The Bisective Funnel Search Algorithm

Michael Simons and Sean Grzenda

April 9, 2023

1 Introduction

Pathfinding algorithms are fundamental in diverse fields such as robotics, network routing, and game AI. Dijkstra's algorithm, while guaranteeing the shortest path in weighted graphs, can be inefficient for large graphs, as it explores all reachable nodes. A* introduces heuristics to guide the search, reducing unnecessary exploration, yet may still struggle with complex graphs. The *Bisective Funnel Search* algorithm bridges the gap by bisecting the search space, partitioning it into dynamically updated regions, each focused on both the start and goal. This approach reduces redundant exploration and directs the search toward a more focused path region, akin to a funnel, resulting in faster convergence while maintaining accuracy, even in large-scale weighted graph environments.

2 Motivation

The motivation for this algorithm stems from the need to balance exhaustive exploration (as seen in Dijkstra's algorithm) and informed, heuristic-driven approaches (as in A*). In weighted graphs with vast search spaces, exploring all nodes may be computationally prohibitive. The bisective funnel approach addresses this by focusing on two dynamic regions: one centered on the start node and the other on the goal node. This localized search allows the algorithm to progressively refine its exploration while maintaining optimality. Furthermore, this method is particularly advantageous in applications where computational efficiency and real-time decision-making are paramount.

3 Derivation

The Bisective Funnel Search is fundamentally a pathfinding algorithm operating over a weighted graph $G = (V, E)$, where V represents the set of vertices (nodes), and E denotes the set of edges, each with an associated weight. The objective is to find a path from a start node $s \in V$ to a target node $t \in V$ such that the total cost of the path is minimized.

The cost function $f(n)$ at each node $n \in V$ is defined as:

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

where:

- $g(n)$ represents the exact cost from the start node s to the node n (actual cost),
- $h(n)$ is a heuristic estimate of the cost from n to the goal node t.

Unlike traditional methods, the bisective search strategy leverages two heuristic-driven explorations, one from the start and another from the goal, progressively narrowing the search space. This bisective nature minimizes the total search space volume by focusing on the high-probability regions, approximating a funnel-like structure as the search converges towards the goal.

In order to ensure that $h(n)$ remains admissible (non-overestimating), we choose a heuristic that satisfies:

$$h(n) \leq \text{true cost}(n, t) \quad \forall n \in V$$

This guarantees the optimality of the solution. When $h(n)$ is consistent (monotonic), i.e., for every edge $(n, m) \in E$, it satisfies:

$$h(n) \leq \text{cost}(n, m) + h(m)$$

the bisective funnel search avoids reprocessing nodes, further enhancing its computational efficiency.

4 Pseudocode

Algorithm 1 Bisective Funnel Search Algorithm

Input: Graph $G = (V, E)$, Start node s, Target node t
Initialize: $g[s] = 0$, $f[s] = h(s, t)$, priority queue $Q = \{s\}$
while Q is not empty **do**
    $current \leftarrow$ node in Q with lowest $f(current)$
    if $current = t$ **then**
        return Reconstruct path from s to t
    end if
    for all neighbors n of $current$ **do**
        $tentative\_g \leftarrow g[current] + \text{cost}(current, n)$
        if $tentative\_g < g[n]$ **then**
            $g[n] \leftarrow tentative\_g$
            $f[n] \leftarrow g[n] + h(n, t)$
            Add n to Q
        end if
    end for
end while

5 Complexity Analysis

5.1 Time Complexity

The time complexity of the Bisective Funnel Search algorithm is primarily influenced by the operations of exploring nodes and managing the priority queue. In a graph $G = (V, E)$, where V represents the vertices and E the edges, we observe the following:

Worst-Case Complexity: In the worst case, the algorithm may need to explore every vertex and edge. Managing the priority queue requires $O(\log V)$ time per operation (insertion or deletion), with a total of $O(V + E)$ operations. Therefore, the overall worst-case time complexity is:

$$O((V + E) \log V)$$

This complexity arises from the logarithmic cost of priority queue operations combined with exploring each vertex and edge.

Average-Case Complexity: Due to the bisective nature of the search, the algorithm often avoids exploring the entire graph. By focusing on a funnel-shaped region between the start and goal, the algorithm reduces the number of explored nodes. Let k represent the number of nodes actually explored (where $k \ll V$). In this case, the average-case complexity becomes:

$$O(k \log V)$$

The bisective strategy's practical effect is that the value of k is often significantly smaller than V, particularly when the heuristic is well-informed.

5.2 Space Complexity

The space complexity is largely driven by the memory required to store the priority queue and the g and f values for each node. In the worst case, up to $O(V)$ nodes may need to be stored in the priority queue. Thus, the space complexity is:

$$O(V)$$

This is comparable to other heuristic search algorithms like A*, although the bisective approach may reduce memory usage in practice by limiting the number of explored nodes.

5.3 Heuristic Impact

The efficiency of the algorithm is highly dependent on the choice of the heuristic $h(n)$. When the heuristic is admissible and consistent, the algorithm maintains optimality while reducing node expansions. The more accurate $h(n)$ is in approximating the true cost, the fewer nodes need to be explored. In cases where $h(n)$ is close to the true cost, the algorithm approaches linear complexity relative to the optimal path length.

6 Use Cases

The Bisective Funnel Search algorithm excels in scenarios where computational efficiency is critical and where graphs are large and weighted. Key use cases include:

- **Robotics**: Navigation through complex, obstacle-filled environments, where real-time pathfinding is essential for autonomy.
- **Game AI**: Pathfinding in dynamic environments, where the ability to quickly compute optimal routes with minimal computational overhead is vital.
- **Network Routing**: Optimization of data transmission paths in large, weighted networks, balancing between heuristic guidance and full exploration to ensure minimal delays.
- **Supply Chain and Logistics**: Route optimization in transportation networks, particularly in systems with weighted nodes or dynamic conditions (e.g., traffic).

7 Conclusion

The Bisective Funnel Search algorithm offers a refined approach to pathfinding by combining the strengths of heuristic methods and exhaustive exploration. Its bisective search nature significantly reduces unnecessary exploration, making it particularly well-suited to large, weighted graphs. The mathematical foundation ensures that it remains optimal, while the practical efficiency derived from narrowing the search space provides notable advantages in real-world applications.