}$	$p_{e_4r_12}$	$p_{e_4r_13}$	$p_{e_4r_21}$	$p_{e_4r_22}$	$p_{e_4r_23}$	$p_{e_4r_11}$	$p_{e_4r_32}$	$p_{e_4r_33}$
0	(1,0.45)	(3,0.45)	(4,0.1)	(1,0.25)	(5,0.35)	(3,0.4)	(2,0.55)	(6,0.1)	(8.0.35)
1	(2,0.55)	(6,0.1)	(8.0.35)	(2,0.55)	(6,0.1)	(8.0.35)	(1,0.45)	(3,0.45)	(4,0.1)
2	(2,0.55)	(6,0.1)	(8.0.35)	(1,0.45)	(3,0.45)	(4,0.1)	(2,0.55)	(6,0.1)	(8.0.35)
3	(2,0.55)	(6,0.1)	(8.0.35)	(1,0.45)	(3,0.45)	(4,0.1)	(2,0.55)	(6,0.1)	(8.0.35)

The expected value of the resources at each iteration of *t* becomes:

$$E'(W(0)) = \begin{bmatrix} (w_{E'(p_1e_1)}, 0) & (w_{E'(p_1e_2)} * (1 - p_1e_1), 0) & (w_{E'(p_1e_3)}, 0) & (w_{E'(p_1e_4)} * (1 - p_1e_3), 0) \\ (w_{E'(p_2e_1)}, 0) & (w_{E'(p_2e_2)} * (1 - p_2e_1), 0) & (w_{E'(p_2e_3)}, 0) & (w_{E'(p_2e_4)} * (1 - p_2e_3), 0) \\ (w_{E'(p_3e_1)}, 0) & (w_{E'(p_3e_2)} * (1 - p_3e_1), 0) & (w_{E'(p_3e_3)}, 0) & (w_{E'(p_3e_4)} * (1 - p_3e_3), 0) \end{bmatrix}$$

$(w_{E'(p_1e_1)}, 0) = 1 * .15 + 3 * .45 + 6 * .4$

$(w_{E'(p_2e_1)}, 0) = 3 * .6 + 2 * .1 + 1 * .3$

$(w_{E'(p_3e_1)}, 0) = 4 * .25 + 5 * .1 + 2 * .65$

$(w_{E'(p_1e_2)} * (1 - p_1e_1), 0) = 2 * .35 * (1 - .15) + 4 * .15 * (1 - .45) + 3 * .5 * (1 - .4)$

$(w_{E'(p_2e_2)} * (1 - p_2e_1), 0) = 3 * .3 * (1 - .6) + 6 * .25 * (1 - .1) + 5 * .45 * (1 - .3)$

$(w_{E'(p_3e_2)} * (1 - p_3e_1), 0) = 1 * .25 * (1 - .25) + 8 * .6 * (1 - .1) + 3 * .15 * (1 - .65)$

$(w_{E'(p_1e_3)}, 0) = 5 * .4 + 2 * .25 + 1 * .35$

$(w_{E'(p_2e_3)}, 0) = 2 * .3 + 5 * .2 + 4 * .5$

$(w_{E'(p_3e_3)}, 0) = 2 * .3 + 10 * .5 + 7 * .2$

$(w_{E'(p_1e_4)} * (1 - p_1e_3), 0) = 1 * .45 * (1 - .4) + 3 * .45 * (1 - .25) + 4 * .1 * (1 - .35)$

$(w_{E'(p_2e_4)} * (1 - p_2e_3), 0) = 1 * .25 * (1 - .3) + 5 * .35 * (1 - .2) + 3 * .4 * (1 - .5)$

$(w_{E'(p_3e_4)} * (1 - p_3e_3), 0) = 2 * .55 * (1 - .3) + 6 * .1 * (1 - .5) + 8 * .35 * (1 - .2)$

$$E'(W(0)) = \begin{bmatrix} 3.9 & 1.83 & 2.85 & 1.54 \\ 2.3 & 3.29 & 3.6 & 2.18 \\ 2.8 & 4.67 & 7 & 3.31 \end{bmatrix}$$

$$E'(W(1)) = \begin{bmatrix} (w_{E'(p_1e_1)}, 1) & (w_{E'(p_1e_2)} * (1 - p_1e_1), 1) & (w_{E'(p_1e_3)}, 1) & (w_{E'(p_1e_4)} * (1 - p_1e_3), 1) \\ (w_{E'(p_2e_1)}, 1) & (w_{E'(p_2e_2)} * (1 - p_2e_1), 1) & (w_{E'(p_2e_3)}, 1) & (w_{E'(p_2e_4)} * (1 - p_2e_3), 1) \\ (w_{E'(p_3e_1)}, 1) & (w_{E'(p_3e_2)} * (1 - p_3e_1), 1) & (w_{E'(p_3e_3)}, 1) & (w_{E'(p_3e_4)} * (1 - p_3e_3), 1) \end{bmatrix}$$

$(w_{E'(p_1e_1)}, 1) = 4 * .25 + 5 * .1 + 2 * .65$

$(w_{E'(p_2e_1)}, 1) = 3 * .6 + 2 * .1 + 1 * .3$

$(w_{E'(p_3e_1)}, 1) = 1 * .15 + 3 * .45 + 6 * .4$

$(w_{E'(p_1e_2)} * (1 - p_1e_1), 1) = 1 * .25 * (1 - .25) + 8 * .6 * (1 - .1) + 3 * .15 * (1 - .65)$

$(w_{E'(p_2e_2)} * (1 - p_2e_1), 1) = 1 * .25 * (1 - .6) + 8 * .6 * (1 - .1) + 3 * .15 * (1 - .3)$

$$\left(w_{E'(p_3e_2)} * (1 - p_3e_1), 1\right) = 2 * .35 * (1 - .15) + 4 * .15 * (1 - .45) + 3 * .5 * (1 - .4)$$

$$\left(w_{E'(p_1e_3)}, 1\right) = 2 * .3 + 5 * .2 + 4 * .5$$

$$\left(w_{E'(p_2e_3)}, 1\right) = 2 * .3 + 10 * .5 + 7 * .2$$

$$\left(w_{E'(p_3e_3)}, 1\right) = 5 * .4 + 2 * .25 + 1 * .35$$

$$\left(w_{E'(p_1e_4)} * (1 - p_1e_3), 1\right) = 2 * .55 * (1 - .3) + 6 * .1 * (1 - .2) + 8 * .35 * (1 - .5)$$

$$\left(w_{E'(p_2e_4)} * (1 - p_2e_3), 1\right) = 2 * .55 * (1 - .3) + 6 * .1 * (1 - .5) + 8 * .35 * (1 - .2)$$

$$\left(w_{E'(p_3e_4)} * (1 - p_3e_3), 1\right) = 1 * .45 * (1 - .4) + 3 * .45 * (1 - .25) + 4 * .1 * (1 - .35)$$

$$E'(W(1)) = \begin{bmatrix} 2.8 & 4.67 & 3.6 & 2.65 \\ 2.3 & 4.74 & 7 & 3.31 \\ 3.9 & 1.83 & 2.85 & 1.54 \end{bmatrix}$$

$$E'(W(2)) = \begin{bmatrix} \left(w_{E'(p_1e_1)}, 2\right) & \left(w_{E'(p_1e_2)} * (1 - p_1e_1), 2\right) & \left(w_{E'(p_1e_3)}, 2\right) & \left(w_{E'(p_1e_4)} * (1 - p_1e_3), 2\right) \\ \left(w_{E'(p_2e_1)}, 2\right) & \left(w_{E'(p_2e_2)} * (1 - p_2e_1), 2\right) & \left(w_{E'(p_2e_3)}, 2\right) & \left(w_{E'(p_2e_4)} * (1 - p_2e_3), 2\right) \\ \left(w_{E'(p_3e_1)}, 2\right) & \left(w_{E'(p_3e_2)} * (1 - p_3e_1), 2\right) & \left(w_{E'(p_3e_3)}, 2\right) & \left(w_{E'(p_3e_4)} * (1 - p_3e_3), 2\right) \end{bmatrix}$$

$$\left(w_{E'(p_1e_1)}, 2\right) = 3 * .6 + 2 * .1 + 1 * .3$$

$$\left(w_{E'(p_2e_1)}, 2\right) = 4 * .25 + 5 * .1 + 2 * .65$$

$$\left(w_{E'(p_3e_1)}, 2\right) = 3 * .6 + 2 * .1 + 1 * .3$$

$$\left(w_{E'(p_1e_2)} * (1 - p_1e_1), 2\right) = 3 * .3 * (1 - .6) + 6 * \ .25 * (1 - .1) + 5 * .45 * (1 - .3)$$

$$\left(w_{E'(p_2e_2)} * (1 - p_2e_1), 2\right) = 2 * .35 * (1 - .25) + 4 * .15 * (1 - .1) + 3 * .5 * (1 - .65)$$

$$\left(w_{E'(p_3e_2)} * (1 - p_3e_1), 2\right) = 3 * .3 * (1 - .6) + 6 * .25 * (1 - .1) + 5 * .45 * (1 - .3)$$

$$\left(w_{E'(p_1e_3)}, 2\right) = 2 * .3 + 10 * .5 + 7 * .2$$

$$\left(w_{E'(p_2e_3)}, 2\right) = 5 * .4 + 2 * .25 + 1 * .35$$

$$\left(w_{E'(p_3e_3)}, 2\right) = 2 * .3 + 5 * .2 + 4 * .5$$

$$\left(w_{E'(p_1e_4)} * (1 - p_1e_3), 2\right) = 2 * .55 * (1 - .3) + 6 * .1 * (1 - .5) + 8 * .35 * (1 - .2)$$

$$\left(w_{E'(p_2e_4)} * (1 - p_2e_3), 2\right) = 1 * .45 * (1 - .4) + 3 * .45 * (1 - .25) + 4 * .1 * (1 - .35)$$

$$\left(w_{E'(p_3e_4)} * (1 - p_3e_3), 2\right) = 2 * .55 * (1 - .3) + 6 * .1 * (1 - .2) + 8 * .35 * (1 - .5)$$

$$E'(W(2)) = \begin{bmatrix} 2.3 & 3.29 & 7 & 3.31 \\ 2.8 & 1.59 & 2.85 & 1.54 \\ 2.3 & 3.29 & 3.6 & 2.65 \end{bmatrix}$$

$$E'(W(3)) = \begin{bmatrix} (w_{E'(p_1e_1)}, 3) & (w_{E'(p_1e_2)}(1 - p_1e_1), 3) & (w_{E'(p_1e_3)}, 3) & (w_{E'(p_1e_4)}(1 - p_1e_3), 3) \\ (w_{E'(p_2e_1)}, 3) & (w_{E'(p_2e_2)}(1 - p_2e_1), 3) & (w_{E'(p_2e_3)}, 3) & (w_{E'(p_2e_4)}(1 - p_2e_3), 3) \\ (w_{E'(p_3e_1)}, 3) & (w_{E'(p_3e_2)}(1 - p_3e_1), 3) & (w_{E'(p_3e_3)}, 3) & (w_{E'(p_3e_4)}(1 - p_2e_3), 3) \end{bmatrix}$$

$(w_{E'(p_1e_1)}, 3) = 4 * .25 + 5 * .1 + 2 * .65$

$(w_{E'(p_2e_1)}, 3) = 1 * .15 + 3 * .45 + 6 * .4$

$(w_{E'(p_3e_1)}, 3) = 4 * .25 + 5 * .1 + 2 * .65$

$(w_{E'(p_1e_2)} * (1 - p_1e_1), 3) = 1 * .25 * (1 - .25) + 8 * .6 * (1 - .1) + 3 * .15 * (1 - .65)$

$(w_{E'(p_2e_2)} * (1 - p_2e_1), 3) = 2 * .35 * (1 - .15) + 4 * .15 * (1 - .45) + 3 * .5 * (1 - .4)$

$(w_{E'(p_3e_2)} * (1 - p_3e_1), 3) = 2 * .35 * (1 - .25) + 4 * .15 * (1 - .1) + 3 * .5 * (1 - .65)$

$(w_{E'(p_1e_3)}, 3) = 2 * .3 + 10 * .5 + 7 * .2$

$(w_{E'(p_2e_3)}, 3) = 5 * .4 + 2 * .25 + 1 * .35$

$(w_{E'(p_3e_3)}, 3) = 2 * .3 + 5 * .2 + 4 * .5$

$(w_{E'(p_1e_4)} * (1 - p_1e_3), 3) = 2 * .55 * (1 - .3) + 6 * .1 * (1 - .5) + 8 * .35 * (1 - .2)$

$(w_{E'(p_2e_4)} * (1 - p_2e_3), 3) = 1 * .45 * (1 - .4) + 3 * .45 * (1 - .25) + 4 * .1 * (1 - .35)$

$(w_{E'(p_3e_4)} * (1 - p_3e_3), 3) = 2 * .55 * (1 - .3) + 6 * .1 * (1 - .2) + 8 * .35 * (1 - .5)$

$$E'(W(3)) = \begin{bmatrix} 2.8 & 4.67 & 7 & 3.31 \\ 3.9 & 1.83 & 2.85 & 1.54 \\ 2.8 & 1.59 & 3.6 & 2.65 \end{bmatrix}$$

We also have our conversion set:

$$C = \left[\frac{1}{10}, \frac{1}{7}, \frac{1}{3}\right]$$

Now, by using the formula $W(E) = CE'(R)$:

$$W(E,0)=\begin{Bmatrix}\left[\frac{1}{10}*3.9+\frac{1}{7}*2.3+\frac{1}{3}*2.8\right],\left[\frac{1}{10}*1.83+\frac{1}{7}*3.29+\frac{1}{3}*4.67\right],\\ \left[\frac{1}{10}*2.85+\frac{1}{7}*3.6+\frac{1}{3}*7\right],\left[\frac{1}{10}1.54+\frac{1}{7}*2.18+\frac{1}{3}*3.31\right]\end{Bmatrix}$$

$$W(E,1)=\begin{Bmatrix}\left[\frac{1}{10}*2.8+\frac{1}{7}*2.3+\frac{1}{3}*3.9\right],\left[\frac{1}{10}*4.67+\frac{1}{7}*4.74+\frac{1}{3}*1.83\right],\\ \left[\frac{1}{10}*3.6+\frac{1}{7}*7+\frac{1}{3}*2.85\right],\left[\frac{1}{10}2.65+\frac{1}{7}*3.31+\frac{1}{3}*1.54\right]\end{Bmatrix}$$

$$W(E,2)=\begin{Bmatrix}\left[\frac{1}{10}*2.3+\frac{1}{7}*2.8+\frac{1}{3}*2.3\right],\left[\frac{1}{10}*3.29+\frac{1}{7}*1.59+\frac{1}{3}*3.29\right],\\ \left[\frac{1}{10}*7+\frac{1}{7}*2.85+\frac{1}{3}*3.6\right],\left[\frac{1}{10}3.31+\frac{1}{7}*1.54+\frac{1}{3}*2.65\right]\end{Bmatrix}$$

$$W(E,3)=\begin{Bmatrix}\left[\frac{1}{10}*2.8+\frac{1}{7}*3.9+\frac{1}{3}*2.8\right],\left[\frac{1}{10}*4.67+\frac{1}{7}*1.83+\frac{1}{3}*1.59\right],\\ \left[\frac{1}{10}*7+\frac{1}{7}*2.85+\frac{1}{3}*3.6\right],\left[\frac{1}{10}3.31+\frac{1}{7}*1.54+\frac{1}{3}*2.65\right]\end{Bmatrix}$$

Implying:

$$w(e_1,0)=\left[\frac{3.9}{10}+\frac{2.3}{7}+\frac{2.8}{3}\right],w(e_2,0)=\left[\frac{1.83}{10}+\frac{3.29}{7}+\frac{4.67}{3}\right],$$

$$w(e_3,0)=\left[\frac{2.85}{10}+\frac{3.6}{7}+\frac{7}{3}\right],w(e_4,0)=\left[\frac{1.54}{10}+\frac{2.18}{7}+\frac{3.31}{3}\right]$$

$$W(0)\approx\{1.65,2.21,3.13,1.57\}$$

$$w(e_1,1)=\left[\frac{2.8}{10}+\frac{2.3}{7}+\frac{3.9}{3}\right],w(e_2,1)=\left[\frac{4.67}{10}+\frac{4.74}{7}+\frac{1.83}{3}\right],$$

$$w(e_3,1)=\left[\frac{3.6}{10}+\frac{7}{7}+\frac{2.85}{3}\right],w(e_4,1)=\left[\frac{2.65}{10}+\frac{3.31}{7}+\frac{1.54}{3}\right]$$

$$W(1)\approx\{1.91,1.75,2.31,1.25\}$$

$$w(e_1,2)=\left[\frac{2.3}{10}+\frac{2.8}{7}+\frac{2.3}{3}\right],w(e_2,2)=\left[\frac{3.29}{10}+\frac{1.59}{7}+\frac{3.29}{3}\right],$$

$$w(e_3, 2) = \left[\frac{7}{10} + \frac{2.85}{7} + \frac{3.6}{3}\right], w(e_4, 2) = \left[\frac{3.31}{10} + \frac{1.54}{7} + \frac{2.65}{3}\right]$$

$$W(2) \approx \{1.4, 1.65, 2.31, 1.43\}$$

$$w(e_1, 3) = \left[\frac{2.8}{10} + \frac{3.9}{7} + \frac{2.8}{3}\right], w(e_2, 3) = \left[\frac{4.67}{10} + \frac{1.83}{7} + \frac{1.59}{3}\right],$$

$$w(e_3, 3) = \left[\frac{7}{10} + \frac{2.85}{7} + \frac{3.6}{3}\right], w(e_4, 3) = \left[\frac{3.31}{10} + \frac{1.54}{7} + \frac{2.65}{3}\right]$$

$$W(3) \approx \{1.77, 1.26, 2.31, 1.43\}$$

The expanded search tree used by A* is illustrated as such:

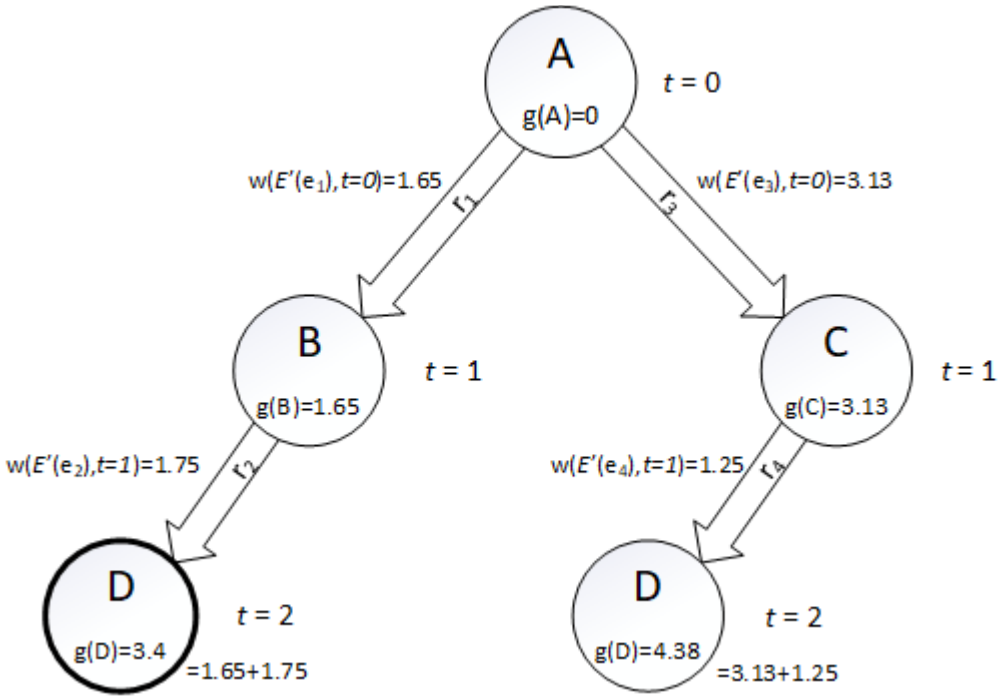


Figure 30: Expanded search tree is featuring multiple resources per edge over dependent events with known temporal values based on the aforementioned Problem Graph (Figure 29).

In short, very few changes need to be made to the original A* Search algorithm to extend its usability across the various problem types described in the aforementioned scenarios. Due to the majority of the changes requiring execution in a pre-processing phase, the time complexity of A* Search is not affected. Furthermore, implementation of this solution may be regarded as a

separate component relative to the actual pathfinding process, as most of the processing must take place before the A* Search algorithm is run. Any modifications to operations (such as the modified weight formula in temporal scenarios) that take place within the algorithm are evaluated in constant time, again, reiterating that performance is not affected when compared to the original A* Search algorithm.

We present a general formula that can be applied to all aforementioned scenarios here:

$$W(t) = CE'\big(R(t)\big)$$

Where:

$$t \rightarrow step\ of\ time\ interval$$

$$C \rightarrow 1\ by\ r\ conversion\ matrix, where\ r\ is\ the\ number\ of\ unique\ resources$$

$$E' \rightarrow expected\ value\ of\ the\ input\ resource\ set$$

$$R \rightarrow r\ by\ |E|\ matrix, where\ r\ is\ the\ number\ of\ unique\ resources, E\ is\ the\ set\ of\ edges$$

Each element can be removed or substituted for a constant depending on which scenario is being handled. All cases will result in a finalized weight value for each edge in the graph. The solution at its core functions similarly to Dijkstra's Algorithm, as each weight becomes quantitative and can thus be used by traditional means. Moreover, our solution possesses a preprocessing phase that manipulates the search map to correspond with traditional pathfinding solutions. By acting as a process that takes place before path planning, our solution can extend the usability of traditional pathfinding solutions to incorporate other properties of the environment they could not detect before.

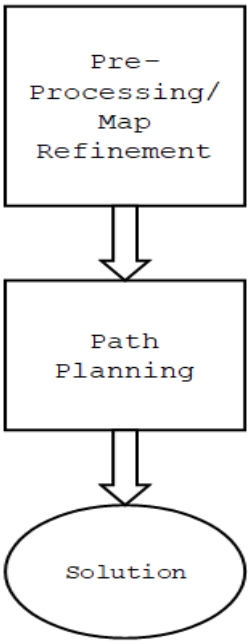


Figure 31: To account for the described scenarios, a preprocessing phase must take place before any actual path planning can commence.

3.3 Performance Analysis

Regarding performance, Dijkstra's Algorithm (and by extension, A* Search) are known to have $O(b^m)$ (or, $O(|E|)$) time complexity, where m is the maximum depth of the search space, and b is the maximum branching factor of the search space [1]. With an admissible heuristic, A* Search is complete and guaranteed to find an optimal path provided one exists, and thus remains at least as efficient as Dijkstra's Algorithm. The average case can always be less than this depending on the admissibility of heuristic that is used. In cases where a Manhattan distance is used as the heuristic, A* Search retains a worst-case complexity of $O(b^m)$, meaning that every edge in the graph is visited once before a solution is realized. This heuristic is applicable to some of the aforementioned scenarios where distance is treated as a resource. In cases where distance is not a resource or where edge-weights contain multiple resources, however, an alternative heuristic must be applied, as that sort of heuristic grants an advantage to distance-based

resources over others [4] [5]. A heuristic that satisfies the requirements of admissibility and consistency is indeed possible for all of the problem types.

3.4 Experimental Setup

The testing environment was developed using C# and the .NET Framework and uses zero resources from outside sources other than the map data that was provided. The application is designed to simulate single-agent pathfinding in environments with varying sizes, which can be configured by the user at runtime. Several features have been included for experimental convenience and performance analysis. Some of these features include custom map creation, optional diagonally linked edges, and real-time nodal updates that can be monitored through the interface. The main feature that was essential to the experimentation process includes a tracker on the number of nodes that were expanded during execution, in addition to the number of operations performed by each algorithm. By our definition, the number of operations pertains to the number of iterations the algorithms uses to expand nodes and update the *g*-values. Some specifications pertaining to the hardware that the tests were housed on can be seen in the following table:

Application	Desktop
Framework	.NET, C#
Processor	Intel Quad Core CPU Q8200 @ 2.33GHz
RAM	8GB
External Sources	Map files (.mapp)

The testing environment currently supports the four pathfinding algorithms that are relevant to this thesis: Dijkstra's Algorithm, A* Search, Strategic Pathfinding, and Multi-Objective

Pathfinding (our proposed solution). The application also accepts map data provided by the Pathfinding Benchmarks that have been made available online for research purposes, specifically data taken from the video game, *Dragon Age: Origins* (2009). The user has the ability to denote the start and goal nodes by using the provided controls, as well as the option to set obstacles, resources, and outcomes with random intensity in the environment. A preference value (or resource-prioritization value) for the agent can also be set in cases where a strategic method is used.

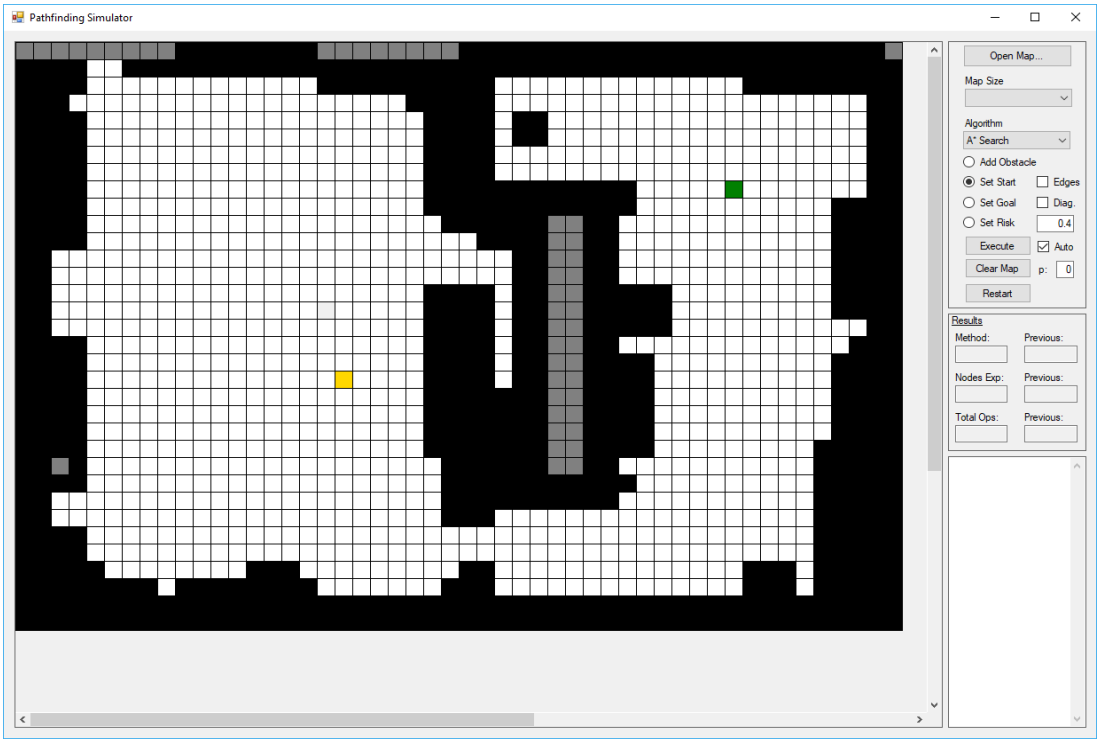


Figure 32: The developed test framework used for experimentation supports up to three algorithms inspired by traditional pathfinding solutions. Path processing can be viewed in real time, although this cuts down on overall performance.

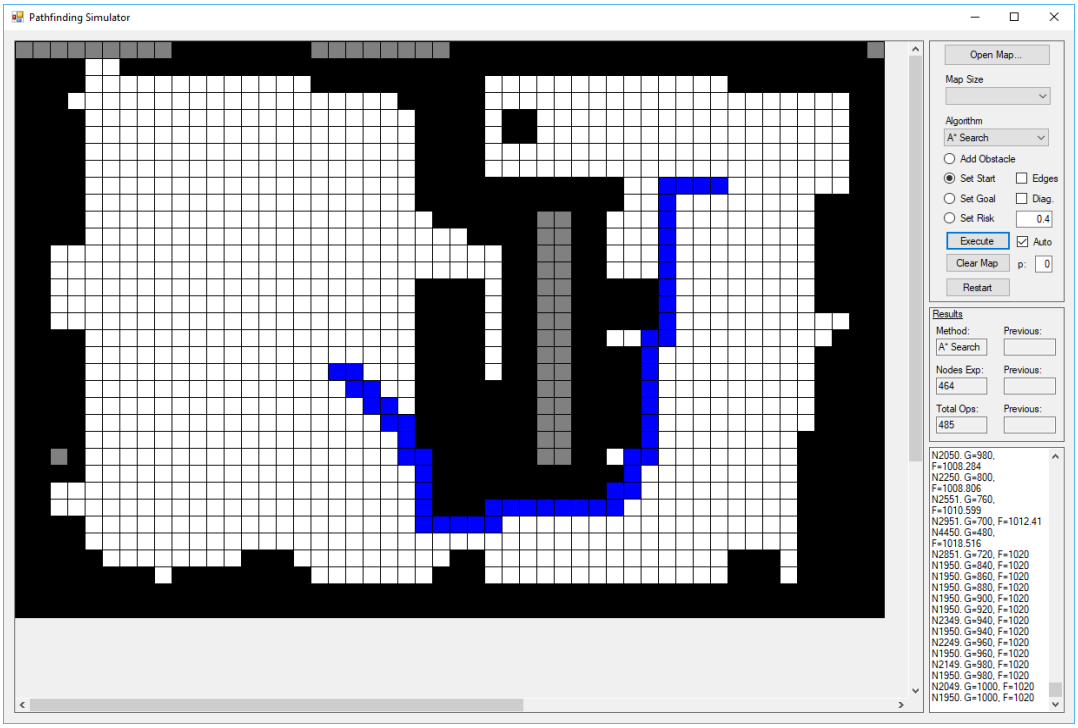


Figure 33: Demonstration of A* Search in the framework. Results are displayed at the end of the test within the sidebars of the window.

Resource values are randomly generated for edge-weights or included in map-data depending on the type of nodes being inputted into the system. Conversion rates can be randomly generated in addition to being "rerolled" if variance across multiple tests is desired. An option for utilizing temporal cases is also included. It should be noted, however, that if an algorithm is executed that does not take into account multiple resources or outcomes, the default values that have been set will be ignored (i.e. each edge-weight will maintain a default value of 1). This provides some form of leniency between tests, as attributes can be denoted without obscuring methods that do not utilize them. Obstacles in the environment play the same role regardless of the algorithm being run, thus if multiple algorithms should be run in sequence, the user does not have to reset the map and replace the nodal data. An option has been provided that allows the user to clear the map of any found paths and reset the nodes to their initial *g*-values through the use of the *Clear Map* button. Additionally, a more rigorous resetter can be found in the form of the *Reset* button,

which reloads the environment to its initial state. Features that provide a better user-experience and not necessarily for testing purposes include an option to make edges visible, the ability to enable diagonal edges, and statistics from the previous test on screen to be compared to the current one.

For monitoring purposes, the program will default to showing each step of iteration during execution. This includes expanding neighbouring nodes originating from the designated starting position, and continually branches out depending on the method that has been chosen for testing. Nodes that have been added to the open list but have not been visited will be a light shade of green on the interface, while nodes that have been visited and considered for the potential solution will be shaded blue. This real-time visual aid has the benefit of allowing the user to observe the algorithm and the decisions it makes at the cost of being slightly sluggish overall. This slower behaviour isn't specific to any one algorithm, but is due to the limitations of the software, as the visuals are drawn using the `OnPaint()` method that is native to the `System.Drawing` library. Calling this method for each update to the panel can be somewhat expensive and results in slower feedback to the viewer and not necessarily from the running algorithm. An option is provided that disables any updates happening in real-time, which saves the time it takes for the screen to render. If this *Auto* option is selected, the algorithms will run without sending updates to the viewer and will present the final results to the screen after path processing has been completed. The finalized path will still be drawn to conclude the simulation, and the data will also be presented in the textboxes.

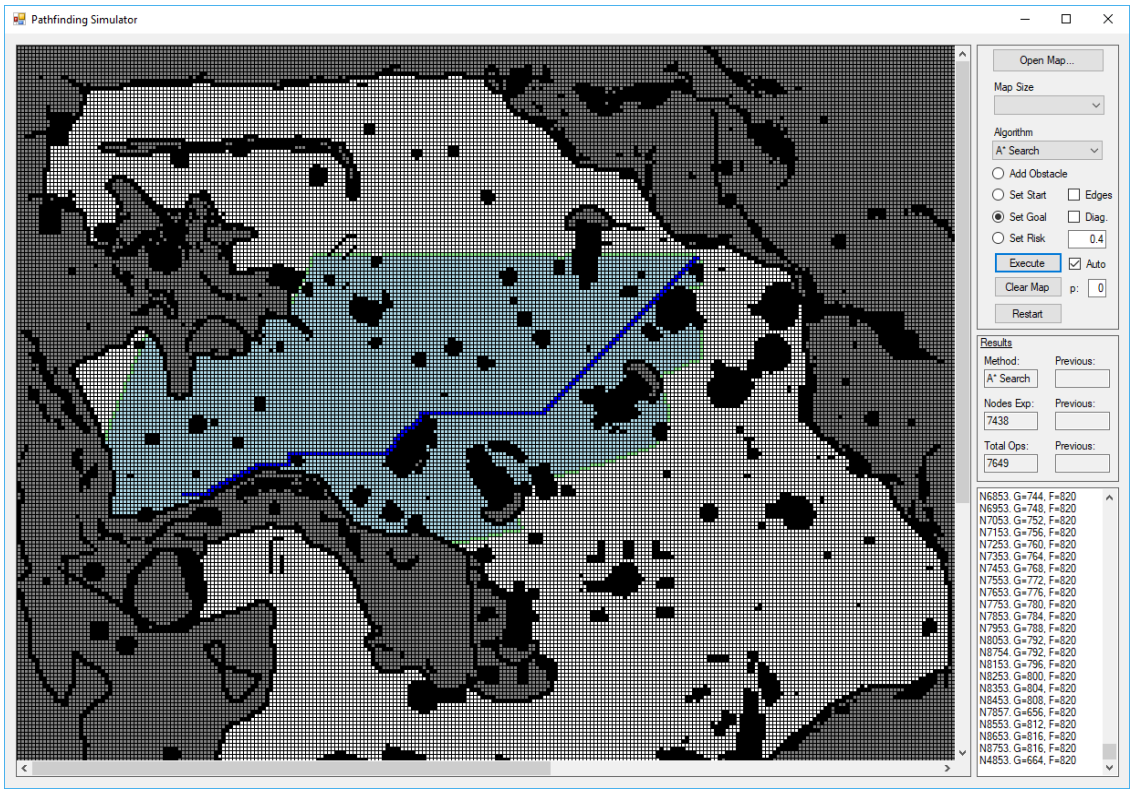


Figure 34: Demonstration of A*Search showing nodes that have been visited and the finalized path.

Path processing takes place in a background worker thread so not to freeze the interface while the program is running. No other operations can take place while path processing has started except for forced cancellation, which will halt processing and display the current results to the user. Although it has no bearing on performance, the user also has the option of selecting a node at any given time and viewing its properties, such as its name, its current *g*-value, and any risks associated to it. The results of the previous test are then displayed next to the current test to provide comparative feedback.

3.5 Obstacle Density

While the map sizes vary between tests, the obstacle density also varies within those maps. In the context of our experiments, obstacle density is defined as the total number of obstacles divided by the total number of vertices in the graph:

$$obstacle\ density = \frac{number\ of\ obstacles}{number\ of\ vertices}$$

The maps to be used in our experiments will be of varying obstacle density, but all solutions will be able to plan a path, provided there is no barrier between the start and goal positions. Obstacles will be distributed in such a way that a path to the destination will always be possible. Assuming no diagonal paths, a typical node in a graph will have at most 4 edge costs associated with traversing either to or from neighbouring nodes, resulting in a total of 8 possible connections for incoming and outgoing edge-weights. While the option of utilizing diagonal paths is available, it is not required and will be omitted from experiments.

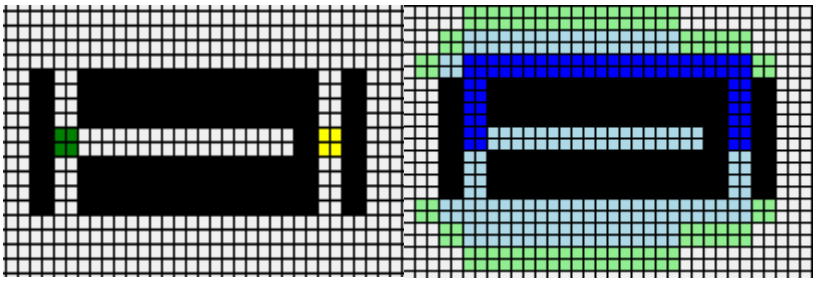


Figure 35: Obstacles may be distributed such that they increase the difficulty of path planning. Despite this, an optimal path will always be planned if it is possible.

In addition to each edge maintaining its own set of resources, they also have an associated *risk* assigned to them. This risk, coming in the form of a probabilistic event, represents the chance of a cost being required for traversal. Risk can be assigned to an edge-cost under the assumption

that a chance of failure can be accurately evaluated by the user or the agent utilizing the algorithm. Currently, risk accounts for one of two outcomes: Either the agent is forced to pay the cost of traversal respective to the required resource, or the cost is voided outright. Ultimately, the chance of one of these outcomes occurring depends on the given risk percentage associated with a resource.

Chapter 4: Experimental Process

4.1 Assumptions

In order to accurately keep experiments consistent and test cases manageable, the following assumptions have been made during the process:

- The pattern in which probabilities change during processing is predictable. Changes in the environment cannot be sporadic even after the agent has planned its estimated path.
- The agent will travel at a consistent speed between nodes. When an agent can change its speed or the rate at which edge costs are traversed, the time it takes to reach specific nodes may not be consistent. To better simplify the problem and to provide a working solution with consistent results, an agent's speed must at the very least be unchanging during runtime.
- Only one agent can be active in the field at a time. Solutions pertaining to cooperative pathfinding have been studied in the past, but the algorithms that were tested in particular can only handle single-agent solutions.
- All (if any) obstacles remain static during planning and execution.
- The weight of an edge does not affect the time it takes to traverse that edge (i.e., time itself is not a resource).

4.2 Map Refinement

The multi-objective pathfinding problem has some differences from traditional pathfinding problems, such as the graphs having multiple resources associated with edge-weights and

expected values related to the probability factor. Traditional maps simply do not come with this information, and so in order to supplement the specifications of the problem, the maps have been altered to accommodate these requirements. We refer to this process of map modification as *map refinement*, and its sole purpose is to prepare the search maps for testing by adding the required properties in order to satisfy the specifications of the multi-objective pathfinding problem. In our case, the search maps will be refined so that graphs contain the following additional data:

- Conversion rate matrix for scaling resource values
- Edge-weights with the following properties:
    - Multiple resources as opposed to a single resource per edge, containing:
        - An event with multiple outcomes and probabilities
    - Time-dependent values (or step-based)

In our setup, a set number of resources can be specified, defaulting to 1 for the simplest of cases, while risks and conversion rates are randomly generated. Risks represent a probability of a value of the associated resource being used up. They are represented by a probabilistic value, and thus range from a value of 0 for no risk to a value of 1 for the highest chance of risk. Agents that encounter nodes with a risk of 1 can expect to use up the exact amount of resources denoted along the specified edge-weight. Likewise, nodes with a risk of 0 can be considered cost-free, and so those nodes will be treated as if they were never refined in the first place, as used in traditional pathfinding. Risks that vary in between these two bounds will have a bearing on the realized path, in addition to the conversion rate, which causes the calculated rates to scale accordingly.

4.3 Framework

Experiments were run in a simulated environment developed in C# for the Microsoft Windows platform on a singular setup as described in the table below:

Application	Standalone Desktop
Framework	.NET, C#
External Sources	Map files (.map)
Processor	Intel Quad Core CPU Q8200 @ 2.33GHz

Exported environment files were taken from the published computer game, *Dragon Age: Origins* in order to provide relevant map data used in the game development industry. These maps come in a variety of sizes and also contain varying levels of obstacle density. Some maps include wide open areas whereas others are composed of narrow corridors. Thus, diversity in the size and obstacle density was considered when choosing map files for experiments. Nevertheless, all of the supplied map files were compatible with the refinement procedure.

Maps used for our research and experiments consist of the following with differentiating sizes:

Filename	**Size**	**Obstacle Density**
arena.map	49x49	~10%
arena2.map	209x281	~80%
brc101d.map	492x512	~80%
orz701d.map	722x839	~80%

Comparisons are made on a case by case basis, referring back to each of the scenarios explained in Chapter 3. Overall, there are a total of 12 scenarios involved:

1. Static Single Resource
2. Static Multiple Resources
3. Static Single Resource with Expected Values over Independent Events
4. Static Multiple Resources with Expected Values over Independent Events
5. Static Single Resource with Expected Values over Dependent Events
6. Static Multiple Resources with Expected Values over Dependent Events
7. Dynamic Single Resource with Known Temporal Values
8. Dynamic Multiple Resources with Known Temporal Values
9. Dynamic Single Resource with Expected Values over Independent Events and Known Temporal Values
10. Dynamic Multiple Resources with Expected Values over Independent Events and Known Temporal Values
11. Dynamic Single Resource with Expected Values over Dependent Events and Known Temporal Values
12. Dynamic Multiple Resources with Expected Values over Dependent Events and Known Temporal Values

A* Search, Strategic Pathfinding, and Multi-Objective Pathfinding solutions will be executed for all scenarios. To compare A* Search, Strategic Pathfinding (and in turn, Dijkstra's Algorithm), and Multi-Objective Pathfinding beyond the scope of Scenario 1, we use only distance in A* Search and Strategic Pathfinding. This results in a minimum distance path per solution. In addition to this, we calculate the cost of that path over an environment catered to Multi-Objective

Pathfinding and compare it to Multi-Objective Pathfinding to get our analysis. The algorithms will process each of the aforementioned map files, using randomly chosen nodes for denoting the start and destination, respectively. Subsequent experiments for the remaining scenarios will follow suit.

Chapter 5: Analysis

5.1 Solution Comparison

Our solution enhances the usability of existing pathfinding solutions and extends their applicability to the following types of problems:

Scenario	A* Search	Dijkstra's Algorithm	Strategic Pathfinding	Multi-Obj.
Static Single Resource	✓	✓	✓	✓
Static Multiple Resources			(Limited)	✓
Static Single Resource with Expected Values over Independent Events				✓
Static Multiple Resources with Expected Values over Independent Events				✓
Static Single Resource with Expected Values over Dependent Events			(Limited)	✓
Static Multiple Resources with Expected Values over Dependent Events				✓
Dynamic Single Resource with Known Temporal Values				✓
Dynamic Multiple Resources with Known Temporal Values				✓
Dynamic Single Resource with Expected Values over Independent Events and Known Temporal Values				✓
Dynamic Multiple Resources with Expected Values over Independent Events and Known Temporal Values				✓
Dynamic Single Resource with Expected Values over Dependent Events and Known Temporal Values				✓
Dynamic Multiple Resources with Expected Values over Dependent Events and Known Temporal Values				✓

5.2 Runtime Analysis

For the first test, the most basic situation is assumed. That is, the search maps contain the following properties:

- Number of resources: 2
- Number of outcomes: 1
- Has Risk: False
- Has dependencies: False
- Temporal values: False
- P = 0 (for Strategic Pathfinding)

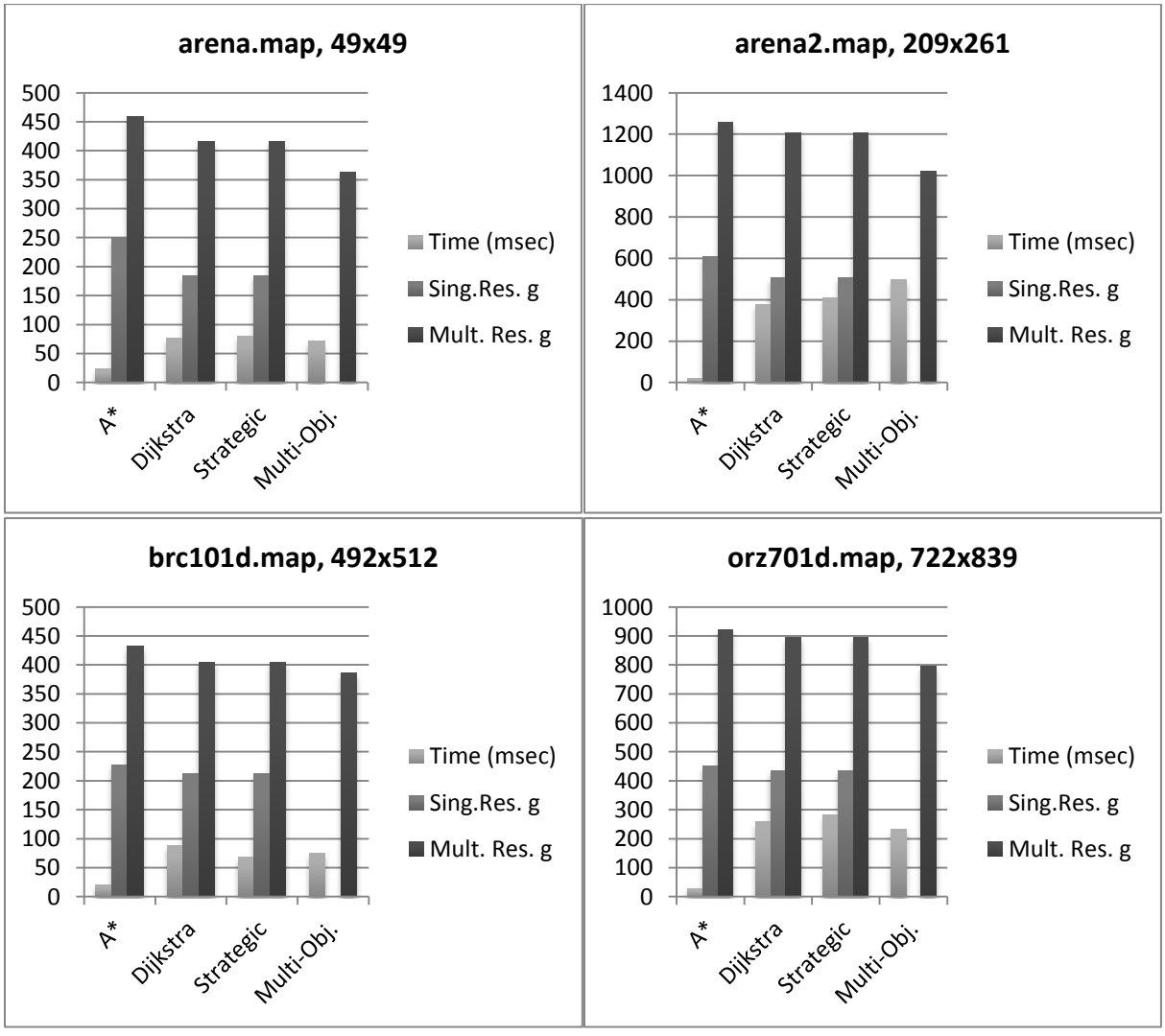


Figure 36: Comparison of algorithms across four different sizes of maps. Each search map contains the properties of a typical graph, such as a single weight per edge and only one outcome.

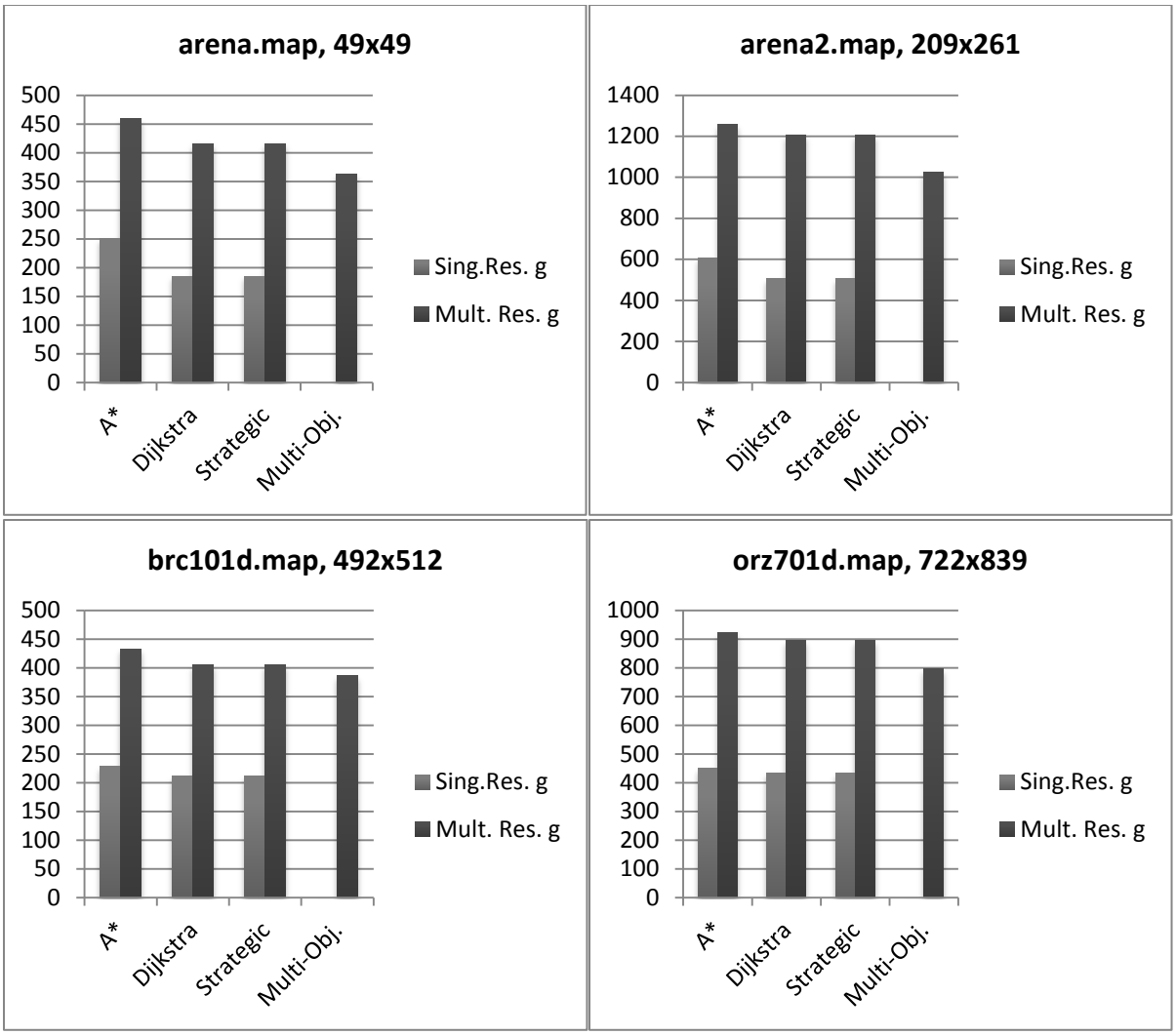


Figure 37: Comparisons of final g-values between solutions.

In this experiment, modified properties of the map were minimal compared to traditional pathfinding problems. All experiments are performed with the same designated start and goal nodes. Four categories are compared: Path Length, Runtime, Single Resource G-Value, and Multiple Resource G-Value. Path Length refers to the finalized length of the path starting from the, while Runtime represents the approximate amount of time each solution took to complete. Note that this time may or may not be the exact same in multiple experiments due to possible alterations in CPU clock speed. However, any slight deviations do not affect the overall

performance. Single Resource G-Value refers to the g-value of the goal node utilizing a single resource (specifically, the first resource that is created in a list of resources). This category will always result in 0 for the Multi-Objective Pathfinding algorithm, as any solution that contains a single resource will still be displayed in the following category. Multiple Resource G-Value represents the g-value of the goal node with all of the map properties taken into consideration. Thus, it is the true cost that would be required by the solution if any solutions that are not the Multi-Objective Pathfinding algorithm are used. Examples of this can be seen in Figure 37, where the A* Search solution results in a much larger Multiple Resource G-Value compared to the Multi-Objective Pathfinding algorithm. We can deduct from this analysis that although A* Search may have a shorter runtime comparatively, it can only utilize a single resource in the environment for path planning, and thus leads to more potential costs in the search map since all other factors are ignored.

With, $p = 0$, the Meta-Weight function in Strategic Pathfinding performs nearly the same as Dijkstra's Algorithm in all cases, as the solution does not take any risk factors into consideration during path planning. The most notable property about this comparison is that Strategic Pathfinding may not output accurate results if the resource values themselves are so great that they over-influence the calculated weight. This is one drawback that our solution circumvents with the use of the conversion rate. Moreover, the performance between Strategic Pathfinding and the Multi-Objective Pathfinding solution are comparable since they both use Dijkstra's Algorithm as a base. As stated previously, however, Strategic Pathfinding has limitations to the amount of resources that can be used in addition to resources being able to out-scale the preference factor.

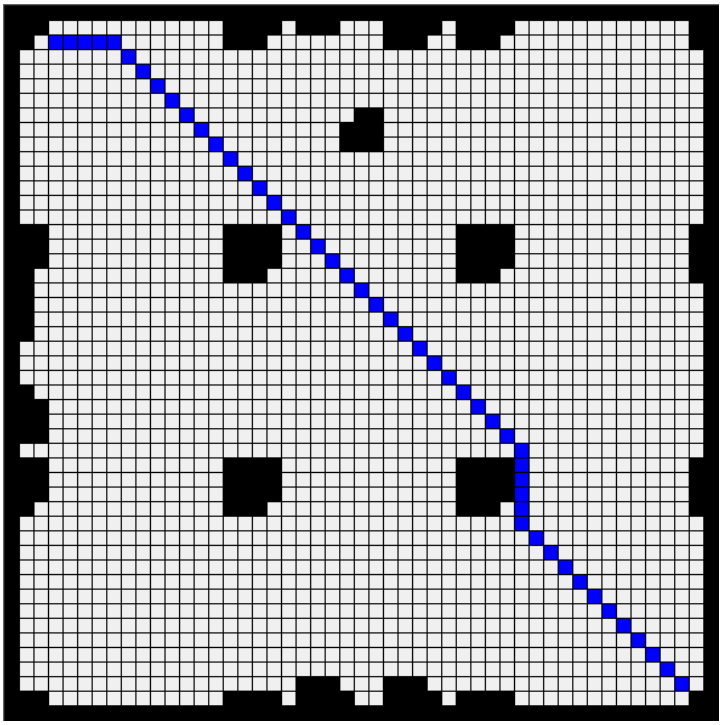


Figure 38: Demonstration of Strategic Pathfinding in our pathfinding framework with $p = 0$. Fundamentally, it works similarly to Dijkstra's Algorithm under these settings.

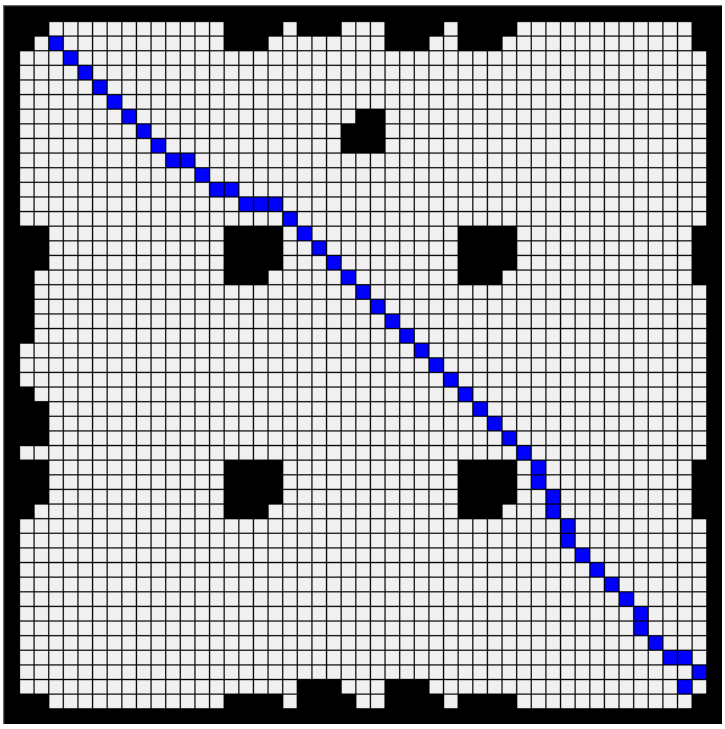


Figure 39: Demonstration of Multi-Objective Pathfinding in our pathfinding framework (with 5 resources, and 4 outcomes). Similarly to Dijkstra's Algorithm, the optimal solution will be realized. The difference, in this case, is the way the weights are outlined in the search map, which visually appears slightly longer than in Figure 38 although the actual weights within the search map are optimized.

For the second test, the search maps are given the following properties:

- Number of resources: 5
- Number of outcomes: 1
- Has Risk: True
- Has dependencies: False
- Temporal values: False
- P = 0.8 (for Strategic Pathfinding)

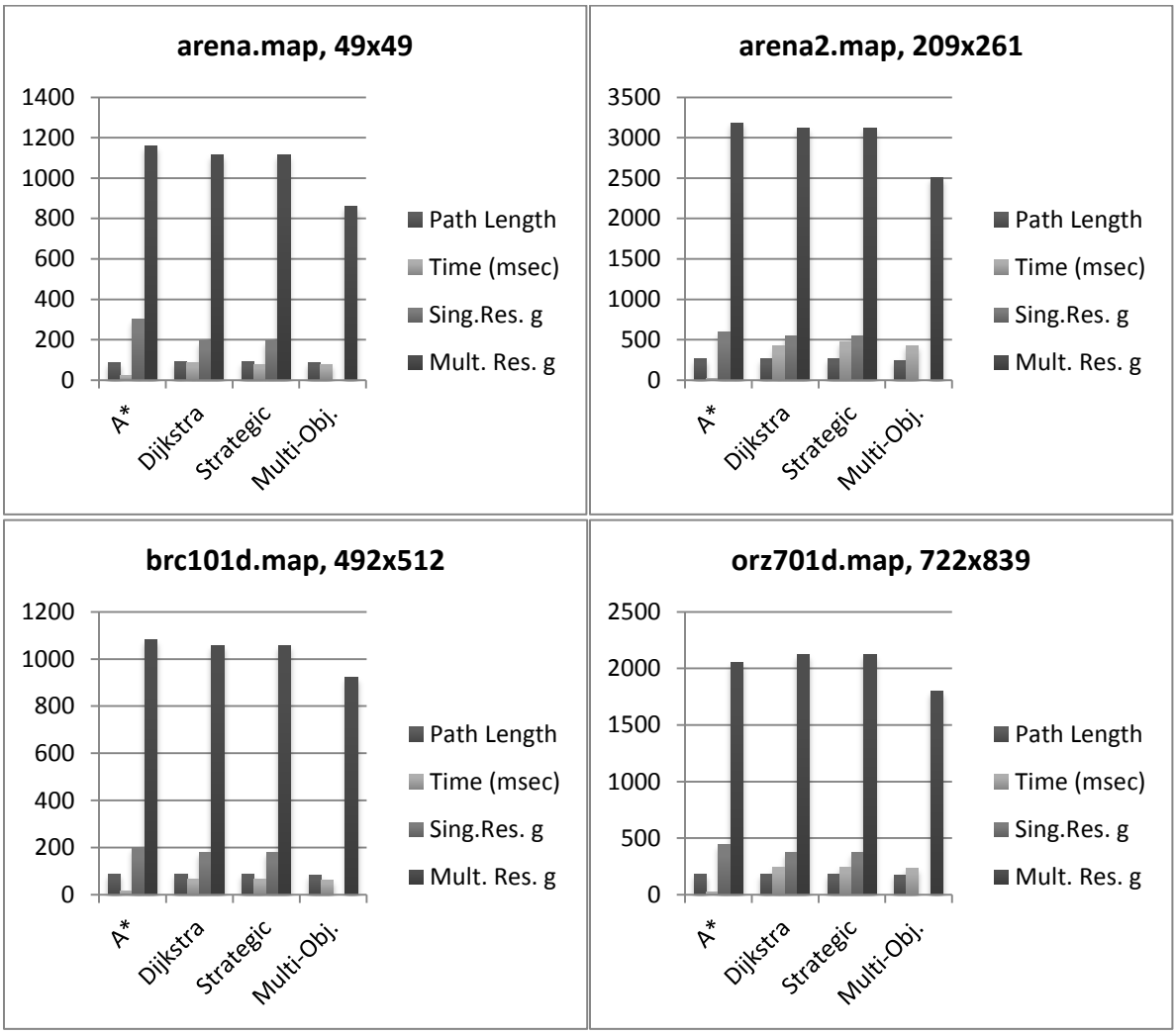


Figure 40: Comparison of algorithms across four different sizes of maps. Each search map caters to more specific scenarios beyond the scope of traditional pathfinding. Nevertheless, the path that A* Search would normally take in an environment with multiple resources can still be compared to Multi-Objective Pathfinding.

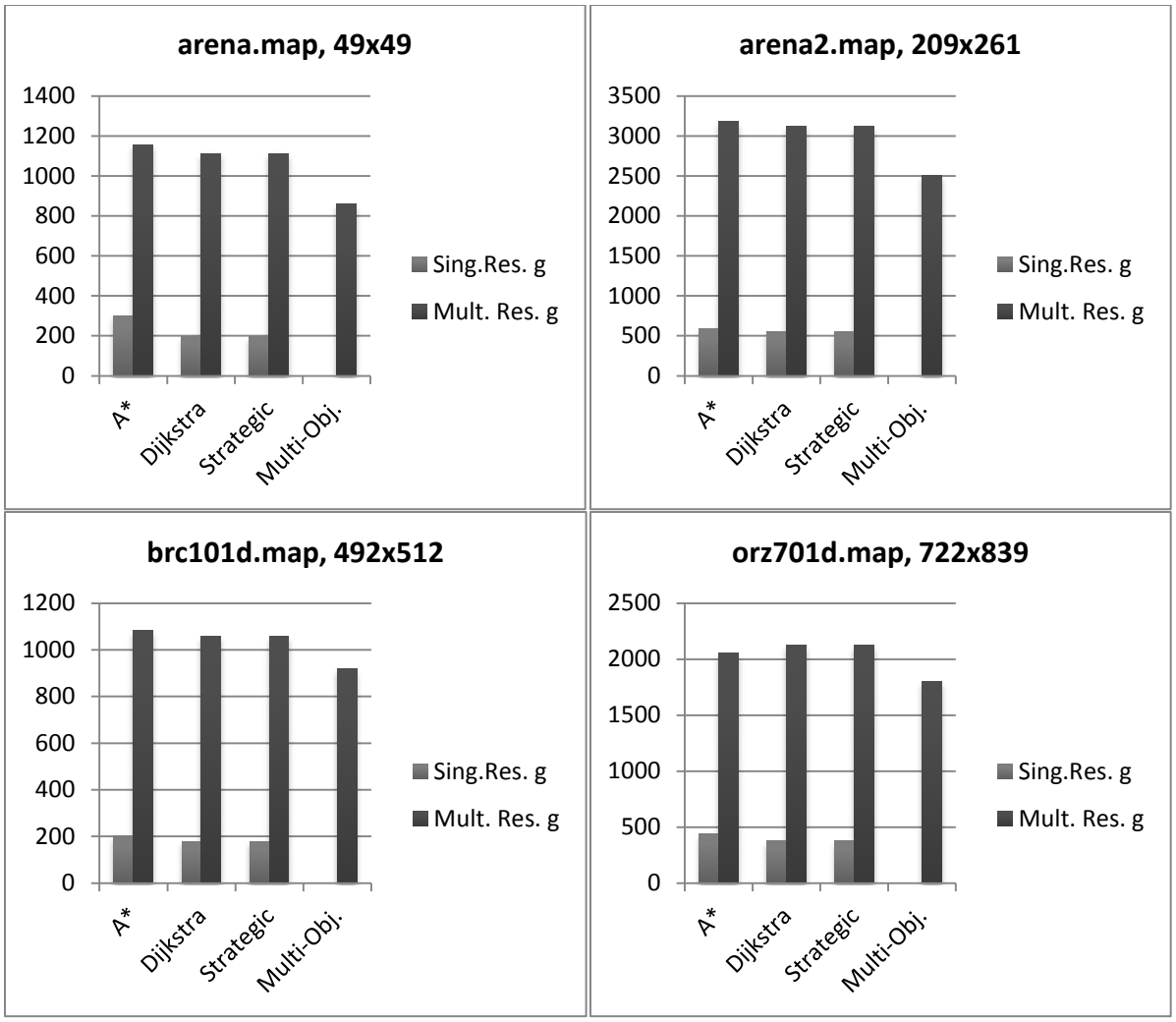


Figure 41: Comparisons of final g-values between solutions.

For this set of experiments, more than 2 resources were included across every edge in the search map and associated with randomly generated risk factors. The value of $p = 0.8$ has been given to add some variance to the path planning in the Strategic Pathfinding solution. From Figure 40 we can make the observation that the Multi-Objective solution is still comparable to Strategic Pathfinding and therefore Dijkstra's Algorithm. The interesting thing to note is that the cost in an environment that utilizes multiple resources is still cheaper if Multi-Objective Pathfinding is used compared to A* Search. As expected A* Search has the fastest compilation time out of all available solutions.

For the third test, the search maps are given the following properties:

- Number of resources: 7
- Number of outcomes: 3
- Has Risk: True
- Has dependencies: True
- Temporal values: False
- P = 0.8 (for Strategic Pathfinding)

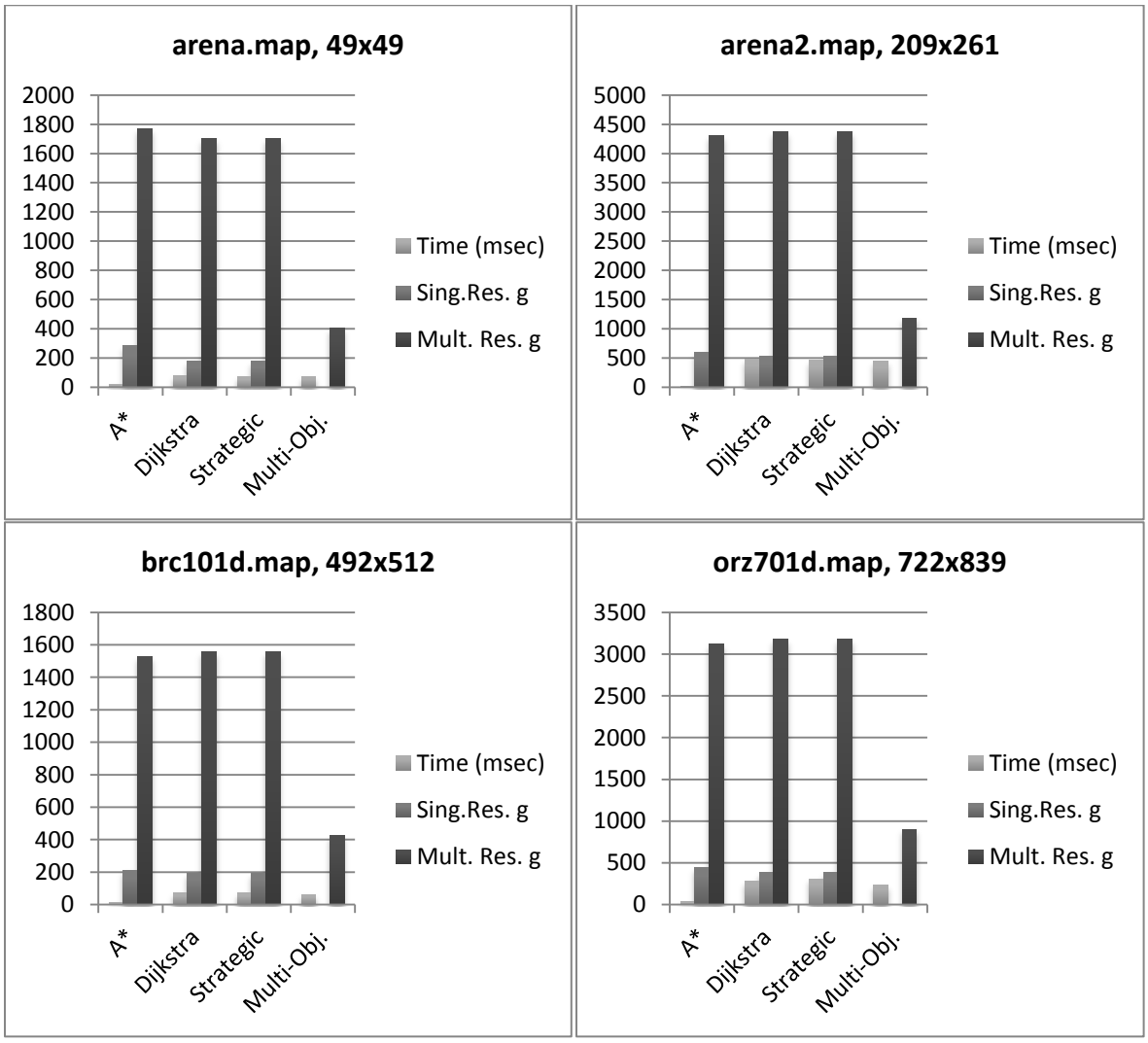


Figure 42: Comparison of algorithms across four different sizes of maps.

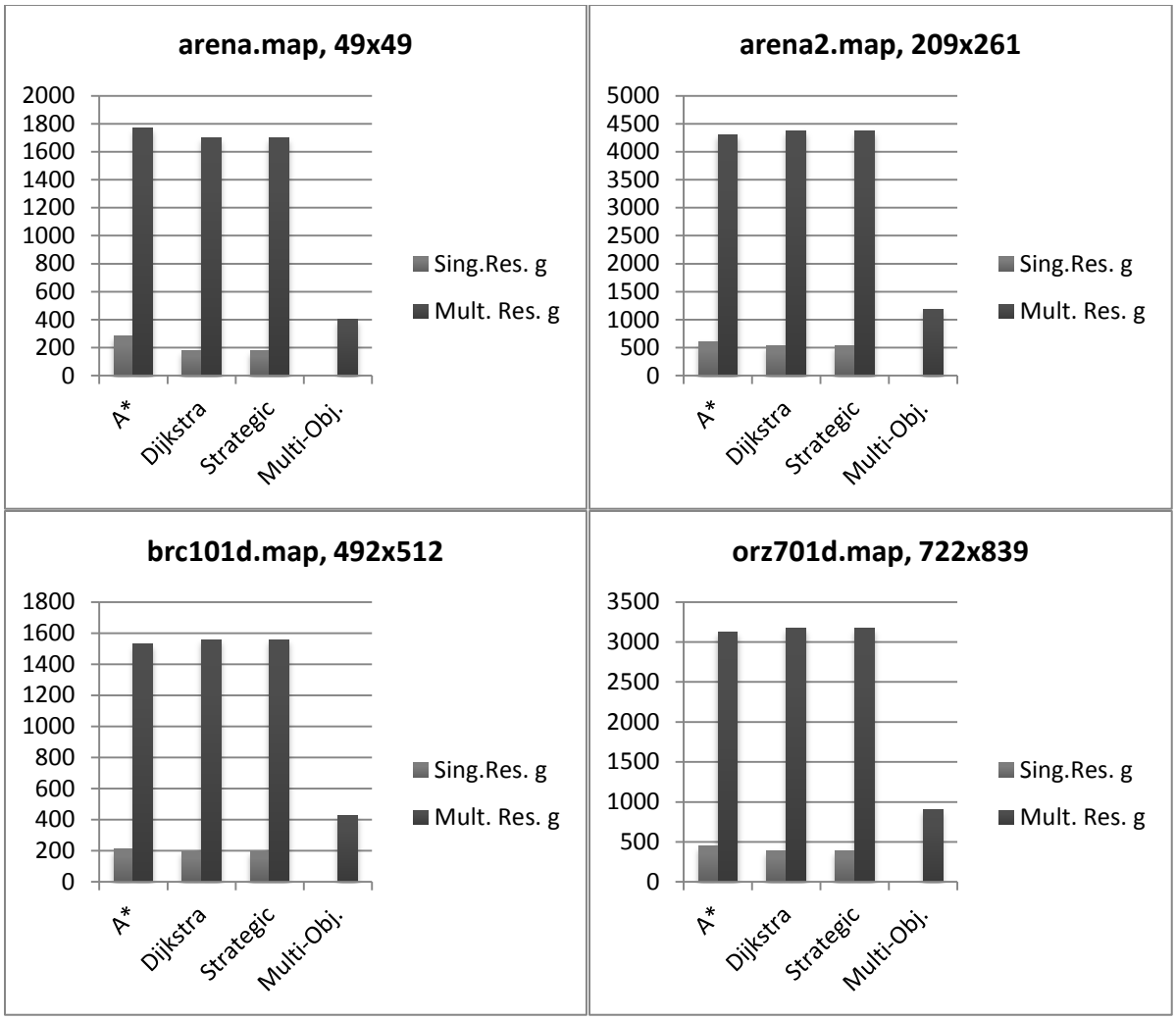


Figure 43: Comparisons of final g-values between solutions.

Like previously, we expect the Multi-Objective solution to be at least as efficient as Strategic Pathfinding. Even with multiple outcomes on each respective edge and dependencies, the pre-processing of the search map does not interfere with the actual path planning taking place. To say that our solution is more efficient than Dijkstra's Algorithm (and Strategic Pathfinding) by observing the results is a false, however, as both solutions follow the same steps to determine an optimal path. Rather, our solution covers many more types of problems that Dijkstra's Algorithm cannot account for as shown by the various scenarios in previous chapters. Additionally, as expected, A* Search leads to higher cost values when relying on a single resource in an

environment that contains multiple additional properties. The Multiple Resource G-Value is noticeably lower for the Multi-Objective Pathfinding solution, as the number of resources and outcomes is increased from previous tests in this case. With more resources and outcomes, there are simply more properties to optimize, which is to be expected when comparing to other traditional solutions that again utilize only one resource.

In the final set of tests, a change is made to the Multi-Objective Pathfinding solution to account for temporal values. The search maps are given the following properties:

- Number of resources: 7
- Number of outcomes: 3
- Has Risk: True
- Has dependencies: True
- Temporal values: True
- P = 0.8 (for Strategic Pathfinding)

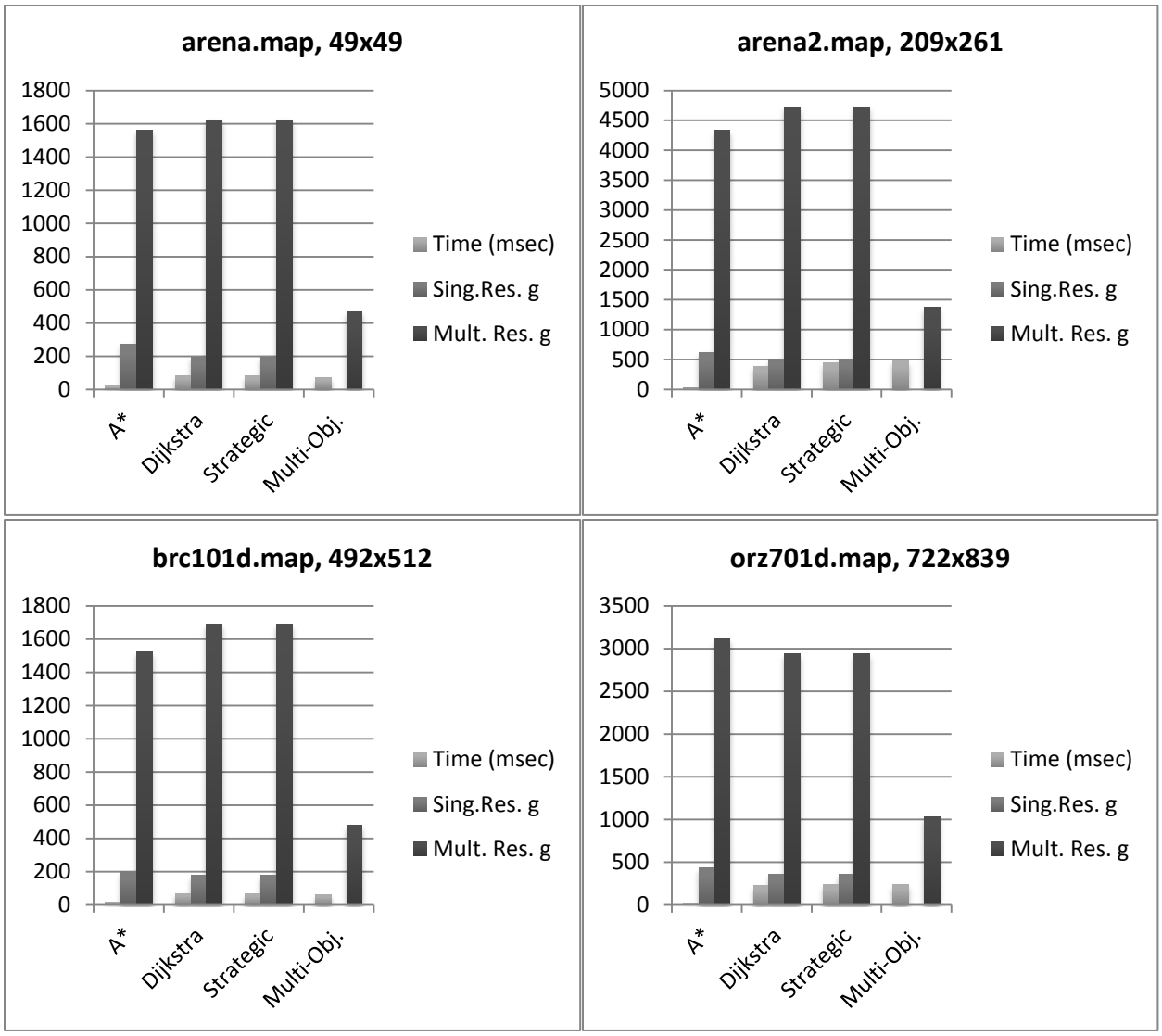


Figure 44: Comparison of algorithms across four different sizes of maps. Utilizing temporal values does not have a notable influence on performance as the underlying algorithm remains the same with the exception of a minor change in the cost formula. This small modification merely allows the accumulating cost to account for time.

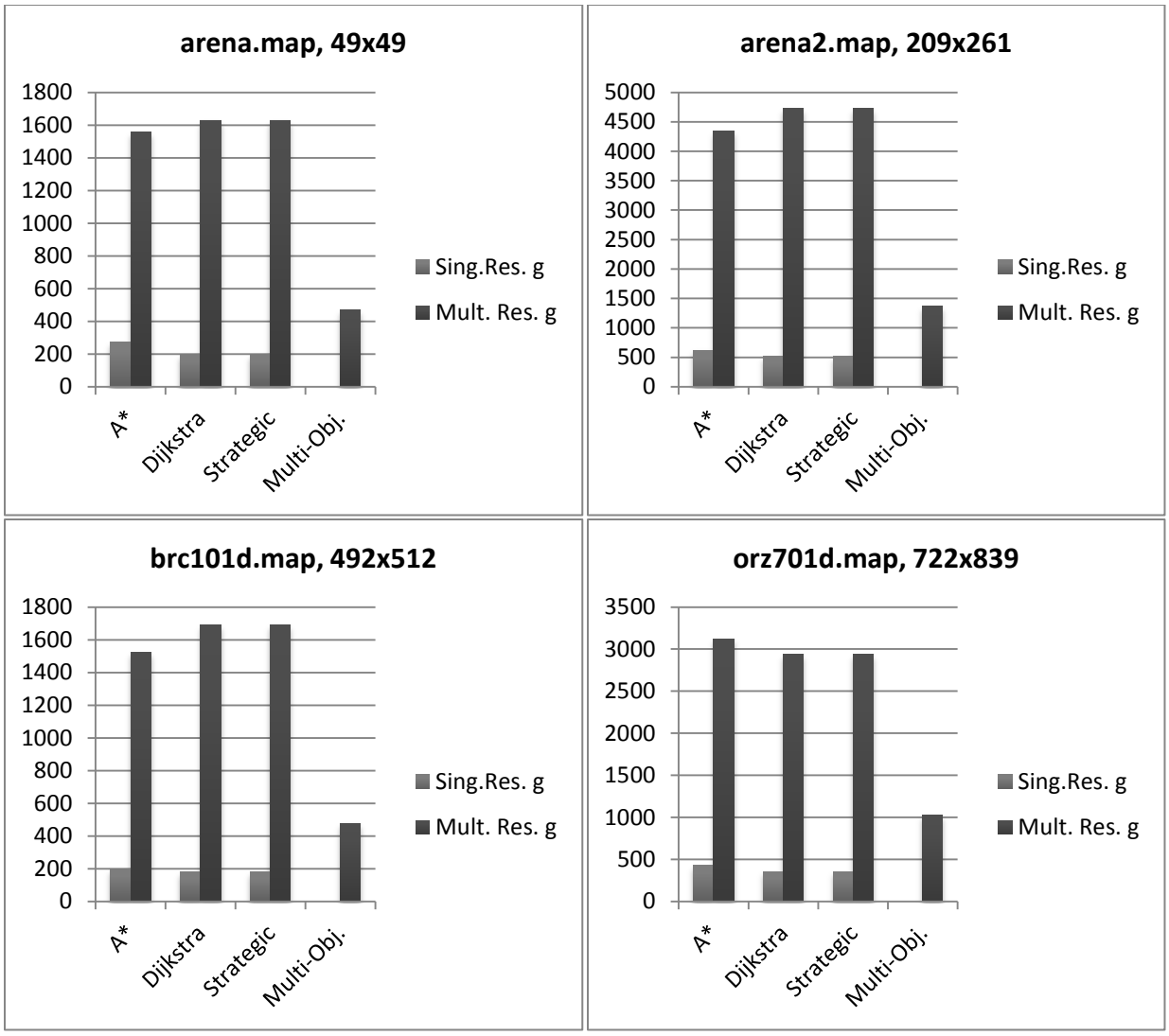


Figure 45: Comparisons of final g-values between solutions.

In these tests, the formula for calculating weight is slightly modified to accept the time at which the current node is expanded, resulting in variations of weights. Although there is a slight modification, the overall performance is barely affected by this. The amount of nodes that are expanded between solutions is comparable. Moreover, the difference in the finalized g-value is once again worth noting here. This can be attributed to the fact that multiple properties in the environment such as numerous resources, risk, and dependencies lean more towards Multi-Objective Pathfinding compared to A* Search and Strategic Pathfinding, which rely on a single

resource among edge costs. Although applying A* Search to a problem catered to Multi-Objective Pathfinding can more likely result in a faster compilation time, the overall cost will be greater than it otherwise would have been if Multi-Objective Pathfinding was used in the first place.

From this analysis, we can conclude that our solution performs at least as effective as traditional pathfinding algorithms that utilize Dijkstra's Algorithm in their core. In addition, problems that can be defined by one of the twelve aforementioned scenarios are best handled by Multi-Objective Pathfinding, as A* Search and Strategic Pathfinding will always result in a larger cost path despite having a smaller compilation time and utilizing a heuristic (provided multiple resources are used). This also true despite the fact that A*Search is suited best for single resource environments. Nevertheless, we cannot use a heuristic with our solution unless we can determine one that is both admissible and is applicable to multiple resource scenarios. In the most basic of cases, our solution will never underperform in comparison to Dijkstra's Algorithm.

Chapter 6: Conclusion

In this work, we have analyzed traditional pathfinding algorithms and extended their usability to several problem types that go beyond the applications of typical implementations. The solutions described in this work involve a pre-processing phase that modifies a search space to be compatible with traditional pathfinding algorithms. Traditional solutions can solve search maps that maintain a single static value representative of each edge-weight. Extending the usability of these solutions makes them applicable to environments with multiple resources contained within an edge by providing a scaling modifier to decide a composite weight. Scenarios proposed by Strategic Pathfinding methods may also be solved in this manner by utilizing the concept of expected values to determine an optimal path. Finally, problems that contain known temporal values in their schematics may also be applied here, as the generic formula for calculating edge-weights also encompasses this case.

Theoretical analysis with results from experiments concludes that our solution fundamentally works within the same realm of complexity as Dijkstra's Algorithm at worst. This is due to the fact that traditional pathfinding solutions such as Dijkstra's Algorithm and A* Search are seldom affected by the preprocessing of the search map. Moreover, the preprocessing in itself does not skew the complexity, as map refinement is performed in constant time per node in the graph. Thus, these solutions also have the benefit of being flexible, as existing traditional pathfinding solutions do not conflict with the refinement process.

Chapter 7: Future Work

There are numerous methods by which this research can be further developed, as some challenges were encountered but excluded from the final dissertation in order to retain maintainability of the scope. Namely, an admissible heuristic with consistency paralleling the A* Search heuristic still needs to be developed for most of the solutions presented in this work. Proposing simpler heuristics can somewhat alleviate this, such as choosing a minimum constant from a pool containing all nodes in the search map. Future research pertaining to optimality, however, may desire a heuristic with more accuracy. Furthermore, scenarios that present expected values in their solutions are assumed not to have a joint-dependency between resources. Although it is not a setback, a case by case analysis with resources containing this property should rightfully be included as an additional scenario in the solution. We could also explore the possibility of the conversion rate (for scenarios that utilize multiple resources on edges) changing during processing, similar to problems where edge-weights change with known temporal values.

References

[1] S. J. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," Upper Saddle River, N.J., Prentice Hall, 2003, p. 97–104.

[2] Buckland, in *Programming Game AI By Example*, Wordware Publishing Inc., 2005, p. 85.

[3] Hamming, "The Art of Probability for Scientists and Engineers," Addison-Wesley Publishing Company Inc, 1991, p. 65.

[4] A. Lavin, "A Pareto Front-Based Multiobjective Path Planning Algorithm," *Proc. of the 6th Workshop on Planning, Perception and Navigation for Intelligent Vehicles, IEEEIRSJ International Conference on Intelligent Robots and Systems (IROS),* 2014.

[5] K. Jeddisaravi, R. J. Alitappeh and F. G. Guimarães, "Multi-objective mobile robot path planning based on A* search," *6th International Conference on Computer and Knowledge Engineering,* p. 7–12, 2016.

[6] S. Aljubayrin, J. Qi, C. S. Jensen, R. Zhang, Z. He and Z. Wen, "The safest path via safe zones," *ICDE,* 2015.

[7] A. Singh and L. Holder, "Strategic Path Planning on the Basis of Risk vs. Time," *International Conference on Entertainment,* pp. 78-87, 2008.

[8] S. Koenig, M. Likhachev and D. Furcy, "Lifelong planning A*," *Artificial Intelligence Journal,* vol. 155, p. 93–146, 2004.

[9] J. Tremblay, P. A. Torres and C. Verbrugge, "Measuring risk in stealth games," *Proceedings of the 9th International Conference on Foundations of Digital Games,* 2014.

[10] N. Sturtevant, "Incorporating human relationships into path planning," *In Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'13),* p. 177–183, 2013.

[11] J. Tremblay, P. A. Torres, N. Rikovitch and C. Verbrugge, "An exploration tool for predicting stealthy behaviour," *Proceedings of the 2013 AIIDE Workshop on Artificial Intelligence in the Game Design Process,* 2013.

[12] A. Johansson and P. Dell'Acqua, "Knowledge-based probability maps for covert pathfinding," p. 339–350, 2010.

[13] M. H. Kim, H. M. Lee, Y. Wei and M. C. Lee, "A Study of New Path Planning Algorithm Using Extended A* Algorithm with Survivability," *Intelligent Robotics and Applications,* vol. 7508, no. 59, pp. 608-617, 2012.

[14] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik,* vol. 1, p. 269–271, 1959.

[15] P. E. Hart, N. J. Nillson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, no. 2, p. 100–107, 1968.

[16] "AI and Navigation," Epic Games Inc., 2012. [Online]. Available: https://docs.unrealengine.com/

udk/Three/AIAndNavigationHome.html. [Accessed November 2017].

[17] "Improved Outcomes Software, Manhattan," [Online]. Available: http://www.improvedoutcomes.com/docs/ WebSiteDocs/image/diagram_euclidean_manhattan_distance_metrics.gif. [Accessed November 2017].

[18] "Pathfinding Javascript Library," Zenva Game Dev Academy, 2013. [Online]. Available: https://gamedevacademy.org/ path-finding-javascript-library/. [Accessed November 2017].

Vita Auctoris

NAME:	Halen Whiston
PLACE OF BIRTH:	Windsor, ON
YEAR OF BIRTH:	1992
EDUCATION:	Ste. Anne High School, Windsor, ON, 2010
	University of Windsor, B.C.S. Honours, Windsor, ON, 2016
	University of Windsor, M.Sc C.S., Windsor, ON, 2018