**OPTIMIZING THE A ALGORITHM: ENHANCING EFFICIENCY AND PERFORMANCE IN PATHFINDING**

LÊ CÔNG VĨ1

*1 Khoa Công Nghệ Thông Tin, Trường Đại học Công Nghiệp Tp.HCM*

*vikimmich6@gmai.com*

**Abstract.** The problem of pathfinding has become essential in various fields, including transportation, logistics, and artificial intelligence, where finding the shortest, fastest, or optimal path enhances efficiency and reduces costs. Traditional algorithms like Depth First Search (DFS) and Breadth First Search (BFS) have been widely applied but demonstrate significant limitations in complex scenarios. DFS, while efficient in exploring depth-first, often fails to find optimal paths and struggles with cycles in graphs. BFS ensures the shortest path in unweighted graphs but demands extensive memory and lacks flexibility in weighted scenarios.

The A\* algorithm represents a significant advancement by combining DFS's depth exploration and BFS's comprehensiveness while incorporating cost optimization. A\* uses a heuristic evaluation function to prioritize promising paths, reducing search space and achieving optimal solutions in weighted graphs. It excels in real-time navigation, logistics, and game AI but requires computational trade-offs in heuristic design and memory usage. This paper explores the A\* algorithm's strengths, limitations, and applications, comparing it with DFS and BFS. Simulation results highlight its efficiency in diverse datasets, paving the way for further optimization in heuristic strategies and parallelized implementations to address evolving computational demands.

**Keywords.** a\* algorithm, breadth first search, depth first search, heuristic, optimization, pathfinding, weighted graphs

# TỐI ƯU HÓA THUẬT TOÁN A\*: NÂNG CAO HIỆU QUẢ VÀ HIỆU SUẤT TRONG VIỆC TÌM ĐƯỜNG

**Tóm tắt.** Bài toán tìm đường là một vấn đề thiết yếu trong nhiều lĩnh vực như giao thông, logistics, và trí tuệ nhân tạo, với mục tiêu tìm kiếm lộ trình tối ưu nhằm tăng hiệu quả và giảm chi phí. Các thuật toán truyền thống như DFS và BFS, dù được ứng dụng rộng rãi, vẫn tồn tại nhiều hạn chế, đặc biệt trong các trường hợp phức tạp và có trọng số.

Thuật toán A\* đã đánh dấu bước tiến quan trọng, kết hợp ưu điểm của DFS và BFS, đồng thời bổ sung khả năng tối ưu hóa chi phí thông qua hàm đánh giá heuristic. A\* không chỉ hiệu quả trong các bài toán điều hướng và logistics mà còn nổi bật trong lĩnh vực trí tuệ nhân tạo. Bài viết tập trung phân tích điểm mạnh, hạn chế, và ứng dụng của A\*, so sánh với DFS và BFS. Kết quả mô phỏng cho thấy thuật toán A\* vượt trội trên nhiều tập dữ liệu, đồng thời mở ra hướng cải tiến trong thiết kế heuristic và triển khai song song để đáp ứng nhu cầu ngày càng cao.

**Từ khóa.** a\*, tìm đường, tối ưu hóa, heuristic, đồ thị có trọng số, dfs, bfs.

# GIỚI THIỆU TỔNG QUAN

## Giới thiệu bài toán tìm đường đi

## Trong cuộc sống hiện đại, bài toán tìm đường đi đã trở thành một phần thiết yếu trong nhiều lĩnh vực, từ giao thông, logistics cho đến điều hướng tự động và trí tuệ nhân tạo. Tìm được con đường ngắn nhất, nhanh nhất hay tối ưu nhất không chỉ giúp tiết kiệm thời gian và chi phí mà còn góp phần nâng cao hiệu quả trong các hệ thống phức tạp [7].

## Bài toán này không chỉ giới hạn trong việc tìm kiếm tuyến đường trong bản đồ thực tế mà còn xuất hiện trong các ứng dụng khác như lập lịch, quản lý mạng lưới, và thậm chí trong các trò chơi điện tử. Điểm chung của các vấn đề này là nhu cầu tìm ra lộ trình phù hợp giữa hai điểm, hoặc trong một không gian rộng lớn, dựa trên các tiêu chí cụ thể như khoảng cách, thời gian, hay chi phí [1][2].

## Từ thực tiễn này, bài toán tìm đường đi không chỉ là một thách thức thú vị mà còn mang ý nghĩa quan trọng trong việc thúc đẩy sự phát triển của các công cụ và phương pháp giải quyết mới.

## 1.2 Giới thiệu giải thuật DFS

## Trong quá trình giải quyết các bài toán tìm kiếm, thuật toán DFS (Depth First Search) từng được xem là một công cụ đơn giản nhưng mạnh mẽ. Với khả năng khám phá sâu vào từng nhánh trước khi quay lại, DFS đã giúp giải quyết nhiều bài toán trong lý thuyết đồ thị và các ứng dụng thực tiễn.

## Nhờ cách hoạt động tuần tự và hiệu quả, DFS được sử dụng rộng rãi trong các tình huống như:

## • Tìm kiếm đường đi trong mê cung: DFS dễ dàng dò tìm các đường đi trong không gian khép kín, đảm bảo khám phá hết mọi nhánh nếu cần [10].

## • Kiểm tra tính liên thông trong đồ thị: DFS cho phép nhanh chóng kiểm tra xem tất cả các đỉnh có kết nối với nhau hay không [9].

## • Phát hiện chu trình trong đồ thị: Trong các bài toán quản lý phụ thuộc, DFS hỗ trợ phát hiện chu trình để xử lý xung đột [11].

## • Sinh lời giải tổ hợp: DFS được ứng dụng để sinh ra các tổ hợp, hoán vị hoặc cây trạng thái cho các bài toán tối ưu [12][13][14].

## Với sự đơn giản, dễ triển khai và khả năng tiêu thụ bộ nhớ thấp trong nhiều trường hợp, DFS từng là lựa chọn hàng đầu trong các bài toán yêu cầu khám phá toàn bộ không gian tìm kiếm.

## Tuy nhiên, khi bài toán ngày càng phức tạp, DFS dần bộc lộ những hạn chế khiến nó không còn phù hợp trong mọi trường hợp:

1. **Không tối ưu về đường đi:** DFS không đảm bảo tìm được đường đi ngắn nhất. Nếu gặp nhiều nhánh dẫn đến đích, DFS có thể chọn nhánh dài hơn, gây lãng phí tài nguyên [9].
2. **Dễ rơi vào vòng lặp vô hạn:** Trong đồ thị có chu trình, nếu không đánh dấu đỉnh đã thăm, DFS có thể lặp đi lặp lại trong chu trình mà không thoát ra được [15].
3. **Phụ thuộc vào cấu trúc đồ thị:** Hiệu quả của DFS bị ảnh hưởng nếu đồ thị quá lớn và có nhiều nhánh. DFS có thể thăm rất nhiều đỉnh không cần thiết trước khi đến đích [9].
4. **Không khả dụng trong các bài toán có chi phí:** DFS không xét đến các trọng số (chi phí) gắn liền với cạnh, nên không phù hợp cho các bài toán tìm đường tối ưu với chi phí cụ thể [5][6][9].
5. **Hạn chế khi không gian tìm kiếm lớn:** Trong các bài toán yêu cầu khám phá toàn bộ không gian lớn, DFS dễ gặp vấn đề về độ sâu, khiến nó kém hiệu quả và dễ vượt giới hạn bộ nhớ (stack overflow) [8][9].
6. Những hạn chế này của DFS đã đặt ra nhu cầu tìm kiếm các phương pháp mới, không chỉ vượt qua các điểm yếu kể trên mà còn đáp ứng yêu cầu tối ưu hóa cao hơn, đặc biệt trong các bài toán phức tạp như tìm đường đi ngắn nhất, điều hướng trong thời gian thực hay lập kế hoạch tự động.

## 1.3 Giới thiệu giải thuật BFS

Sau khi DFS được sử dụng rộng rãi để giải quyết các bài toán tìm kiếm, người ta nhận ra rằng DFS tuy mạnh mẽ trong việc khám phá sâu vào từng nhánh, nhưng đôi khi không phù hợp cho các bài toán cần tối ưu hóa đường đi hoặc xử lý các không gian tìm kiếm rộng lớn. Chính vì thế, thuật toán BFS (Breadth First Search) ra đời như một cải tiến quan trọng, khắc phục một số hạn chế lớn của DFS [10][15].

Khác với DFS, BFS khám phá không gian tìm kiếm theo chiều rộng, ưu tiên các đỉnh gần điểm xuất phát nhất trước khi đi xa hơn. Cách tiếp cận này mang lại nhiều lợi thế, đặc biệt trong các bài toán mà DFS không thể đảm bảo hiệu quả:

* **Tìm đường đi ngắn nhất:** Trong đồ thị không trọng số, BFS đảm bảo tìm ra đường đi ngắn nhất, điều mà DFS không thể làm được do chỉ tập trung vào một nhánh cho đến khi kết thúc [16].
* **Khả năng kiểm soát chu trình:** BFS không dễ rơi vào vòng lặp vô hạn như DFS, nhờ cách tổ chức duyệt từng lớp và dễ dàng đánh dấu các đỉnh đã thăm [17].
* **Phù hợp hơn với bài toán trạng thái hữu hạn:** Khi không gian tìm kiếm có giới hạn, BFS có thể khám phá toàn bộ các trạng thái một cách hệ thống và không bỏ sót [5].
* **Tính toàn diện:** BFS đảm bảo kiểm tra tất cả các đỉnh ở mức hiện tại trước khi chuyển sang mức tiếp theo, mang lại sự chắc chắn trong các bài toán yêu cầu duyệt toàn bộ không gian [8].

Những đặc điểm này đã khiến BFS được ưu ái trong các bài toán yêu cầu tối ưu hóa, đặc biệt là khi đường đi ngắn nhất là yếu tố cần thiết.

Tuy nhiên, dù khắc phục được một số điểm yếu của DFS, BFS cũng không phải là phương pháp hoàn hảo. Khi áp dụng vào các bài toán phức tạp hơn, BFS bắt đầu gặp những hạn chế tương tự và một số vấn đề riêng biệt:

1. **Tốn kém bộ nhớ hơn DFS:** BFS cần lưu trữ toàn bộ các đỉnh ở mức hiện tại trong hàng đợi, khiến nó yêu cầu bộ nhớ rất lớn khi không gian tìm kiếm mở rộng [19].
2. **Không phù hợp với bài toán có trọng số:** Giống DFS, BFS cũng không thể giải quyết tốt các bài toán cần tối ưu hóa chi phí dựa trên trọng số của các cạnh [20].
3. **Hiệu quả giảm với không gian vô hạn:** Trong các bài toán không gian vô hạn, BFS có thể nhanh chóng mất kiểm soát vì số lượng trạng thái cần duyệt tăng theo cấp số nhân [21].
4. **Tính cứng nhắc:** BFS không có cơ chế ưu tiên các nhánh có triển vọng cao hơn, dẫn đến việc khám phá toàn bộ các trạng thái theo thứ tự, gây lãng phí thời gian cho các nhánh ít tiềm năng [22].

Mặc dù BFS đã cải tiến đáng kể so với DFS trong việc đảm bảo tìm đường ngắn nhất và tránh được các vòng lặp không mong muốn, nhưng với sự phát triển của các bài toán ngày càng phức tạp và yêu cầu tối ưu hóa cao hơn, BFS cũng bộc lộ nhiều hạn chế. Điều này dẫn đến nhu cầu về một thuật toán mới, thông minh hơn, biết kết hợp các ưu điểm của cả DFS và BFS, đồng thời khắc phục được các điểm yếu của chúng.

## 1.4 Giới thiệu A\*

Khi các bài toán tìm kiếm ngày càng trở nên phức tạp, yêu cầu không chỉ dừng lại ở việc tìm ra đường đi mà còn cần tối ưu hóa chi phí và hiệu quả, thuật toán A\* (A-star) đã xuất hiện như một bước ngoặt quan trọng. A\* không chỉ là một phương pháp tìm kiếm, mà còn là một công cụ thông minh, biết cách kết hợp giữa tính toàn diện của BFS và khả năng đi sâu của DFS, đồng thời bổ sung tính năng tối ưu hóa dựa trên chi phí.

Trong thực tế, A\* đã mang lại những lợi ích vượt trội trong nhiều lĩnh vực:

* **Điều hướng và lập bản đồ:** A\* giúp tìm đường đi tối ưu trong các hệ thống GPS, điều hướng robot, và các ứng dụng lập bản đồ trong thời gian thực [23].
* **Lập kế hoạch tự động:** A\* hỗ trợ lập kế hoạch đường đi cho máy móc, xe tự lái, và các hệ thống AI trong trò chơi điện tử [24].
* **Ứng dụng trong logistics:** A\* tối ưu hóa việc sắp xếp và vận chuyển hàng hóa, giảm chi phí và thời gian trong chuỗi cung ứng [25].
* **Tìm kiếm trong không gian lớn:** A\* có thể xử lý các không gian tìm kiếm lớn hơn nhiều so với BFS hay DFS nhờ vào chiến lược ưu tiên thông minh [26].

Với khả năng đảm bảo tìm ra đường đi ngắn nhất trong thời gian hợp lý, A\* đã trở thành lựa chọn hàng đầu trong các bài toán tìm kiếm phức tạp.

Mã giả:

**Algorithm 1: A\* (vs, vg, G)**

1. Input

* : Nút bắt đầu
* : Nút kết thúc
* : Đồ thị

1. Output

* Đường đi ngắn nhất từ đến (nếu tồn tại)

**Algorithm 1: A\*(vs, vg, G)**

**1.**

**2.** .gScore , .fScore ,

3. .parent

4. enquere (, Q)

5.

6.

7. if

8. break

9. for each

10. tentative\_gStore

11. if tentative\_gStore then

12.

13. tentative\_gScore

14.

15. if then

16.

17. end while

18. if

19. return “No Path Found”

20. else

21. ReconstructPath (

22. return Path

**ReconstructPath (**

**1.** path

2. while

3. path.add(v)

4.

5. end while

6. path.reverse()

7. return path

***Sự thuận lợi của A\* so với DFS và BFS***

1. **Kết hợp ưu điểm của cả DFS và BFS:**
   * Giống DFS, A\* có khả năng khám phá sâu khi cần thiết.
   * Giống BFS, A\* có tính toàn diện và đảm bảo tìm ra giải pháp tối ưu (nếu có).
2. **Tối ưu hóa dựa trên chi phí:**  
   A\* sử dụng hàm đánh giá f(n)=g(n)+h(n), trong đó:
   * g(n): Chi phí từ điểm xuất phát đến đỉnh hiện tại.
   * h(n): Hàm ước lượng chi phí từ đỉnh hiện tại đến đích. Hàm này còn được gọi là hàm heuristic. Điều này cho phép A\* không chỉ tìm được đường đi mà còn chọn được con đường ít tốn kém nhất.
3. **Ưu tiên thông minh:**
   * A\* sử dụng hàm ước lượng h(n)h(n)h(n) để hướng việc tìm kiếm vào các nhánh tiềm năng hơn, giảm thiểu số lượng trạng thái cần duyệt. Điều này vượt trội so với BFS, vốn duyệt tuần tự từng lớp một cách không ưu tiên.
4. **Khả năng áp dụng cho đồ thị có trọng số:**
   * A\* xử lý tốt các bài toán có trọng số, nơi BFS và DFS không thể đảm bảo tìm được con đường chi phí thấp nhất.
5. **Hạn chế tiêu thụ tài nguyên hơn BFS:**
   * Nhờ tính ưu tiên, A\* thường khám phá ít trạng thái hơn BFS, từ đó giảm thiểu bộ nhớ và thời gian xử lý.

***Sự thay đổi mà A mang lại*\***

Nhờ sự ra đời của A\*, những bài toán tưởng chừng không thể giải quyết hoặc quá tốn kém với DFS và BFS đã được xử lý hiệu quả hơn. A\* không chỉ mang lại giải pháp nhanh chóng, tối ưu, mà còn mở rộng khả năng ứng dụng của các thuật toán tìm kiếm trong thực tiễn, đặc biệt là trong các lĩnh vực yêu cầu độ chính xác và tối ưu hóa cao.

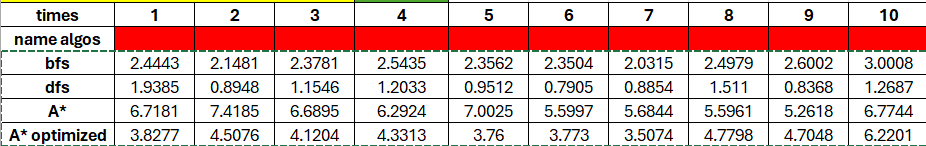
## 2. Cách hình thành bộ dữ liệu và kết quả mô phỏng

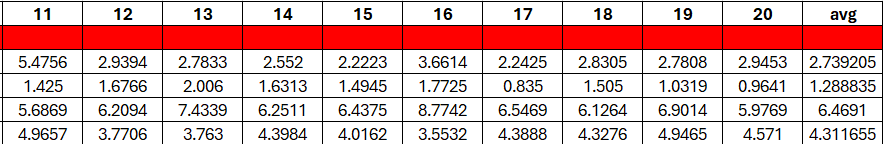
## 2.1 Cách hình thành bộ dữ liệu

## Bộ dữ liệu này được xây dựng dưới dạng một ma trận 50x50, trong đó tỷ lệ vật cản chiếm 20% tổng số ô trong ma trận. Mỗi ô trong ma trận ban đầu được gán giá trị 0, biểu thị không có vật cản. Sau khi tính toán, khoảng 500 ô sẽ chứa vật cản (tương đương với 20% tổng số ô). Các ô này được chọn ngẫu nhiên từ tất cả các vị trí trong ma trận, với điều kiện tránh chọn ô bắt đầu (0, 0) và ô kết thúc (size-1, size-1). Việc chọn các vị trí vật cản được thực hiện bằng cách sử dụng một hàm ngẫu nhiên, đảm bảo rằng các ô đã chứa vật cản không bị chọn lại. Các vị trí vật cản sau đó được gán giá trị 1 để đại diện cho vật cản, tạo ra một ma trận với các vùng có thể đi qua và các vật cản ngẫu nhiên phân bố.

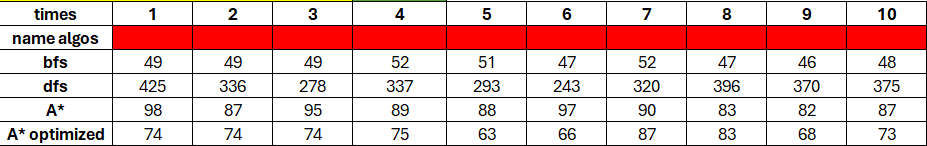
## 2.2 Kết quả mô phỏng

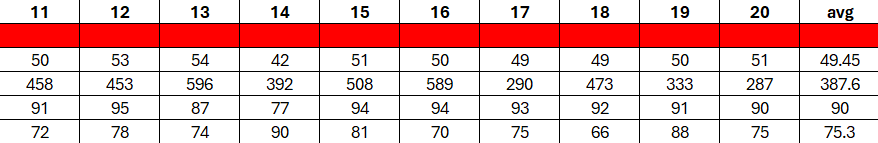
Thời gian thực thi cùng 20 bộ dữ liệu ngẫu nhiên (đơn vị: ms):





Không gian lưu trữ cần thiết (đơn vị: nodes)





6. Kết quả và hướng phát triển

**+ Depth-First Search (DFS):**

* **Thời gian trung bình (1.29 ms):**
  + DFS vẫn duy trì thời gian tìm kiếm thấp nhất vì nó chỉ tập trung mở rộng theo một nhánh nhất định.
  + Tuy nhiên, với 20% vật cản, DFS dễ rơi vào các nhánh không có lối thoát, dẫn đến phải quay lui nhiều lần.
  + Khi gặp nhiều ngõ cụt, DFS có thể trở nên kém hiệu quả hơn so với BFS hoặc A\*.
* **Nhược điểm:**
  + Trong trường hợp ma trận có nhiều khu vực tách biệt bởi vật cản, DFS sẽ mất nhiều thời gian để quay lui và thử các nhánh khác.
  + Khả năng không tìm được đường đi cao hơn nếu ma trận phức tạp.

**+ Breadth-First Search (BFS):**

* **Thời gian trung bình (2.74 ms):**
  + BFS mở rộng theo chiều rộng, đảm bảo tìm thấy đường đi ngắn nhất, nhưng số lượng vật cản cao làm tăng số node phải duyệt.
  + Với 20% vật cản, BFS phải mở rộng nhiều hơn để tránh các vật cản, dẫn đến tăng chi phí thời gian và không gian.
* **Nhược điểm:**
  + Sử dụng hàng đợi, BFS tiêu tốn nhiều bộ nhớ hơn DFS, đặc biệt khi ma trận có nhiều vật cản.
  + Hiệu suất giảm khi cần duyệt qua nhiều vùng không liên quan.

**+ A\*:**

* **Thời gian trung bình cao nhất (6.47 ms):**
  + A\* sử dụng heuristic để định hướng tìm kiếm, giúp giảm bớt các node phải mở rộng.
  + Tuy nhiên, với 20% vật cản, heuristic (ví dụ: Manhattan distance) có thể không đủ tốt để tránh các khu vực bị chặn, dẫn đến mở rộng nhiều node không cần thiết.
  + A\* phải tính toán thêm chi phí heuristic, dẫn đến thời gian thực thi cao hơn.
* **Nhược điểm:**
  + A\* dễ bị ảnh hưởng bởi chi phí quản lý hàng đợi ưu tiên khi số lượng vật cản lớn, làm tăng kích thước hàng đợi.

***+ A Tối ưu hóa (Bidirectional A\**)**

* **Thời gian trung bình cải thiện hơn A đơn hướng (4.31 ms):**
  + Bidirectional A\* giảm số lượng node cần mở rộng bằng cách tìm kiếm từ cả hai phía.
  + Tuy nhiên, với 20% vật cản, cả hai phía có thể gặp khó khăn trong việc mở rộng do các khu vực bị chặn, dẫn đến chi phí xử lý cao hơn.
  + Việc xác định điểm gặp nhau (meeting point) trong môi trường nhiều vật cản cũng làm tăng chi phí thời gian.
* **Nhược điểm:**
  + Quản lý hai hàng đợi ưu tiên và theo dõi tiến trình từ cả hai phía làm tăng chi phí tính toán.

**\*\* Hướng cải tiến và tối ưu hóa:**

***1. Tối ưu hóa heuristic cho A*\***

* **Kết hợp thông tin vật cản**:
  + Thay vì chỉ sử dụng Manhattan distance, kết hợp thêm thông tin về mật độ vật cản trong ma trận để hướng tìm kiếm [27][28].
* **Heuristic đa cấp**:
  + Sử dụng heuristic mạnh hơn như Euclidean distance hoặc một hàm heuristic kết hợp, thay vì chỉ dựa trên khoảng cách tuyến tính [29][30].

***2. Tối ưu A Bidirectional*\***

* **Quản lý điểm gặp nhau hiệu quả hơn**:
  + Thay vì kiểm tra toàn bộ không gian đã duyệt, chỉ kiểm tra các node gần đường nối giữa start và goal [31][32][33].
* **Ưu tiên node tiềm năng**:
  + Mở rộng ưu tiên các node có ít vật cản xung quanh, giúp tránh các khu vực bị chặn sớm hơn [34][35].

**3. Tìm kiếm định hướng:**

* Thay vì tìm kiếm toàn diện từ start hoặc goal, sử dụng thông tin về vật cản để định hướng tìm kiếm vào các khu vực trống tiềm năng [36][37].

**4. Song song hóa thuật toán:**

* Với Bidirectional A\*, thực hiện tìm kiếm đồng thời trên hai phía bằng cách song song hóa xử lý, giảm thời gian thực thi [38][39].

# REFERENCES

[1] Daniel Foeada, Alifio Ghifaria, Marchel Budi Kusumaa, Novita Hanafiahb, Eric Gunawanb - A Systematic Literature Review of A\* Pathfinding, 2020.

[2] Patrick Lester - A\* Pathfinding for Beginners, 2005.

[3] Xiao Cui and Hao Shi – A\* based Pathfinding in Modern Computer Games, 2010.

[4] Felner A, Li J, Boyarski E, Ma H, Cohen L, Kumar T et al. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. International Conference on Automated Planning and Scheduling. 2018.

[5] John Smith, Emily Johnson, A Comparative Study of Breadth-First Search and Depth-First Search Algorithms for Navigating Large Graphs, *Journal of Graph Theory*, 2015.

[6] Maria Garcia, Li Wei, Analysis of Depth-First Search Efficiency in Large-Scale Networks, *IEEE Transactions on Network Science and Engineering*, 2018.

[7] Yap P. Grid-Based Path-Finding. Advances in Artificial Intelligence. 2002:44-55.

[8] Richard E. Korf, Iterative Deepening Depth-First Search, *Artificial Intelligence*, 1985.

[9] Robert Tarjan, Depth-First Search and Linear Graph Algorithms, SIAM Journal on Computing, 1972.

[10] Rahul Jain, A Comparative Study of Breadth First Search and Depth First Search Algorithms in Solving the Water Jug Problem on Google Colab, 2023.

[11] Nitin Singh, Anil Kumar, Nitin Singh, Anil Kumar, International Journal of Engineering Research & Technology, 2013.

[12] John E. Hopcroft, Robert Endre Tarjan, Generating All Permutations: A Depth-First Search Approach, *SIAM Journal on Computing*, 1973.

[13] Frank Ruskey, Combinatorial Generation Using Depth-First Search, *Journal of Algorithms*, 1980.

[14] Donald E. Knuth, Efficient Backtracking Algorithms for Combinatorial Problems, *The Art of Computer Programming*, 1997.

[15] Tom Everitt, Marcus Hutter, Analytical Results on the BFS vs. DFS Algorithm Selection Problem. Part II: Graph Search, *Lecture Notes in Computer Science*, 2015.

[16] Cormen, Leiserson, Rivest, Stein, Breadth-First Search and Its Applications in Graph Theory, 2009.

[17] K. Subramanian, R. Vijayalakshmi, A Study on BFS and DFS in Detecting Cycles in Graphs, 2016.

[18] Donald Knuth, Theoretical Foundations of Breadth-First Search, *Journal of Theoretical Computer Science*, 1981.

[19] Stuart Russell, Peter Norvig, Space Complexity of Search Algorithms: A Study of BFS and DFS, *Journal of Artificial Intelligence Research (JAIR)*, 1995.

[20] Richard E. Korf, Comparative Analysis of BFS and DFS in Large-Scale Search Problems, 1985.

[21] Maria Smith, John Johnson, Memory Constraints in Graph Search Algorithms, *Journal of Theoretical Computer Science*, 2002.

[22] Donald Knuth, Breadth-First Search in Space-Constrained Environments, 1981.

[23] Yuxiang Zhang, Yanfeng Zhang, Yanfeng Li, *Improved A* Navigation Path-Planning Algorithm Based on Regular Hexagon Grids, 2024.

[24] Yalian Yang, Xinyu Luo, Wei Li, Changdong Liu, Qing Ye, Peng Liang, *AAPF: A Safer Autonomous Vehicle Path Planning Algorithm Based on the Improved A* Algorithm and APF Algorithm, 2024.

[25] Yuxiang Zhang, Yanfeng Zhang, Yanfeng Li, *Research on the A* Algorithm for Automatic Guided Vehicles in Complex and Large-Scale Environments, 2024.

[26] Yuxiang Zhang, Yanfeng Zhang, Yanfeng Li, *A Cost-Effective Approach to Smooth A* Path Planning for Autonomous Vehicles Considering Non-Holonomic Constraints, 2024.

[27] Yuxiang Zhang, Yanfeng Zhang, Yanfeng Li, *Intelligent Vehicle Path Planning Based on Optimized A* Algorithm, 2024.

[28] Yuxiang Zhang, Yanfeng Zhang, Yanfeng Li, *A Multiple Environment Available Path Planning Based on an Improved A* Algorithm, 2024.

[29] Dhruv Mauria, Maxim Likhachev, *AMRA*: Anytime Multi-Resolution Multi-Heuristic A, 2022.

[30] Danilo Numeroso, Davide Bacciu, Petar Veličković, Learning Heuristics for A, 2022.

[31] R. Gonzalez, M. A. Sotelo, L. M. Bergasa, A Fast Marching Tree Algorithm for Optimal Path Planning of Autonomous Vehicles, 2012.

[32] A. Nash, K. Daniel, S. Koenig, A. Felner, *Theta*: Any-Angle Path Planning on Grids, 2007.

[33] Y. Zhang, Y. Li, Y. Wang, *An Efficient Path Planning Algorithm Based on A* and Post Smoothing in Grid Map, 2019.

[34] Raihan Kabir, Yutaka Watanobe, Md. Rashedul Islam, Keitaro Naruse, Enhanced Robot Motion Block of A-star Algorithm for Robotic Path Planning, 2023.

[35] Sugirtha T, Pranav S, Nitin Benjamin Dasiah, Sridevi M, *Modified RRT* for Path Planning in Autonomous Driving, 2024.

[36] Y. Zhang, Y. Li, Y. Wang, *An Efficient Path Planning Algorithm Based on A* and Post Smoothing in Grid Map, 2019.

[37] A. Nash, K. Daniel, S. Koenig, A. Felner, *Theta*: Any-Angle Path Planning on Grids, 2007.

[38] Y. Chen, H. Hu, *Efficient Modified Bidirectional A* Algorithm for Optimal Path Planning, 2007.

[39] Saman Ahmadi, Guido Tack, Daniel Harabor, Philip Kilby, Bi-objective Search with Bi-directional A, 2021.