LNS-based ACO with GA repair mechanic algorithm for solving large scale TSP instances

Trần Thành Đạt
VNU University of Engineering and Technology
ttdat.vnu@gmail.com

Tóm tắt. Bài toán Người chào hàng (Travelling Salesman Problem - TSP) là một bài toán tối ưu tổ hợp kinh điển với nhiều ứng dụng thực tiễn. Tuy nhiên, các thuật toán hiện có vẫn đối mặt với thách thức về cả hiệu suất tính toán và chất lượng lời giải. Nghiên cứu này phân tích đặc điểm của các thuật toán TSP hàng đầu hiện nay, từ đó đề xuất một thuật toán giải mới dựa trên mô hình Tìm kiếm lân cận rộng (Large Neighborhood Search - LNS). Mô hình của thuật toán mới là quá trình lặp lại phá hủy và sửa lại lời giải nhằm tìm kiếm lời giải tối ưu. Đặc điểm nổi trội của thuật toán mới là kĩ thuật sử dụng Tối ưu đàn kiến (Ant Colony Optimization - ACO) với tính chất học tăng cường nhằm giữ lại các cạnh tốt trong quá trình phá hủy lời giải, sau đó nén bài toán lại thành bài toán con rồi sử dụng Giải thuật di truyền (Genetic Algorithm - GA) để xây dựng lại lời giải nhằm tìm kiếm lân cận tốt. Thực nghiệm trên các bộ dữ liệu lớn (kích thước lên đến một trăm nghìn đỉnh) cho thấy thuật toán đề xuất đạt được chất lượng lời giải cận tối ưu với hiệu suất vượt trội, với sai số không quá 0.1% trong thời gian không quá 30 phút, vượt trội so với mọi thuật toán ACO hiện tại và cạnh tranh với những phương pháp tiên tiến nhất của TSP.

1. Giới thiệu

Tối ưu hóa là một nhánh cốt lõi của vận trù học, trong đó các bài toán tối ưu hóa được có thể được chia thành hai loại: tối ưu hóa liên tục và tối ưu hóa rời rạc, tùy thuộc vào việc các biến quyết định là liên tục hay rời rạc. Bài toán tối ưu hóa rời rạc, còn được gọi là bài toán tối ưu tổ hợp (TƯTH), hầu hết là những bài toán thuộc lớp NP-Khó. Các thuật toán chính xác như Quy hoạch tuyến tính [1], Nhánh và cận [2], hay Quy hoạch động [3] có thể được áp dụng để tìm lời giải tối ưu toàn cục cho các bài toán TƯTH NP-khó. Tuy nhiên, khi kích thước bài toán tăng lên, độ phức tạp tính toán của các thuật toán này thường tăng theo cấp số mũ, dẫn đến việc chúng trở nên không khả thi trên quy mô lớn. Bởi vậy, những mô hình giải thuật mô phỏng tự nhiên (meta-heuristic) được đề xuất và đã đưa ra nhiều kết quả khả quan [4].

Bài toán Người chào hàng (TSP) là trong những bài toán TƯTH kinh điển và được nghiên cứu nhiều nhất trong lĩnh vực này [5]. Bài toán có thể miêu tả như sau: Cho đồ thị $G = (v, \Omega, d)$ trong đó $v = \{v_i\}_{i=1}^{n}$ là tập đỉnh (thành phố), $\Omega = (\{i, j\} | (i, j) \in v \times v)$ là tập cạnh (đường đi trực tiếp giữa các thành phố), và $d : \Omega \mapsto \mathbb{N}$ là hàm giá trị của mỗi cạnh (i, j), thể hiện độ dài đường đi trực tiếp tương ứng. TSP là bài toán chuẩn để đánh giá hiệu quả của các phương pháp giải TƯTH. Hơn nữa, bài toán này có nhiều ứng dụng thực tiễn quan trọng, bao gồm: tối ưu hóa vận tải và lập kế hoạch trong logistics [6], giải trình tự ADN trong tin sinh học, và tối ưu hóa quy trình sản xuất [7], chẳng hạn như lập trình đường khoan trên bảng mạch in [8]. Nhiều thuật toán được đề xuất đề giải TSP với quy mô vừa và nhỏ đã đem lại những kết quả tốt, ví dụ như Tối ưu đàn kiến [9], Giải thuật di truyền [10], hay Lin-Kernighan heuristic (LKH) [11].

Với sự phát triển của công nghệ hiện đại, nhu cầu giải bài toán Người chào hàng (TSP) trên quy mô lớn ngày càng trở nên cấp thiết. Một mặt, các lĩnh vực ứng dụng TSP trong thực tiễn ngày càng mở rộng và trở nên phức tạp hơn. Mặt khác, TSP quy mô lớn đóng vai trò quan trọng trong khoa học máy tính và vận trù học, và việc nghiên cứu các phương pháp giải bài toán

này có thể mang lại những đột phá quan trọng trong cả hai lĩnh vực [12]. Do đó, nhiều chiến lược thuật toán đã được đề xuất, chủ yếu tập trung vào việc điều chỉnh và cải tiến các thuật toán tiên tiến nhất để có thể xử lý các bài toán quy mô lớn hiệu quả hơn [13]. Một số phương pháp tiêu biểu bao gồm FACO (Focused ACO [14]) nhằm cải thiện ACO, EAX (Edge Assembly Crossover [15]) cho GA, và POPMUSIC (Partial Optimization Metaheuristic Under Special Intensification [16]) cho LKH. Tuy nhiên, khi áp dụng trên các bộ dữ liệu có quy mô rất lớn (hàng trăm nghìn đỉnh), các thuật toán này vẫn gặp phải những hạn chế nhất định về hiệu suất và chất lượng lời giải.

Trong nghiên cứu này, dựa trên phân tích đặc điểm của các thuật toán tiên tiến hiện nay, chúng tôi đề xuất một mô hình giải thuật mới cho TSP, kết hợp ý tưởng của mô hình Tìm kiếm lân cận rộng (LNS - Large Neighborhood Search [17]) với ACO và GA. Hiệu quả của thuật toán đề xuất được đánh giá trên các bộ dữ liệu chuẩn như TSPLIB [18] và DIMACS [19], cho thấy những cải thiện đáng kể so với các phương pháp hiện hữu.

2. Một số phương pháp hiện thời

2.1. Giải thuật chính xác

Bài toán TSP thuộc lớp tối ưu tổ hợp (TƯTH), do đó có thể được giải bằng các bộ giải dựa trên Quy hoạch tuyến tính [1]. Hiện nay, Concorde [20] được xem là bộ giải chính xác hàng đầu cho TSP, có khả năng tìm lời giải tối ưu cho các bộ dữ liệu có cấu trúc và quy luật lên đến hàng chục nghìn đỉnh. Tuy nhiên, đối với các bộ dữ liệu ngẫu nhiên không có cấu trúc rõ ràng, Concorde gặp nhiều khó khăn trong việc tìm lời giải tối ưu. Trên các bài toán ngẫu nhiên có kích thước chỉ vài nghìn đỉnh, Concorde thường không thể tìm ra lời giải trong thời gian hợp lý [21].

2.2. Giải thuật Heuristic

Khi các bộ dữ liệu có quy mô quá lớn khiến các bộ giải chính xác không thể tìm ra lời giải trong thời gian hợp lý, các giải thuật heuristic trở thành

phương pháp thay thế hiệu quả [22]. Các thuật toán heuristic mạnh nhất cho TSP có thể được chia thành hai nhóm chính: các thuật toán dựa trên tìm kiếm cục bộ và các thuật toán mô phỏng quá trình tự nhiên dựa trên quần thể.

2.2.1. Tìm kiếm cục bộ

Tìm kiếm cục bộ (Local Search) là phương pháp khởi tạo từ một lời giải ban đầu và cải tiến dần dần bằng cách di chuyển đến các lời giải tốt hơn trong lân cận của lời giải hiện tại. Để thực hiện phương pháp này, thuật toán cần xác định cấu trúc lân cận, thường được xây dựng bằng cách thay đổi một số ít thành phần trong lời giải. Một cách tiếp cận phổ biến là k-opt, trong đó thuật toán thay thế k cạnh trong lời giải hiện tại bằng k cạnh khác để tạo ra lời giải mới.

Mặc dù tìm kiếm cục bộ có thể cải thiện đáng kể chất lượng lời giải, nhưng phương pháp này thường mắc kẹt tại cực trị địa phương. Hơn nữa, tốc độ hội tụ của thuật toán cũng là một yếu tố quan trọng, ảnh hưởng đến hiệu quả tìm kiếm lời giải tối ưu.

2.2.2. Mô phỏng tự nhiên dựa trên quần thể

Tối ưu đàn kiến: Tối ưu đàn kiến (Ant Colony Optimization - ACO) là một thuật toán mô phỏng tự nhiên do Dorigo đề xuất vào năm 1991, được nghiên cứu và ứng dụng rộng rãi trong nhiều bài toán tối ưu tổ hợp [23]. ACO dựa trên hành vi tìm kiếm thức ăn của quần thể kiến, trong đó cấu trúc vết mùi được lưu lại trên các cạnh và tận dụng bởi các cá thể kiến khác. Vết mùi này được bổ sung và bay hơi theo cơ chế tương tự như học tăng cường, giúp ưu tiên những cạnh tiềm năng tốt nhất. [24]

```
ACO(#iterations, #ants, ρ):
    Initialize global_best
    for i = 1, ..., #iterations do
        for j = 1, ..., #ants do
            Construct route_j
        iteration_best ← select_shortest(ants)
        if iteration_best < global_best then
            global_best ← iteration_best
        Evaporate phermone based on ρ
        Deposit pheremone based on iteration_best or global_best
```

Giải thuật 1: ACO

Đối với bài toán TSP trên bộ dữ liệu quy mô lớn, thuật toán ACO hiệu quả nhất hiện nay là Focused ACO (FACO [14], [25]). FACO kết hợp nhiều cải tiến tiên tiến trong ACO và áp dụng hạn chế số cạnh mới, trong đó mỗi cá thể kiến chỉ sửa đổi một phần nhỏ của lời giải thay vì tái tạo hoàn toàn. Bằng việc giới hạn số cạnh mới xuất hiện, FACO có thể sử dụng tìm kiếm cục bộ 2-opt trên mọi lời giải mới tìm được, một điều không thể trước đó vì 2-opt là một giải thuật có chi phí tính toán rất cao. FACO đạt được kết quả khả quan, với chất lượng lời giải sai lệch dưới 1% so với giá trị tối ưu hiện có, ngay cả trên các bộ dữ liệu có hàng chục nghìn đến hàng trăm nghìn đỉnh, trong thời gian ngắn. Tuy nhiên, FACO gặp khó khăn trong hội tụ đến lời giải tốt nhất, và tốc độ cải thiện lời giải giảm đáng kể khi thời gian chạy tăng lên.

```
ACO(#iterations, #ants, ρ, MNE):
    Initialize global_best
    for i = 1, ..., #iterations do
        for j = 1, ..., #ants do
            Copy route_j from source_solution
            Modify route_j with limit on MNE
        iteration_best ← select_shortest(ants)
        if iteration_best < global_best then
            global_best ← iteration_best
        Evaporate phermone based on ρ
        Choose source_solution based on iteration_best or global_best
        Deposit pheremone based on source_solution
```

Giải thuật 2: Focused ACO

Giải thuật di truyền: Giải thuật di truyền (Genetic Algorithm - GA) là một trong những thuật toán mô phỏng tự nhiên đầu tiên, được Holland đề xuất vào đầu những năm 1970 [10], [26]. GA hoạt động trên một tập quần thể lời giải, trong đó các cá thể trải qua quá trình lai tạo và chọn lọc nhằm tạo ra thế hệ kế tiếp với chất lượng cao hơn.

```
GA(#population, #offspring):
    Initialize population
    while not_terminated() do
        Randomly permutate population for mating
        for i = 1, ..., #population - 1 do
            Generate offspring based on population[i] and population[i+1]
            Select offspring for next generation
```

Giải thuật 3: GA

Với những cải tiến như EAX (Edge Assembly Crossover [15]), GA đạt được chất lượng lời giải gần tối ưu trên các bộ dữ liệu vừa (lên đến hàng nghìn đỉnh) với hiệu suất đáng kinh ngạc. Tuy nhiên, đối với các bộ dữ liệu lớn (vài chục nghìn đỉnh trở lên), GA-EAX gặp khó khăn trong việc hội tụ lời giải trong thời gian hợp lý.

2.3. Giải thuật dựa trên Học Máy

Trong những năm gần đây, học máy (Machine Learning - ML), đặc biệt là học sâu (Deep Learning - DL), đã phát triển nhanh chóng và vượt trội so với các phương pháp truyền thống trong nhiều lĩnh vực như xử lý ngôn ngữ tự nhiên (NLP), xử lý ảnh, v.v. Nhờ những thành tựu này, nhiều nghiên cứu đã đề xuất ứng dụng ML vào các bài toán tối ưu tổ hợp (TƯTH), bao gồm TSP.

Các phương pháp này thường đào tạo mô hình trí tuệ nhân tạo trên tập dữ liệu mẫu, sau đó sử dụng mô hình đã học để giải quyết các bộ dữ liệu mới. Tuy nhiên, cho đến nay, các thuật toán dựa trên ML vẫn chưa đủ sức cạnh tranh với các phương pháp tối ưu tổ hợp truyền thống tiên tiến nhất, bất kể kích thước bài toán. Theo hiểu biết của chúng tôi, chưa có nghiên cứu nào sử dụng thuật toán thuần ML để giải TSP trên bộ dữ liệu lớn hơn 20,000 đỉnh. Ngay cả trên các tập dữ liệu nhỏ hơn, sai số của các phương pháp ML vẫn cao hơn vài phần trăm so với các thuật toán truyền thống tối ưu nhất [27], [28], [29].

3. Công trình liên quan

3.1. Tìm kiếm lân cận rộng

Tìm kiếm lân cận rộng (Large Neighborhood Search - LNS) là một thuật toán metaheuristic được đề xuất bởi Shaw (1998). Phương pháp này hoạt động bằng cách cải tiến lời giải ban đầu thông qua cơ chế "phá hủy" và "sửa lại" nhằm khám phá không gian tìm kiếm hiệu quả hơn [17].

Mọi thuật toán LNS đều dựa trên quan sát rằng việc mở rộng cấu trúc lân cận trong tìm kiếm cục bộ có thể giúp tìm ra lời giải tốt hơn. Tuy nhiên, tìm kiếm lân cận rộng thuần túy thường có chi phí tính toán rất cao, do đó, nhiều kỹ thuật định hướng lân cận đã được đề xuất để cải thiện hiệu suất.

Trong LNS, cấu trúc lân cận được xác định bởi phương pháp phá hủy và sửa lại lời giải. Cụ thể:

- Phương pháp phá hủy ảnh hưởng đến kích thước và chất lượng của tập lân cận, quyết định phạm vi tìm kiếm.

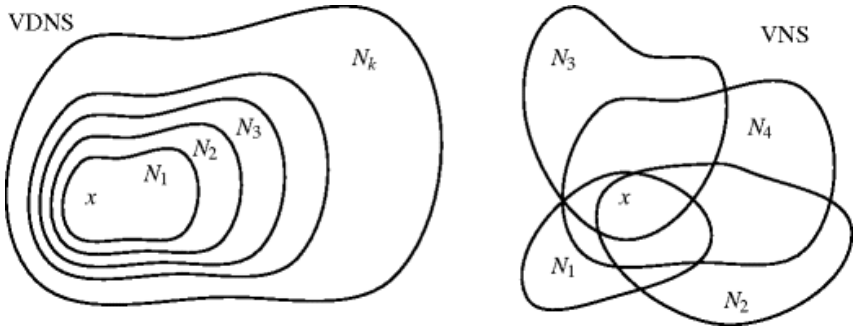
- Phương pháp sửa lại đóng vai trò xây dựng lại lời giải từ phần bị phá hủy, tác động trực tiếp đến chất lượng lời giải cuối cùng.

Có thể thấy, khi phương pháp sửa lại được cố định, hiệu suất và chất lượng lời giải của thuật toán sẽ phụ thuộc vào mức độ phá hủy trong mỗi vòng lặp.

- Nếu phá hủy quá nhiều, phương pháp sửa lại sẽ cần nhiều thời gian hơn để tái cấu trúc lời giải, khiến thuật toán chậm hơn.
- Ngược lại, nếu phá hủy quá ít, thuật toán dễ bị kẹt trong cực trị địa phương (local optima) và khó cải thiện lời giải.

Bên cạnh đó, bản chất của phần bị phá hủy cũng có ảnh hưởng lớn đến chất lượng lời giải, do đó, việc lựa chọn chiến lược phá hủy phù hợp là một yếu tố quan trọng trong thiết kế thuật toán LNS hiệu quả.

Trong LNS, hai kĩ thuật thường được sử dụng là Variable-Depth Neighborhood Search (VDNS) và Variable Neighborhood Search (VNS). Hai kĩ thuật này cho phép lận cận được biến đổi theo thời gian, từ đó giúp LNS tránh khỏi cực trị địa phương.



Minh họa 1: Lân cận trong VDNS và VNS. [17]

3.2. Focused ACO

Focused ACO (FACO [14], [25]), như đã đề cập, là một biến thể của ACO có cơ chế giới hạn số lượng thay đổi mà mỗi con kiến có thể thực hiện trên lời giải gốc, sau đó áp dụng tìm kiếm cục bộ trên các cạnh bị thay đổi. Nhờ chiến lược này, FACO đạt chất lượng lời giải khá tốt trên các bộ dữ liệu lớn trong thời gian ngắn, vượt qua các thuật toán dựa trên ACO khác như [30], [31], [32]. Tuy nhiên, thuật toán gặp khó khăn trong việc hội tụ đến lời giải tối ưu khi đạt ngưỡng nhất định và dễ mắc kẹt trong cực trị địa phương (local

optima). Bảng 1 trình bày so sánh chất lượng lời giải của FACO với ACO trên các bộ dữ liệu nhỏ.

Từ góc độ Tìm kiếm lân cận rộng (LNS), FACO có thể được xem như một phương pháp phá hủy một phần lời giải cũ, trong đó quá trình chọn cạnh được điều hướng theo tiêu chí của ACO, và sau đó tái cấu trúc lời giải bằng phép toán 2-opt. Trong nghiên cứu gốc, tác giả đề xuất giới hạn số lượng cạnh mới ở mức khá nhỏ $(8 \rightarrow 32)$, điều này có thể là một nguyên nhân quan trọng khiến FACO không thể hội tụ đến lời giải tối ưu.

3.3. GA-EAX

GA-EAX [15] là một trong những thuật toán hiệu quả nhất hiện nay cho TSP. Thuật toán này cải tiến Giải thuật di truyền (GA) bằng phương pháp lai tạo EAX, trong đó sử dụng kỹ thuật lựa chọn chu trình AB để tạo ra các cá thể con có chất lượng cao từ các cá thể cha. Tuy nhiên, khi áp dụng trên các bộ dữ liệu lớn, GA-EAX gặp khó khăn trong việc hội tụ. Do đó, GA-EAX chỉ thực sự phù hợp để tìm kiếm lời giải tối ưu trên các bộ dữ liệu vừa và nhỏ.

4. Đề xuất mô hình thuật toán mới

Trong TSP, có thể định nghĩa phá hủy là việc loại bỏ một tập cạnh khỏi lời giải hiện tại, còn sửa lại là quá trình giải TSP trên bài toán con nhận được, trong đó các cạnh không bị xóa sẽ được nối lại để tạo thành một cạnh duy nhất.

Có thể thấy, tập cạnh bị xóa ảnh hưởng đến hiệu quả thuật toán theo hai khía cạnh chính. Thứ nhất, kích thước của tập cạnh bị xóa tác động trực tiếp đến chi phí tính toán của việc giải bài toán con. Thứ hai, cấu trúc của tập cạnh bị xóa quyết định chất lượng của lân cận được tìm kiếm. Ví dụ, trong TSP trên mặt phẳng Euclid, nếu các cạnh bị xóa cách nhau quá xa, bài toán con thu được sẽ không có nhiều ý nghĩa trong việc cải thiện lời giải. Một hướng tiếp cận tự nhiên là xóa các cạnh gần nhau trong cùng một cụm, giúp tăng khả năng cải thiện lời giải.

Tuy nhiên, nếu số cạnh bị xóa quá nhỏ so với kích thước bài toán, không gian lân cận của thuật toán bị thu hẹp, khiến thuật toán dễ mắc kẹt trong cực trị địa phương mà khó thoát ra. Một giải pháp là cho phép giữ lại một số cạnh trong cụm bị xóa, đồng thời vẫn đảm bảo tổng số cạnh bị loại bỏ đạt đến một ngưỡng nhất định. Khi một cạnh được giữ lại, thuật toán sẽ phải xóa một cạnh khác xa hơn, giúp mở rộng không gian lân cận được tìm kiếm mà không làm tăng đáng kể chi phí tính toán. Cách tiếp cận này tương tự Mô hình Tìm kiếm Lân cận Biến đổi (Variable Neighborhood Search - VNS). Tuy nhiên, việc quyết định cạnh nào được giữ lại là một bài toán khó, đòi hỏi sự cân bằng giữa việc mở rộng lân cận và đảm bảo cấu trúc không thay đổi quá nhiều. Ngoài ra, các cạnh được giữ lại cũng nên là những cạnh có khả năng thuộc về lời giải tối ưu cao.

Dựa trên mô hình LNS (VNS) và những đặc điểm của FACO, chúng tôi đề xuất một thuật toán mới để giải TSP. Đặc điểm nổi bật của thuật toán là sử dụng ACO để phá hủy lời giải cũ, tạo ra bài toán con có quy mô nhỏ, sau đó sử dụng GA-EAX để tối ưu lời giải trên bài toán con. Sau khi bài toán con được giải, lời giải sẽ được tích hợp trở lại vào bài toán gốc, và quá trình lặp lại. Thuật toán tận dụng vết mùi của ACO để phá hủy lời giải một cách hiệu quả, giúp các cạnh xuất hiện nhiều trong lời giải tốt có nhiều khả năng được giữ lại. Điều này khiến không gian lân cận dần mở rộng theo thời gian, giúp thuật toán thoát khỏi cực trị địa phương dễ dàng hơn. Mã giả tổng quát của thuật toán được trình bày trong Giải thuật 4.

```
Algorithm(CDE):
    Initialize global_best
    for i = 1, ..., #iterations do
        for j = 1, ..., #ants do
            Copy route_j from source_solution
            Destroy route_j based on CDE and pheremone
            Repair route_j subproblem using GA
        iteration_best ← select_shortest(ants)
        if iteration_best < global_best then
            global_best ← iteration_best
        Evaporate phermone based on ρ
        Choose source_solution based on iteration_best or global_best
        Deposit pheremone based on source_solution
```

Giải thuật 4: Thuật toán đề xuất

Cần lưu ý rằng, mặc dù thuật toán đề xuất sử dụng mô hình ACO, nhưng cách tiếp cận của nó hoàn toàn khác biệt so với ACO truyền thống khi áp dụng vào TSP. Trong ACO truyền thống, các kiến xây dựng lời giải mới dựa trên vết mùi và thông tin heuristic. Khi đó, vết mùi trên cạnh (i, j) phản ánh kinh nghiệm tích lũy của đàn kiến về chất lượng của cạnh này.

Thuật toán đề xuất cũng tận dụng vết mùi, nhưng thay vì sử dụng để xây dựng lời giải mới, vết mùi được khai thác nhằm phá hủy lời giải hiện tại một cách có định hướng. Nói cách khác, thuật toán đề xuất sử dụng ACO như một cơ chế để giải một lớp bài toán con của TSP: xác định tập cạnh cần loại bỏ khỏi lời giải hiện tại, sao cho sau khi tái tối ưu hóa, khả năng thu được lời giải tốt hơn là cao nhất.

Chính vì sự khác biệt này, các tham số truyền thống của ACO trong TSP không thể được áp dụng trực tiếp vào thuật toán đề xuất. Do đó, cần tiến hành thử nghiệm và lựa chọn các tham số tối ưu riêng biệt để phù hợp với mô hình này.

4.1. Phương pháp khởi tạo lời giải ban đầu

Trong thuật toán đề xuất, việc khởi tạo lời giải ban đầu đóng vai trò quan trọng trong hiệu suất của thuật toán. Tính chất thuật toán yêu cầu lời giải ban đầu cần phải có chất lượng đủ tốt với chi phí tính toán thấp, để không mất thời gian tìm kiếm lân cận rộng khi lời giải vẫn đang có thể dễ dàng cải thiện. Điều này đòi hỏi giải thuật sử dụng để khởi tạo phải có tốc độ hội tụ lời giải nhanh với chất lượng lời giải chấp nhận được. Dựa trên công trình nghiên cứu trước đó ([33], 2024), chúng tôi đề xuất sử dụng giải thuật AACO - một biến thể của FACO được chúng tôi cải tiến bằng một số cải tiến như cập nhật Max-Min trơn [34] và Ép lời giải sớm. Thực nghiệm từ [33] cho thấy AACO trong thời gian ngắn có thể hội tụ đến lời giải có chất lượng khá tốt, với sai số dưới 1% cho những bộ dữ liệu hàng chục nghìn đỉnh trong thời gian chạy một vài phút. Dù khó tìm được lời giải cận tối ưu, đặc điểm hội tụ nhanh của thuật toán khiến nó phù hợp để làm phương pháp khởi tạo lời giải ban đầu trong thuật toán mới được đề xuất.

4.2. Phương pháp phá hủy lời giải

Thuật toán đề xuất phương pháp phá hủy lời giải như sau: bắt đầu từ một đỉnh ngẫu nhiên s, thuật toán lần lượt xóa CDE (Count Destroy Edges) cạnh. Các cạnh được duyệt theo thứ tự tăng dần của khoảng cách đến s. Tại bước thứ t, kiến k có thể giữ lại cạnh (i, j) với xác suất được định nghĩa như sau:

$$p_{ij}^k = \frac{\tau_{ij}(t)}{\tau_{\max}}$$

Trong đó, τ_{ij} là thông tin vết mùi được lưu trên cạnh (i, j), và $\tau_{\max}$ là cận trên của vết mùi. Trong cài đặt, lấy $\tau_{\max} = 1.0$.

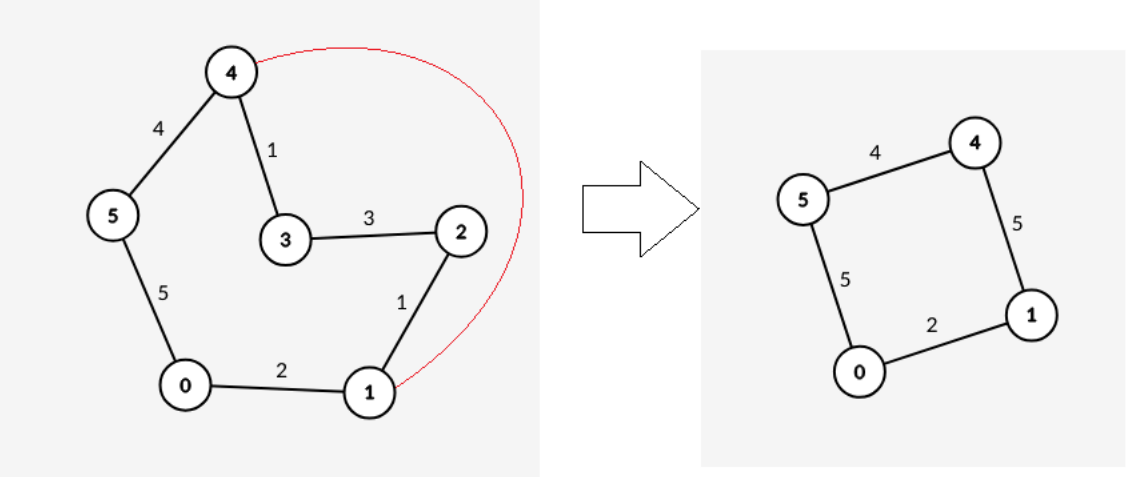
Sau khi loại bỏ đủ số cạnh, thuật toán tiến hành nén bài toán gốc thành một bài toán con, trong đó các đỉnh có thể thuộc tập hai đầu mút của các cạnh bị xóa. Các đường đi không bị xóa giữa hai đỉnh đầu mút sẽ được nén thành một cạnh mới, với trọng số bằng tổng trọng số của các cạnh trên đường đi ban

đầu. Dễ dàng nhận thấy rằng, nếu số cạnh bị xóa là CDE, thì kích thước tối đa (số đỉnh) của bài toán con mới được giới hạn với cận trên là $2 * \text{CDE}$.

Một quan sát quan trọng là nếu vết mùi trong đàn kiến dần hội tụ về lời giải tốt, thì ở các giai đoạn sau của thuật toán, những cạnh có chất lượng cao sẽ có xác suất được giữ lại cao hơn. Nhờ đó, quá trình phá hủy lời giải có thể mở rộng không gian lân cận được tìm kiếm, giúp thuật toán thoát khỏi tối ưu cục bộ do giới hạn trong lân cận gần.

4.3. Phương pháp sửa lại lời giải

Sau khi xóa đủ số cạnh, phần còn lại của lời giải được nén lại thành một bài toán con có kích thước nhỏ hơn, trong đó, các cạnh liên tiếp được giữ lại của lời giải cũ được nén lại thành một cạnh có tổng trọng số bằng độ dài đường đi và được cố định lại. Sau khi đường đi được nén lại, bài toán con sẽ được giải quyết trên tập đỉnh của đường đi đó, những cạnh được cố định sẽ bắt buộc phải có trong lời giải đích.



Hình 2: Một đường đi được nén lại sau khi cố định 3 cạnh

Khi CDE có giá trị nhỏ, bài toán con thu được có thể được giải quyết bằng bất kỳ bộ giải TSP nào có hiệu suất và chất lượng đủ tốt. Trong thuật toán đề xuất, một phiên bản của GA-EAX được sử dụng, trong đó cho phép cố định một số cạnh. Với CDE nhỏ, chi phí tính toán cho quá trình sửa lời giải có thể coi là gần như hằng số.

4.4. Phương pháp cập nhật vết mùi

Ở ACO-TSP thông thường, phương pháp cập nhật vết mùi nổi trội nhất là MMAS (Min-Max Ant System). Trong đó với mỗi lần lặp, thuật toán sẽ bổ sung vết mùi cho mọi cạnh nằm trong lời giải tốt nhất của lần lặp đó hoặc lời giải tốt nhất toàn cục theo công thức sau, trong đó $w(t)$ là đường đi được chọn để bổ sung vết mùi tại thời điểm t:

$$\tau_{i,j} := \begin{cases} \tau_{i,j} * (1-\rho) + \frac{1}{\text{route\_cost}} & \text{nếu } (i,j) \in w(t) \\ \tau_{i,j} * (1-\rho) & \text{nếu không} \end{cases}$$

Thuật toán đề xuất không thể sử dụng phương pháp cập nhật vết mùi trong ACO-TSP thông thường. Lý do là với mỗi lần lặp, thuật toán chỉ phá hủy và sửa lại *một phần* của lời giải trước đó. Nếu áp dụng trực tiếp phương pháp cập nhật của ACO-TSP, tất cả mọi cạnh của lời giải đều sẽ được bổ sung, kể cả những cạnh không hề bị ảnh hưởng bởi việc phá hủy và sửa lại. Ngoài ra, thuật toán đề xuất sử dụng vết mùi làm xác xuất để giữ lại cạnh, nên việc bổ sung vết mùi bằng nghịch đảo độ dài đường đi là không hợp lý. Tạm thời, thuật toán đề xuất phương pháp cập nhật vết mùi như sau:

$$\tau_{i,j} := \begin{cases} \tau_{i,j} * (1-\rho) + g & \text{nếu } (i,j) \in w(t) \wedge (i,j) \notin w(t-1) \\ \tau_{i,j} * (1-\rho) & \text{nếu không} \end{cases}$$

Trong đó $w(t)$ là đường đi tốt nhất tại thời điểm t, và g là tham số thuật toán. Với phương pháp cập nhật vết mùi trên, chỉ những cạnh mới bị điều chỉnh được cập nhật vết mùi, từ đó loại bỏ tình trạng cạnh kém được bổ sung khả năng giữ lại. Ngoài ra, do ban đầu thuật toán chỉ sửa một phần nhỏ của lời giải, thuật toán đề xuất đặt tham số ρ rất nhỏ so với ACO gốc, nhằm giúp cho những cạnh được sửa giữ được được vết mùi của chúng lâu hơn.

4.5. Lựa chọn tham số

Trong thuật toán đề xuất, có hai tham số mới: CDE quyết định số cạnh bị xóa, và g là lượng vết mùi được bổ sung vào các cạnh mới. Với CDE, nếu quá lớn có thể khiến việc sửa lại lời giải mất nhiều thời gian, còn nếu quá nhỏ thì lời khó có thể được cải thiện. Tương tự, tham số g quyết định tốc độ tăng số cạnh được giữ lại của thuật toán. Dựa trên kết quả thực nghiệm, thuật toán đề xuất đặt tham số như Bảng 1:

Tham số	Mô tả	Giá trị mặc định
m	Số lượng kiến	5
ρ	Tham số bay hơi	0.01
g	Tham số bổ sung vết mùi	0.01
CDE	Số cạnh bị xóa	500
#iterations	Số lần lặp	500

Bảng 1: Tham số thuật toán đề xuất

5. Thực nghiệm

Động lực chính của thuật toán mới được đề xuất là giải quyết và tìm lời giải cho các bộ dữ liệu TSP với quy mô lớn, lên đến hàng chục nghìn đỉnh với chất lượng lời giải cận tối ưu trong thời gian ngắn.

Để kiểm chứng sự hiệu quả của thuật toán mới đề xuất, thuật toán mới (được đặt tên là HLAGA - Hybrid LNS-based ACO with GA repair mechanic Algorithm) được cài đặt bằng ngôn ngữ C++ và chạy trên một số lượng những bộ dữ liệu cỡ lớn. Chương trình được chạy trên máy tính cá nhân với cấu hình CPU i5-13400F @ 2.50Ghz và 32 GB RAM. Kết quả thực nghiệm sau đó được so sánh với một số thuật toán tiên tiến nhất hiện tại của giải thuật ACO và GA.

Trong các bảng thực nghiệm tiếp theo, các bộ dữ liệu được chạy và đánh giá qua 10 lần chạy, "%" thể hiện sai số của kết quả so với lời giải tốt nhất hiện có của mỗi bộ dữ liệu. Sai số được tính bằng công thức sau:

$$\text{Sai số} = \frac{\text{cost} - \text{best known}}{\text{best known}} * 100\%$$

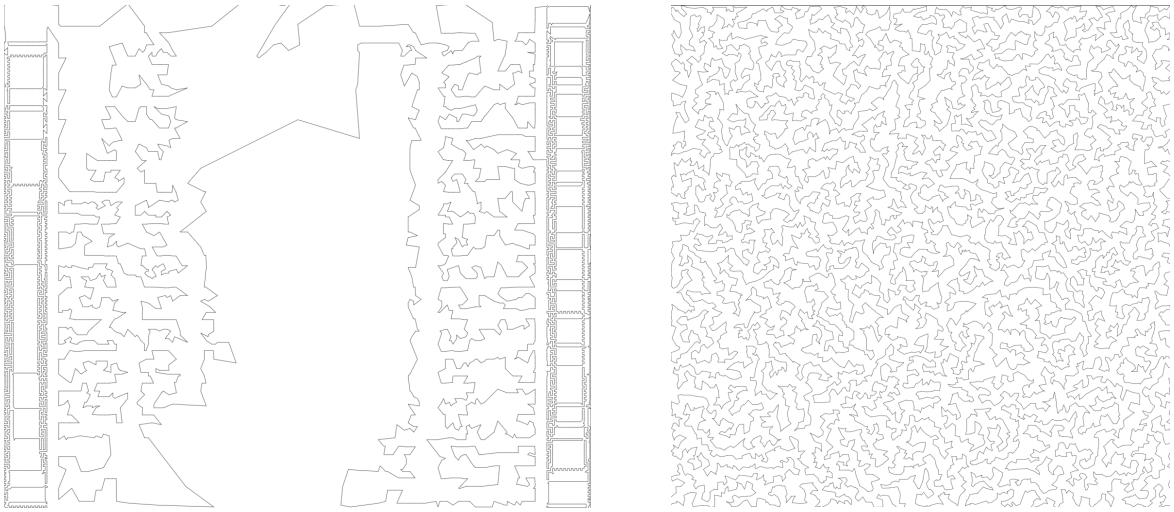
Đầu tiên là bảng kết quả thực nghiệm của thuật toán đề xuất trên 5 bộ dữ liệu từ TSPLIB [18] và 7 bộ dữ liệu từ DIMACS [19]. Thông số được cài như Bảng 1 mục 4.5.

Instances	BKS	HLAGA			
		Best	Worst	Mean	Time
pla7397	23260728	23280007 **(0.066%)**	23280151 **(0.083%)**	23280296 **(0.084%)**	827
rl11849	923288	924099 (0.082%)	924283 (0.107%)	924163 (0.095%)	1143
usa13509	19982859	19984790 (0.000%)	19994654 (0.005%)	19989689 (0.003%)	593
d15112	1573084	1573248 (0.001%)	1573419 (0.021%)	1573360 (0.017%)	827
d18512	645238	645263 (0.000%)	645313 (0.001%)	645303 (0.001%)	1178
E10k.0	71865826	71873310 (0.010%)	71887372 (0.029%)	71878868 (0.018%)	609
E10k.1	72031630	72046421 (0.020%)	72056036 (0.033%)	72051400 (0.027%)	687
E10k.2	71822483	71828497 (0.008%)	71838510 (0.022%)	71833659 (0.015%)	635
E31k.0	127281803	127330860 (0.038%)	127359197 (0.060%)	127348519 (0.052%)	782
E31k.1	127452384	127512737 (0.047%)	127537011 (0.066%)	127527456 (0.058%)	791
E100k.0	225783795	225972765 (0.083%)	225992452 (0.092%)	225984623 (0.088%)	1605
E100k.1	225653450	225838800 (0.082%)	225860400 (0.091%)	225849600 (0.086%)	1724

Bảng 1: Kết quả thực nghiệm thuật toán đề xuất

Kết quả thực nghiệm từ Bảng 1 cho thấy, thuật toán đề xuất HLAGA đạt được chất lượng cận tối ưu trên tất cả các bộ dữ liệu được thử nghiệm, với sai số tối đa không quá 0.01% và sai số trung bình không quá 0.095%. Thời gian chạy trên toàn bộ các bộ dữ liệu nằm trong khoảng từ 10 đến 30 phút.

Ngoài ra, có sự khác biệt về tỉ lệ giữa chất lượng và kích thước dữ liệu giữa hai bộ dữ liệu TSPLIB và DIMACS. Điều này có thể giải thích được là bộ TSPLIB là một tập hợp nhiều dạng dữ liệu (đồ thị) khác nhau, ví dụ như pla7397 biểu diễn Programmed logic array trên thiết kế vi mạch, còn usa13509 thể hiện đồ thị thành phố trên nước Mỹ. Trong khi đó, DIMACS E là tập hợp những bộ dữ liệu được sinh ngẫu nhiên trên đồ thị Euclid. Do thuật toán đề xuất không đặt ra giả thiết nào, việc nó đạt hiệu quả tốt hơn trên tập ngẫu nhiên là có thể hiểu được.



Hình 3: So sánh đường đi kết quả trong 2 bộ dữ liệu pla7397 và E10k.0

Bảng 2 trình bày kết quả thực nghiệm so sánh ba phương pháp FACO, GA-EAX và HLAGA với 7 bộ dữ liệu từ DIMACS challenges, với tập đỉnh có kích thước từ 10,000 đến 100,000 đỉnh. Phương pháp FACO (Focused ACO) hiện là thuật toán ACO tốt nhất cho bộ dữ liệu lớn, được dịch trực tiếp mã nguồn từ chính tác giả Rafał Skinderowicz cung cấp. Thuật toán GA-EAX hiện là thuật toán GA cho TSP tốt nhất hiện nay, đồng thời cũng là một trong những thuật

[1]https://github.com/nagata-yuichi/GA-EAX

Bảng 2: Kết quả thực nghiệm so sánh ba phương pháp FACO, GA-EAX và HLAGA

Instances	BKS	FACO			GA-EAX			HLAGA		
		Length	Error	Time	Length	Error	Time	Length	Error	Time
E10k.0	71865826	72111677	0.34%	886	**71865826**	**0.00%**	11382	71873310	0.01%	609
		72115762	0.35%		**71866651**	**0.00%**		71878868	0.01%	
E10k.1	72031630	72238812	0.28%	714	**72031630**	**0.00%**	11103	72046421	0.02%	687
		72277073	0.34%		**72032293**	**0.00%**		72051400	0.02%	
E10k.2	71822483	71959338	0.19%	854	**71822483**	**0.00%**	11212	71828497	0.00%	635
		71970784	0.20%		**71822549**	**0.00%**		71833659	0.01%	
E31k.0	127281803	127837530	0.42%	863	127569710	0.22%	63252	**127330860**	**0.03%**	782
		127845236	0.44%		127608868	0.25%		**127348519**	**0.05%**	
E31k.1	127452384	128047893	0.46%	902	127760790	0.24%	63251	**127512737**	**0.04%**	791
		128052527	0.47%		127791639	0.26%		**127527456**	**0.05%**	
E100k.0	225783795	227173640	0.61%	1736	238006870	5.41%	200034	**225972765**	**0.08%**	1605
		227199340	0.63%		238185059	5.49%		**225984623**	**0.08%**	
E100k.1	225653450	227043452	0.61%	1772	237978179	5.46%	200037	**225838800**	**0.08%**	1724
		227054452	0.62%		238152143	5.53%		**225849600**	**0.08%**	

toán state-of-the-art của TSP, được biên dịch từ mã nguồn[1] của tác giả Nagata Yuichi. Cả ba thuật toán được so sánh đều được chạy trên cùng một máy tính.

Trong FACO, các thông số được thiết đặt như tác giả công bố trong bài báo, riêng số lời giải được tăng lên cho phù hợp với thời gian chạy của mọi thuật toán. Tương tự, GA-EAX được để tham số là mặc định. Kết quả của từng phương pháp với từng bộ dữ liệu nằm trong ô giao giữa cột và hàng tương ứng, trong đó hàng trên là kết quả tốt nhất thu được, hàng dưới là kết quả trung bình thu được sau 10 lần chạy. Trong các bảng kết quả dưới đây, kết quả được tô đậm là kết quả tốt nhất trong các phương pháp. Thời gian chạy tính bằng giây.

Kết quả thực nghiệm cho thấy, ở những bộ dữ liệu với kích thước 10,000 đỉnh, thuật toán GA-EAX vẫn thể hiện sự vượt trội trong việc hội tụ đến lời giải tốt. Dù vậy, với quy mô dữ liệu trên 30,000 đỉnh, GA-EAX gặp khó khăn với việc hội tụ lời giải, với thời gian chạy lên tới hàng chục tiếng. Còn FACO dù luôn cho ra lời giải tương đối tốt trong thời gian hợp lý nhưng chất lượng vẫn đang còn tương đối kém so với tối ưu, với sai số tối đa 0.6%. So với hai thuật toán trên, HLAGA luôn cho ra kết quả tốt cận tối ưu với sai số luôn dưới 0.09%, và thời gian chạy và thời gian chạy luôn dưới 30 phút. Với những bộ

dữ liệu trên 30,000 đỉnh, HLAGA vượt GA-EAX trở thành thuật toán có chất lượng lời giải tốt nhất trong cả ba thuật toán.

6. Kết luận

6.1. Nhận xét

Trong nghiên cứu này, chúng tôi đã phân tích các cách tiếp cận tốt nhất hiện nay cho bài toán TSP với kích thước lớn, đồng thời giới thiệu mô hình Tìm kiếm lân cận rộng. Từ đó, chúng tôi đã đề xuất ra một thuật toán mới dựa trên mô hình tìm kiếm lân cận rộng kết hợp với Tối ưu đàn kiến, đồng thời sử dụng Giải thuật di truyền. Từ những đề xuất về phương pháp khởi tạo dựa trên nghiên cứu trước, phương pháp phá hủy dựa trên tối ưu đàn kiến và phương pháp sửa lại dựa trên giải thuật di truyền, chúng tôi đã cài đặt và thử nghiệm thuật toán mới đề xuất trên các bộ dữ liệu có kích thước lớn, đồng thời so sánh kết quả với những giải thuật tốt nhất hiện nay của Tối ưu đàn kiến và Giải thuật di truyền. Thuật toán đã dành được những thành tựu đáng kể, đạt được chất lượng lời giải cận tối ưu với những bộ dữ liệu lên đến 100,000 đỉnh với sai số không quá 0.08%, với hiệu suất đáng kinh ngạc.

Giải thuật tìm kiếm lân cận rộng là một metaheuristic rất tiềm năng để giải quyết nhiều bài toán NP-Khó, trong đó có bài toán người chào hàng. Thuật toán đề xuất đã vượt qua GA-EAX, một trong những thuật toán tốt nhất hiện nay cho TSP, chứng minh tiềm năng của cách tiếp cận này cho các bộ dữ liệu TSP kích thước lớn. HLAGA là một thuật toán mới và với những kết quả nhận được, nó là một bổ sung có giá trị cho nhóm các thuật toán giải quyết bài toán TSP, đặc biệt là những bộ dữ liệu TSP với kích thước lớn.

6.2. Hướng nghiên cứu tương lai

Giải thuật Hybrid LNS-based ACO with GA repair mechanic Algorithm được đề xuất đã đạt được nhiều kết quả tốt cho những bộ dữ liệu TSP với kích thước lớn. Tuy nhiên, vẫn còn nhiều hướng nghiên cứu tiềm năng mở rộng từ nghiên cứu trên. Một hướng nghiên cứu tiềm năng là về phương pháp phá

hủy lời giải: thay vì chỉ tập trung phá hủy 1 phần nhỏ, có thể đồng thời phá hủy nhiều phần, thậm chí chia lời giải thành nhiều phần bị phá hủy nhỏ, sau đó sửa lại từng phần bị phá hủy đó. Cách phá hủy này sẽ giúp lân cận được tìm kiếm của lời giải trở nên rộng hơn, từ đó cải thiện khả năng hội tụ lời giải. Một hướng nghiên cứu khác là việc song song hóa quá trình phá và sửa lời giải, từ đó cải thiện tốc độ chạy của thuật toán.

Trong tương lai, thuật toán sẽ được chạy trên bộ tham số mạnh hơn và thời gian lâu hơn trên các bộ dữ liệu lớn hơn, nhằm so sánh trực tiếp chất lượng của thuật toán với mọi thuật toán TSP tốt nhất trên thế giới hiện nay, nhằm cạnh tranh và tìm ra lời giải tốt nhất mới cho các bộ dữ liệu trên. Giải thuật tìm kiếm lân cận rộng là một giải thuật đã chứng minh độ hiệu quả của nó trong nhiều bài toán, và có nhiều tiềm năng chưa được khai phá khi kết hợp với Tối ưu đàn kiến và Giải thuật di truyền, và nghiên cứu này hi vọng sẽ đưa ra một hướng tiếp cận tối ưu mới nhằm giải những bộ dữ liệu với TSP kích thước lớn và rất lớn.

6.2.1. Credit

Trần Thành Đạt: Quan sát, thiết kế thuật toán và thực nghiệm. Thu thập và phân tích dữ liệu. Viết bài báo.

6.2.2. Lời cảm ơn

Lời cảm ơn đến thầy Đỗ Đức Đông với sự hỗ trợ của thầy khi nghiên cứu, và lời cảm ơn đến Rafał Skinderowicz [14], [25] đã hỗ trợ cài đặt và so sánh kết quả thực nghiệm.

Bibliography

[1] C. H. Papadimitriou, “On the complexity of integer programming,” *J. ACM*, vol. 28, no. 4, pp. 765–768, Oct. 1981, doi: 10.1145/322276.322287.

[2] E. L. Lawler and D. E. Wood, “Branch-and-Bound Methods: A Survey,” *Oper. Res.*, vol. 14, no. 4, pp. 699–719, Aug. 1966, doi: 10.1287/opre.14.4.699.

[3] G. Lera-Romero, J. J. Miranda Bront, and F. J. Soulignac, "Dynamic Programming for the Time-Dependent Traveling Salesman Problem with Time Windows," *INFORMS J. on Computing*, vol. 34, no. 6, pp. 3292–3308, Nov. 2022, doi: 10.1287/ijoc.2022.1236.

[4] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, vol. 137, p. 106040, 2019.

[5] K. L. Hoffman, M. Padberg, G. Rinaldi, and others, "Traveling salesman problem," *Encyclopedia of operations research and management science*, vol. 1, pp. 1573–1578, 2013.

[6] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE transactions on systems, man, and cybernetics-Part A: systems and humans*, vol. 33, no. 5, pp. 560–572, 2003.

[7] T. P. Bagchi, J. N. Gupta, and C. Sriskandarajah, "A review of TSP based approaches for flowshop scheduling," *European Journal of Operational Research*, vol. 169, no. 3, pp. 816–854, 2006, doi: https://doi.org/10.1016/j.ejor.2004.06.040.

[8] J. Gomez *et al.*, "Ant colony system algorithm for the planning of primary distribution circuits," *IEEE Transactions on power systems*, vol. 19, no. 2, pp. 996–1004, 2004.

[9] M. Dorigo and T. Stützle, *Ant colony optimization: overview and recent advances*. Springer, 2019.

[10] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[11] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European journal of operational research*, vol. 126, no. 1, pp. 106–130, 2000.

[12] R. Matai, S. P. Singh, and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches,"

Traveling salesman problem, theory and applications, vol. 1, no. 1, pp. 1–25, 2010.

[13] S. Lin, "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[14] R. Skinderowicz, "Improving Ant Colony Optimization efficiency for solving large TSP instances," *Applied Soft Computing*, vol. 120, p. 108653, May 2022, doi: 10.1016/j.asoc.2022.108653.

[15] Y. Nagata, "Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem," *Proceedings of the 7th Internatinal Conferencenon Genetic Algorithms*, pp. 450–457, 1997.

[16] É. D. Taillard and K. Helsgaun, "POPMUSIC for the travelling salesman problem," *European Journal of Operational Research*, vol. 272, no. 2, pp. 420–429, 2019, doi: https://doi.org/10.1016/j.ejor.2018.06.039.

[17] D. Pisinger and S. Ropke, "Large Neighborhood Search," 2010, pp. 399–419. doi: 10.1007/978-1-4419-1665-5_13.

[18] H. University, "TSP Test Data," [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/

[19] C. for Discrete Mathematics and T. C. Science, "DIMACS TSP Challenge," [Online]. Available: http://dimacs.rutgers.edu/archive/Challenges/TSP/

[20] W. Cook, "Concorde TSP Solver," [Online]. Available: https://www.math.uwaterloo.ca/tsp/concorde.html

[21] B. Tüű-Szabó, P. Földesi, and L. T. Kóczy, "Analyzing the performance of TSP solver methods," *Computational Intelligence and Mathematics for Tackling Complex Problems 2*. Springer, pp. 65–71, 2022.

[22] S. Gupta, A. Rana, and V. Kansal, "Comparison of Heuristic techniques: A case of TSP," in *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2020, pp. 172–177.

[23] A. Akhtar, "Evolution of Ant Colony Optimization Algorithm–A Brief Literature Review," *arXiv preprint arXiv:1908.08007*, 2019.

[24] T. Stützle, M. Dorigo, and others, "ACO algorithms for the traveling salesman problem," *Evolutionary algorithms in engineering and computer science*, vol. 4, pp. 163–183, 1999.

[25] R. Skinderowicz, "Focused ACO with node relocation procedure for solving large TSP instances," *Procedia Computer Science*, vol. 225, pp. 2992–3000, 2023, doi: https://doi.org/10.1016/j.procs.2023.10.292.

[26] J. Scholz, "Genetic algorithms and the traveling salesman problem a historical review," *arXiv preprint arXiv:1901.05737*, 2019.

[27] Z. Sun and Y. Yang, "DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization." [Online]. Available: https://arxiv.org/abs/2302.08224

[28] X. Pan *et al.*, "H-TSP: Hierarchically Solving the Large-Scale Travelling Salesman Problem." [Online]. Available: https://arxiv.org/abs/2304.09395

[29] R. Qiu, Z. Sun, and Y. Yang, "DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems." [Online]. Available: https://arxiv.org/abs/2210.04123

[30] M. Guntsch and M. Middendorf, "A population based approach for ACO," in *Applications of Evolutionary Computing: EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN Kinsale, Ireland, April 3–4, 2002 Proceedings*, 2002, pp. 72–81.

[31] D. M. Chitty, "Applying ACO to large scale TSP instances," in *Advances in Computational Intelligence Systems: Contributions Presented at the 17th UK Workshop on Computational Intelligence, September 6-8, 2017, Cardiff, UK*, 2018, pp. 104–118.

[32] J. Peake, M. Amos, P. Yiapanis, and H. Lloyd, "Scaling techniques for parallel ant colony optimization on large problem instances," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 47–54.

[33] T. T. Dat, “Improving Solutions Quality for Large-Scale Datasets of the Traveling Salesman Problem,” 2024.

[34] D. D. Dong,.