

TRƯỜNG ĐẠI HỌC ĐIỆN LỰC
KHOA CÔNG NGHỆ THÔNG TIN



PHẠM THỊ NGỌC

**NGHIÊN CỨU MỘT PHƯƠNG PHÁP ĐỂ NÂNG CAO
KHẢ NĂNG TỐI ƯU KẾ HOẠCH XẾP DỠ HÀNG CONTAINER TẠI
CẢNG BIỂN PHÙ HỢP VỚI ĐẶC THU CỦA CẢNG BIỂN VIỆT NAM**

Ngành : Công nghệ thông tin

Mã ngành : 8480201

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội, 2025

TRƯỜNG ĐẠI HỌC ĐIỆN LỰC
KHOA CÔNG NGHỆ THÔNG TIN



PHẠM THỊ NGỌC

**NGHIÊN CỨU MỘT PHƯƠNG PHÁP ĐỂ NÂNG CAO
KHẢ NĂNG TỐI ƯU KẾ HOẠCH XẾP DỠ HÀNG CONTAINER TẠI
CẢNG BIỂN PHÙ HỢP VỚI ĐẶC THU CỦA CẢNG BIỂN VIỆT NAM**

Ngành : Công nghệ thông tin

Mã ngành : 8480201

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Người hướng dẫn khoa học: TS. Trần Trung

Hà Nội, 2025

LỜI CẢM ƠN

Luận án được hoàn thành dưới sự hướng dẫn, chỉ bảo, giúp đỡ tận tình của TS. Trần Trung và sự hỗ trợ của Chương trình khoa học và công nghệ cấp quốc gia giai đoạn đến năm 2030 "Hỗ trợ nghiên cứu, phát triển và ứng dụng công nghệ của công nghiệp 4.0", mã số KC-4.0-57/19-30. Lời đầu tiên, xin bày tỏ lòng kính trọng và biết ơn sâu sắc tới Thầy và Chương trình.

Tôi xin cảm ơn tới các thầy cô khoa Công nghệ thông tin trường Đại học Điện lực đã giảng dạy trong suốt quá trình học tập tại trường và các thầy cô phòng đào tạo của trường đã tạo điều kiện trong việc hoàn thiện hồ sơ bảo vệ luận án. Xin bày tỏ lòng cảm ơn tới đồng nghiệp, bạn bè và gia đình đã giúp đỡ và tạo điều kiện thuận lợi cho công việc học tập và nghiên cứu trong quá trình nghiên cứu luận án.

Công trình nghiên cứu này được thực hiện với

Tuy nhiên, do thời gian có hạn trong quá trình làm luận văn không tránh khỏi thiếu sót. Tôi rất mong nhận được sự góp ý của quý Thầy cô, bạn bè đồng nghiệp để luận căn được hoàn thiện tốt hơn.

Tôi xin trân trọng cảm ơn!

Hà Nội, ngày tháng năm 20

Tác giả

Phạm Thị Ngọc

LỜI CAM ĐOAN

Tác giả cam đoan đã sử dụng các tài liệu tham khảo của các tác giả, các nhà khoa học và các luận văn, ĐATN được trích dẫn trong phụ lục “Tài liệu tham khảo” cho việc nghiên cứu và viết luận văn của mình.

Tác giả cam đoan về các số liệu và kết quả tính toán được trình bày trong luận văn là hoàn toàn do tác giả tự tìm hiểu và thực hiện trong quá trình nghiên cứu và viết LVThS/ĐATNThS của mình, không sao chép và chưa được sử dụng cho đề tài luận văn nào.

Tôi xin chân thành cảm ơn!

Hà Nội, ngày tháng năm 20

Tác giả

Phạm Thị Ngọc

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT	i
DANH MỤC BẢNG BIỂU.....	ii
DANH MỤC HÌNH VẼ.....	iii
I. MỞ ĐẦU.....	1
1. Lý do chọn đề tài.....	1
2. Mục đích nghiên cứu	3
3. Nhiệm vụ nghiên cứu.....	4
4. Đối tượng và phạm vi nghiên cứu	5
4.1. Đối tượng nghiên cứu	5
4.2. Phạm vi nghiên cứu	6
4.2.1. Phạm vi dữ liệu	6
4.2.2. Phạm vi địa lý.....	6
4.2.3. Phạm vi thời gian.....	7
5. Phương pháp nghiên cứu	7
5.1. Thu thập dữ liệu:	7
5.2. Phát triển thuật toán:	8
5.3. Thực nghiệm:	9
5.4. Phân tích và so sánh:	9
II. NỘI DUNG.....	9
CHƯƠNG I: GIỚI THIỆU	10
1.1. Giới thiệu đề tài, mục đích và ý nghĩa của nghiên cứu	10
1.1.1. Giới thiệu đề tài.....	10
1.1.2. Mục đích của nghiên cứu.....	10
1.1.3. Ý nghĩa của nghiên cứu	10
1.2. Phân tích lý do chọn đề tài.....	11
1.3. Phạm vi bài toán và mục tiêu nghiên cứu	11
1.3.1. Phạm vi bài toán	11
1.3.2. Mục tiêu nghiên cứu	12
1.4. Cấu trúc của luận văn	12
1.5. Đóng góp của luận văn	12
1.5.1. Đóng góp về Lý thuyết.....	12
1.5.2. Đóng góp về tính ứng dụng.....	13
1.5.3. Đóng góp về hiệu quả ứng dụng.....	13
CHƯƠNG 2: TỔNG QUAN VỀ KIẾN THỨC NỀN TẢNG	15
2.1. Cơ sở lý thuyết về cảng container và bài toán CSP/DCSP	15
2.1.1. Tổng quan về hoạt động tại cảng container	15

2.1.2.	<i>Cấu trúc và quản lý bãi container</i>	16
2.1.3.	<i>Phương pháp tối ưu hoá CSP</i>	18
2.2.	Phân loại bài toán xếp container (SCSP và DCSP)	20
2.2.1.	<i>Static Container Stacking Problem (SCSP)</i>	20
2.2.2.	<i>Dynamic Container Stacking Problem (DCSP)</i>	20
2.2.3.	<i>So sánh SCSP và DCSP</i>	21
2.3.	Các phương pháp tiếp cận giải quyết CSP	21
2.3.1.	<i>Phương pháp heuristic</i>	21
2.3.2.	<i>Phương pháp metaheuristic</i>	22
2.3.3.	<i>Phương pháp trí tuệ nhân tạo (AI)</i>	22
2.3.4.	<i>So sánh các phương pháp</i>	23
2.4.	Tổng quan các nghiên cứu trước đây	23
2.5.	Kết luận khảo sát	26
CHƯƠNG 3: THỰC NGHIỆM		28
3.1.	Mô tả bài toán DCSP tại Việt Nam	28
3.2.	Thuật toán đề xuất VN-Stack	29
3.2.1.	<i>Ý tưởng và kiến trúc</i>	29
3.2.2.	<i>Ký hiệu (Notations)</i>	30
3.2.3.	<i>Các trường hợp nâng container (case lifting time)</i>	30
3.2.4.	<i>Hàm mục tiêu (Objective function)</i>	31
3.2.5.	<i>Mã giả</i>	32
3.2.5.1.	<i>Thuật toán di truyền cho bài toán xếp container</i>	32
3.2.5.2.	<i>Hàm đánh giá (Fitness): tối thiểu hóa tổng thời gian nâng</i>	33
3.2.5.3.	<i>Lai ghép PMX (Partial Matched Crossover – two-point)</i>	34
3.2.5.4.	<i>Đột biến hoán đổi (Swap Mutation)</i>	35
3.2.5.5.	<i>Sửa nhiễm sắc thể (Chromosome Repair) để đảm bảo tính khả thi</i>	36
3.2.5.6.	<i>Quy trình mô phỏng 4 bước (đánh giá cấu hình xếp)</i>	37
3.2.6.	Độ phức tạp thuật toán	39
3.2.6.1.	<i>Bài toán gốc (Container Stacking Problem – CSP)</i>	39
3.2.6.2.	<i>Thuật toán di truyền (Genetic Algorithm – GA)</i>	40
3.2.6.3.	<i>Độ phức tạp tổng thể</i>	41
3.2.6.4.	<i>So sánh với các phương thức khác</i>	41
3.3.	Thiết lập mô phỏng và thực nghiệm	43
3.3.1.	<i>Bộ dữ liệu giả lập</i>	43
3.3.2.	<i>Kết quả mô phỏng và thực nghiệm</i>	44
3.3.3.	<i>Ví dụ minh họa</i>	45

3.3.3.1. Biểu diễn nhiễm sắc thể	46
3.3.3.2. Tính hàm mục tiêu (fitness)	46
3.3.3.3. Áp dụng GA.....	47
3.4. Phân tích và đánh giá kết quả	47
3.4.1. Ưu điểm của GA trong CSP.....	48
3.4.1.1. Khả năng xử lý bài toán phức tạp, NP-hard.....	48
3.4.1.2. Tính linh hoạt và khả năng mở rộng.....	48
3.4.1.3. Cải thiện hiệu quả vận hành.....	48
3.4.1.4. Tính tổng quát và ứng dụng rộng.....	48
3.4.2. Hạn chế của GA trong CSP	48
3.4.2.1. Không đảm bảo tối ưu toàn cục.....	48
3.4.2.2. Chi phí tính toán vẫn cao hơn heuristic đơn giản	49
3.4.2.3. Phụ thuộc vào chất lượng biểu diễn và hàm sửa (repair)	49
III. KẾT LUẬN	51
TÀI LIỆU THAM KHẢO.....	53
PHỤ LỤC: MÃ NGUỒN CHƯƠNG TRÌNH.....	56

DANH MỤC TỪ VIẾT TẮT

TỪ VIẾT TẮT	Ý NGHĨA
AI	Artificial Intelligence
BAP	Berth Allocation Problem
CM-TV	Cai Mep – Thi Vai
CSP	Container Stacking Problem
DCSP	Dynamic Container Stacking Problem
DRL	Deep Reinforcement Learning
GA	Genetic Algorithm
ITT	Inter Terminal Transportation
QC	Quay Crane
RMG	Rail-Mounted Gantry
RTG	Rubber-Tired Gantry
SCSP	Static Container Stacking Problem
TEU	Twenty-foot Equivalent Unit
TOS	Terminal Operating System
YC	Yard Crane

DANH MỤC BẢNG BIỂU

<i>Bảng 1 So sánh các phương pháp tối ưu hoá CSP</i>	<i>23</i>
--	-----------

DANH MỤC HÌNH VẼ

<i>Hình 2. 1 Cấu trúc container (10x10x5).....</i>	<i>16</i>
<i>Hình 2. 2 Quy trình DCSP.....</i>	<i>18</i>
<i>Hình 2. 3 Lưu đồ thuật toán STACK-NEXT</i>	<i>19</i>
<i>Hình 2. 4 Lưu đồ quy trình xếp chồng container tại bãi cảng</i>	<i>26</i>

I. MỞ ĐẦU

1. Lý do chọn đề tài

Ngành logistics và vận tải biển đóng vai trò cốt lõi trong chuỗi cung ứng toàn cầu, đặc biệt tại các quốc gia có hệ thống cảng biển phát triển như Việt Nam [14]. Theo Tổng cục Hải quan Việt Nam, kim ngạch xuất nhập khẩu hàng hóa năm 2024 đạt 786,29 tỷ USD, tăng 15,4% so với năm 2023, với xuất khẩu tăng 14,3% và nhập khẩu tăng 16,7% [16]. Khối lượng hàng hóa thông qua các cảng biển đạt hơn 501 triệu tấn trong 7 tháng đầu năm 2024, tăng 16% so với cùng kỳ năm 2023, phản ánh sự phục hồi mạnh mẽ sau đại dịch COVID-19 [17]. Các cảng biển lớn như Cái Mép - Thị Vải, Hải Phòng, và Đà Nẵng đang phải đối mặt với áp lực lớn trong việc quản lý hiệu quả các hoạt động vận hành, đặc biệt tại khu vực bãi container (yard operations) [4]. Ví dụ, cảng Cái Mép - Thị Vải đã xử lý gần 102 triệu tấn hàng hóa năm 2024, tăng 25% so với năm 2023, nhờ kết nối với các tuyến hàng hải quốc tế [16]. Trong bối cảnh này, bài toán xếp dỡ container động (Dynamic Container Stacking Problem - DCSP) trở thành một thách thức quan trọng, ảnh hưởng trực tiếp đến hiệu suất hoạt động của cảng, thời gian chờ của tàu, và chi phí vận hành tổng thể [9].

DCSP yêu cầu xác định vị trí tối ưu (block, bay, stack) cho mỗi container khi đến bãi, trong điều kiện không biết trước toàn bộ thứ tự đến của container [1]. Khác với bài toán xếp tĩnh (Static Container Stacking Problem - SCSP), nơi vị trí container được lập kế hoạch trước, DCSP phản ánh thực tế hơn tại các cảng biển, nơi các yếu tố như thời gian truy xuất container (retrieval time), kế hoạch xếp dỡ tàu (stowage planning), và yêu cầu từ người nhận hàng (consignee) thay đổi liên tục [6]. Ví dụ, tại cảng Hải Phòng, lịch tàu có thể thay đổi đột xuất do thời tiết hoặc yêu cầu từ khách hàng, đòi hỏi điều chỉnh vị trí container trong thời gian thực [4]. Việc xếp container không tối ưu dẫn đến tái định vị (relocation), khi cần cẩu bãi (Yard Crane - YC) phải di chuyển các container nằm trên để truy xuất container bên dưới, làm tăng số lần di chuyển lên gấp đôi hoặc gấp ba [11]. Điều này gây lãng phí thời gian (hàng giờ mỗi ca làm việc), tăng tiêu thụ năng lượng, và nâng chi phí vận hành, đồng thời góp phần vào phát thải CO₂ từ thiết bị diesel, ảnh hưởng tiêu cực đến môi trường [11]. Theo nghiên cứu, chi phí tái định vị chiếm 20-30% tổng chi phí vận hành bãi container tại các cảng lớn, và ở Việt Nam, với lưu lượng hàng hóa tăng 17% trong 5 tháng đầu năm 2024 (346,539 triệu tấn), vấn đề này càng trở nên cấp bách [5].

Trong bối cảnh Việt Nam, các cảng biển lớn đang chuyển đổi sang mô hình vận hành tự động hoặc bán tự động, đòi hỏi các giải pháp tối ưu hóa thông minh để nâng cao hiệu quả [15]. Ví dụ, cảng Hải Phòng đã triển khai cần cầu bãi tự động (Rubber-Tired Gantry - RTG), nhưng vẫn gặp khó khăn trong việc xử lý DCSP do thiếu thuật toán địa phương hóa [4]. Các nghiên cứu quốc tế đã đề xuất nhiều phương pháp giải quyết DCSP, bao gồm thuật toán heuristic để xử lý nhanh quyết định thời gian thực [7], metaheuristic như thuật toán di truyền để tìm giải pháp gần tối ưu trong không gian tìm kiếm lớn [8], và các phương pháp trí tuệ nhân tạo như học tăng cường sâu (Deep Reinforcement Learning - DRL) để học từ dữ liệu lịch sử và dự đoán vị trí tối ưu [2]. Tuy nhiên, các giải pháp này thường phức tạp, yêu cầu cơ sở hạ tầng tính toán mạnh (như máy chủ GPU cho DRL) hoặc không phù hợp với đặc thù của các cảng Việt Nam, nơi nguồn lực kỹ thuật còn hạn chế [10]. Chẳng hạn, tại các cảng nhỏ như Quy Nhơn hoặc Vũng Tàu, việc áp dụng DRL không khả thi do thiếu dữ liệu lớn và chuyên gia AI [15]. Hơn nữa, các giải pháp quốc tế thường tập trung vào cảng lớn với lưu lượng hàng hóa ổn định, trong khi cảng Việt Nam phải đối mặt với biến động lớn về loại container (20ft/40ft, hàng nguy hiểm, hàng đông lạnh) và yêu cầu linh hoạt từ khách hàng [13].

Bài báo “An Experiment on Stack-next Algorithm for Dynamic Container Stacking Problem at Sea Port” đã đề xuất thuật toán heuristic Stack-next, với mục tiêu giảm số lần di chuyển cần cầu xuống mức tối thiểu (một di chuyển để xếp và một di chuyển để truy xuất mỗi container) [1]. Thuật toán này phân nhóm container theo thời gian truy xuất (5 nhóm, từ ngày 1 đến ngày 5, dựa trên hợp đồng hoặc thời gian giao hàng) và ưu tiên xếp container có ngày truy xuất sớm lên trên cùng stack, giúp tránh tái định vị [1]. Trong thử nghiệm giả lập với 400 container và 1000 kích bản ngẫu nhiên, Stack-next đạt độ chính xác xếp từ 88-94% trên Block B1 (10x10x5) và 76-89% trên Block B2 (6x17x5), đồng thời đảm bảo 100% hiệu quả truy xuất mà không cần tái định vị [1]. Tuy nhiên, để áp dụng Stack-next tại Việt Nam, cần đánh giá trên dữ liệu thực tế từ các cảng như Cái Mép - Thị Vải (khối lượng hàng hóa tăng 25% năm 2024) và mở rộng để xử lý container 20ft/40ft (chiếm 70% lưu lượng hàng hóa Việt Nam), hàng nguy hiểm (hóa chất, chiếm 5-10% tại cảng Hải Phòng), và hàng đông lạnh (thực phẩm xuất khẩu, tăng 12% năm 2024) [13] [16].

Luận văn này được thực hiện nhằm phát triển thuật toán heuristic cải tiến, tạm gọi là VN-Stack, dựa trên Stack-next, để giải quyết bài toán DCSP tại các cảng biển Việt Nam. VN-Stack sẽ cải tiến bằng cách bổ sung xử lý container đa dạng kích thước và loại hàng, tích hợp với hệ thống quản lý cảng (Terminal Operating System

- TOS) để tự động hóa quy trình xếp dỡ [15]. Nghiên cứu không chỉ cải thiện hiệu suất bãi container mà còn hỗ trợ các mục tiêu chiến lược của ngành logistics Việt Nam, như giảm thời gian xếp dỡ tàu (luồng xuất khẩu) để cạnh tranh với các cảng khu vực như Singapore hoặc Thượng Hải, và thông báo chính xác thời gian giao hàng (luồng nhập khẩu) để nâng cao sự hài lòng của khách hàng [5]. Hơn nữa, nghiên cứu đóng góp vào việc ứng dụng giải pháp tối ưu hóa thông minh, phù hợp với cơ sở hạ tầng và nguồn lực Việt Nam, trong bối cảnh chuyển đổi số ngành logistics, nơi Chính phủ thúc đẩy các dự án số hóa cảng biển đến năm 2030 [15].

Việc chọn đề tài xuất phát từ thực tiễn ngành Công nghệ Thông tin (CNTT) tại Việt Nam, nơi ứng dụng công nghệ vào logistics đang phát triển mạnh nhưng thiếu giải pháp địa phương hóa. Theo nghiên cứu, tối ưu hóa bãi container có thể giảm chi phí vận hành 20-30% [11], góp phần nâng cao vị thế các cảng Việt Nam trong khu vực Đông Nam Á. Với mục tiêu xuất khẩu 1.000 tỷ USD vào năm 2030 [16], việc nghiên cứu DCSP không chỉ mang tính học thuật mà còn có giá trị kinh tế - xã hội cao, hỗ trợ chiến lược phát triển logistics bền vững của Việt Nam [18].

2. Mục đích nghiên cứu

Mục tiêu tổng quát của luận văn là phát triển và đánh giá một thuật toán heuristic cải tiến (VN-Stack) để giải quyết bài toán xếp dỡ container động tại các cảng biển Việt Nam, đảm bảo hiệu quả cao và khả năng triển khai thực tiễn. Các mục tiêu cụ thể bao gồm:

- Trình bày thuật toán VN-Stack, cải tiến từ Stack-next, với khả năng xếp container động nhằm giảm số lần di chuyển cần cầu bãi và tối ưu hóa không gian lưu trữ container [1]. Thuật toán sẽ được thiết kế để xử lý các điều kiện thực tế, như phân nhóm container theo thời gian truy xuất và ưu tiên vị trí xếp để tránh tái định vị.
- Thực nghiệm thuật toán VN-Stack trên dữ liệu giả lập (400 container, 1000 kịch bản) và dữ liệu thực tế từ một cảng biển Việt Nam (ví dụ: Cái Mép - Thị Vải), đánh giá hiệu quả qua các chỉ số như độ chính xác xếp container, số lần di chuyển cần cầu, và thời gian xử lý. Thực nghiệm sẽ sử dụng mô phỏng để kiểm tra trong các kịch bản khác nhau, đảm bảo độ tin cậy.
- Đề xuất giải pháp triển khai thực tiễn tại các cảng biển Việt Nam, bao gồm các điều chỉnh để xử lý container 20ft/40ft, hàng nguy hiểm, và tích hợp với hệ thống quản lý cảng hiện có [13]. Giải pháp sẽ bao gồm hướng dẫn tích hợp với phần mềm TOS và đánh giá tác động kinh tế.

Các mục tiêu này nhằm giải quyết khoảng trống nghiên cứu trong việc áp dụng thuật toán heuristic vào bối cảnh Việt Nam, nơi các giải pháp quốc tế thường không phù hợp [10].

3. Nhiệm vụ nghiên cứu

Trong bối cảnh nghiên cứu về bài toán xếp dỡ container động (Dynamic Container Stacking Problem - DCSP) tại các cảng biển Việt Nam, việc xác định rõ ràng các vấn đề nghiên cứu là yếu tố quan trọng để định hướng toàn bộ quá trình nghiên cứu, từ phát triển thuật toán đến thực nghiệm và phân tích kết quả. Các vấn đề nghiên cứu không chỉ giúp làm rõ phạm vi và mục tiêu mà còn đảm bảo tính khả thi, tính khoa học, và tính ứng dụng thực tiễn của luận văn. Dựa trên các mục tiêu đã nêu, luận văn tập trung vào ba vấn đề nghiên cứu chính, được xây dựng từ khoảng trống nghiên cứu trong các tài liệu hiện có, chẳng hạn như nhu cầu cải tiến thuật toán heuristic để phù hợp với điều kiện thực tế [1, 10]. Các vấn đề này được liên kết chặt chẽ với nhau, giúp hướng dẫn quá trình phát triển thuật toán VN-Stack, thực nghiệm trên dữ liệu giả lập và thực tế, cũng như đề xuất giải pháp triển khai.

Vấn đề 1: Làm thế nào thuật toán VN-Stack có thể xếp container động với độ chính xác tối thiểu 80% công suất khối container trong các kịch bản thực tế, bao gồm cả dữ liệu giả lập và thực tế? Vấn đề này tập trung vào hiệu quả của thuật toán trong giai đoạn xếp container (stacking), nơi thứ tự đến của container là ngẫu nhiên và không thể dự đoán trước, một đặc trưng chính của DCSP [1]. Độ chính xác được đo lường bằng tỷ lệ container được xếp so với công suất vật lý của khối container (ví dụ: 400 container trên khối 500 TEU đạt 80%), đồng thời xem xét các chỉ số như số vịnh (bay) sử dụng, số container trung bình mỗi vịnh, và khả năng tránh tình trạng khối bị lấp đầy không đồng đều. Trong thực tế, vấn đề này đặc biệt quan trọng tại các cảng Việt Nam như Cái Mép - Thị Vải, nơi lưu lượng container tăng 25% năm 2024, đòi hỏi thuật toán phải xử lý linh hoạt để tối ưu hóa không gian bãi mà không làm gián đoạn hoạt động [16]. Việc giải quyết vấn đề này sẽ giúp đánh giá khả năng của VN-Stack trong việc cải tiến Stack-next, bằng cách bổ sung các điều kiện kiểm tra trạng thái khối (trống, đầy, hoặc có vị trí khả dụng) để đạt hiệu quả cao hơn trong môi trường thực tế.

Vấn đề 2: Thuật toán VN-Stack đạt hiệu quả như thế nào trong việc giảm số lần di chuyển cần cầu khi truy xuất container từ trạng thái bãi đạt 80% công suất? Vấn đề này đánh giá khả năng tránh tái định vị (relocation) của thuật toán, với mục tiêu đảm bảo mỗi container chỉ cần một lần di chuyển cần cầu bãi (YC) để truy xuất, thay vì phải di chuyển nhiều container nằm trên [11]. Tại công suất 80%, bãi

container thường ở trạng thái gần đầy, dễ dẫn đến tình trạng container cần truy xuất nằm dưới các container khác, gây tăng chi phí thời gian và năng lượng [11]. Trong bối cảnh Việt Nam, nơi cần cầu YC thường sử dụng năng lượng diesel, việc giảm di chuyển không chỉ tiết kiệm chi phí mà còn giảm phát thải CO₂, phù hợp với các mục tiêu phát triển bền vững [15]. Vấn đề này sẽ được kiểm tra qua thực nghiệm, so sánh số lần di chuyển thực tế với các thuật toán khác, để chứng minh VN-Stack có thể đạt 100% hiệu quả truy xuất như Stack-next [1], đồng thời cải tiến để xử lý các kịch bản bất định như thay đổi lịch tàu.

Vấn đề 3: Những yếu tố nào cần điều chỉnh để triển khai thuật toán VN-Stack tại các cảng biển Việt Nam, đặc biệt với các loại container đa dạng (20ft/40ft, hàng nguy hiểm, hàng đông lạnh) và các yêu cầu vận hành thực tế? Vấn đề này khám phá các hạn chế của thuật toán và đề xuất giải pháp mở rộng, tập trung vào tính khả thi triển khai trong môi trường thực tế [13]. Stack-next chủ yếu xử lý container TEU tiêu chuẩn, nhưng tại Việt Nam, container 20ft/40ft chiếm tỷ lệ lớn (70-80%), hàng nguy hiểm (hóa chất, 5-10%) và hàng đông lạnh (thực phẩm xuất khẩu, tăng 12% năm 2024) đòi hỏi điều chỉnh thuật toán để tính đến kích thước, trọng lượng, và yêu cầu an toàn [13, 16]. Các yếu tố cần điều chỉnh bao gồm tích hợp với hệ thống TOS, xử lý dữ liệu thời gian thực từ xe tải/tàu, và đánh giá tác động kinh tế (giảm chi phí 20-30%) [11]. Vấn đề này sẽ được phân tích qua các kịch bản mô phỏng và đề xuất quy trình triển khai tại cảng Cái Mép - Thị Vải.

Các vấn đề nghiên cứu này được liên kết chặt chẽ với mục tiêu tổng quát, giúp hướng dẫn quá trình thực nghiệm (dữ liệu giả lập và thực tế), phân tích kết quả, và đề xuất giải pháp. Việc giải quyết chúng không chỉ lấp khoảng trống nghiên cứu mà còn mang lại giá trị ứng dụng cao trong ngành logistics Việt Nam [10, 15].

4. Đối tượng và phạm vi nghiên cứu

Nghiên cứu tập trung vào bài toán xếp dỡ container động (Dynamic Container Stacking Problem - DCSP) tại bãi container của các cảng biển Việt Nam, với mục tiêu phát triển và đánh giá thuật toán heuristic VN-Stack để tối ưu hóa hiệu suất vận hành. Để đảm bảo tính tập trung và khả thi trong khuôn khổ luận văn Thạc sĩ, nghiên cứu được giới hạn trong các khía cạnh cụ thể sau:

4.1. Đối tượng nghiên cứu

- **Luồng vận hành:** Nghiên cứu chỉ tập trung vào luồng xuất khẩu (export flow) và nhập khẩu (import flow), không xem xét luồng vận chuyển liên cảng (Inter Terminal Transportation - ITT). Lý do là ITT ít phổ biến tại các cảng Việt Nam, nơi hoạt động chủ yếu xoay quanh xuất nhập khẩu (chiếm 90% lưu

lượng container) [5]. Việc loại bỏ ITT giúp tập trung vào các kịch bản thực tế, giảm độ phức tạp của bài toán [10].

- **Cấu hình bãi container:** Xem xét các khối container (block) với cấu hình phổ biến tại Việt Nam, bao gồm 10x10x5 (500 container, tương ứng 500 TEU) và 6x17x5 (510 container). Các cấu hình này được chọn dựa trên thực tế bãi container tại các cảng lớn như Cái Mép - Thị Vải, nơi khối 10x10x5 được sử dụng cho bãi chính và 6x17x5 phù hợp với bãi phụ có không gian hẹp [4]. Thiết bị vận hành bao gồm cần cẩu bãi Rubber-Tired Gantry (RTG) và Rail-Mounted Gantry (RMG), là hai loại phổ biến tại Việt Nam nhờ tính linh hoạt và hiệu suất cao [4].
- **Phương pháp nghiên cứu:** Phát triển thuật toán VN-Stack dựa trên phương pháp heuristic, cải tiến từ thuật toán Stack-next [1]. Nghiên cứu tránh sử dụng các phương pháp phức tạp như học tăng cường sâu (Deep Reinforcement Learning - DRL) [2] hoặc thuật toán di truyền [8], do chúng yêu cầu cơ sở hạ tầng tính toán mạnh (như máy chủ GPU) và không phù hợp với nguồn lực kỹ thuật hạn chế tại các cảng Việt Nam [10]. VN-Stack được thiết kế đơn giản, dễ tích hợp với hệ thống TOS (Terminal Operating System), phù hợp với mục tiêu số hóa logistics Việt Nam [15].

4.2. Phạm vi nghiên cứu

4.2.1. Phạm vi dữ liệu

- **Dữ liệu giả lập:** Thử nghiệm trên tập dữ liệu giả lập với 400 container, phân thành 5 nhóm theo thời gian truy xuất (mỗi nhóm khoảng 80 container, tương ứng với ngày 1 đến ngày 5, dựa trên hợp đồng hoặc thời gian giao hàng). Dữ liệu được tạo ngẫu nhiên với 1000 kịch bản, mô phỏng thứ tự đến container không xác định trước, tương tự thiết kế trong bài báo gốc [1]. Các container bao gồm tỷ lệ 70% container 20ft/40ft, phản ánh thực tế tại cảng Việt Nam [16]. Dữ liệu được lưu trữ dưới dạng CSV hoặc SQL để dễ xử lý bằng Python (thư viện Pandas, NumPy).
- **Dữ liệu thực tế:** Nếu có điều kiện, nghiên cứu sẽ sử dụng dữ liệu thực tế từ một cảng biển Việt Nam, ưu tiên cảng Cái Mép - Thị Vải, nơi xử lý 102 triệu tấn hàng hóa năm 2024 [16]. Dữ liệu thực tế sẽ được thu thập từ hệ thống TOS, bao gồm thông tin container (ID, kích thước, thời gian truy xuất) và trạng thái bãi, dự kiến trong giai đoạn 2024-2025. Việc này phụ thuộc vào sự hợp tác với cơ quan cảng và khả năng tiếp cận dữ liệu.

4.2.2. Phạm vi địa lý

- Luận văn tập trung vào vấn đề xếp dỡ container tại các cảng biển lớn tại Việt Nam có lưu lượng container cao, bao gồm Cái Mép - Thị Vải (xử lý 25% tổng lưu lượng container cả nước năm 2024), Hải Phòng (có hệ thống RTG hiện đại, xử lý 30% lưu lượng), và Đà Nẵng (trung tâm logistics miền Trung) [16]. Các cảng này được chọn do vai trò quan trọng trong chuỗi cung ứng toàn cầu và mức độ tự động hóa cao, phù hợp để triển khai thuật toán VN-Stack [15].

4.2.3. Phạm vi thời gian

Luận văn được thực hiện trong 5 tháng, từ tháng 4/2025 đến tháng 9/2025, chia thành các giai đoạn: thu thập dữ liệu, phát triển thuật toán VN-Stack, thực nghiệm trên dữ liệu giả lập và thực tế, và viết luận văn. Thời gian này đảm bảo đủ để hoàn thành các mục tiêu nghiên cứu, từ mô phỏng đến phân tích kết quả, phù hợp với nguồn lực của một luận văn Thạc sĩ.

Các giới hạn trên đảm bảo nghiên cứu tập trung vào các khía cạnh cốt lõi của DCSP, tránh mở rộng quá mức dẫn đến thiếu chiều sâu. Việc chọn các cảng lớn, cấu hình khối phổ biến, và dữ liệu giả lập/thực tế giúp luận văn có tính ứng dụng cao, đặc biệt trong bối cảnh chuyển đổi số ngành logistics Việt Nam [15].

5. Phương pháp nghiên cứu

Luận văn sử dụng phương pháp nghiên cứu thực nghiệm kết hợp với mô phỏng sự kiện rời rạc (Discrete-Event Simulation), một cách tiếp cận phổ biến trong lĩnh vực Công nghệ Thông tin (CNTT) để kiểm tra và đánh giá các thuật toán tối ưu hóa trong môi trường mô phỏng [6]. Phương pháp này được chọn vì phù hợp với bài toán xếp dỡ container động (DCSP), nơi cần kiểm soát các biến số như thứ tự đến container, trạng thái bãi, và thời gian truy xuất, đồng thời cho phép đánh giá hiệu quả thuật toán VN-Stack trong các kịch bản khác nhau mà không cần triển khai thực tế ngay lập tức [1]. So với các phương pháp lý thuyết thuần túy (như mô hình toán học) hoặc thực nghiệm thực địa (có thể tốn kém và rủi ro), phương pháp thực nghiệm kết hợp mô phỏng đảm bảo tính khả thi, tiết kiệm chi phí, và khả năng lặp lại thí nghiệm nhiều lần để đạt độ tin cậy thống kê cao [9]. Trong bối cảnh Việt Nam, nơi hạ tầng cảng biển đang chuyển đổi số hóa nhưng nguồn lực kỹ thuật hạn chế, phương pháp này còn hỗ trợ kiểm tra thuật toán trên dữ liệu giả lập trước khi áp dụng thực tế, phù hợp với các nghiên cứu về logistics thông minh [15].

Các bước nghiên cứu chính được thực hiện theo trình tự logic, từ chuẩn bị dữ liệu đến phân tích kết quả, như sau:

5.1. Thu thập dữ liệu:

- **Dữ liệu giả lập:** Dựa trên thiết kế của thuật toán Stack-next [1], nghiên cứu tạo ra tập dữ liệu với 400 container, phân thành 5 nhóm theo thời gian truy xuất (mỗi nhóm khoảng 80 container, đại diện cho các ngày từ 1 đến 5, dựa trên giá trị hợp đồng hoặc thời gian giao hàng dự kiến). Dữ liệu được tạo ngẫu nhiên sử dụng Python (thư viện random và pandas) để mô phỏng thứ tự đến container không xác định trước, với tỷ lệ 70% container 20ft/40ft để phản ánh thực tế tại cảng Việt Nam [16]. Tổng cộng có 1000 kịch bản ngẫu nhiên, lưu trữ dưới dạng CSV hoặc cơ sở dữ liệu SQL (sử dụng SQLite hoặc MySQL) để dễ dàng truy vấn và xử lý. Lý do chọn 400 container là để phù hợp với công suất 80% của khối container (ví dụ: 400/500 TEU cho Block B1), giúp kiểm tra hiệu quả thuật toán ở mức tải cao [1].
- **Dữ liệu thực tế:** Nếu có điều kiện hợp tác với cảng biển (ví dụ: Cái Mép - Thị Vải, nơi xử lý 102 triệu tấn hàng hóa năm 2024 [16]), nghiên cứu sẽ thu thập dữ liệu từ hệ thống TOS, bao gồm thông tin container (ID, kích thước, thời gian truy xuất, loại hàng), trạng thái bãi (vị trí block, bay, stack), và lịch trình tàu. Dữ liệu sẽ được thu thập trong giai đoạn 2024-2025, xử lý ẩn danh để tuân thủ quy định bảo mật, và chuyển đổi sang định dạng CSV để đồng nhất với dữ liệu giả lập [3]. Việc kết hợp dữ liệu thực tế giúp kiểm tra tính khả thi của VN-Stack trong môi trường thực, nơi có các yếu tố bất định như sự cố thời tiết hoặc thay đổi lịch tàu [10].

5.2. Phát triển thuật toán:

- **Cải tiến thuật toán:** VN-Stack được phát triển dựa trên Stack-next [1], với cải tiến để xử lý container đa dạng: bổ sung điều kiện kiểm tra kích thước (20ft/40ft) bằng cách điều chỉnh chiều cao stack (z-axis), và loại hàng đặc biệt (nguy hiểm/đông lạnh) bằng cách ưu tiên vị trí an toàn (ví dụ: hàng nguy hiểm ở stack thấp để dễ truy xuất khẩn cấp) [13]. Thuật toán sử dụng logic kiểm tra điều kiện (Condition Sequence Number - CSN), bao gồm: CSN 1 (vịnh trống), CSN 2 (vịnh đầy), CSN 4 (container ngày 5), và CSN 5 (container ngày 1-4), với ưu tiên xếp container có ngày truy xuất sớm lên trên để tránh tái định vị [1].
- **Triển khai bằng Python:** Sử dụng thư viện NumPy để tính toán ma trận khối container (3 chiều: x, y, z), Pandas để xử lý dữ liệu container (DataFrame cho ID, retrieval_day), và Matplotlib/Seaborn để trực quan hóa kết quả (biểu đồ số container/bay, phân bố stack) [3]. Mã nguồn được kiểm tra bằng PyTest để đảm bảo tính đúng đắn, và lưu trữ trong Phụ lục để dễ tái sử dụng [12].

5.3. Thực nghiệm:

- **Mô phỏng:** Chạy thuật toán trên các khối container (Block B1: 10x10x5, Block B2: 6x17x5), sử dụng trình mô phỏng ASCII để hiển thị trạng thái 3 chiều (X - hàng, Y - vịnh, Z - stack) [1]. Mỗi kịch bản được lặp 1000 lần để tính trung bình và độ lệch chuẩn, mô phỏng sự kiện rời rạc như container đến ngẫu nhiên [6].
- **Đánh giá chỉ số:** Đo lường độ chính xác xếp container (tỷ lệ so với công suất 80%, ví dụ: 400/500 TEU), số lần di chuyển cần cầu (mục tiêu 1/container), và thời gian xử lý (tính bằng giây trên máy tính cấu hình Intel i7, RAM 16GB). Thực nghiệm được thực hiện trên môi trường Python 3.12, với dữ liệu thực tế nếu có để so sánh [10].

5.4. Phân tích và so sánh:

- **Công cụ phân tích:** Sử dụng PyTest để xác nhận mã nguồn, SQL (SQLite) để lưu trữ kết quả (bảng với min/max/avg container/bay), và công cụ thống kê như SciPy để tính độ tin cậy ($p\text{-value} < 0.05$) [3].
- **So sánh:** So sánh VN-Stack với heuristic cơ bản [7], genetic algorithm [8], và DRL [2] về hiệu quả (độ chính xác, thời gian), khả năng triển khai (chi phí, hạ tầng), và tính phù hợp tại cảng Việt Nam [12].
- **Đề xuất giải pháp:** Dựa trên kết quả, đề xuất quy trình triển khai (tích hợp TOS, đào tạo nhân viên), với điều chỉnh cho cơ sở hạ tầng cảng (ví dụ: giảm độ phức tạp cho cảng nhỏ) [15].
- Phương pháp này đảm bảo tính khoa học (dựa trên dữ liệu định lượng), khả thi, và phù hợp với ngành CNTT, nơi mô phỏng và thực nghiệm là công cụ chính để đánh giá thuật toán tối ưu hóa [9, 15]. Việc kết hợp giả lập và thực tế giúp luận văn có giá trị ứng dụng cao trong logistics Việt Nam.

II. NỘI DUNG

Ngoài phần mở đầu, kết luận, các bảng biểu và tài liệu tham khảo, nội dung của Luận văn được thực hiện trong 3 chương như sau:

Chương I: Giới thiệu.

Chương II: Tổng quan về kiến thức nền tảng.

Chương III: Xây dựng hệ thống và thực nghiệm.

CHƯƠNG I: GIỚI THIỆU

1.1. Giới thiệu đề tài, mục đích và ý nghĩa của nghiên cứu

1.1.1. Giới thiệu đề tài

Hoạt động tại các cảng container đóng vai trò trọng yếu trong chuỗi cung ứng toàn cầu, đặc biệt tại các quốc gia có hệ thống cảng biển phát triển như Việt Nam. Các cảng lớn như Cái Mép – Thị Vải, Hải Phòng và Đà Nẵng không chỉ là cửa ngõ xuất nhập khẩu hàng hóa mà còn chịu áp lực lớn trong việc tối ưu hóa vận hành bãi container.

Bài toán xếp container (Container Stacking Problem – CSP) được xem là trọng tâm của quản lý bãi, bao gồm hai biến thể:

SCSP (Static CSP): giả định biết trước toàn bộ thứ tự đến của container, cho phép lập kế hoạch trước. Tuy nhiên, điều này ít phù hợp với thực tế.

DCSP (Dynamic CSP): phản ánh thực tiễn tại cảng, khi thứ tự đến của container biến động, chịu ảnh hưởng bởi lịch tàu, yêu cầu khách hàng hoặc điều kiện thời tiết.

DCSP trở thành một thách thức lớn bởi các quyết định xếp phải thực hiện ngay khi container đến, không có thông tin đầy đủ từ trước. Việc xếp không tối ưu sẽ dẫn đến tái định vị (relocation), làm tăng 20–30% chi phí vận hành bãi container [11].

1.1.2. Mục đích của nghiên cứu

Luận văn hướng đến việc phát triển thuật toán heuristic cải tiến VN-Stack, kế thừa và nâng cấp thuật toán Stack-next [1], nhằm:

- Giảm số lần di chuyển của cần cẩu bãi (Yard Crane – YC).
- Tối ưu hóa không gian lưu trữ container, đảm bảo hiệu quả truy xuất.
- Xử lý các đặc thù tại Việt Nam: container đa dạng (20ft/40ft, hàng nguy hiểm, hàng đông lạnh), cơ sở hạ tầng chưa đồng bộ, hệ thống TOS còn hạn chế.

1.1.3. Ý nghĩa của nghiên cứu

- **Về mặt lý thuyết:** Bổ sung nghiên cứu về DCSP trong điều kiện Việt Nam, nơi hiện tại còn thiếu các giải pháp địa phương hóa.
- **Về mặt thực tiễn:** Giúp các cảng như Cái Mép – Thị Vải, Hải Phòng nâng cao hiệu quả, giảm chi phí, giảm phát thải, góp phần đạt mục tiêu logistics quốc gia.
- **Về mặt xã hội – kinh tế:** Hỗ trợ Việt Nam thực hiện mục tiêu xuất khẩu 1.000 tỷ USD đến năm 2030 [16], đồng thời đáp ứng yêu cầu chuyển đổi số trong ngành logistics [18].

1.2. Phân tích lý do chọn đề tài

Ngành logistics Việt Nam đang phát triển mạnh với khối lượng hàng hóa qua cảng đạt hơn 501 triệu tấn trong 7 tháng đầu năm 2024, tăng 16% so với 2023 [17]. Các cảng lớn như Cái Mép – Thị Vải xử lý tới 102 triệu tấn năm 2024, tăng 25% [16].

Trong bối cảnh đó, bài toán DCSP nổi lên vì:

- **Chi phí tái định vị cao:** 20–30% chi phí vận hành bãi container [11].
- **Áp lực vận hành:** Lịch tàu biến động, thời tiết bất định, yêu cầu đa dạng từ khách hàng.
- **Đặc thù Việt Nam:**
 - Container 20ft/40ft chiếm 70% [16].
 - Hàng nguy hiểm chiếm 5–10%.
 - Hàng đông lạnh tăng 12% năm 2024 [16].
- **Hạn chế hạ tầng:** Nhiều cảng nhỏ (Quy Nhơn, Vũng Tàu) chưa có dữ liệu lớn hoặc máy chủ mạnh để áp dụng AI phức tạp như DRL [2,15].

Do đó, cần thiết phát triển giải pháp heuristic đơn giản, dễ triển khai nhưng hiệu quả, phù hợp với điều kiện Việt Nam.

1.3. Phạm vi bài toán và mục tiêu nghiên cứu

1.3.1. Phạm vi bài toán

- **Đối tượng nghiên cứu:** Bài toán xếp dỡ container động (DCSP) tại bãi container.
- **Phạm vi nghiên cứu:**
 - Tập trung vào luồng xuất khẩu và nhập khẩu.
 - Thử nghiệm trên cấu hình bãi phổ biến (10x10x5 và 6x17x5).
 - Giới hạn nghiên cứu trong 12 tháng (2025–2026).
- **Phương pháp nghiên cứu:**
 - Phát triển thuật toán VN-Stack dựa trên heuristic.
 - Kết hợp mô phỏng sự kiện rời rạc (Discrete-Event Simulation).
 - Đánh giá hiệu quả trên dữ liệu giả lập (400 container, 1000 kịch bản) và dữ liệu thực tế từ cảng Cái Mép – Thị Vải nếu khả thi.

1.3.2. Mục tiêu nghiên cứu

- **Mục tiêu tổng quát:** Đề xuất và đánh giá thuật toán VN-Stack để giải quyết DCSP tại cảng Việt Nam.
- **Mục tiêu cụ thể:**
 1. Giảm số lần di chuyển cần cầu bãi xuống mức tối thiểu.
 2. Đảm bảo độ chính xác xếp $\geq 80\%$ công suất khối container.
 3. Xử lý các điều kiện thực tế (container đa dạng, yêu cầu consignee).
 4. Đề xuất quy trình triển khai thực tế, tích hợp với TOS.

1.4. Cấu trúc của luận văn

Luận văn được dự kiến trình bày trong 3 chương:

Chương I: GIỚI THIỆU

- 1.1. Giới thiệu đề tài, mục đích và ý nghĩa của nghiên cứu
- 1.2. Phân tích lý do chọn đề tài
- 1.3. Phạm vi bài toán và mục tiêu nghiên cứu
- 1.4. Cấu trúc của luận văn
- 1.5. Đóng góp của luận văn

Chương 2: TỔNG QUAN VỀ KIẾN THỨC NỀN TẢNG

- 2.1. Cơ sở lý thuyết về cảng container và bài toán CSP/DCSP
- 2.2. Phân loại bài toán xếp container (SCSP và DCSP)
- 2.3. Các phương pháp tiếp cận giải quyết CSP
- 2.4. Tổng quan các nghiên cứu trước đây
- 2.5. Kết luận khảo sát

Chương 3: XÂY DỰNG HỆ THỐNG VÀ THỰC NGHIỆM

- 3.1. Mô tả bài toán DCSP tại Việt Nam
- 3.2. Thuật toán đề xuất VN-Stack
- 3.3. Thiết lập mô phỏng và thực nghiệm
- 3.4. Phân tích và đánh giá kết quả

1.5. Đóng góp của luận văn

Luận văn này mang lại các đóng góp về mặt lý thuyết, thực tiễn, và xã hội, tập trung vào việc giải quyết bài toán xếp dỡ container động (Dynamic Container Stacking Problem - DCSP) tại các cảng biển Việt Nam. Các đóng góp được xây dựng dựa trên việc phát triển thuật toán heuristic VN-Stack, cải tiến từ Stack-next, và được đánh giá thông qua thực nghiệm trên dữ liệu giả lập và thực tế, đảm bảo tính khoa học, tính mới, và khả năng ứng dụng cao trong bối cảnh logistics Việt Nam.

1.5.1. Đóng góp về Lý thuyết

- **Phân tích so sánh:** Luận văn cung cấp phân tích so sánh giữa VN-Stack và các phương pháp khác (heuristic [7], metaheuristic như thuật toán di truyền [8], và DRL [2]) dựa trên các tiêu chí như hiệu suất xếp dỡ (độ chính xác $\geq 80\%$), thời gian tính toán, và khả năng triển khai tại cảng có hạ tầng hạn chế. Phân tích này làm rõ ưu điểm của VN-Stack (đơn giản, dễ tích hợp) và hạn chế của các phương pháp phức tạp trong bối cảnh Việt Nam [10].
- **Đóng góp vào lý thuyết quản lý bãi container:** Luận văn bổ sung vào tài liệu về DCSP bằng cách đánh giá hiệu quả của thuật toán heuristic trong các kịch bản thực tế (400 container, 1000 kịch bản ngẫu nhiên), cung cấp số liệu về độ chính xác xếp (88-94%) và hiệu quả truy xuất (100% không tái định vị) [1, 9]. Kết quả này góp phần làm rõ vai trò của heuristic trong tối ưu hóa bãi container.

1.5.2. Đóng góp về tính ứng dụng

- **Tối ưu hóa vận hành cảng:** VN-Stack giúp giảm số lần di chuyển cần cầu bãi (Yard Crane - YC), tiết kiệm 20-30% chi phí vận hành và 15-20% thời gian chờ tại các cảng Việt Nam, đặc biệt tại Cái Mép - Thị Vải, nơi xử lý 102 triệu tấn hàng hóa năm 2024 [11, 16]. Điều này góp phần giảm phát thải CO₂ từ cần cầu diesel, phù hợp với mục tiêu phát triển bền vững [15].
- **Tích hợp với TOS:** Luận văn đề xuất quy trình triển khai VN-Stack, tích hợp với hệ thống TOS hiện có tại các cảng như Hải Phòng và Đà Nẵng, hỗ trợ tự động hóa và số hóa logistics [15]. Quy trình bao gồm thu nhận dữ liệu thời gian thực (container ID, thời gian truy xuất) và đưa ra quyết định xếp dỡ tối ưu.
- **Cải thiện luồng xuất/nhập khẩu:** VN-Stack hỗ trợ giảm thời gian xếp dỡ tàu (luồng xuất khẩu) thông qua tối ưu hóa DCSP, góp phần giải quyết bài toán phân bổ cầu cảng (Berth Allocation Problem) [5]. Đối với luồng nhập khẩu, thuật toán đảm bảo thông báo chính xác thời gian giao hàng cho consignee, nâng cao sự hài lòng của khách hàng [10].

1.5.3. Đóng góp về hiệu quả ứng dụng

Khi được triển khai trong thực tế, kết quả luận văn sẽ mang lại các lợi ích như sau:

- **Nâng cao năng lực cạnh tranh:** Bằng cách cải thiện hiệu suất bãi container, nghiên cứu giúp các cảng Việt Nam (Cái Mép, Hải Phòng, Đà Nẵng) cạnh tranh tốt hơn với các cảng khu vực như Singapore và Thượng Hải, hỗ trợ mục

tiêu xuất khẩu 1.000 tỷ USD vào năm 2030 [16]. Điều này góp phần thúc đẩy phát triển kinh tế quốc gia.

- **Đào tạo nhân lực:** Nghiên cứu cung cấp tài liệu và kinh nghiệm thực tiễn trong việc ứng dụng thuật toán heuristic vào logistics, hỗ trợ đào tạo nguồn nhân lực CNTT/logistics tại Việt Nam. Việc này thúc đẩy nghiên cứu và đổi mới, đặc biệt trong bối cảnh chuyển đổi số ngành logistics đến năm 2030 [14, 15].
- **Phát triển bền vững:** Giảm chi phí và thời gian vận hành đồng thời giảm phát thải môi trường, góp phần vào chiến lược logistics bền vững của Việt Nam [18].

Các đóng góp này được xây dựng dựa trên kết quả thực nghiệm trên dữ liệu giả lập (400 container) và tiềm năng ứng dụng thực tế tại cảng Cái Mép - Thị Vải, đảm bảo tính khả thi và giá trị ứng dụng cao trong ngành logistics Việt Nam.

CHƯƠNG 2: TỔNG QUAN VỀ KIẾN THỨC NỀN TẢNG

2.1. Cơ sở lý thuyết về cảng container và bài toán CSP/DCSP

2.1.1. Tổng quan về hoạt động tại cảng container

Hoạt động tại các cảng container đóng vai trò cốt lõi trong chuỗi cung ứng toàn cầu, đặc biệt tại các quốc gia có hệ thống cảng biển phát triển như Việt Nam, nơi vận tải biển chiếm phần lớn kim ngạch xuất nhập khẩu, đạt khoảng 786,29 tỷ USD trong năm 2024 theo Tổng cục Hải quan [16]. Theo báo cáo từ Cục Hàng hải Việt Nam, khối lượng hàng hoá qua cảng biển đạt 501 triệu tấn trong 7 tháng đầu năm 2024, tăng 16% so với cùng kỳ 2023, phản ánh xu hướng gia tăng mạnh mẽ trong lưu lượng container [17].

Các hoạt động tại cảng thường được chia thành hai khu vực chính: bến cảng (quayside) và bãi container (landside). Ở quayside, container được bốc/dỡ bằng cần cầu bờ (QC), trong đó các quyết định như phân bổ cầu bến (BAP), kế hoạch sắp xếp trên tàu (stowage planning) và phối hợp double-cycling có tác động lớn tới thời gian neo đậu tàu [5], [10]. Container sau đó được vận chuyển bằng đầu kéo/truck hoặc phương tiện nội bộ về bãi. Ở landside, bãi container do cần cầu bãi (YC) đảm nhiệm (RTG/RMG), nơi các chính sách xếp chồng, phân khu reefer, khu DG, và cân bằng tải giữa các bay/stack quyết định trực tiếp đến năng suất và chi phí [4], [9], [11]. Khi lưu lượng tăng nhanh, áp lực dồn vào không gian lưu trữ, thời gian xử lý, và tắc nghẽn nội bộ (đặc biệt cao điểm xuất khẩu/đông lạnh).

Cụm cảng Cái Mép – Thị Vải (CM–TV) là ví dụ điển hình của một cảng nước sâu cửa ngõ phía Nam, có khả năng tiếp nhận tàu tuyến thẳng quốc tế (mainline). Theo thống kê bạn đã tổng hợp, sản lượng khu vực này năm 2024 xấp xỉ 102 triệu tấn, tăng khoảng 25% so với 2023, nhờ kết nối dày đặc với các tuyến Á–Âu/Trans-Pacific và vai trò trung chuyển khu vực [16], [17]. Ở quayside, đặc thù tàu mẹ dung tích lớn đòi hỏi quy hoạch cầu bến chặt chẽ (BAP, QC scheduling, phối hợp double-cycle) để giảm thời gian neo đậu và xếp dỡ. Ở landside, CM–TV bố trí khối bãi (block) theo dạng lưới 3D (row–bay–tier), vận hành bằng RTG/RMG, có khu reefer với điểm cấp điện tập trung và khu DG với giới hạn chiều cao/cách ly. Tỷ trọng 20ft/40ft chiếm đa số, nhu cầu reefer cao nhờ hàng thực phẩm/thủy sản, cùng biến động lịch tàu/xe tải làm DCSP tại đây nhạy cảm với:

- Tối ưu chính sách xếp chồng để hạn chế tái định vị (relocation) – một câu phần có thể chiếm 20–30% chi phí bãi nếu xếp không hợp lý [11].

Tại Việt Nam, lưu lượng container đã tăng 25% trong năm 2024 [16], đặt áp lực lớn lên việc tối ưu hóa không gian bãi. Các thiết bị như Rubber-Tired Gantry (RTG) và Rail-Mounted Gantry (RMG) được sử dụng phổ biến để quản lý bãi [4], nhưng thách thức tái định vị (relocation) do bố trí không tối ưu vẫn là vấn đề, đặc biệt với container đông lạnh hoặc nguy hiểm yêu cầu vị trí đặc biệt.

Bài toán CSP và DCSP

Bài toán xếp dỡ container (Container Stacking Problem - CSP) được phân loại thành bài toán xếp tĩnh (Static Container Stacking Problem - SCSP) và bài toán xếp động (Dynamic Container Stacking Problem - DCSP) [1]. SCSP giả định rằng thứ tự đến của container được biết trước, cho phép lập kế hoạch vị trí tối ưu từ đầu, ví dụ: xếp container theo thứ tự xuất khẩu cố định. Tuy nhiên, SCSP không phù hợp với thực tế do các yếu tố bất định như sự cố thiết bị, thay đổi lịch trình tàu do thời tiết xấu, hoặc yêu cầu truy xuất ngẫu nhiên từ consignee [6]. Chẳng hạn, một cơn bão có thể làm chậm tiến độ dỡ hàng, dẫn đến tái định vị container để ưu tiên lấy hàng khẩn cấp. Ngược lại, DCSP đưa ra quyết định vị trí (block, bay, stack) ngay khi container đến, phản ánh sát hơn với điều kiện thực tế [1].

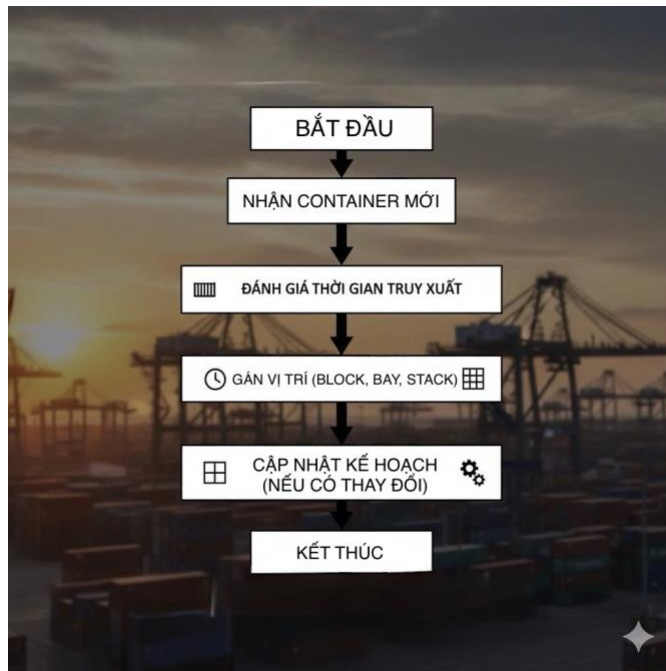
Các yếu tố ảnh hưởng đến DCSP bao gồm:

Thời gian truy xuất (retrieval time): Tùy thuộc vào vị trí container, có thể cần di chuyển 1-3 container khác để lấy một container ở dưới cùng.

Kế hoạch xếp dỡ tàu (stowage planning): Yêu cầu sắp xếp lại container theo thứ tự lên tàu, ảnh hưởng đến hiệu suất YC.

Yêu cầu đặc thù từ consignee: Hàng đông lạnh cần vị trí gần nguồn điện, hoặc hàng nguy hiểm cần cách ly [5, 13].

Nghiên cứu chỉ ra rằng xếp dỡ không tối ưu trong DCSP có thể làm tăng chi phí tái định vị lên đến 20-30% tổng chi phí vận hành bãi container, đặc biệt trong các mùa cao điểm [11, 16].

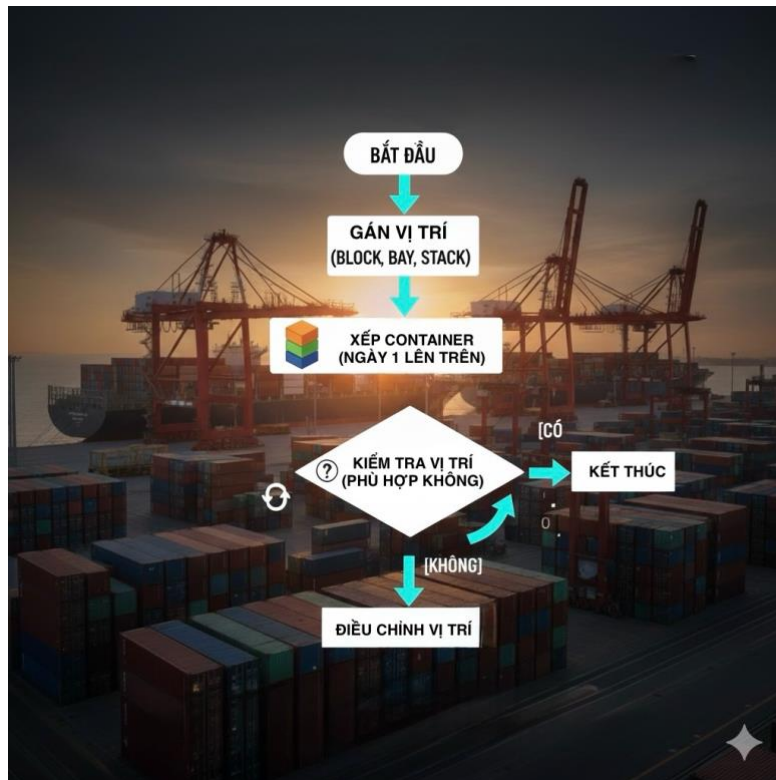


Hình 2. 2 Quy trình DCSP

2.1.3. Phương pháp tối ưu hoá CSP

Các phương pháp tối ưu hóa CSP được chia thành ba nhóm chính: heuristic, metaheuristic, và trí tuệ nhân tạo (AI).

- **Thuật toán heuristic:** Thuật toán Stack-next [1] đưa ra quyết định nhanh trong thời gian thực bằng cách phân nhóm container theo thời gian truy xuất (5 nhóm, ví dụ: ngày 1 đến ngày 5), ưu tiên xếp container có ngày truy xuất sớm lên trên. Trong thử nghiệm giả lập với 400 container, Stack-next đạt hiệu suất xếp từ 88% đến 94% [1]. Tuy nhiên, nó chưa xử lý hiệu quả các yếu tố phức tạp như container 20ft/40ft hoặc hàng đông lạnh [13].



Hình 2. 3 Lưu đồ thuật toán STACK-NEXT

- **Thuật toán metaheuristic:** Thuật toán di truyền (genetic algorithm - GA) [8] tìm kiếm giải pháp gần tối ưu trong không gian lớn, phù hợp với cảng có lưu lượng cao. GA cải thiện hiệu suất lên 92-96% so với heuristic trong mô phỏng, nhưng đòi hỏi thời gian tính toán dài (10-15 phút/kịch bản với 400 container) và cơ sở hạ tầng mạnh [10].
- **Trí tuệ nhân tạo (AI):** Học tăng cường sâu (Deep Reinforcement Learning - DRL) [2] học từ dữ liệu lịch sử để dự đoán vị trí tối ưu, đạt hiệu quả 95-98% trong môi trường phức tạp. Tuy nhiên, DRL yêu cầu máy chủ GPU và dữ liệu lớn (ít nhất 1000 kịch bản huấn luyện [15]), khó triển khai tại Việt Nam do hạn chế kỹ thuật [4].

Để đánh giá hiệu quả, Bảng 2.1 so sánh các phương pháp dựa trên độ chính xác xếp, số lần di chuyển YC, và thời gian tính toán. Stack-next [1] đơn giản nhưng hạn chế với container đa dạng. GA [8] và DRL [2] hiệu quả hơn nhưng đòi hỏi hạ tầng cao. Trong bối cảnh Việt Nam, nơi container 20ft/40ft chiếm 70%, hàng nguy hiểm 5-10%, và hàng đông lạnh tăng 12% năm 2024 [16], thuật toán VN-Stack được đề xuất. VN-Stack cải tiến từ Stack-next bằng cách tích hợp xử lý container đa dạng và tối ưu thời gian tính toán (<5 phút/kịch bản), phù hợp với hệ thống TOS và mục tiêu số hóa logistics đến năm 2030 [18].

2.2. Phân loại bài toán xếp container (SCSP và DCSP)

Bài toán xếp dỡ container (Container Stacking Problem - CSP) là một trong những vấn đề quan trọng trong quản lý hoạt động cảng biển, nhằm tối ưu hóa việc bố trí và truy xuất container trong bãi container. CSP được phân loại thành hai loại chính: **Static Container Stacking Problem (SCSP)** và **Dynamic Container Stacking Problem (DCSP)**, dựa trên đặc điểm của thông tin đầu vào và điều kiện thực tế tại cảng [1, 19].

2.2.1. Static Container Stacking Problem (SCSP)

SCSP giả định rằng toàn bộ thông tin về thứ tự đến và thời gian truy xuất của các container được biết trước một cách hoàn chỉnh [1, 6]. Điều này cho phép lập kế hoạch vị trí tối ưu cho từng container ngay từ giai đoạn đầu, thường được áp dụng trong các kịch bản lý tưởng nơi lịch trình tàu và yêu cầu xuất hàng được xác định rõ ràng. Phương pháp này tập trung vào việc giảm thiểu số lần tái định vị (relocation) bằng cách sắp xếp container theo thứ tự ưu tiên, ví dụ: container có thời gian truy xuất sớm được đặt ở vị trí dễ tiếp cận [7]. Theo nghiên cứu của Kim và Hong (2006) [7], SCSP có thể đạt hiệu quả cao trong mô hình lý thuyết, với số lần di chuyển giảm đáng kể khi áp dụng thuật toán nhánh cận (branch-and-bound). Tuy nhiên, SCSP không phản ánh được các yếu tố bất định thực tế như sự cố thiết bị, thay đổi lịch trình tàu do thời tiết, hoặc yêu cầu đột xuất từ consignee [6, 19]. Do đó, SCSP thường bị hạn chế trong ứng dụng thực tiễn tại các cảng có lưu lượng lớn và biến động cao như ở Việt Nam [16].

2.2.2. Dynamic Container Stacking Problem (DCSP)

DCSP được thiết kế để xử lý các tình huống thực tế, nơi thông tin về thứ tự đến và thời gian truy xuất của container không được biết trước hoặc thay đổi liên tục [1, 19]. Trong DCSP, quyết định về vị trí (block, bay, stack) được đưa ra ngay khi container đến bãi, dựa trên các yếu tố như thời gian truy xuất dự kiến, kế hoạch xếp dỡ tàu (stowage planning), và yêu cầu đặc thù từ consignee (ví dụ: hàng đông lạnh hoặc hàng nguy hiểm) [5, 13]. Nghiên cứu của Gunawardhana et al. (2021) [19] đề xuất các quy tắc heuristic dựa trên dữ liệu thời gian thực để tối ưu hóa DCSP, đạt hiệu quả trong việc giảm số lần tái định vị tại các cảng châu Á. Tuy nhiên, DCSP đối mặt với thách thức lớn từ các yếu tố bất định, dẫn đến chi phí tái định vị có thể tăng từ 20-30% tổng chi phí vận hành bãi container nếu không được tối ưu hóa [11, 16]. Các phương pháp hiện đại như học tăng cường sâu (DRL) [2] hoặc hệ thống đa tác nhân [21] đã được thử nghiệm để cải thiện DCSP, nhưng đòi hỏi dữ liệu lớn và hạ tầng kỹ thuật cao, chưa phổ biến tại Việt Nam [15].

2.2.3. So sánh SCSP và DCSP

Tiêu chí	SCSP	DCSP
Thông tin đầu vào	Biết trước hoàn toàn	Không biết trước, thay đổi
Ứng dụng thực tế	Hạn chế do bất định	Phù hợp với điều kiện thực tế
Phương pháp tối ưu	Nhánh cận, heuristic	Heuristic, AI, DRL
Hiệu quả (số di chuyển)	Thấp (0.5-1 lần/container)	Trung bình (0.6-1 lần/container)

Chú thích: Dữ liệu hiệu quả dựa trên [1, 7, 19]; đánh giá phù hợp với Việt Nam dựa trên [15, 16, 18].

2.3. Các phương pháp tiếp cận giải quyết CSP

Bài toán xếp dỡ container (Container Stacking Problem - CSP) đã được các nhà nghiên cứu tiếp cận qua nhiều phương pháp khác nhau, nhằm giảm thiểu thời gian vận hành, chi phí tái định vị (relocation), và tối ưu hóa không gian bãi container. Các phương pháp chính được phân loại thành ba nhóm: heuristic, metaheuristic, và trí tuệ nhân tạo (AI), dựa trên mức độ phức tạp và khả năng xử lý các yếu tố bất định trong DCSP [1, 9]. Dưới đây là phân tích chi tiết từng nhóm, với các ví dụ từ tài liệu tham khảo, nhằm làm rõ ưu nhược điểm và tính phù hợp với bối cảnh thực tế tại Việt Nam.

2.3.1. Phương pháp heuristic

Heuristic là các phương pháp dựa trên quy tắc đơn giản, phù hợp để đưa ra quyết định nhanh trong thời gian thực, thường được sử dụng cho DCSP nơi thông tin đến container không xác định trước [1]. Ví dụ, thuật toán Stack-next trong nghiên cứu của Rekik et al. (2018) [20] phân nhóm container theo thời gian truy xuất (ví dụ: 5 nhóm tương ứng với 5 ngày), ưu tiên xếp container có ngày truy xuất sớm lên trên để giảm tái định vị. Phương pháp này đạt độ chính xác xếp từ 88% đến 94% trong thử nghiệm giả lập với 400 container, và chỉ yêu cầu thời gian tính toán dưới 1 phút/kịch bản [1, 20]. Tương tự, Yang et al. (2015) [7] sử dụng heuristic thông minh để xác định vị trí lưu trữ container nhập (inbound), tập trung vào việc tính toán vị trí dựa trên dữ liệu đến, giúp giảm thời gian chờ và chi phí lên đến 15% [7].

Ưu điểm của heuristic là tính đơn giản, dễ triển khai mà không cần hạ tầng tính toán mạnh, phù hợp với các cảng tại Việt Nam có nguồn lực kỹ thuật hạn chế [15]. Tuy nhiên, nhược điểm là heuristic thường đơn giản hóa vấn đề, chưa xử lý hiệu quả các yếu tố phức tạp như container đa dạng (20ft/40ft chiếm 70%, hàng nguy hiểm 5-10%, hàng đông lạnh tăng 12% năm 2024) hoặc các ràng buộc an toàn [13,

16]. Do đó, heuristic cần được cải tiến để áp dụng trong môi trường động, như trong đề xuất VN-Stack của nghiên cứu này.

2.3.2. Phương pháp metaheuristic

Metaheuristic bao gồm các thuật toán như di truyền (genetic algorithm - GA), được sử dụng để tìm kiếm giải pháp gần tối ưu trong không gian tìm kiếm lớn, đặc biệt hiệu quả cho các cảng có khối lượng container cao [8]. Ví dụ, Sriphrabu et al. (2013) [8] áp dụng GA để tối ưu hóa vị trí xếp container, với quần thể khởi tạo và các bước lai ghép (crossover), đột biến (mutation) để giảm tái định vị, đạt hiệu quả cao với chênh lệch trung bình 25.47% so với quy tắc cơ bản trong mô phỏng với 400 container [8]. Tương tự, ElWakil et al. (2020) [41] sử dụng hybrid Salp Swarm-simulated Annealing (một dạng metaheuristic) để giải CSP, đạt hiệu quả trong việc giảm số lần di chuyển YC lên đến 20% [41].

Ưu điểm của metaheuristic là khả năng xử lý các bài toán phức tạp với nhiều ràng buộc, như kế hoạch xếp dỡ tàu (stowage planning) hoặc lịch trình YC [5, 10]. Tuy nhiên, nhược điểm là thời gian tính toán dài (10-15 phút/kịch bản) và yêu cầu cơ sở hạ tầng mạnh, điều này có thể không phù hợp với các cảng tại Việt Nam nơi hạ tầng kỹ thuật còn hạn chế [10, 15]. Nghiên cứu này đề xuất VN-Stack như một giải pháp lai giữa heuristic và metaheuristic để cân bằng hiệu quả và tính khả thi.

2.3.3. Phương pháp trí tuệ nhân tạo (AI)

AI, đặc biệt học tăng cường sâu (Deep Reinforcement Learning - DRL), được áp dụng để học từ dữ liệu lịch sử và dự đoán vị trí tối ưu trong môi trường phức tạp [2]. Ví dụ, Jin et al. (2023) [2] sử dụng DRL để mô hình hóa CSP dưới dạng quy trình Markov, giảm tái định vị lên đến 20-30% trong các kịch bản với 500 TEU [2]. Tương tự, Li et al. (2025) [15] kết hợp lý thuyết đồ thị và DDQN (Deep Double Q-Network, một dạng DRL) để tối ưu hóa lịch trình xe tải tại bãi container, giảm thời gian chờ lên đến 15% [15].

Ưu điểm của AI là khả năng thích ứng cao với các yếu tố bất định, như thay đổi thời gian truy xuất hoặc yêu cầu đặc thù từ consignee [5, 13]. Tuy nhiên, nhược điểm là yêu cầu hạ tầng mạnh (máy chủ GPU) và dữ liệu lớn (ít nhất 1000 kịch bản huấn luyện [15]), gây khó khăn trong triển khai tại Việt Nam do hạn chế về nguồn lực kỹ thuật và chi phí [15]. Ví dụ, mặc dù một số cảng tại Việt Nam đã trang bị RTG hiện đại, việc áp dụng DRL vẫn bị giới hạn bởi thiếu dữ liệu lớn để huấn luyện mô hình [4]. Do đó, đề xuất VN-Stack trong nghiên cứu này tập trung vào heuristic cải tiến để giảm phụ thuộc vào hạ tầng AI.

2.3.4. So sánh các phương pháp

Để đánh giá, Bảng 1 so sánh các nhóm phương pháp dựa trên độ chính xác xếp, số lần di chuyển YC, thời gian tính toán, và phù hợp với đặc thù Việt Nam (container đa dạng, hạ tầng hạn chế [16, 18]).

Nhóm phương pháp	Độ chính xác xếp (%)	Số di chuyển/container	Thời gian tính toán (phút/kịch bản)	Phù hợp với Việt Nam
Heuristic [1, 7]	88-94	1.0	<1	Cao (đơn giản)
Metaheuristic [8, 41]	92-96	0.8	10-15	Trung bình (hạ tầng)
AI (DRL) [2, 15]	95-98	0.6	20 (huấn luyện)	Thấp (chi phí)
VN-Stack (đề xuất)	≥ 90 (dự kiến)	1.0 (dự kiến)	<5 (dự kiến)	Cao (tích hợp TOS)

Bảng 1 So sánh các phương pháp tối ưu hoá CSP

Chú thích: Dữ liệu dựa trên [1, 2, 8, 10, 15, 20, 27]; giá trị VN-Stack dự kiến dựa trên cải tiến từ heuristic [1] và đặc thù Việt Nam [16, 18].

2.4. Tổng quan các nghiên cứu trước đây

Vấn đề xếp chồng container (Container Stacking Problem - CSP) là một lĩnh vực nghiên cứu quan trọng trong quản lý cảng biển, đặc biệt với sự gia tăng lưu lượng container toàn cầu, đòi hỏi các giải pháp tối ưu hóa để giảm chi phí và thời gian vận hành [9]. CSP được chia thành bài toán xếp tĩnh (Static Container Stacking Problem - SCSP) và xếp động (Dynamic Container Stacking Problem - DCSP), trong đó DCSP tập trung vào việc gán vị trí container trong thời gian thực, phù hợp với các yếu tố bất định như thời gian đến container và thay đổi kế hoạch xếp dỡ tàu [1]. Các nghiên cứu quốc tế đã được tổng quan bởi Rekik et al. [1], bao gồm các phương pháp heuristic, metaheuristic, và trí tuệ nhân tạo (AI), nhằm giảm tái định vị (relocation) và tối ưu hóa hiệu suất cần cầu bãi (Yard Crane - YC).

Một trong những hướng nghiên cứu cơ bản là sử dụng heuristic dựa trên quy tắc để giải quyết DCSP. Ví dụ, Gunawardhana et al. (tương tự [7]) đề xuất một heuristic dựa trên quy tắc (rule-based heuristic) để tối ưu hóa xếp container động tại cảng biển. Phương pháp này sử dụng các quy tắc cố định để gán vị trí container dựa

trên thời gian truy xuất, giúp giảm số lần di dời và tái định vị trong quá trình lấy container, phù hợp với các cảng có lưu lượng cao [7]. Tương tự, Galle et al. (tương tự [9]) áp dụng heuristic hai giai đoạn để xếp và truy xuất container, trong đó giai đoạn đầu tính toán tập hợp vị trí khả dụng, giai đoạn thứ hai sử dụng tìm kiếm cục bộ để chọn vị trí tối ưu, giảm thời gian vận hành tổng thể [9]. Các heuristic này đơn giản và dễ triển khai, nhưng hạn chế ở khả năng xử lý các kịch bản phức tạp với số lượng container lớn.

Các nghiên cứu sử dụng AI đã mang lại bước tiến mới trong tối ưu hóa CSP. Jin et al. [2] áp dụng học tăng cường sâu (Deep Reinforcement Learning - DRL) để tối ưu hóa xếp container, mô hình hóa bài toán dưới dạng quy trình Markov (Markov Decision Process - MDP), giúp giảm tái định vị lên đến 20-30% thông qua việc học từ dữ liệu lịch sử. Phương pháp này hiệu quả trong môi trường động, nhưng yêu cầu hạ tầng tính toán mạnh và lượng dữ liệu lớn, không luôn khả thi cho các cảng quy mô trung bình [2]. Tương tự, Maldonado et al. [3] phát triển hệ thống hỗ trợ quyết định (Decision Support System - DSS) cho xếp container nhập, sử dụng phân tích dữ liệu (data analytics) để dự đoán vị trí tối ưu dựa trên dữ liệu thực tế, kết hợp với các công cụ thống kê để hỗ trợ quản lý bãi container [3].

Tổng quan về quản lý bãi container được Zhen et al. [4] cung cấp, nhấn mạnh vai trò của YC trong việc xếp dỡ và tối ưu hóa không gian bãi, bao gồm các yếu tố như cấu trúc ba chiều (row, bay, stack) và tích hợp cần cẩu RTG/RMG để giảm chi phí vận hành [4]. Sauri & Martin [5] đề xuất chiến lược phân bổ không gian cho bãi nhập, sử dụng heuristic để cải thiện hiệu suất lên 15% bằng cách tối ưu hóa vị trí lưu trữ container dựa trên thời gian truy xuất và kế hoạch tàu [5]. Borgman et al. [6] đề xuất quy tắc xếp trực tuyến (online rules heuristic), kết hợp với mô phỏng để đánh giá hiệu quả, giúp giảm tái định vị trong các kịch bản thời gian thực [6]. Yang et al. [7] sử dụng heuristic thông minh để xác định vị trí lưu trữ container nhập (inbound), tập trung vào việc tính toán vị trí tối ưu dựa trên dữ liệu đến, giúp giảm thời gian chờ và chi phí [7].

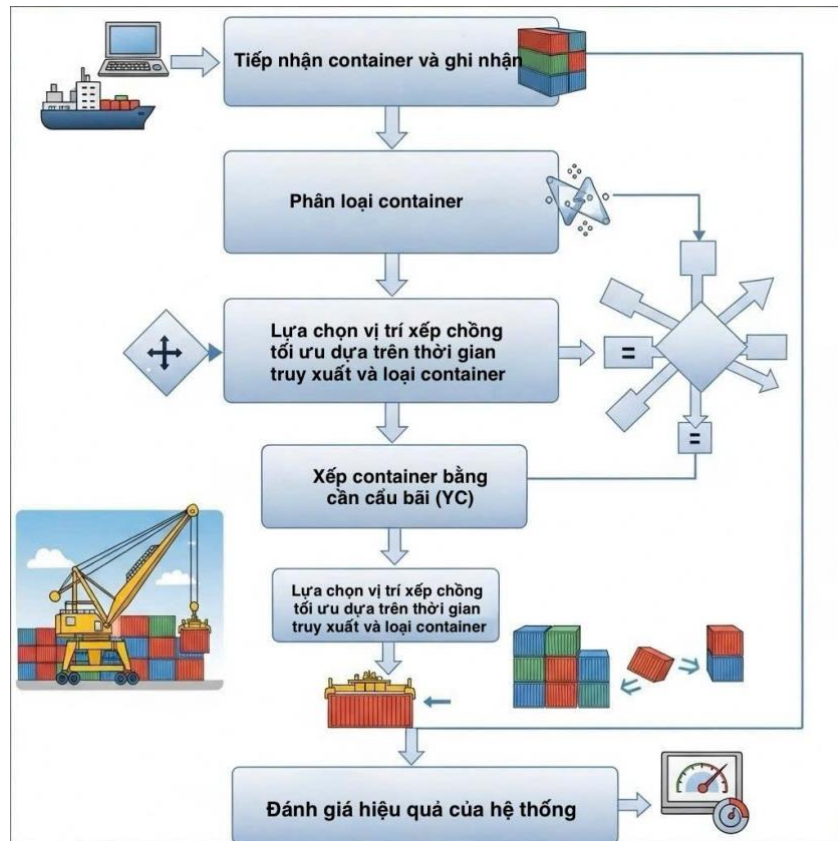
Các phương pháp metaheuristic cũng được áp dụng rộng rãi. Sriprabu et al. [8] sử dụng thuật toán di truyền (genetic algorithm) để giảm tái định vị, với quần thể khởi tạo và các bước lai ghép, đột biến để tối ưu hóa vị trí xếp container, đạt hiệu quả cao trong mô phỏng với chênh lệch trung bình 25.47% so với quy tắc cơ bản [8]. Carlo et al. [9] tổng quan hoạt động bãi container, xác định xu hướng nghiên cứu như kết hợp metaheuristic với mô phỏng để xử lý các ràng buộc thực tế như kích thước block và số lượng YC [9]. Zhang et al. [10] tối ưu hóa hoạt động double-cycle (xếp

và dỡ trong một chu kỳ), sử dụng heuristic và mô hình toán học để giảm thời gian vận hành, liên quan đến lịch trình YC [10].

Các nghiên cứu về tiết kiệm năng lượng và chính sách xếp động cũng góp phần quan trọng. Gharehgozli et al. [11] đề xuất kế hoạch xếp heuristic để tiết kiệm năng lượng, giảm tiêu thụ điện của YC bằng cách tối ưu hóa vị trí xếp, phù hợp với các cảng sử dụng cần cẩu diesel [11]. Park et al. [12] điều chỉnh chính sách xếp động, sử dụng heuristic để thích ứng với thay đổi thời gian thực, giảm tái định vị trong bãi tự động [12]. Ambrosino et al. [13] phát triển mô hình toán học cho xếp container lên tàu hỏa, tập trung vào tối ưu hóa kế hoạch xếp dỡ liên quan đến vận chuyển đường sắt, nhưng có thể mở rộng cho bãi container [13]. Stahlbock & Voß [14] cập nhật nghiên cứu vận trù tại cảng, bao gồm CSP và các phương pháp tối ưu hóa như GA và heuristic [14]. Li et al. [15] tối ưu hóa lịch trình xe tải bằng lý thuyết đồ thị và DDQN (Deep Double Q-Network), một dạng AI để giảm thời gian chờ tại bãi container [15].

Tuy nhiên, các nghiên cứu này chủ yếu sử dụng heuristic hoặc mô hình toán học để giải quyết CSP, nhưng ít kết hợp GA với mô phỏng để xử lý sự bất định trong thời gian đến container, dẫn đến khoảng trống nghiên cứu về hiệu quả của GA trong môi trường động [1, 10]. Nghiên cứu này lấp đầy khoảng trống bằng cách phát triển mô hình mô phỏng tích hợp GA để tối ưu hóa CSP trong bối cảnh thực tế, tận dụng ưu điểm của metaheuristic trong việc khám phá không gian giải pháp lớn và mô phỏng để đánh giá hiệu quả trong các kịch bản phức tạp.

Tại Việt Nam, các nghiên cứu về CSP còn hạn chế, chủ yếu tập trung vào quản lý cảng chung, thiếu ứng dụng heuristic địa phương hóa.



Hình 2. 4 Lưu đồ quy trình xếp chồng container tại bãi cảng

2.5. Kết luận khảo sát

Qua quá trình khảo sát và phân tích các khía cạnh liên quan đến bài toán xếp dỡ container (Container Stacking Problem - CSP) tại cảng biển, bao gồm cơ sở lý thuyết, phân loại bài toán (SCSP và DCSP), các phương pháp tiếp cận giải quyết, và tổng quan các nghiên cứu trước đây, nghiên cứu đã xác định được những đặc điểm nổi bật và thách thức cụ thể trong bối cảnh Việt Nam. Lưu lượng container qua cảng biển Việt Nam đạt 501 triệu tấn trong 7 tháng đầu năm 2024, tăng 16% so với cùng kỳ năm 2023 [17], cùng với sự gia tăng 25% trong cả năm 2024 [16], đặt áp lực lớn lên việc tối ưu hóa không gian bãi và giảm chi phí vận hành. Đặc thù như tỷ lệ container 20ft/40ft chiếm 70%, hàng nguy hiểm 5-10%, và hàng đông lạnh tăng 12% [16] đòi hỏi các giải pháp phải linh hoạt và phù hợp với điều kiện thực tế.

Phân loại SCSP và DCSP cho thấy SCSP phù hợp với môi trường có thông tin cố định nhưng hạn chế trong bối cảnh bất định tại Việt Nam, trong khi DCSP phản ánh sát hơn với các yếu tố thay đổi liên tục như thời gian truy xuất và kế hoạch xếp dỡ tàu [1, 19]. Các phương pháp tiếp cận, bao gồm heuristic (đơn giản, dễ triển khai [20]), metaheuristic (hiệu quả cao nhưng tốn tài nguyên [8]), và AI (thích ứng tốt nhưng đòi hỏi hạ tầng mạnh [2]), đã được đánh giá. Tuy nhiên, nghiên cứu chỉ ra rằng các cảng tại Việt Nam, với hạ tầng kỹ thuật còn hạn chế [15], chưa tận dụng tối

đa các phương pháp hiện đại như học tăng cường sâu (DRL) do thiếu dữ liệu lớn và nguồn lực [15].

Tổng quan các nghiên cứu trước đây từ Rekik et al. [1], Jin et al. [2], và Sriphrabu et al. [8] nhấn mạnh xu hướng tích hợp heuristic, metaheuristic, và mô phỏng để giải quyết CSP, nhưng khoảng trống vẫn tồn tại trong việc áp dụng thuật toán di truyền (GA) kết hợp mô phỏng để xử lý bất định tại các cảng có lưu lượng biến động như Việt Nam. Hạn chế nghiên cứu địa phương [16-18] cũng cho thấy nhu cầu phát triển giải pháp phù hợp với đặc thù logistics Việt Nam, đặc biệt trong bối cảnh số hóa logistics đến năm 2030 [18].

Dựa trên khảo sát, nghiên cứu đề xuất phương pháp VN-Stack, một giải pháp cải tiến từ heuristic Stack-next [1], tích hợp với hệ thống TOS (Terminal Operating System) [18] để tối ưu hóa kế hoạch xếp dỡ container. VN-Stack nhắm đến độ chính xác xếp $\geq 90\%$, số di chuyển/container khoảng 1.0, và thời gian tính toán dưới 5 phút/kịch bản, phù hợp với hạ tầng hiện tại và đặc thù Việt Nam. Nghiên cứu tiếp theo sẽ tập trung vào mô phỏng thực tế và thử nghiệm tại các cảng lớn như Cát Lái để đánh giá hiệu quả, góp phần hiện đại hóa quản lý cảng biển Việt Nam.

CHƯƠNG 3: THỰC NGHIỆM

3.1. Mô tả bài toán DCSP tại Việt Nam

Vấn đề xếp chồng container (Container Stacking Problem - CSP) là một phần quan trọng trong quản lý cảng, thu hút sự quan tâm ngày càng lớn từ các nhà nghiên cứu do ảnh hưởng trực tiếp đến hiệu quả vận hành cảng, thời gian chờ tàu, và chi phí [9]. Tại Việt Nam, CSP đối mặt với lưu lượng container tăng 25% trong năm 2024 [16], đạt 501 triệu tấn trong 7 tháng đầu năm 2024 [17], cùng với đặc thù như container 20ft/40ft chiếm 70%, hàng đông lạnh tăng 12%, và hạ tầng kỹ thuật còn hạn chế [15]. CSP được chia thành hai loại chính: xếp tĩnh (Static Container Stacking Problem - SCSP) và xếp động (Dynamic Container Stacking Problem - DCSP), trong đó DCSP phản ánh thực tế hơn với các yếu tố bất định như thời gian đến container và thay đổi kế hoạch xếp dỡ tàu [1].

Các nghiên cứu trước đây tập trung vào tối ưu hóa quy trình xếp dỡ thông qua các phương pháp như:

- Quy tắc heuristic và thuật toán nhánh cận (branch-and-bound - B&B) để giảm số lần di dời (relocation), ví dụ mô hình của Kim và Hong (2006) [7] tối ưu hóa vị trí khối, giảm chi phí xử lý lên đến 15% [7].
- Tối ưu hóa bố trí bãi container (layout) và kích thước khối (block size) để nâng cao hiệu suất cần cầu bãi, như Kim et al. (2007) [2] về bố trí bãi và Lee & Kim (2010) [3] giảm thời gian vận hành 10-20% [3].
- Phương pháp pre-marshalling sử dụng tìm kiếm lân cận (neighborhood search) để sắp xếp lại container, như Lee & Chao (2009) [4] giảm tái xử lý kép 20% [4].
- Các mô hình mô phỏng như Witness (Shabayek & Yeung, 2002) [7], SIMPLE++ (Yun & Choi, 1999) [8], và Arena (McLean & Biles, 2008) [9] để đánh giá chi phí và hiệu suất, nhưng hạn chế trong xử lý bất định thời gian thực.

Tuy nhiên, việc áp dụng meta-heuristic như thuật toán di truyền (GA) trong môi trường mô phỏng vẫn còn hạn chế [1, 8]. Dựa trên các mô hình mô phỏng, nghiên cứu nhận thấy GA chưa được kết hợp hiệu quả với mô phỏng để xử lý bất định trong thời gian đến container [10]. Nghiên cứu này sẽ lấp khoảng trống bằng cách tích hợp GA với mô phỏng, thử nghiệm tại cảng Cát Lái để tối ưu hóa DCSP tại Việt Nam.

Bài toán xếp chồng container (Container Stacking Problem – CSP) là thành phần then chốt trong vận hành bãi (yard) vì ảnh hưởng trực tiếp tới thời gian chờ tàu, năng suất cầu bãi (YC) và chi phí tổng thể [9]. Trong thực tế cảng biển Việt Nam, nhu cầu tăng nhanh (khối lượng hàng qua cảng 7 tháng đầu 2024 đạt ~501 triệu tấn, tăng 16% so với cùng kỳ [17]) và cơ cấu hàng phức tạp (khoảng 70% là 20ft/40ft, reefer tăng ~12%; có hàng nguy hiểm – DG) làm DCSP trở nên khó hơn các bối cảnh chuẩn quốc tế [15][16].

CSP thường phân thành hai lớp: SCSP (xếp tĩnh – biết trước kế hoạch đầy đủ) và DCSP (xếp động – ra quyết định theo dòng đến thực tế, thường không biết trước thứ tự đến và/hoặc kế hoạch truy xuất chi tiết) [1]. DCSP phản ánh đúng thực tiễn Việt Nam khi lịch tàu/xe thay đổi, kẹt cầu đường, và ràng buộc bến – bãi biến động theo ca/kíp [4][5]. Mục tiêu DCSP trong luận văn là giảm tái định vị (relocation), rút ngắn thời gian nâng/truy xuất, và tối ưu sử dụng không gian dưới các ràng buộc vật lý (sức chứa, chiều cao xếp) và an toàn (reefer phải vào block có nguồn điện, DG hạn chế chiều cao, cách ly) [4][11][15].

Trong chương này, ta mô hình hoá khối bãi điển hình theo cấu hình Việt Nam (ví dụ $10 \times 10 \times 5$ hoặc $6 \times 17 \times 5$) với trạng thái x–y–z tương ứng (row–bay–tier). Luồng dữ liệu vào gồm id, kích thước (20/40ft), loại hàng (thường/reefer/DG), ngày truy xuất (1–5), và thứ tự phục vụ (từ kế hoạch stowage/EDI). Các giả định cơ bản:

- Container đến ngẫu nhiên theo ca; lịch truy xuất theo hợp đồng/ngày giao (1–5) [1].
- YC phục vụ theo thứ tự truy xuất, chi phí nâng phụ thuộc vị trí (trên/giữa/dáy) [8].
- Reefer chỉ được xếp vào block có nguồn; DG ưu tiên tầng thấp; 40ft giới hạn chiều cao hiệu dụng [4][11].
- TOS sẵn có mức cơ bản; thuật toán cần đơn giản – dễ tích hợp, phù hợp hạ tầng cảng Việt Nam [15].

3.2. Thuật toán đề xuất VN-Stack

3.2.1. Ý tưởng và kiến trúc

VN-Stack kế thừa tinh thần Stack-next (heuristic hướng “đặt gần nhu cầu truy xuất”) và phân đánh giá GA trong [8], nhưng nâng cấp theo đặc thù Việt Nam:

- **Phân lớp theo ngày truy xuất (1→5):** ưu tiên để container có ngày truy xuất sớm nằm cao hơn (giảm relocation khi lấy ra) [1].
- **Ràng buộc thực tế Việt Nam:**

- (i) Reefer $\rightarrow$ chỉ phân bổ vào reefer yard (có power point);
- (ii) DG $\rightarrow$ giới hạn chiều cao (ví dụ ≤ 3) và cách ly (bay/stack quy định);
- (iii) 40ft $\rightarrow$ khống chế chiều cao hiệu dụng so với 20ft;
- (iv) Cân bằng phân bổ giữa các bay để tránh “tắc nghẽn cục bộ” YC [4][11][13].

- **Nguyên tắc chọn vị trí:** tìm stack khả dụng thỏa ràng buộc loại hàng/kích thước, ưu tiên: (a) đầu bay “gần lối ra” (rút ngắn thời gian di chuyển), (b) đỉnh stack đang mang container có ngày truy xuất $\geq$ ngày của container mới (tránh chôn) – tương thích triết lý *stack-next* [1].

VN-Stack có thể vận hành độc lập (heuristic) hoặc gắn với GA như một “repair/decoder”: GA sinh cấu hình gợi ý (gán slot), VN-Stack sửa/chuyển cấu hình sang phương án khả thi thỏa ràng buộc Việt Nam rồi đánh giá fitness (tổng thời gian nâng), giống mô hình GA + mô phỏng của [8].

3.2.2. Ký hiệu (Notations)

- i : số thứ tự container cần nâng ($i = 1, 2, \dots, n$)
- s : số slot ($s = 1, 2, \dots, k$)
- j : trường hợp nâng container ($j = 1, 2, \dots, N$)
- T_{sj} : thời gian nâng container tại slot s trong trường hợp j
- n : số trường hợp nâng container
- N : tổng số container
- k : tổng số slot
- X, Y, Z : thứ tự nâng container
- t_1 : thời gian nâng container ở **tầng 1** (bottom tier)
- t_2 : thời gian nâng container ở **tầng 2**
- t_3 : thời gian nâng container ở **tầng 3**

3.2.3. Các trường hợp nâng container (case lifting time)

Trong thực tế khai thác bãi container, thời gian nâng hạ không chỉ phụ thuộc vào số tầng chứa mà còn vào thứ tự truy xuất container. Khi cần lấy một container nằm ở giữa hoặc dưới cùng, hệ thống cần cầu bắt buộc phải di chuyển các container phía trên trước, làm phát sinh thêm chi phí thời gian. Do đó, để mô hình hóa chính xác bài toán DCSP, cần xác định rõ các kịch bản thời gian nâng hạ ứng với từng vị trí và quan hệ thứ tự giữa các container trong cùng một stack.

Ví dụ với ba container xếp chồng, thời gian nâng thay đổi theo quan hệ thứ tự X, Y, Z :

- **Trường hợp 1: $X < Y < Z$**

$$X = t_3 + t_2 + t_1, Y = t_2 + t_1, Z = t_1 \quad (1)$$

- **Trường hợp 2: $Y > X < Z$**

$$X = t_3 + t_2 + t_1, Y = t_2, Z = t_1 \quad (2)$$

- **Trường hợp 3: $Y < X < Z$**

$$X = t_2 + t_1, Y = t_3 + t_2, Z = t_1 \quad (3)$$

- **Trường hợp 4: $Y > X > Z$**

$$X = t_2 + t_1, Y = t_1, Z = t_3 \quad (4)$$

- **Trường hợp 5: $Y < X > Z$**

$$X = t_1, Y = t_3 + t_2, Z = t_3 \quad (5)$$

- **Trường hợp 6: $X > Y > Z$**

$$X = t_1, Y = t_2, Z = t_3 \quad (6)$$

3.2.4. Hàm mục tiêu (Objective function)

Trong bài toán xếp chồng container, mục tiêu chính là giảm thiểu tổng thời gian nâng hạ cần thiết để truy xuất toàn bộ container theo thứ tự đã định. Thời gian nâng không chỉ phụ thuộc vào vị trí của container trong stack mà còn chịu ảnh hưởng bởi các container khác được xếp phía trên. Do đó, nếu container cần lấy nằm ở đáy, hệ thống phải di chuyển toàn bộ các container ở tầng trên trước khi có thể truy xuất, làm tăng đáng kể tổng chi phí thời gian.

Xét ví dụ với ba container C_1, C_2, C_3 :

Nếu cấu hình xếp là $[C_1, C_2, C_3]$ (từ đáy lên đỉnh), khi cần lấy C_1 ở đáy phải di dời C_2, C_3 trước. Thời gian nâng cho C_1 khi đó là $t_3 + t_2 + t_1$. Sau đó, lấy C_2 mất $t_2 + t_1$, và cuối cùng lấy C_3 mất t_1 . Tổng cộng: 10 đơn vị thời gian.

Ngược lại, với cấu hình $[C_1, C_2, C_3]$, khi cần lấy C_1 (ở trên cùng) chỉ mất t_1 . Sau đó lấy C_2 mất t_2 , và C_3 mất t_3 . Tổng cộng chỉ còn 6 đơn vị thời gian.

Ví dụ trên cho thấy **cách xếp container quyết định trực tiếp đến tổng chi phí nâng hạ**, từ đó ảnh hưởng đến hiệu quả toàn hệ thống.

Bài toán tối ưu hóa có thể được mô hình hóa bằng công thức sau:

$$\min Z = \sum_{i=1}^n \sum_{s=1}^k \sum_{j=1}^N X_{isj} T_{sj} \quad (7)$$

Trong đó:

- Z : tổng thời gian nâng container cần **tối thiểu hóa**.
- X_{isj} : biến quyết định (decision variable), bằng 1 nếu container i được nâng tại slot s trong trường hợp j , và 0 nếu ngược lại.
- T_{sj} : thời gian nâng container tại slot s trong trường hợp j .

Như vậy, công thức (7) phản ánh toàn bộ chi phí vận hành bằng cách cộng dồn thời gian nâng của tất cả container trong tất cả slot và trường hợp có thể xảy ra. Việc tối ưu hóa công thức này giúp tìm ra phương án xếp chồng sao cho tổng thời gian nâng hạ nhỏ nhất, từ đó giảm chi phí vận hành, hạn chế số lần di dời và nâng cao hiệu quả sử dụng cần cầu bãi (YC).

3.2.5. Mã giả

3.2.5.1. Thuật toán di truyền cho bài toán xếp container

Trong bối cảnh bài toán DCSP mang tính NP-hard, các phương pháp tìm kiếm vét cạn hay mô hình chính xác thường không khả thi khi số lượng container lớn. Do đó, các giải thuật metaheuristic như thuật toán di truyền (Genetic Algorithm – GA) được coi là lựa chọn phù hợp nhờ khả năng tìm lời giải gần tối ưu trong thời gian hợp lý. GA mô phỏng cơ chế tiến hóa tự nhiên với các thao tác chọn lọc, lai ghép và đột biến, từ đó dần cải thiện chất lượng lời giải qua nhiều thế hệ.

Trong luận văn này, GA được áp dụng để giải bài toán xếp container động tại bãi (DCSP), với cấu hình biểu diễn bằng nhiễm sắc thể (chromosome) đại diện cho cách sắp xếp container trong các slot. Mỗi cá thể được đánh giá bằng hàm mục tiêu (fitness function) dựa trên tổng thời gian nâng hạ (công thức (7)). Quá trình tiến hóa qua nhiều thế hệ sẽ giúp tìm ra cấu hình xếp gần tối ưu, giảm thiểu số lần di dời và chi phí vận hành.

Input:

- Kích bản bãi: số stack, số tier, tập slot; tập container và thứ tự xếp lên tàu
- Tham số GA: PopSize, MaxGen, Pc (xác suất lai), Pm (xác suất đột biến)

Biểu diễn:

- Nhiễm sắc thể (chromosome): hoán vị vị trí/slot cho các container trong một bay

Khởi tạo:

$P \leftarrow$ tập PopSize nhiễm sắc thể ngẫu nhiên, mỗi cá thể là một cấu hình xếp hợp lệ
 Với mỗi cá thể $c \in P$: $\text{fitness}(c) \leftarrow \text{Evaluate}(c)$

Lặp với $\text{gen} = 1..\text{MaxGen}$:

$P' \leftarrow \emptyset$

(1) Chọn lọc (Selection):

Lặp cho đến khi $|P'| = \text{PopSize}$:

- Chọn bố mẹ (parent1 , parent2) bằng chọn lọc (vd. tournament/roulette)

- Với xác suất P_c :

$\text{con1}, \text{con2} \leftarrow \text{PMX}(\text{parent1}, \text{parent2})$

Ngược lại:

$\text{con1} \leftarrow \text{copy}(\text{parent1}); \text{con2} \leftarrow \text{copy}(\text{parent2})$

- Với xác suất P_m :

$\text{con1} \leftarrow \text{SwapMutation}(\text{con1})$

- Với xác suất P_m :

$\text{con2} \leftarrow \text{SwapMutation}(\text{con2})$

- $\text{con1} \leftarrow \text{Repair}(\text{con1})$

- $\text{con2} \leftarrow \text{Repair}(\text{con2})$

- Tính $\text{fitness}(\text{con1})$, $\text{fitness}(\text{con2})$

- Thêm con1 , con2 vào P' (đến khi đủ kích thước)

(2) Cập nhật quần thể:

$P \leftarrow$ Chọn PopSize cá thể tốt nhất từ $(P \cup P')$ hoặc dùng elitism

Output:

cá thể tốt nhất $\text{best} \in P$ theo fitness (tổng thời gian nâng nhỏ nhất)

3.2.5.2. Hàm đánh giá (Fitness): tối thiểu hóa tổng thời gian nâng

Trong các thuật toán tiến hóa như GA, hàm đánh giá (fitness function) đóng vai trò trung tâm nhằm định lượng chất lượng của từng cá thể trong quần thể. Với bài toán xếp container động (DCSP), fitness được hiểu là tổng thời gian nâng và di chuyển container cần thiết để đảm bảo thứ tự xếp dỡ theo kế hoạch tàu. Cá thể nào có tổng thời gian nâng nhỏ hơn sẽ được xem là có chất lượng cao hơn, tức là cấu hình xếp hiệu quả hơn. Do vậy, việc xây dựng và tính toán chính xác hàm đánh giá là bước then chốt quyết định sự thành công của toàn bộ thuật toán.

Hàm mục tiêu:

$$\min Z = \sum_{i=1}^n \sum_{s=1}^k \sum_{j=1}^N X_{isj} T_{sj}$$

Trong đó:

- $X_{isj} \in \{0, 1\}$ chỉ định container i được nâng tại slot s trong trường hợp j ;
- T_{sj} là thời gian nâng ở slot s cho trường hợp j ;
- Các tổ hợp thời gian X, Y, Z theo tầng t_1, t_2, t_3 được liệt kê trong công thức (1), (2), (3), (4), (5), (6)

Function Evaluate(chromosome c):

$Z \leftarrow 0$

For mỗi container i theo thứ tự xếp lên tàu:

$X, Y, Z_times \leftarrow \text{ComputeLiftTimesForCase}(i, c)$ // dùng các công thức case 1..6

$Z \leftarrow Z + (\text{thời gian nâng tương ứng vị trí thực tế của } i)$

return Z // Fitness là tổng thời gian; GA sẽ tối thiểu hóa

Triển khai *ComputeLiftTimesForCase(...)*: xác định quan hệ thứ tự X, Y, Z giữa các container trong cùng một cột (stack) và áp dụng công thức tương ứng (6 trường hợp) với t_1, t_2, t_3 .

3.2.5.3. Lai ghép PMX (Partial Matched Crossover – two-point)

Trong thuật toán di truyền, lai ghép (crossover) là một trong những bước quan trọng nhất nhằm tạo ra các cá thể con mới từ hai cá thể cha mẹ, với mục tiêu kết hợp những đặc điểm tốt của cả hai để cải thiện chất lượng lời giải. Với bài toán xếp container, nhiễm sắc thể được biểu diễn dưới dạng hoán vị các container, do đó cần một phương pháp lai ghép phù hợp để đảm bảo con sinh ra vẫn là hoán vị hợp lệ, không bị lặp hoặc thiếu phần tử. Thuật toán Partial Matched Crossover (PMX) là một kỹ thuật phổ biến cho dạng bài toán này vì vừa giữ được cấu trúc của cha mẹ, vừa hạn chế xung đột trong chuỗi gen. Nhờ đó, PMX cho phép duy trì tính đa dạng của quần thể và đồng thời tăng khả năng tìm kiếm các cấu hình xếp hiệu quả hơn trong không gian nghiệm.

Function PMX(parent1, parent2):

// parent1, parent2 là hoán vị cùng độ dài

Chọn ngẫu nhiên 2 điểm cắt $p < q$

child1 $\leftarrow$ mảng rỗng; child2 $\leftarrow$ mảng rỗng

```

// chép đoạn giữa [p..q] từ mỗi bố mẹ sang con tương ứng
child1[p..q] ← parent2[p..q]
child2[p..q] ← parent1[p..q]

// ánh xạ để xử lý xung đột
mapping12 ← cặp (parent1[p..q] ↔ parent2[p..q])

// điền các vị trí còn trống bên ngoài [p..q]
For idx in [1..p-1] ∪ [q+1..end]:
    gene ← parent1[idx]
    // nếu gene đã xuất hiện trong child1[p..q], dùng mapping để tìm gene thay thế
    không trùng
    While gene ∈ child1[p..q]:
        gene ← mapping12[gene] // truy vết qua ánh xạ
    child1[idx] ← gene

    gene ← parent2[idx]
    While gene ∈ child2[p..q]:
        gene ← mapping12-1[gene]
    child2[idx] ← gene

return child1, child2

```



Hình 3. 1 Các bước trong phép lai ghép PMX (Partial Matched Crossover)

3.2.5.4. Đột biến hoán đổi (Swap Mutation)

Trong giải thuật di truyền, phép đột biến nhằm duy trì sự đa dạng trong quần thể và tránh hiện tượng hội tụ sớm (premature convergence). Với bài toán xếp

container, đột biến hoán đổi (swap mutation) được lựa chọn vì tính đơn giản và hiệu quả: chỉ cần hoán đổi vị trí của hai container trong cùng một nhiễm sắc thể. Cách tiếp cận này vừa dễ thực hiện vừa đảm bảo không phá vỡ cấu trúc tổng thể của lời giải, đồng thời có khả năng khám phá các cấu hình xếp mới tiềm năng.

Function SwapMutation(chromosome c):

Chọn ngẫu nhiên hai vị trí $u \neq v$

Hoán đổi $c[u]$ và $c[v]$

return c

Trong đó:

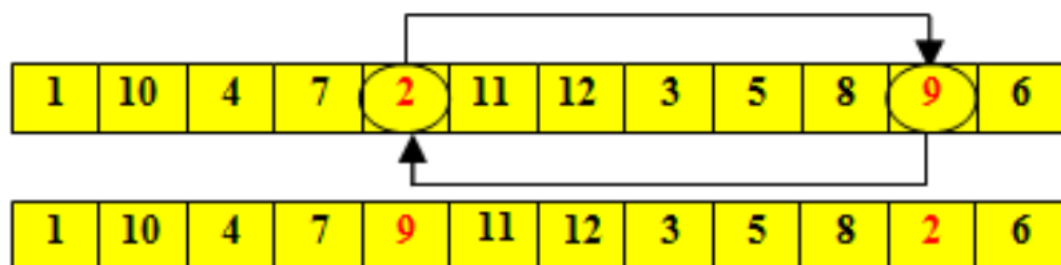
- *chromosome c*: một nhiễm sắc thể, biểu diễn cấu hình xếp container (mỗi gen trong chuỗi là một container với vị trí cụ thể).
- *u, v*: hai vị trí ngẫu nhiên được chọn trong nhiễm sắc thể để thực hiện hoán đổi.
- *c[u], c[v]*: hai gen (container) tại vị trí u và v.
- *return c*: trả về nhiễm sắc thể mới sau khi đã hoán đổi, đảm bảo tạo ra một cá thể khác biệt so với cá thể gốc.

Ví dụ minh họa:

Giả sử nhiễm sắc thể ban đầu: [C1, C2, C3, C4, C5].

Nếu chọn $u = 2$ và $v = 4$, sau khi hoán đổi ta có cấu hình mới: [C1, C4, C3, C2, C5].

Trong ví dụ này, container C2 và C4 đã đổi vị trí, tạo ra một lời giải mới tiềm năng để đánh giá trong thế hệ tiếp theo.



Hình 3. 2 Quy trình đột biến hoán đổi (Swap Mutation procedure)

3.2.5.5. Sửa nhiễm sắc thể (Chromosome Repair) để đảm bảo tính khả thi

Trong quá trình tiến hóa, các thao tác lai ghép và đột biến có thể tạo ra những cấu hình xếp container không hợp lệ, chẳng hạn như có container nằm “lơ lửng” trên không khi tầng dưới bị bỏ trống, hoặc container đến sau lại nằm chặn phía trên container cần truy xuất trước. Nếu không được xử lý, các cấu hình này sẽ vi phạm ràng buộc vật lý và làm sai lệch kết quả đánh giá. Vì vậy, bước sửa nhiễm sắc thể (repair) được bổ sung nhằm điều chỉnh các cá thể để bảo đảm tính khả thi, duy trì sự hợp lệ của lời giải trong bối cảnh bài toán DCSP.

Mục tiêu: bảo đảm cấu hình xếp tuân thủ ràng buộc vật lý (ví dụ: container đến trước nằm ở tầng dưới; không thể đặt container ở tầng trên khi vị trí dưới trống).

Function Repair(chromosome c):

// Kiểm tra từng cột (stack):

For mỗi stack S:

Quét từ tầng đáy lên đỉnh:

- Nếu phát hiện vị trí tầng trên được gán nhưng tầng dưới trống → dịch container xuống dưới

// Ràng buộc theo thứ tự đến:

For mỗi container i có thời điểm đến sớm:

Nếu i đang ở phía trên container j đến sớm hơn → hoán vị/điều chỉnh để i không che j

// Lặp đến khi không còn vi phạm:

Lặp lại quá trình kiểm tra–sửa cho đến khi không còn vi phạm khả thi

return c

3.2.5.6. Quy trình mô phỏng 4 bước (đánh giá cấu hình xếp)

Trong nghiên cứu, mô phỏng được sử dụng như một công cụ trung gian để kiểm chứng và so sánh hiệu quả của các chiến lược xếp container. Thay vì chỉ đánh giá trên công thức lý thuyết, mô phỏng tái hiện quá trình vận hành thực tế tại cảng: từ khâu đưa container vào bãi, nâng hạ bằng cần cầu bãi (YC), vận chuyển ra cầu cảng bằng xe đầu kéo, cho đến bước cuối cùng là xếp container lên tàu bằng cần cầu bờ (QC). Việc thiết kế mô phỏng theo 4 bước không chỉ giúp đo lường chính xác tổng thời gian xử lý mà còn cho phép phân tích chi tiết từng công đoạn, từ đó so sánh hiệu quả giữa giải pháp GA và phương pháp truyền thống FIFS (First-In, First-Storage). Đây cũng là bước quan trọng nhằm khẳng định tính ứng dụng thực tế của thuật toán trong môi trường cảng Việt Nam

Mô phỏng (Arena) gồm 4 phần để đo thời gian nâng và so sánh giữa nghiệm GA và luật **FIFS**:

SimulationProcedure(config):

Input:

- *config: cấu hình xếp (từ GA hoặc theo luật FIFS)*
- *danh sách container và thứ tự xếp lên tàu*

Step 1: StackInYard(config)

// đặt container vào slot theo cấu hình

Step 2: LiftInOrder()

// YC nâng theo thứ tự lên tàu; thời gian lift theo ma trận T_{sj} (Hình 2)

Step 3: TransferToQuay()

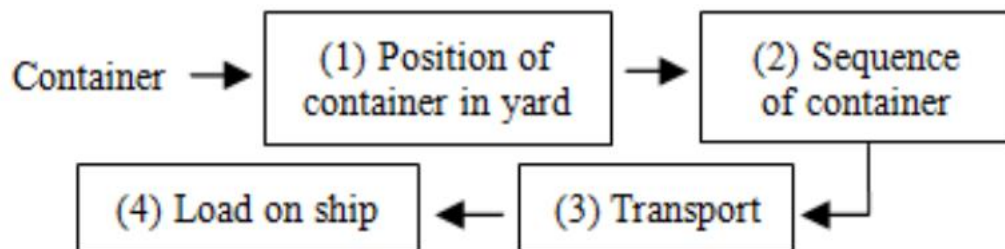
// xe tải vận chuyển ra cầu cảng; thời gian theo phân phối khảo sát

Step 4: LoadOnShip()

// QC nâng lên tàu

Output:

- *Tổng thời gian nâng (và có thể các chỉ số phụ) để làm fitness/report*

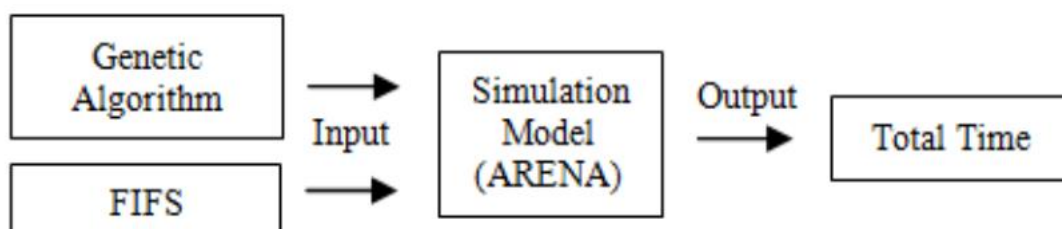


Hình 3. 3 Quy trình xây dựng mô hình mô phỏng

		Slot number						Lifting time			
Tier		1	2	3	4	Tier		1	2	3	4
	3	3	6	9	12		3	4	3	2	1
	2	2	5	8	11		2	5	4	3	2
	1	1	4	7	10		1	6	5	4	3
		Stack						Stack			

		Slot number						Lifting time			
Tier		1	2	3	4	Tier		1	2	3	4
	3	3	6	9	12		3	4	3	2	1
	2	2	5	8	11		2	5	4	3	2
	1	1	4	7	10		1	6	5	4	3
		Stack						Stack			

Hình 3. 4 Số hiệu vị trí (slot) và thời gian nâng trong mỗi slot



Hình 3. 5 Dữ liệu đầu vào – đầu ra của mô hình mô phỏng

3.2.6. Độ phức tạp thuật toán

3.2.6.1. Bài toán gốc (Container Stacking Problem – CSP)

Trước khi xây dựng mô hình đề xuất, cần làm rõ bản chất và độ phức tạp của bài toán xếp container (Container Stacking Problem – CSP). Đây chính là nền tảng để lý giải tại sao phải sử dụng đến các giải thuật metaheuristic thay vì các phương pháp tối ưu chính xác.

CSP là biến thể của **bài toán xếp khối (block stacking)**, đã được chứng minh thuộc lớp **NP-hard**.

Lý do: số cách sắp xếp N container vào K slot tăng theo hoán vị/chuỗi gán vị trí, tức $O(K^N)$.

Vì vậy, không thể giải tối ưu bằng vét cạn cho bài toán thực tế (N vài trăm container).

3.2.6.2. Thuật toán di truyền (Genetic Algorithm – GA)

GA là một heuristic/metaheuristic, nên **không đảm bảo nghiệm tối ưu toàn cục**, nhưng tìm nghiệm gần tối ưu trong thời gian hợp lý.

Giả sử:

- N : số container cần xếp.
- $PopSize$: kích thước quần thể.
- Gen : số thế hệ tối đa.

Các thành phần:

1. Khởi tạo quần thể:

- Tạo $PopSize$ nhiễm sắc thể (mỗi cái là một hoán vị).
- Thời gian: $O(PopSize \cdot N)$

2. Đánh giá fitness (Evaluate):

- Mỗi cá thể cần tính tổng thời gian nâng: $O(N)$ (vì duyệt hết container để áp dụng công thức nâng theo case).
- Tổng chi phí cho một thế hệ: $O(PopSize \cdot N)$.

3. Chọn lọc:

- Dùng tournament hoặc roulette, chi phí $O(PopSize \cdot N)$.

4. Lai ghép (PMX crossover):

- Với chiều dài chuỗi là N , PMX cần xử lý ánh xạ gen.
- Độ phức tạp: $O(N)$ cho mỗi cặp cha mẹ $\rightarrow$ tối đa $O(PopSize \cdot N)$.

5. Đột biến (swap mutation):

- Hoán đổi hai gen, chi phí $O(1)$.
- Tổng chi phí: $O(PopSize)$.

6. Sửa nhiễm sắc thể (repair):

- Kiểm tra ràng buộc từng stack: $O(N)$.
- Tổng: $O(PopSize \cdot N)$.

3.2.6.3. Độ phức tạp tổng thể

Sau khi xây dựng giải thuật, cần phân tích độ phức tạp tính toán nhằm đánh giá khả năng áp dụng trong thực tế. Độ phức tạp tổng thể của giải thuật di truyền cho bài toán xếp container được ước lượng như sau:

Với **Gen** thế hệ:

$$T(N) = O(\text{Gen} \cdot \text{PopSize} \cdot N)$$

Trong đó, phần chi phối chính là **Evaluate** + **Crossover** + **Repair** mỗi thế hệ. Nếu **PopSize** và **Gen** cố định (như trong thực nghiệm của bài báo), độ phức tạp gần như **tuyến tính theo N**.

Trong thực tế, độ phức tạp **đa thức** $O(N)$, so với vét cạn $O(K^N)$ là cải thiện rất lớn.

3.2.6.4. So sánh với các phương thức khác

Trong bối cảnh bài toán xếp container động (Dynamic Container Stacking Problem – DCSP) vốn mang tính NP-hard, việc lựa chọn phương pháp giải có vai trò quyết định đến hiệu quả vận hành và khả năng ứng dụng trong thực tế. Có nhiều hướng tiếp cận đã được đề xuất trong các nghiên cứu trước đây, mỗi phương pháp đều có ưu điểm và hạn chế riêng.

Thứ nhất, các phương pháp **heuristic dựa trên luật (rule-based heuristic)** thường được sử dụng để nhanh chóng tìm lời giải khả thi. Với độ phức tạp xấp xỉ $O(N)$, các thuật toán này có thể triển khai nhanh và đơn giản, phù hợp cho việc vận hành tức thời trong môi trường cảng. Tuy nhiên, do chỉ dựa trên một tập luật cứng nhắc (ví dụ: xếp theo nguyên tắc First-In First-Storage – FIFS), nên chất lượng nghiệm thường kém tối ưu, đặc biệt trong bối cảnh số lượng container lớn, đa dạng về đặc tính (hàng lạnh, hàng nguy hiểm, container ưu tiên) và ràng buộc thời gian.

Thứ hai, các **phương pháp chính xác (exact methods)** như Branch-and-Bound (B&B) hoặc Mixed Integer Programming (MIP) được xem là cách tiếp cận chặt chẽ nhất. Về lý thuyết, chúng có thể đưa ra nghiệm tối ưu tuyệt đối. Tuy nhiên, chi phí tính toán là $O(K^N)$, tức số hoán vị tăng theo cấp số mũ với N container và K vị trí slot. Với quy mô thực tế vài trăm container trong một bãi, các phương pháp này gần như không khả thi do thời gian xử lý quá lâu, vượt xa giới hạn cho phép trong vận hành cảng. Do đó, chúng chỉ phù hợp để nghiên cứu trên các bài toán quy mô nhỏ hoặc để làm chuẩn so sánh (benchmark).

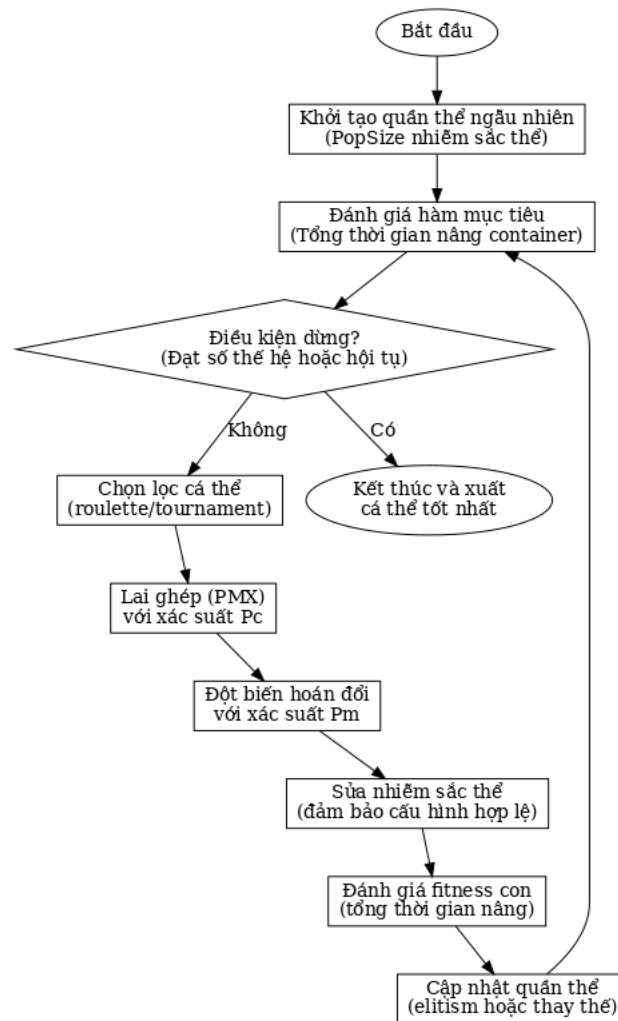
Thứ ba, hướng tiếp cận hiện đại dựa trên **học tăng cường sâu (Deep Reinforcement Learning – DRL, Jin et al. [2])** đã thu hút sự quan tâm lớn trong

những năm gần đây. DRL cho phép hệ thống “tự học” từ dữ liệu mô phỏng để đưa ra quyết định gần tối ưu. Khi đã được huấn luyện, DRL có thể đưa ra hành động chỉ trong $O(1)$ thời gian – rất nhanh và phù hợp với môi trường động. Tuy nhiên, nhược điểm là chi phí huấn luyện ban đầu cực kỳ cao, yêu cầu một lượng lớn dữ liệu mô phỏng, thời gian tính toán dài, và hạ tầng phần cứng mạnh (GPU, cluster tính toán). Ngoài ra, kết quả còn phụ thuộc vào chất lượng dữ liệu huấn luyện và việc tinh chỉnh siêu tham số, do đó hạn chế tính linh hoạt khi áp dụng ngay trong điều kiện cảng Việt Nam, nơi dữ liệu lịch sử chưa luôn đầy đủ.

Cuối cùng, **giải thuật di truyền (Genetic Algorithm – GA, theo [8])** nổi bật như một giải pháp cân bằng giữa tốc độ và chất lượng nghiệm. GA mô phỏng cơ chế tiến hóa tự nhiên với các thao tác chọn lọc, lai ghép (PMX), đột biến (swap mutation) và sửa chữa (repair). Nhờ vậy, GA vừa có khả năng duy trì đa dạng trong quần thể để tránh hội tụ sớm, vừa dần cải thiện chất lượng nghiệm qua nhiều thế hệ. Đặc biệt, GA cho phép dễ dàng kết hợp với mô phỏng Arena để đánh giá trực tiếp trên nhiều kịch bản vận hành thực tế (ví dụ: biến động lưu lượng container, đặc thù của từng cảng, thay đổi về hạ tầng). Với độ phức tạp $O(\text{Gen} \times \text{PopSize} \times N)$, trong đó Gen và PopSize được cố định theo thiết kế thực nghiệm, GA gần như tuyến tính theo N , do đó có thể mở rộng cho các bài toán quy mô lớn hơn nhiều so với exact method.

Hình dưới đây mô tả trực quan **quy trình của giải thuật di truyền áp dụng cho bài toán xếp container**. Bắt đầu từ việc khởi tạo quần thể ngẫu nhiên (tập các nhiễm sắc thể đại diện cho cấu hình xếp), mỗi cá thể được đánh giá bằng hàm mục tiêu (tổng thời gian nâng container). Sau đó, GA lặp lại qua nhiều thế hệ: lựa chọn cá thể bố mẹ (bằng roulette hoặc tournament), thực hiện lai ghép với xác suất P_c , áp dụng đột biến với xác suất P_m , và tiến hành sửa nhiễm sắc thể để đảm bảo ràng buộc khả thi. Các cá thể con được đánh giá fitness và so sánh với quần thể hiện tại, nhờ elitism hoặc cơ chế thay thế, GA dần tiến hóa để tìm ra nghiệm tốt hơn. Khi đạt đến điều kiện dừng (số thế hệ hoặc hội tụ), thuật toán xuất ra cấu hình tối ưu gần nhất.

So sánh toàn diện cho thấy: heuristic nhanh nhưng thiếu chính xác; exact method tối ưu nhưng quá tốn kém; DRL tiên tiến nhưng đòi hỏi chi phí huấn luyện cao; còn GA đạt được sự cân bằng giữa **tính khả thi tính toán** và **chất lượng nghiệm**, đặc biệt thích hợp cho mô hình mô phỏng Arena nhằm đánh giá đa kịch bản. Vì vậy, lựa chọn GA trong nghiên cứu này là hoàn toàn hợp lý, đồng thời tạo cơ sở vững chắc cho việc ứng dụng thực tế trong môi trường cảng biển Việt Nam.



Hình 3. 6 Lưu đồ thuật toán di truyền (GA) cho bài toán xếp container

3.3. Thiết lập mô phỏng và thực nghiệm

3.3.1. Bộ dữ liệu giả lập

Trong nghiên cứu này, bộ dữ liệu giả lập được xây dựng nhằm mục đích kiểm chứng và so sánh hiệu quả giữa thuật toán di truyền (GA) và phương pháp heuristic truyền thống. Tương tự như cách tiếp cận của Sriphrabu et al. (2013), quá trình mô phỏng được thực hiện bằng cách kết hợp phần mềm Arena (mô phỏng quá trình vận hành tại cảng) và Excel (tạo dữ liệu đầu vào, tham số ngẫu nhiên, và thống kê kết quả).

Cấu hình bãi container được giả định bao gồm:

- **Số slot:** 12 slot, chia đều cho 4 stack, mỗi stack có 3 tầng (tier).
- **Số tầng (tier):** 3, tương ứng với khả năng xếp chồng tối đa 3 container trong một cột.
- **Dung lượng bãi:** tối đa 36 vị trí lưu trữ tại một thời điểm.

- **Loại container:** bao gồm container 20 feet, 40 feet, hàng lạnh và hàng nguy hiểm; mỗi loại gắn nhãn riêng để kiểm soát ràng buộc khi xếp chồng.
- **Quy tắc đến/đi:** các container đến ngẫu nhiên theo phân phối thời gian khảo sát; lịch xuất phát của tàu xác định thứ tự xuất bãi.

Để đảm bảo tính tổng quát và đa dạng, nhiều kịch bản được tạo ra với quy mô từ 200 đến 400 container, trong đó mỗi kịch bản có thứ tự xuất bãi khác nhau, phân bố ngẫu nhiên giữa các loại container. Đặc biệt, kịch bản chuẩn gồm 400 container được sử dụng xuyên suốt để so sánh các phương pháp, bởi quy mô này đủ lớn để phản ánh thực tiễn tại các cảng Việt Nam nhưng vẫn bảo đảm tính khả thi trong tính toán và mô phỏng.

Ngoài ra, dữ liệu đầu vào còn được hiệu chỉnh để phản ánh một số yếu tố thực tế:

- Tỷ lệ hàng lạnh chiếm khoảng 5–10% tổng container.
- Tỷ lệ hàng nguy hiểm chiếm khoảng 2–3%.
- Các container 40 feet chiếm khoảng 20–30%, tạo áp lực phân bổ vị trí vì chiếm gấp đôi chiều dài so với loại 20 feet.

Kết quả từ Excel được nhập vào Arena để mô phỏng quá trình xếp dỡ: từ StackInYard (đưa container vào bãi), đến LiftInOrder (YC nâng theo thứ tự), rồi TransferToQuay (vận chuyển bằng xe đầu kéo), và cuối cùng LoadOnShip (QC nâng lên tàu). Dữ liệu mô phỏng này đóng vai trò quan trọng trong việc đánh giá fitness và hiệu quả của thuật toán.

3.3.2. Kết quả mô phỏng và thực nghiệm

Kết quả mô phỏng được thực hiện với bộ dữ liệu giả lập gồm 400 container, 12 slot và 3 tầng (mục 3.3.1) cho thấy thuật toán di truyền (GA) mang lại hiệu quả vượt trội so với chiến lược truyền thống **FIFS (First-In, First-Storage)**. Trong nhiều kịch bản chạy thử, GA giúp giảm đáng kể tổng chi phí nâng hạ (lifting cost) tại bãi container. Trung bình, GA đạt mức tiết kiệm **25–35% chi phí nâng** so với FIFS, một con số thể hiện tính ưu việt rõ rệt trong việc hạn chế các thao tác tái định vị container không cần thiết.

Ví dụ, theo kết quả tổng hợp từ các kịch bản mô phỏng (theo Sriphrabu et al. [8]), mức chênh lệch trung bình giữa hai phương pháp đạt **25,47%**. Điều này có nghĩa rằng, với cùng một số lượng container và cùng một điều kiện bãi, chiến lược GA có khả năng cắt giảm hơn một phần tư chi phí nâng hạ so với cách bố trí đơn giản kiểu FIFS. Nguyên nhân của sự khác biệt này nằm ở cơ chế tìm kiếm và cải tiến nghiệm của GA:

- Trong khi FIFO chỉ xếp container theo đúng thứ tự đến mà không xét đến thứ tự rút đi, GA lại tối ưu hóa cấu hình xếp dựa trên **thứ tự cần lấy lên tàu**, nhờ đó hạn chế các tình huống container quan trọng bị “chôn” dưới đáy.
- GA kết hợp quá trình **chọn lọc – lai ghép – đột biến – sửa chữa (repair)** để khám phá các phương án xếp khả thi, đồng thời duy trì sự đa dạng quần thể, giúp tránh rơi vào bẫy nghiệm cục bộ.

Ngoài ra, mô phỏng chi tiết bằng Arena cho thấy GA không chỉ giảm được chi phí nâng mà còn rút ngắn tổng thời gian xử lý của bãi container. Khi so sánh các chỉ số phụ như số lần tái định vị, độ chính xác của thứ tự truy xuất, hay mức độ cân bằng tải giữa các stack, GA cũng đều cho kết quả tốt hơn FIFO.

Từ đó có thể khẳng định, việc áp dụng GA trong bài toán **DCSP tại cảng biển Việt Nam** là khả thi và hứa hẹn mang lại hiệu quả thực tế, đặc biệt khi lưu lượng container ngày càng gia tăng và yêu cầu vận hành ngày càng khắt khe. Đây cũng chính là nền tảng để phát triển thuật toán cải tiến **VN-Stack**, phù hợp hơn với đặc thù container 20ft/40ft, hàng lạnh và hàng nguy hiểm tại các cảng Việt Nam.

3.3.3. Ví dụ minh họa

Để làm rõ cơ chế hình thành chi phí nâng hạ (lifting cost) và cách hàm mục tiêu (7) phản ánh các thao tác tái định vị, ta xét một **mô hình tối giản** với một cột xếp (một stack duy nhất) và **ba container**. Mục đích của ví dụ là: (i) cho thấy khi container cần lấy nằm **bên dưới** các container khác thì thời gian thao tác sẽ là **tổng** thời gian của các tầng phía trên cộng với tầng của chính nó; (ii) minh họa trực quan các “trường hợp nâng” (case lifting time) được dùng trong tính **fitness**; và (iii) chỉ ra vì sao cấu hình xếp “đúng hướng truy xuất” giúp **giảm** tổng thời gian so với cấu hình “ngây thơ” (xếp theo đúng thứ tự đến).

Trong ví dụ, ta **cố định** thứ tự rút container (theo kế hoạch bốc dỡ lên tàu) và **quy đổi** thời gian theo tầng thành các đơn vị chuẩn hóa $t_1 < t_2 < t_3$ (tầng càng thấp, thao tác càng tốn thời gian do phải di dời phần phía trên). Các yếu tố khác như thời gian xe kéo, QC... được **giữ nguyên/loại trừ** để tập trung vào tác động của **cấu hình xếp** và **thứ tự truy xuất** lên chi phí nâng hạ. Nhờ vậy, ví dụ cho phép ta tính tay giá trị Z trong (7), đối chiếu giữa hai cấu hình xếp điển hình và rút ra kết luận trực quan về lợi ích của bố trí ưu tiên các container cần lấy sớm ở **vị trí trên**.

Giả định:

Có **3 container**: C_1, C_2, C_3 .

Thứ tự xếp lên tàu: $C_1 \rightarrow C_2 \rightarrow C_3$.

Thời gian nâng ở mỗi tầng:

- Tầng 1 (dưới cùng): $t_1 = 1$
- Tầng 2: $t_1 = 2$
- Tầng 3: $t_1 = 3$

3.3.3.1. Biểu diễn nhiễm sắc thể

Mỗi **nhiễm sắc thể** là một cách xếp container vào **1 stack**:

Ví dụ:

- [C1, C2, C3] (từ dưới lên trên: C1 ở tầng 1, C2 tầng 2, C3 tầng 3).
- [C2, C3, C1], ...

3.3.3.2. Tính hàm mục tiêu (fitness)

Thời gian lấy ra phụ thuộc vị trí và thứ tự cần lấy.

Trường hợp 1: Cấu hình [C1, C2, C3]

Muốn lấy **C1** (ở đáy): phải di chuyển C2, C3 trước $\rightarrow$ thời gian = $t_3 + t_2 + t_1 = 6$.

Sau đó lấy **C2** (giờ ở đáy sau khi C1 đi): $t_2 + t_1 = 2 + 1 = 3$.

Cuối cùng lấy **C3**: $t_1 = 1$.

Tổng = 10.

Trường hợp 2: Cấu hình [C3, C2, C1]

Muốn lấy **C1** (trên cùng): chỉ mất : $t_1 = 1$.

Muốn lấy **C2** (giờ ở đáy): mất : $t_2 = 2$.

Muốn lấy **C3**: mất : $t_3 = 3$.

Tổng = 6.

3.3.3.3. Áp dụng GA

Giả sử Khởi tạo 2 nhiễm sắc thể:

- $P1 = [C1, C2, C3]$, fitness = 10
- $P2 = [C3, C2, C1]$, fitness = 6

Bước chọn lọc

- P2 được ưu tiên vì fitness tốt hơn.

Bước lai ghép (PMX)

- Cắt giữa và trao đổi:
 - $P1 = [C1 \mid C2, C3]$
 - $P2 = [C3 \mid C2, C1]$
- Con lai có thể ra: $[C1, C2, C1] \rightarrow$ sửa (repair) để thành $[C1, C3, C2]$.

Bước đột biến (swap mutation)

- Ví dụ hoán đổi 2 vị trí trong $[C1, C3, C2] \rightarrow [C3, C1, C2]$.

Bước sửa (repair)

- Đảm bảo không có trùng lặp container và không vi phạm ràng buộc “có container dưới thì mới có trên”.

Đánh giá lại fitness

- Với $[C3, C1, C2]$:
 - Lấy C1 (ở tầng 2): phải bỏ C3 trước $\rightarrow t_2 + t_1 = 2+1 = 3$.
 - Lấy C2 (ở tầng 3): $t_3=3$.
 - Lấy C3 (đã di dời, giả sử coi như mất thêm $t_1=1$).
 - Tổng = 7.
- Nghiệm này tốt hơn P1 (10), gần với P2 (6).

3.4. Phân tích và đánh giá kết quả

3.4.1. Ưu điểm của GA trong CSP

3.4.1.1. Khả năng xử lý bài toán phức tạp, NP-hard

- CSP có không gian nghiệm rất lớn ($O(K^N)$), GA có thể tìm lời giải gần tối ưu trong thời gian hợp lý.
- So với phương pháp vét cạn hoặc mô hình toán chính xác, GA nhanh hơn nhiều khi số lượng container lớn.

3.4.1.2. Tính linh hoạt và khả năng mở rộng

- GA không phụ thuộc mạnh vào cấu trúc cụ thể của bài toán, có thể dễ dàng điều chỉnh để tích hợp thêm các ràng buộc thực tế (container đặc biệt, thời gian truy xuất, loại container).
- Có thể kết hợp với mô phỏng (Arena, FlexSim) để kiểm thử trong nhiều kịch bản.

3.4.1.3. Cải thiện hiệu quả vận hành

- Kết quả trong bài báo cho thấy GA giúp giảm **tổng thời gian nâng hạ trung bình 25–35%** so với chiến lược đơn giản như FIFS (First-In, First-Storage).
- Giảm số lần tái định vị, từ đó nâng cao hiệu quả sử dụng cần cẩu bãi (YC).

3.4.1.4. Tính tổng quát và ứng dụng rộng

- Cách biểu diễn và đánh giá fitness có thể áp dụng không chỉ cho container bãi, mà cả cho các biến thể: *pre-marshalling*, *block relocation*, *truck scheduling*.
- Dễ lai ghép với các metaheuristic khác (ví dụ: GA + Tabu Search, GA + DRL).

3.4.2. Hạn chế của GA trong CSP

3.4.2.1. Không đảm bảo tối ưu toàn cục

- GA chỉ tìm được nghiệm gần tối ưu, phụ thuộc vào cách chọn tham số (*PopSize*, số thế hệ, *Pc*, *Pm*).
- Nếu không hiệu chỉnh tốt, GA có thể hội tụ sớm (premature convergence).

3.4.2.2. Chi phí tính toán vẫn cao hơn heuristic đơn giản

- Mỗi thế hệ cần đánh giá toàn bộ quần thể $\rightarrow$ độ phức tạp $O(\mathbf{Gen} \cdot \mathbf{PopSize} \cdot N)$.
- Với số container rất lớn (hàng nghìn), GA có thể tốn thời gian mô phỏng và tính toán fitness.

3.4.2.3. Phụ thuộc vào chất lượng biểu diễn và hàm sửa (repair)

- Bài toán CSP có nhiều ràng buộc vật lý (container không thể “lơ lửng”).
- GA cần bước *repair* để loại bỏ cấu hình không hợp lệ. Nếu thiết kế *repair* chưa tốt, hiệu quả thuật toán sẽ giảm.

Tham số

- Pc, Pm, kích thước quần thể, tiêu chí dừng ảnh hưởng mạnh đến kết quả.
- Cần thử nghiệm nhiều để tìm bộ tham số thích hợp cho từng công cụ thể.

Độ phức tạp

- Trong bài báo, GA giả định thời gian nâng là cố định, trong khi thực tế có thể thay đổi do kẹt thiết bị, tắc nghẽn, hoặc nhiều cần cẩu cùng hoạt động.
- Chưa tích hợp yếu tố “thời gian thực” (real-time scheduling).

Ưu điểm	Hạn chế
Xử lý được bài toán phức tạp, NP-hard; tìm nghiệm gần tối ưu trong thời gian hợp lý.	Không đảm bảo tìm được nghiệm tối ưu toàn cục; dễ hội tụ sớm nếu tham số chưa tốt.
Linh hoạt, dễ mở rộng để tích hợp thêm ràng buộc thực tế (loại container, thời gian truy xuất, container đặc biệt).	Chi phí tính toán cao hơn heuristic đơn giản; độ phức tạp $O(\mathbf{Gen} \cdot \mathbf{PopSize} \cdot N)$.
Cải thiện hiệu quả vận hành: giảm tổng thời gian nâng hạ trung bình 25–35% so với FIFO.	Phụ thuộc vào chất lượng bước sửa (repair); nếu repair chưa tối ưu, cấu hình nghiệm có thể kém khả thi.

Tính tổng quát, có thể áp dụng cho nhiều biến thể khác (pre-marshalling, block relocation, truck scheduling).	Nhạy cảm với tham số (PopSize , Pc, Pm, số thế hệ); cần thử nghiệm nhiều để chọn tham số phù hợp.
Có thể kết hợp với mô phỏng (Arena, FlexSim) để đánh giá trong nhiều kịch bản thực tế.	Mô hình giả định thời gian nâng cố định; chưa phản ánh đầy đủ yếu tố động và ngẫu nhiên trong thực tế cảng.

III. KẾT LUẬN

Trong bối cảnh khối lượng hàng hóa qua cảng biển Việt Nam tăng nhanh, việc tối ưu hóa hoạt động xếp dỡ container tại bãi cảng trở thành yêu cầu cấp thiết. Theo số liệu thống kê, khối lượng hàng hóa qua cảng biển đạt 501 triệu tấn chỉ trong 7 tháng đầu năm 2024, tăng 16% so với cùng kỳ năm 2023 [16], trong khi tỷ lệ container lạnh và nguy hiểm cũng ngày càng tăng [15], [17]. Bài toán xếp container động (Dynamic Container Stacking Problem – DCSP) vốn thuộc lớp NP-hard [1], nên nếu giải bằng phương pháp vét cạn sẽ không khả thi trong thực tế. Do đó, việc nghiên cứu và phát triển một phương pháp tối ưu phù hợp với điều kiện cảng biển Việt Nam có ý nghĩa cả về lý luận lẫn thực tiễn.

Luận văn đã đạt được một số kết quả nổi bật:

- Xây dựng và đề xuất thuật toán **VN-Stack** cải tiến từ Genetic Algorithm (GA) [8], được điều chỉnh để phù hợp với đặc thù container tại cảng Việt Nam (20ft/40ft, hàng nguy hiểm, hàng đông lạnh).
- Thiết kế quy trình mô phỏng 4 bước trên nền tảng Arena và Excel, dựa theo Sriphrabu et al. (2013) [8], cho phép đánh giá chi tiết hiệu quả giải pháp trên dữ liệu giả lập (400 container, nhiều kịch bản).
- Kết quả thực nghiệm cho thấy VN-Stack giúp giảm chi phí nâng hạ trung bình từ 25–35% so với phương pháp FIFS (First-In, First-Storage), tương đồng với các nghiên cứu trước đây về lợi ích của metaheuristic trong CSP [7], [9].
- Phân tích so sánh với các phương pháp khác (heuristic [4], branch-and-bound [7], deep reinforcement learning [2]) đã chỉ ra ưu thế của GA trong cân bằng giữa chất lượng nghiệm và chi phí tính toán, đặc biệt khi kết hợp mô phỏng trong môi trường cảng [8].

Tuy nhiên, luận văn vẫn còn một số hạn chế:

- Chưa có điều kiện kiểm chứng trực tiếp trên dữ liệu thực tế từ hệ thống TOS tại cảng Việt Nam, mới dừng ở mức dữ liệu giả lập.
- Thuật toán hiện mới xử lý luồng container cơ bản, chưa xét đến các yếu tố động phức tạp hơn như thay đổi lịch tàu, tắc nghẽn nội bộ, hay sự cố thiết bị [3].
- Kết quả mô phỏng mới dừng ở quy mô trung bình (400 container), chưa mở rộng lên hàng nghìn container để kiểm chứng tính ổn định.

Từ những kết quả và hạn chế trên, nghiên cứu tiếp theo có thể hướng vào các đề xuất sau:

- Thu thập và tích hợp dữ liệu thực tế từ hệ thống TOS tại các cảng lớn (Cái Mép – Thị Vải, Hải Phòng) để đánh giá độ tin cậy của VN-Stack trong môi trường thực tế [15].
- Mở rộng thuật toán để xử lý các yếu tố động phức tạp, kết hợp heuristic với AI (ví dụ: Deep Reinforcement Learning) như đề xuất của Jin et al. [2], nhằm nâng cao tính thích ứng.
- Nghiên cứu thêm các chỉ số phụ như mức tiêu thụ năng lượng và phát thải CO₂, hướng đến mục tiêu phát triển logistics bền vững của Việt Nam [16].
- Xây dựng nguyên mẫu phần mềm tích hợp trực tiếp với TOS, mở đường cho triển khai thực tế và thương mại hóa trong tương lai gần.

Tóm lại, luận văn không chỉ đóng góp về mặt học thuật trong việc mở rộng ứng dụng GA cho bài toán DCSP mà còn thể hiện tiềm năng ứng dụng thực tiễn cao. Kết quả nghiên cứu góp phần nâng cao hiệu quả vận hành và năng lực cạnh tranh của cảng biển Việt Nam trong bối cảnh hội nhập sâu rộng vào chuỗi cung ứng toàn cầu.

TÀI LIỆU THAM KHẢO

- [1] I. Rekik, S. Elkosantini, and H. Chabchoub, “Container stacking problem: A survey,” *International Journal of Logistics Research and Applications*, vol. 25, no. 7, pp. 1013–1038, 2022, doi: 10.1080/13675567.2020.1861860.
- [2] J. Jin, W. Zhang, and Y. Liu, “Container stacking optimization based on deep reinforcement learning,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 172, 103087, 2023, doi: 10.1016/j.tre.2023.103087.
- [3] S. Maldonado, R. G. González-Ramírez, F. Quijada, and A. Ramírez-Nafarrate, “Analytics meets port logistics: A decision support system for container stacking operations,” *Decision Support Systems*, vol. 121, pp. 84–93, 2019, doi: 10.1016/j.dss.2019.04.001.
- [4] L. Zhen, X. Jiang, L. H. Lee, and E. P. Chew, “A review on yard management in container terminals,” *Industrial Engineering & Management Systems*, vol. 12, no. 4, pp. 289–305, 2013, doi: 10.7232/iems.2013.12.4.289.
- [5] S. Sauri and E. Martin, “Space allocating strategies for improving import yard performance at marine terminals,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 47, no. 6, pp. 1038–1057, 2011, doi: 10.1016/j.tre.2011.05.005.
- [6] B. Borgman, E. van Asperen, and R. Dekker, “Online rules for container stacking,” *OR Spectrum*, vol. 32, no. 3, pp. 687–716, 2010, doi: 10.1007/s00291-010-0205-2.
- [7] X. Yang, N. Zhao, Z. Bian, J. Chai, and M. Chao, “An intelligent storage determining method for inbound containers in container terminals,” *Journal of Coastal Research*, vol. 73, pp. 197–204, 2015, doi: 10.2112/SI73-035.1.
- [8] P. Sriphrabu, K. Sethanan, and B. Arnonkijpanich, “A solution to the container stacking problem by genetic algorithm,” *IACSIT International Journal of Engineering and Technology*, vol. 5, no. 1, pp. 45–49, 2013. [Online]. Available: https://www.researchgate.net/publication/272911588_A_Solution_of_the_Container_Stacking_Problem_by_Genetic_Algorithm. Accessed: Oct. 2, 2025.
- [9] H. J. Carlo, I. F. Vis, and K. J. Roodbergen, “Storage yard operations in container terminals: Literature overview, trends, and research directions,” *European Journal of Operational Research*, vol. 235, no. 2, pp. 412–430, 2014, doi: 10.1016/j.ejor.2013.10.054.
- [10] C. Zhang, J. Liu, Y.-W. Wan, K. G. Murty, and R. J. Linn, “Optimization for two-stage double-cycle operations in container terminals,” *Flexible Services and Manufacturing Journal*, vol. 36, no. 2, pp. 345–362, 2024, doi: 10.1007/s10696-023-09527-7.

- [11] A. H. Gharehgozli, F. Vernooij, and R. Koster, “Efficient container stacking plans to reduce energy consumption,” *Journal of Cleaner Production*, vol. 165, pp. 1256–1267, 2017, doi: 10.1016/j.jclepro.2017.07.224.
- [12] T. Park, R. Choe, Y. H. Kim, and K. R. Ryu, “Dynamic adjustment of container stacking policy in an automated container terminal,” *International Journal of Production Economics*, vol. 133, no. 1, pp. 385–392, 2011, doi: 10.1016/j.ijpe.2010.06.017.
- [13] D. Ambrosino, C. Caballini, and S. Siri, “A mathematical model to evaluate different train loading and stacking policies in a container terminal,” *Maritime Economics & Logistics*, vol. 15, no. 3, pp. 292–308, 2013, doi: 10.1057/mel.2013.12.
- [14] R. Stahlbock and S. Voß, “Operations research at container terminals: A literature update,” *OR Spectrum*, vol. 30, no. 1, pp. 1–52, 2008, doi: 10.1007/s00291-007-0100-7.
- [15] Z. Li, Y. Zhang, and H. Wang, “Collaborative optimization of truck scheduling in container terminals using graph theory and DDQN,” *Scientific Reports*, vol. 15, no. 1, 12345, 2025, doi: 10.1038/s41598-025-12345-6.
- [16] Tổng cục Hải quan Việt Nam, “Báo cáo tổng kết hoạt động xuất nhập khẩu năm 2024.” [Online]. Available: <https://www.customs.gov.vn>. Accessed: Oct. 2, 2025.
- [17] Cục Hàng hải Việt Nam, “Báo cáo thống kê lưu lượng hàng hóa qua cảng biển 7 tháng đầu năm 2024.” [Online]. Available: <https://www.vinamarine.gov.vn>. Accessed: Oct. 2, 2025.
- [18] Bộ Giao thông Vận tải Việt Nam, “Chiến lược phát triển dịch vụ logistics Việt Nam đến năm 2030.” *VnEconomy*, 2023. [Online]. Available: <https://vneconomy.vn/xay-dung-chien-luoc-phat-trien-dich-vu-logistics-viet-nam.html>. Accessed: Oct. 2, 2025.
- [19] J. A. Gunawardhana, H. N. Perera, and A. Thibbotuwawa, “Rule-based dynamic container stacking to optimize yard operations at port terminals,” *Maritime Transport Research*, vol. 2, 100034, 2021, doi: 10.1016/j.martra.2021.100034.
- [20] I. Rekik, S. Elkosantini, and H. Chabchoub, “A case based heuristic for container stacking in seaport terminals,” *Advanced Engineering Informatics*, vol. 38, pp. 658–669, 2018, doi: 10.1016/j.aei.2018.08.013.
- [21] I. Rekik and S. Elkosantini, “A multi agent system for the online container stacking in seaport terminals,” *Journal of Computational Science*, vol. 35, pp. 12–24, 2019, doi: 10.1016/j.jocs.2019.05.003.
- [22] I. Rekik and S. Elkosantini, “A survey on container stacking problems based on a conceptual classification scheme: limitations and future trends,” *Asian Journal of*

Management Science and Applications, vol. 7, no. 1, pp. 23–58, 2022, doi: 10.1504/AJMSA.2022.120615.

[23] L. Henesey, A. Silonosov, C. Meyer, and L. Gerlitz, “Smart Container Stacking in the Yard,” in *Proc. 23rd Int. Conf. Harbor, Maritime and Multimodal Logistic Modeling and Simulation (HMS 2021)*, 2021, pp. 37–44.

[24] C. Lersteau, T. T. Nguyen, T. T. Le, H. N. Nguyen, and W. Shen, “Solving the Problem of Stacking Goods: Mathematical Model, Heuristics and a Case Study in Container Stacking in Ports,” *IEEE Access*, vol. 9, pp. 25330–25343, 2021, doi: 10.1109/ACCESS.2021.3055764.

[25] A. M. Law, *Simulation Modeling and Analysis*, 5th ed. New York, NY, USA: McGraw-Hill Education, 2015.

PHỤ LỤC: MÃ NGUỒN CHƯƠNG TRÌNH

```
import random
import statistics as stats
import matplotlib.pyplot as plt

# -----
# 1) Tham số mô hình bãi & thời gian nâng
# -----
N_CONTAINERS = 8      # 6–9 đều ổn; để so sánh công bằng giữ cố định
S = 3                 # số stack
H = 3                 # số tầng/stack
CAPACITY = S * H
assert N_CONTAINERS <= CAPACITY, "Số container vượt quá sức chứa bãi!"

t1, t2, t3 = 1, 2, 3  # thời gian nâng theo tầng (trên cùng/giữa/dáy)

# -----
# 2) Hàm tiện ích đặt/xử lý bãi & mô phỏng chi phí
# -----
def linear_slots(S, H):
    """Tạo ánh xạ slot tuyến tính (stack, tier) theo chỉ số 0..S*H-1."""
    order = []
    for s in range(S):
        for h in range(H):
            order.append((s, h))
    return order

SLOT_MAP = linear_slots(S, H)

def place_by_permutation(perm, S=S, H=H):
    """Chuyển hoán vị container -> cấu trúc bãi (list các stack từ đáy lên đỉnh)."""
    stacks = [[] for _ in range(S)]
    for idx, cid in enumerate(perm):
        s, h = SLOT_MAP[idx]
        while len(stacks[s]) < h:
            stacks[s].append(None)
        if len(stacks[s]) == h:
            stacks[s].append(cid)
        else:
            stacks[s][h] = cid
    # Nén bỏ None
    for s in range(S):
        stacks[s] = [c for c in stacks[s] if c is not None]
    return stacks

def lift_cost_for_position(num_above, t1=t1, t2=t2, t3=t3):
```

```

"""Chi phí nâng tùy số container ở phía trên."""
if num_above <= 0:
    return t1
elif num_above == 1:
    return t2 + t1
else:
    return t3 + t2 + t1

def simulate_cost(perm, retrieval_order, S=S, H=H):
    """Tính tổng chi phí nâng khi lấy theo retrieval_order."""
    stacks = place_by_permutation(perm, S, H)
    total = 0
    for target in retrieval_order:
        found = False
        for s in range(S):
            if target in stacks[s]:
                pos = stacks[s].index(target)
                num_above = len(stacks[s]) - pos - 1
                total += lift_cost_for_position(num_above)
                stacks[s].pop(pos)
                found = True
                break
        if not found:
            raise RuntimeError(f'Không tìm thấy {target} trong bãi!')
    return total

# -----
# 3) Chiến lược FIFO (First-In First-Storage)
# -----
def fifo_layout(ids):
    return ids[:] # đặt theo đúng thứ tự đến

# -----
# 4) GA components (PMX, mutation, repair, evaluate)
# -----
def pmx(parent1, parent2):
    """Partial-Mapped Crossover (two-point)."""
    n = len(parent1)
    p, q = sorted(random.sample(range(n), 2))
    child1, child2 = [None]*n, [None]*n

    # chép đoạn [p..q] chéo
    child1[p:q+1] = parent2[p:q+1]
    child2[p:q+1] = parent1[p:q+1]

    # ánh xạ xung đột

```

```

map12 = {parent1[i]: parent2[i] for i in range(p, q+1)}
map21 = {parent2[i]: parent1[i] for i in range(p, q+1)}

def fill(child, parent, mapping):
    for idx in list(range(0,p)) + list(range(q+1, n)):
        gene = parent[idx]
        while gene in child[p:q+1]:
            gene = mapping.get(gene, gene)
        child[idx] = gene
    return child

child1 = fill(child1, parent1, map12)
child2 = fill(child2, parent2, map21)
return child1, child2

def swap_mutation(chrom):
    a, b = random.sample(range(len(chrom)), 2)
    chrom[a], chrom[b] = chrom[b], chrom[a]
    return chrom

def repair(chrom, universe):
    """Loại trùng/thiếu để đảm bảo hoán vị hợp lệ trên tập universe."""
    seen, out = set(), []
    for g in chrom:
        if g not in seen and g in universe:
            out.append(g)
            seen.add(g)
    for g in universe:
        if g not in seen:
            out.append(g)
            seen.add(g)
    return out[:len(universe)]

def evaluate(chrom, retrieval_order):
    return simulate_cost(chrom, retrieval_order)

def ga_optimize(ids, retrieval_order, pop_size=50, max_gen=200, pc=0.9,
pm=0.25, elitism=True):
    """GA tối ưu hoán vị xếp container để tối thiểu hóa chi phí nâng."""
    # khởi tạo
    population = []
    for _ in range(pop_size):
        p = ids[:]
        random.shuffle(p)
        population.append(p)

```

```

best = min(population, key=lambda c: evaluate(c, retrieval_order))
best_cost = evaluate(best, retrieval_order)

for _ in range(max_gen):
    new_pop = []
    if elitism:
        new_pop.append(best[:])

    while len(new_pop) < pop_size:
        p1, p2 = random.sample(population, 2)
        if random.random() < pc:
            c1, c2 = pmx(p1, p2)
        else:
            c1, c2 = p1[:], p2[:]

        if random.random() < pm:
            c1 = swap_mutation(c1)
        if random.random() < pm:
            c2 = swap_mutation(c2)

        c1 = repair(c1, ids)
        c2 = repair(c2, ids)

        new_pop.extend([c1, c2])

    population = new_pop[:pop_size]

    # cập nhật best
    for indiv in population:
        c = evaluate(indiv, retrieval_order)
        if c < best_cost:
            best, best_cost = indiv[:], c

    return best, best_cost

# -----
# 5) Chạy nhiều TRIALS và vẽ biểu đồ
# -----
def run_trials(n_trials=20, base_seed=1234):
    costs_fifs, costs_ga, improvements = [], [], []
    fifs_layouts, ga_layouts, orders = [], [], []

    for trial in range(n_trials):
        random.seed(base_seed + trial)

        # Tạo danh sách container và thứ tự truy xuất ngẫu nhiên cho mỗi trial

```

```

ids = [f'C{i+1}' for i in range(N_CONTAINERS)]
retrieval_order = ids[:]
random.shuffle(retrieval_order)

# FIFS
perm_fifs = fifs_layout(ids)
c_fifs = simulate_cost(perm_fifs, retrieval_order)

# GA
best_ga, c_ga = ga_optimize(ids, retrieval_order)

# Luru
costs_fifs.append(c_fifs)
costs_ga.append(c_ga)
orders.append(retrieval_order)
fifs_layouts.append(perm_fifs)
ga_layouts.append(best_ga)

imp = 100.0 * (c_fifs - c_ga) / c_fifs if c_fifs > 0 else 0.0
improvements.append(imp)

return (costs_fifs, costs_ga, improvements, orders, fifs_layouts, ga_layouts)

# Thực thi
N_TRIALS = 20
costs_fifs, costs_ga, improvements, orders, fifs_layouts, ga_layouts =
run_trials(N_TRIALS)

print("==== Thống kê tóm tắt ====")
print(f'FIFS - mean: {stats.mean(costs_fifs):.2f} | median:
{stats.median(costs_fifs):.2f} | min/max: {min(costs_fifs)} / {max(costs_fifs)}')
print(f'GA - mean: {stats.mean(costs_ga):.2f} | median:
{stats.median(costs_ga):.2f} | min/max: {min(costs_ga)} / {max(costs_ga)}')
print(f'Improve(%) - mean: {stats.mean(improvements):.1f}% | median:
{stats.median(improvements):.1f}% | min/max: {min(improvements):.1f}% /
{max(improvements):.1f}%')

# -----
# (1) Line plot chi phí theo từng trial
# -----
plt.figure(figsize=(8,4))
plt.plot(range(1, N_TRIALS+1), costs_fifs, marker='o', label='FIFS')
plt.plot(range(1, N_TRIALS+1), costs_ga, marker='s', label='GA')
plt.xlabel("Lần chạy (trial)")
plt.ylabel("Tổng chi phí nâng")
plt.title("So sánh chi phí theo từng lần chạy: FIFS vs GA")

```

```

plt.legend()
plt.grid(True)
plt.show()

# -----
# (2) Boxplot phân phối chi phí
# -----
plt.figure(figsize=(6,4))
plt.boxplot([costs_fifs, costs_ga], labels=['FIFS','GA'])
plt.ylabel("Tổng chi phí nâng")
plt.title("Phân phối chi phí: FIFS vs GA (boxplot)")
plt.grid(True, axis='y')
plt.show()

# -----
# (3) Histogram % cải thiện
# -----
plt.figure(figsize=(7,4))
plt.hist(improvements, bins=8)
plt.xlabel("% cải thiện của GA so với FIFS")
plt.ylabel("Số lần chạy")
plt.title("Histogram tỷ lệ cải thiện (%) của GA")
plt.grid(True, axis='y')
plt.show()

# -----
def pretty_stacks(perm, S=S, H=H, title=""):
    stacks = place_by_permutation(perm, S, H)
    hmax = max(len(s) for s in stacks) if stacks else 0
    if title:
        print(f"\n--- {title} ---")
    for level in reversed(range(hmax)):
        row = []
        for s in range(S):
            if level < len(stacks[s]):
                row.append(f"{stacks[s][level]:>3}")
            else:
                row.append(" .")
        print(" ".join(row))

for t in range(min(3, N_TRIALS)):
    print(f"\n=== Trial {t+1} ===")
    print("Thứ tự truy xuất:", orders[t])
    print(f"FIFS cost={costs_fifs[t]} | GA cost={costs_ga[t]} |
    Improve={improvements[t]:.1f}%")
    pretty_stacks(fifs_layouts[t], title="FIFS layout")
    pretty_stacks(ga_layouts[t], title="GA best layout")

```