ỦY BAN NHÂN DÂN TP.HCM
SỞ KHOA HỌC VÀ CÔNG NGHỆ

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HCM
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

BÁO CÁO TỔNG KẾT

(Báo cáo nghiệm thu đã thông qua hội đồng)

XÂY DỰNG THƯ VIỆN LẬP TRÌNH HỖ TRỢ TỐI ƯU TỔ HỢP TRÊN MÔI TRƯỜNG TÍNH TOÁN SONG SONG VÀ PHÂN BỐ

Chủ nhiệm đề tài
(Họ tên và chữ ký)

TS. Trần Văn Hoài

Chủ tịch hội đồng nghiệm thu
(Họ tên và chữ ký)

PGS. TS. Dương Anh Đức

Phản biện 1
(Họ tên và chữ ký)

Phản biện 2
(Họ tên và chữ ký)

Cơ quan chủ trì

Cơ quan chủ quản

THÀNH PHỐ HỒ CHÍ MINH
THÁNG 5/ 2014

PHẦN MỞ ĐẦU

Tên đề tài: Xây dựng thư viện lập trình hỗ trợ tối ưu tổ hợp trên môi trường tính toán song song và phân bố

Chủ nhiệm đề tài: TS. Trần Văn Hoài

Cơ quan chủ trì: Trường Đại học Bách khoa TP.Hồ Chí Minh

Thời gian thực hiện: từ tháng 12/2011 đến tháng 3/2014

Kinh phí được duyệt: 300.000 ngàn đồng

Kinh phí đã cấp: 190.000 ngàn đồng (Theo thông báo số: 156/TB-KHCN ngày 02 tháng 12 năm 2011)

Mục tiêu: Đề tài "*xây dựng thư viện lập trình hỗ trợ tối ưu tổ hợp trên môi trường tính toán song song/phân bố*" với các mục tiêu sau:

- Hỗ trợ các nhà nghiên cứu trong các lĩnh vực khác giải quyết bài toán tối ưu tổ hợp sử dụng các cách tiếp cận dựa trên meta-heuristic.
- Xây dựng thư viện có các thành phần cơ bản với giao diện lập trình C++ phát hiện để có thể dễ dàng mô tả tiến trình tối ưu.
- Thử nghiệm thư viện vào các ứng dụng tối ưu trong kỹ thuật và quản lý để minh chứng khả năng.

Tiếp tục phát triển nhóm nghiên cứu tính toán khoa học đã được xây dựng qua nhiều hoạt động nghiên cứu từ trước đến nay tại ĐH Bách khoa TP.HCM.

Nội dung: Có 4 nội dung chính như sau (theo đề cương đã duyệt):

- Hỗ trợ làm việc với máy tính cụm sử dụng MPI (trong khối *computing resources* của hình 2) (MPI – Message Passing Interface): hiện nay thư viện chỉ mới sử dụng POP-C++ để làm việc trên lưới tính toán (phát triển trong khuôn khổ 2 đề tài EDAGrid và POP-C++). Tuy nhiên, một môi trường tính toán mạnh thông dụng nhất là máy tính cụm sử dụng thư viện lập trình truyền thông điệp MPI. Vì thế đề tài muốn bổ sung phần này để tăng khả năng sử dụng của thư viện.

- Bổ sung genetic algorithm (GA) (trong phần *meta-heuristics*): xây dựng thêm cơ chế cho thư viện hỗ trợ các phương pháp dựa trên dân số và hiện thực thuật toán GA.

- Bổ sung các cấu trúc cho workflow (trong phần *workflow engine*): lặp (có giới hạn số lần lặp), chọn lựa nghiệm trong tất cả các nghiệm đầu vào (có thể phục vụ cho GA), cơ chế đa dạng hóa giá trị của thông số trên các nhánh song song.

- Thử nghiệm tính toán cho 2 bài toán (trong phần ứng dụng): định lộ trình chuyên chở chất thải nguy hại, thiết khí động lực học cánh máy bay.

Và cụ thể là các nội dung được thể hiện qua các công việc được đăng ký như bảng sau (được thêm cột kết quả đạt được):

TT	**Các nội dung, công việc chủ yếu cần được thực hiện**	**Kết quả đạt được**
1	Xây dựng phần mềm hỗ trợ máy tính cụm dùng MPI	Phần mềm, C++, dùng MPI, triển khai dễ dàng trên bất kỳ máy tính cụm nào có thư viện MPI
2	Bổ sung GA cơ bản vào trong thư viện	Thuật toán GA chạy riêng biệt với các giao diện lập trình cơ bản. Thuật toán MA chạy riêng biệt với các giao diện lập trình cơ bản
3	Bổ sung các cấu trúc/cú pháp mô tả GA trong workflow hiện có	Đã xây dựng xong, cho phép kết nối GA, MA trong workflow.
4	Bổ sung các cú pháp cơ bản mô tả cấu trúc lặp, rẽ nhánh có điều kiện	Đã xây dựng xong giải pháp mô tả, và cơ chế trong workflow engine để thực hiện tính toán
5	Giải bài toán định lộ trình chuyên chở chất thải nguy hại bằng các meta-heuristic riêng biệt	Sử dụng MA (kết hợp GA và tìm kiếm cục bộ) và các phương pháp xây dựng Bài báo khoa học [Thai13b]
6	Giải bài toán định lộ trình chuyên chở chất thải nguy hại bằng cách kết hợp các meta-heuristic dưới dạng workflow	Đã thực hiện nhiều dạng workflow khác nhau để tính toán.
7	Giải bài toán tối ưu biên dạng cánh máy bay	Có GA, đã sử dụng thành công một thư viện CFD khác và cho kết quả tốt hơn

		hẳn so với [Khai11]
8	Giải bài toán tối ưu biên dạng cánh máy bay bằng cách kết hợp các meta-heuristic dưới dạng workflow	Đã xây dựng thuật toán MA dưới dạng kết hợp bằng workflow.
9	Thử nghiệm tính toán song song	Đã thử nghiêm tính toán tất cả các workflow.
10	Viết báo cáo tổng hợp	Đã hoàn thành tài liệu báo cáo cấp cơ sở và tài liệu hướng dẫn sử dụng.

Sản phẩm của đề tài (theo đề cương đã duyệt):

TT	**Tên sản phẩm**	**Yêu cầu khoa học đạt được**	**Nơi công bố** (Tạp chí, Nhà xuất bản)	**Ghi chú**
1	Bài báo khoa học	Công bố đầy đủ (full paper)	International Conference on Computational Science and Its Applications, LNCS (Springer)	
2	Luận văn thạc sĩ	- 1 luận văn (trong thời gian thực hiện đề tài) [Thai13a]	Luận văn Thạc sỹ, ngành Khoa học Máy tính, ĐH Bách khoa TP.HCM	
3	Thư viện lập trình hỗ trợ tối ưu dạng meta-heuristic	- Có hỗ trợ GA, MA. - Có hỗ trợ việc kết nối bằng các cú pháp khác nhau - Có hỗ trợ chạy trên máy tính cụm		

TÓM TẮT NỘI DUNG NGHIÊN CỨU

Tối ưu là một dạng bài toán có nhiều ứng dụng trong thực tế, không chỉ trong công nghiệp mà còn trong quản lý. Một quy trình ứng dụng các giải pháp tối ưu hóa thông thường qua các bước cơ bản sau: định nghĩa bài toán, mô hình hóa bài toán, giải bài toán bằng các phương pháp số, diễn giải/xác thực nghiệm của mô hình với các điều kiện thực tế. Dĩ nhiên, đây là một quy trình lặp theo nghĩa là các bước có thể quay lại các bước trước đó để cải thiện hướng đến chất lượng của lời giải được tốt nhất. Nhìn chung, quy trình trên có thể dẫn đến một nhận định là kiến thức đa ngành là cần thiết, và nếu đứng dưới góc nhìn của ngành công nghệ thông tin thì để giải hiệu quả một bài toán tối ưu thì rất cần sự tham gia của kiến thức các ngành khác, nhất là kiến thức về ứng dụng cụ thể. Đây cũng chính là kim chỉ nam cho nội dung đề xuất trong đề tài: *hướng đến một công cụ hỗ trợ cho người làm ứng dụng (có biết về lập trình) có thể sử dụng meta-heuristic (một công cụ tối ưu gần đúng) để giải các bài toán tối ưu*.

Tuy nhiên, việc xây dựng một môi trường tiện lợi cho người dùng không hoàn toàn đơn giản. Và để thực hiện điều đó và trong cách được nhóm nghiên cứu lựa chọn, đề tài hướng đến các nội dung nghiên cứu chính sau:

- Xây dựng một sườn chương trình dạng ngôn ngữ C++ để cho các nhà nghiên cứu ứng dụng triển khai nhanh các phương pháp dựa trên meta-heuristic để giải bài toán tối ưu. Ngoài ra, thư viện cũng cung cấp khả năng triển khai tính toán trên các máy tính song song.
- Triển khai thử nghiệm thư viện để giải quyết hai bài toán: xây dựng lộ trình chuyên chở chất thải rắn có xung đột; tối ưu biên dạng cánh máy bay.

SUMMARY OF RESEARCH CONTENT

Optimization is not only a theoretical research, but also applied in many practical areas, including both the industry and the management. A process of applying optimization-based approaches normally has four steps: problem statement, optimization modelling, problem solving by numerical methods, representation/implementation numerical solutions into real-world conditions. Certainly, this process is iterative in which each step can go back to the previous steps in order to obtain better solution quality. In general, it can be concluded that an interdisciplinary knowledge is quite critical in the solution process, and moreover, under the angle of information technology, to solve the optimization problem efficiently and effectively requires the involvement of application-specific knowledge. This is also a principle proposed in this project: towards a tool to support application-specific scientists (having basic programming skills) in applying meta-heuristics to find good solutions.

However, to achieve those expectation is not easy, and a tool to comfort all kinds of users cannot be obtained within the scope of this project. Instead, the project aims at the following objectives.

- To build a programming framework to help scientists to deploy their meta-heuristics based research quickly and effectively to solve optimization problems. Additionally, the framework also has an ability to deploy the computation parallel computers, although the scientists do not have much knowledge on parallel.
- To experiments the framework to solve real-world practical applications: vehicle routing to collect industrial waste with conflict; airfoil optimization.

MỤC LỤC

PHẦN MỞ ĐẦU III
TÓM TẮT NỘI DUNG NGHIÊN CỨU VII
MỤC LỤC IX
DANH SÁCH CHỮ VIẾT TẮT XI
QUYẾT TOÁN KINH PHÍ XVI
CHƯƠNG 1. TỔNG QUAN 1
1.1 CƠ SỞ LÝ LUẬN 1
1.2 CÁC CÔNG TRÌNH NGHIÊN CỨU LIÊN QUAN 2
1.3 Ý NGHĨA VÀ TÍNH MỚI VỀ KHOA HỌC VÀ THỰC TIỄN 7
CHƯƠNG 2. NỘI DUNG NGHIÊN CỨU 11
2.1 MỤC TIÊU NGHIÊN CỨU 11
2.2 HƯỚNG TIẾP CẬN CỦA NGHIÊN CỨU 12
2.3 PHÂN TÍCH VÀ TÌM HIỂU CÁC VẤN ĐỀ LIÊN QUAN 12
CHƯƠNG 3. NGUYÊN LÝ THIẾT KẾ CHUNG CỦA THƯ VIỆN 17
3.1 ĐẶC TẢ CÁC NỘI DUNG LIÊN QUAN ĐẾN BÀI TOÁN 19
3.2 ĐÓNG GÓI VÀ TÁI TẠO THÔNG TIN 21
3.3 LỰA CHỌN GIẢI THUẬT TÌM KIẾM 22
3.4 ĐẶC TẢ WORKFLOW 22
3.5 THỰC THI TÌM KIẾM 23
CHƯƠNG 4. KIẾN TRÚC CỦA THƯ VIỆN 24
4.1 TẦNG POPC++/MPI 25
4.2 TẦNG WORKFLOW ENGINE 26
4.2.1 Cấu trúc của workflow 26
4.2.2 Các lớp đối tượng thích nghi dữ liệu 29
4.3 TẦNG SEARCH ALGORITHM 30
4.3.1 Các giải thuật dựa trên dịch chuyển 31
4.3.2 Các giải thuật dựa trên quần thể 38
4.4 KIẾN TRÚC LỚP TRONG THƯ VIỆN 46
4.5 CẤU TRÚC CƠ SỞ DỮ LIỆU TRONG THƯ VIỆN 50
CHƯƠNG 5. CÁC CÔNG CỤ PHỤ TRỢ KHÁC 55
5.1 PHÁT HIỆN LỖI 55
5.2 NHẬT KÝ VÀ BIỂU ĐỒ 58
CHƯƠNG 6. THỬ NGHIỆM ĐÁNH GIÁ HIỆU NĂNG CỦA THƯ VIỆN 60
6.1 PHÁT BIỂU CỦA BÀI TOÁN 60
6.2 ĐẶC TẢ NGHIỆM 64
6.3 ĐẶC TẢ LÂN CẬN 67
6.3.1 *Lân cận 2-Opt* 68

6.3.2 Lân cận Or-Opt *69*
6.4 CƠ CHẾ PHÁT SINH LÂN CẬN 70
6.5 ĐẶC TẢ GEN VÀ CƠ CHẾ MÃ HÓA 72
6.6 CƠ CHẾ MÃ HÓA 73
6.7 XÂY DỰNG WORKFLOW TỐI ƯU 75
6.8 MÔI TRƯỜNG THỬ NGHIỆM 77
6.9 KẾT QUẢ THỬ NGHIỆM VÀ HIỆU NĂNG 78
CHƯƠNG 7: ỨNG DỤNG TRONG TỐI ƯU THỰC TẾ 88
7.1 HỖ TRỢ CỦA THƯ VIỆN 88
7.2 BÀI TOÁN VRPTWC 89
7.2.1 Phát biểu của bài toán: *90*
7.2.2 Đặc tả nghiệm: *93*
7.2.3 Cơ chế phát sinh lân cận *97*
7.2.4 Cơ chế giải mã và mã hóa: *100*
7.2.5 Chiến lược tối ưu: *101*
7.2.6 Kết quả thực nghiệm của bài toán VRP *103*
7.3 BÀI TOÁN AIRFOIL 109
7.3.1 Đặc tả bài toán Airfoil *110*
7.3.2 Đặc tả nghiệm: *112*
7.3.3 Phương pháp mã hóa *114*
7.3.4 Quy trình tối ưu cho bài toán Airfoil *115*
7.3.5 Kết quả thực nghiệm của bài toán Airfoil *118*
CHƯƠNG 8. KẾT LUẬN VÀ ĐỀ NGHỊ 122
TÀI LIỆU THAM KHẢO 126

DANH SÁCH CHỮ VIẾT TẮT

HC: Hill Climbing (Leo đồi)

SA: Simulated Annealing (Mô phỏng luyện kim)

TS: Tabu Search (Tìm kiếm Tabu)

GA: Genetic Algorithm (Thuật toán Gene)

MA: Memetic Algorithm (Thuật toán Memetic)

TSP: Travelling Salesman Problem (Bài toán người du lịch)

VRP: Vehicle Routing Problem (Bài toán tìm lộ trình xe)

VRPTWC: Vehicle Routing Problem with Time Windows and Conflict (Bài toán tìm lộ trình xe với ràng buộc khung thời gian và xung đột chất thải)

DANH SÁCH HÌNH

Hình 1. Ảnh hưởng của độ phức tạp không gian nghiệm đến quá trình tìm kiếm. 7
Hình 2. Ý tưởng phối hợp các quá trình tìm kiếm khác nhau. 7
Hình 3. Tìm kiếm dạng workflow trên 4 nhánh cho mạch pair. 8
Hình 4. Sơ đồ các bước tự của chương trình sử dụng thư viện. 18
Hình 5. Các interface cần thực hiện đối với giải thuật HC. 19
Hình 6. Các interface cần thực hiện đối với giải thuật TS. 19
Hình 7. Các interface cần thực hiện đối với giải thuật SA. 20
Hình 8. Các interface cần thực hiện đối với giải thuật GA. 20
Hình 9. Các interface cần thực hiện đối với giải thuật MA. 20
Hình 10. Các cấu trúc workflow được sử dụng trong sinh học, thiên văn học, động đất, vật lý [Dong13]. 22
Hình 11. Kiến trúc của thư viện lập trình hỗ trợ tối ưu tổ hợp. 24
Hình 12. Workflow tối ưu với cấu trúc rẽ nhánh và vòng. 29
Hình 13. Workflow tối ưu với cạnh vòng vi phạm điều kiện lặp. 29
Hình 14. Cơ chế tái tạo đối tượng dựa vào ID. 29
Hình 15. Các lớp giải thuật được hỗ trợ trong hệ thống thư viện. 31
Hình 16. Sơ đồ khối minh họa quá trình cải tiến lời giải ban đầu bằng các hàm sẵn có trong thư viện meta-heuristic. 31
Hình 17. Các cơ chế lựa chọn lân cận trong thư viện. 33
Hình 18. Các kiểu điều kiện dừng được hỗ trợ trong thư viện. 33
Hình 19. Hai cơ chế làm lạnh nhiệt độ được hỗ trợ trong thư viện. 35
Hình 20. Các kiểu điều kiện dừng được hỗ trợ trong thư viện. 35
Hình 21. Các loại danh sách Tabu được hỗ trợ trong thư viện. 37
Hình 22. Các tiêu chí tuyển chọn thực hiện bước chuyển cho các đối tượng nằm trong danh sách Tabu. 38
Hình 23. Các cơ chế khai phá không gian được hỗ trợ trong thư viện. 38
Hình 24. Sơ đồ khối của giải thuật GA. 40
Hình 25. Các tiêu chí chọn lọc tự nhiên trong thư viện. 42
Hình 26. Các cơ chế lai chéo được hỗ trợ trong thư viện. 43

Hình 27. Các cơ chế đột biến được hỗ trợ trong thư viện. 44

Hình 28. Quy trình thực hiện cơ chế cải tiến lời giải theo MA. 45

Hình 29. Biểu đồ hiển thị tập tin nhật ký của quá trình tối ưu với các tác vụ tối HC, SA, TS với 4 đơn vị tính toán song song..... 59

Hình 30. Cơ chế tạo lời giải khả thi của *2-Opt.* 68

Hình 31. Cơ chế tạo lời giải khả thi của *Or-Opt.* 69

Hình 32. Workflow tính toán với ngữ cảnh lặp lồng..... 75

Hình 33. Các sơ đồ tối ưu đối với từng loại giải thuật meta-heuristic đơn lẽ. 78

Hình 34. Hiện thực giải thuật VNS bằng giải thuật HC kết hợp với sự thay thông số λ của toán tử Or-Opt. 78

Hình 35. Hiện thực giải thuật MA song song (MPIMA) bằng workflow trên nền tảng kết hợp giải thuật HC và GA. 79

Hình 36. Workflow của SEQ01 & MPI01. 79

Hình 37. Workflow của SEQ02 & MPI02. 79

Hình 38. Workflow của SEQ03 & MPI03. 79

Hình 39. Workflow của SEQ04 & MPI04. 80

Hình 40. Quá trình tối ưu của giải thuật HC (2-Opt) với tập benchmark ali535. 82

Hình 41. Quá trình tối ưu của giải thuật SA cho tập benchmark ali535. 82

Hình 42. Quá trình tối ưu của giải thuật TS cho tập benchmark ali535. 83

Hình 43. Quá trình tối ưu của giải thuật GA cho tập benchmark ali535. 83

Hình 44. Quá trình tối ưu của MA cho tập benchmark ali535. 85

Hình 45. Quá trình tối ưu của MPIMA cho tập benchmark ali535. 85

Hình 46. So sánh thời gian tối ưu giữa workflow SEQ02 (1 processor) và MPI02 (21 processors) với 3 tập benchmark berlin52, eil101 và ali535. 87

Hình 47. Một lộ trình thu gom chất thải nguy hại thực giữa các điểm trong thành phố. 95

Hình 48. Cơ chế tạo lời giải khả thi của toán tử Interchange với $\lambda = 1$. 98

Hình 49. Cơ chế tạo lời giải khả thi của toán tử Composite Cyclic với $\lambda = 1$ và 2.... 100

Hình 50. Workflow tuần tự SEQ và song song MPI được thiết lập cho bài toán vận chuyển chất thải nguy hại. 102

Hình 51. Quá trình tối ưu của sơ đồ MA cho tập benchmark 335_stop..... 107

Hình 52. Quá trình tối ưu của sơ đồ MPIMA cho tập benchmark 335_stop. 107
Hình 53. Biên dạng cánh PARSEC với 11 thông số hình học cơ bản [Sobiec99].....110
Hình 54 Quy trình giải thuật tối ưu tự động cho bài toán tối ưu biên dạng cánh. 115
Hình 55. Đồ thị của quá trình tối ưu của sơ đồ MPI MA cho bài toán tối ưu hóa biên dạng cánh 2D trong chuyển động cận thanh. 120
Hình 56. So sánh giữa biên dạng RAE2822 và kết quả tối ưu của quy trình MA. 120
Hình 57. Phần đầu của biên dạng RAE2822 và kết quả tối ưu của quy trình MA. 121

DANH SÁCH BẢNG

Bảng 1. Log file được ghi lại trong quá trình tối ưu của giải thuật GA. 58
Bảng 2. Các thông số được thiết lập cho các workflow tính toán. 80
Bảng 3. Kết quả tối ưu với các workflow tính toán khác nhau trên dữ liệu berlin52. . 81
Bảng 4. Kết quả tối ưu với các workflow tính toán khác nhau trên dữ liệu eil101..... 84
Bảng 5. Kết quả tối ưu với các workflow tính toán khác nhau trên dữ liệu ali535..... 86
Bảng 6. Các thông số được thiết lập cho bài toán VRPTWC. 102
Bảng 7. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 102_stop. 105
Bảng 8. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 277_stop. 106
Bảng 9. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 335_stop. 108
Bảng 10. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 444_stop. 108
Bảng 11. Miền trị được thiết lập đối với từng biến thiết kế. 113
Bảng 12. Tập các biến thiết kế đầu ra từ quá trình tối ưu. 119

QUYẾT TOÁN KINH PHÍ

Thời gian đăng ký trong hợp đồng: *Từ tháng 12/2011 đến tháng 06/2013*

Gia hạn đến tháng 12/2013.

Gia hạn đến tháng 3/2014

Tổng kinh phí được duyệt: *300.000 ngàn đồng*

Kinh phí cấp giai đoạn 1: *(190.000 ngàn đồng) (Theo thông báo số: 156/TB-KHCN ngày 02 tháng 12 năm 2011) và đã được quyết toán như trong bảng sau*

Đơn vị: *ngàn đồng*

TT	Nội dung	Kinh phí	Trong đó	
			Ngân sách	Nguồn khác
I	Kinh phí đã được cấp (giai đoạn 1)	190.000	190.000	
II	Kinh phí quyết toán trong năm			
1.	Công chất xám	6.000	6.000	
2.	Công thuê khoán	177.000	177.000	
3.	Nguyên, nhiên, vật liệu, dụng cụ, phụ tùng, văn phòng phẩm	2.000	2.000	
4.	Thiết bị	0	0	
5.	Xét duyệt, giám định, nghiệm thu	0	0	
6.	Hội nghị, hội thảo	0	0	
7.	Đánh máy tài liệu	2.000	2.000	
8.	Giao thông liên lạc	0	0	
9.	Chi phí điều hành	3.000	3.000	
III	Tiết kiệm 5%	0	0	
IV	Kinh phí chuyển sang năm sau			

Kinh phí cấp giai đoạn 2: *do có thay đổi về quản lý nên kinh phí giai đoạn 2 chưa nhận được, và sẽ bổ sung thông tin sau.*

CHƯƠNG 1. TỔNG QUAN

1.1 CƠ SỞ LÝ LUẬN

Trong các ứng dụng thực tế ở Việt Nam, (chẳng hạn nhóm bài toán vận trù học) hầu như đa số cách tiếp cận hiện nay là xây dựng các hệ thống thông tin phục vụ cho quá trình quản lý. Một mục tiêu cao nhất của các hệ thống này là giám sát được toàn bộ hoạt động của hệ thống và có thể cung cấp những thông tin thống kê phục vụ cho việc hoạch định. Tuy nhiên, việc hoạch định này chủ yếu là dựa vào kinh nghiệm, không có sự hỗ trợ mạnh mẽ từ máy tính. Kể cả khi điều hành một hoạt động (chẳng hạn như điều hành xe buýt), các quy trình và tham số của quy trình cũng chưa được xem xét một cách kỹ lưỡng để hoạt động điều hành đó đạt được độ hiệu quả cao nhất. Những điểm yếu nêu trên có thể được giải quyết phần nào nếu biết sử dụng những thuật toán tối ưu được hiện thực trên máy tính. Rõ ràng đây là nhu cầu thấy rõ trong thực tế ngày nay ở Việt Nam.

Trong thời gian làm việc với các nhóm nghiên cứu khác nhau (cả thiên về ứng dụng và về công nghệ thông tin), chúng tôi (những người làm về toán – tin) thấy nhu cầu có những thư viện (công cụ) tính toán mạnh, nhưng dễ sử dụng để đưa đến cho các nhà nghiên cứu ứng dụng. Nếu không có những thư viện như thế thì hầu như cách tiếp cận thông thường nhất của họ là mua (sử dụng) các sản phẩm của nước ngoài. Điều này có thể đưa đến 2 điểm yếu :

- Khó tiếp cận đến được một giải pháp tốt cho các bài toán đặc thù (ở Việt Nam).
- Không nắm bắt được kiến thức, công nghệ của nước ngoài để sản sinh ra các sản phẩm (bài báo, phần mềm) mang dấu ấn của riêng mình.

Ngoài ra, nếu không có những thư viện như trên thì việc sử dụng tài nguyên đã được đầu tư là lãng phí. Trong một thời gian dài quản lý PTN Tính toán khoa học có hệ thống tính toán khá mạnh [Nam07] (nhưng đã khá lỗi thời), nhưng hiệu suất sử dụng chưa đạt như mong đợi. Một phần chính là các thư viện (công cụ) cung cấp trên đó còn khá hạn chế. Do đó chúng tôi thấy rất cần thiết để phát triển các thư

viện hỗ trợ để cung cấp cho các chuyên gia lĩnh vực ứng dụng. Trong khuôn khổ của đề tài, chúng tôi mong muốn tạo ra một thư viện cho lĩnh vực tối ưu rời rạc để ứng dụng vào các bài toán vận trù học trong thực tế.

1.2 CÁC CÔNG TRÌNH NGHIÊN CỨU LIÊN QUAN

Tính toán mềm (soft computing) không phải là một lĩnh vực mới mẻ trên thế giới và các thuật toán tối ưu (tìm kiếm) trong lĩnh vực này cũng đã được nghiên cứu rất nhiều. Các thuật toán meta-heuristic trong xu thế này đã được sử dụng rộng rãi trong nhiều nghiên cứu cơ bản và ứng dụng, cũng như đem lại nhiều hiệu quả kinh tế. Tham khảo chi tiết về lĩnh vực nghiên cứu này có thể tìm thấy trong [Bian09]. Tuy nhiên cho đến ngày nay vẫn còn rất nhiều nghiên cứu trong lĩnh vực này nhằm giải quyết các thách thức chính sau:

- *Đặc thù của ứng dụng:* có khá nhiều loại meta-heuristic và mỗi loại đều có những ưu và khuyết điểm đặc thù (ví dụ, SA khá tốt trong các bài toán tối ưu kỹ thuật với các hàm mục tiêu liên tục, nhưng lại không hoạt động tốt lắm trong các bài toán tối ưu trên đồ thị). Do đó, để có thể ứng dụng một loại meta-heuristic vào một ứng dụng cụ thể một cách hiệu quả, nhiều nghiên cứu và thử nghiệm tính toán phải được thực hiện. Đó là lý do chúng ta không thể áp dụng máy móc những tiếp cận dựa trên meta-heuristic đã có vào giải quyết những bài toán tối ưu xuất phát từ thực tế ngày nay. Và đây cũng là động lực cho các nghiên cứu dựa trên meta-heuristic ngày nay.
- *Meta-optimization:* như đã nêu trên, sử dụng một loại meta-heuristic cho một ứng dụng cụ thể có thể không đạt hiệu quả như mong muốn. Đó là thách thức mà các nghiên cứu ngày nay hướng đến một cách tiếp cận phối hợp các meta-heuristic khác nhau nhằm tận dụng các ưu thế đặc thù. Ngoài ra, lựa chọn một bộ các tham số tốt cho các meta-heuristic cũng không thực hiện được dễ dàng, và thường phải dựa trên nhiều tính toán thử nghiệm và kinh nghiệm của nhà khoa học. Do đó, meta-optimization là một xu hướng phối hợp để cung cấp những giá trị tốt cho các thông số một cách tự động.
- *Sử dụng môi trường tính toán mạnh:* giải quyết các bài toán tối ưu có không gian nghiệm lớn (hoặc phức tạp) yêu cầu những tài nguyên tính toán rất

mạnh, không chỉ hướng đến việc tìm ra lời giải nhanh, mà còn hướng đến chất lượng của lời giải tốt hơn. Đây cũng là xu thế đang được sử dụng mạnh mẽ ngày nay trong tính toán khoa học. Sử dụng hạ tầng tính toán mạnh trong meta-heuristic cũng không nằm ngoài xu thế này [Alba05, Hoff08].

Nhìn chung, các thách thức nêu trên không dễ giải quyết. Rõ ràng là để có thể tạo ra các tiếp cận meta-heuristic đặc thù cho một ứng dụng thì phải có sự hiểu biết khá tường tận về ứng dụng này (ví dụ về cấu trúc không gian nghiệm, cách biến đổi/di chuyển tốt trong không gian nghiệm, hàm lượng giá nghiệm …). Thách thức khoa học càng khó thì sự hiểu biết này càng quan trọng, và chỉ có những nhà nghiên cứu ứng dụng mới nắm bắt được. Vì thế việc xây dựng một công cụ hỗ trợ mạnh giúp các nhà khoa học tiếp cận nhanh đến các meta-heuristic là một nhu cầu rất lớn. Và công cụ càng mạnh hơn nếu có khả năng triển khai tính toán trên những hệ thống tính toán mạnh, mà không cần nhiều hiểu biết (và thay đổi trong chương trình) khi làm việc trên những hệ thống như thế. Đây cũng chính là định hướng của đề xuất trong đề tài này. Cụ thể hơn đề tài mong muốn giải quyết một thách thức cụ thể hơn "*xây dựng một thư viện lập trình hỗ trợ tối ưu tổ hợp*".

Như ta đã biết, chất lượng của các giải pháp tìm kiếm tùy thuộc rất nhiều vào không gian và phương thức tìm kiếm. Vì thế càng mở rộng vùng không gian tìm kiếm và mở rộng tính đa dạng tìm kiếm một cách hợp lý, kết quả tìm kiếm sẽ đạt gần giá trị tối ưu toàn cục. Tuy nhiên, dễ nhận ra rằng, việc tìm kiếm càng mở rộng, thời gian tìm kiếm sẽ gia tăng một cách đáng kể. Vì vậy nếu ứng dụng giải pháp tính toán song song, vừa tận dụng được ưu thế của từng thuật toán, tận dụng nguồn tài nguyên tính toán mạng lưới cải thiện đáng kể hiệu quả tính toán.

Trên thế giới hiện nay cũng có những nghiên cứu tương tự. Chúng cố gắng tạo ra những công cụ linh hoạt cho phép xây dựng các thuật toán dựa trên meta-heuristic, cũng như bổ sung các meta-heuristic mới khi cần thiết. Có hai xu hướng phát triển công cụ chính là mô hình hộp trắng và hộp đen. jMetal [Duri06], ParadisEO [Caho04], Shark [Igel08] theo mô hình hộp trắng, trong đó, người dùng có thể sử dụng các thành phần cơ bản được xây dựng sẵn, hoặc có thể thêm thuật toán mới bằng cách sử dụng các thành phần cơ bản có sẵn của công cụ. Ngược lại,

trong mô hình hộp đen, người dùng phải viết từ đầu nếu muốn thêm thuật toán mới. Đây là mô hình áp dụng trong PISA [Bleu03]. Tuy nhiên với PISA, việc phát triển thuật toán có sẵn khá dễ dàng.

Phần lớn các thư viện tìm kiếm meta-heuristic này đều không hỗ trợ cho tính toán song song. Chẳng hạn như jMetal, Shark, PISA or MOMHLib++ chỉ đáp ứng cho việc thực hiện tính toán tuần tự. Bên cạnh đó, có nhiều khung ứng dụng có khả năng chạy song song như DGPE và ParadisEO. Có thể nói, công cụ mạnh nhất trong lĩnh vực nghiên cứu này là ParadisEO. Nó cung cấp khá nhiều tính năng mạnh (giải quyết các thách thức khoa học nêu trên) và đã ứng dụng khá hiệu quả vào nhiều bài toán tính toán lớn ngày nay trong: sinh – tin học [Tant06], thiết kế mạng di động [Talb07]. Một đóng góp quan trọng của ParadisEO là khả năng triển khai tính toán trên các hệ thống song song (sử dụng MPI) và phân bố (dùng MPICH-G2). Tuy nhiên chúng không hỗ trợ cho xây dựng các ứng dụng trên mạng lưới. Các version ParadisEO sử dụng MPI, nhưng chúng chỉ chạy trên cụm hoặc các mạng máy trạm. ParadisEO-G là một cải tiến ParadisEO và nó có thể chạy trên môi trường mạng lưới bằng cách sử dụng MPICH-G2. Ví dụ khác là thư viện DGPF, nó hỗ trợ nhiều cách thực hiện như tuần tự, client-server hoặc P2P, nhưng để thực hiện ứng dụng, các dịch vụ thời gian chạy (runtime) phải được cài đặt trên tất cả các máy.

Tuy nhiên, sự phối hợp các meta-heuristic chưa được quan tâm nhiều trong ParadisEO. Việc kết hợp các meta-heuristic lại với nhau lấy cảm hứng từ tự nhiên là một chủ đề đang được quan tâm hiện nay. Người đọc có thể tham khảo trong [Yang13] để biết thêm tổng quan về hướng này. Việc phối hợp các meta-heuristic thành công có thể xem trong [Dom13], trong đó nhóm tác giả lấy cảm hứng từ việc kết hợp nên các hydro-carbon. Đề tài của chúng tôi cũng đề xuất cách kết hợp các meta-heuristic lại với nhau, nhưng ở dạng tổng quát hơn trong một mô hình workflow, được sử dụng nhiều ngày nay trên các hệ thống tính toán lớn (ASKALON [Fahr05], Pegasus [Deel05]) và trong doanh nghiệp (BPEL, tham khảo thêm trong [Lour08]). Kết quả nghiên cứu ban đầu cũng đã được đưa vào sử dụng trong một đề tài khoa học công nghệ của Sở KHCN TP.HCM 2007-2009, "*Xây*

dựng công cụ lập trình cho tính toán thích nghi hiệu năng cao trên môi trường tính toán lưới" (đã nghiệm thu thành công). Hơn thế nữa, trong xu thế ảo hóa ngày nay thì việc sử dụng các hạ tầng tính toán có tính mềm dẻo cao là một hướng đi được các nhà nghiên cứu theo đuổi. Nghiên cứu công bố trong [Truong11] đã chỉ ra rằng với các nhóm nghiên cứu nhỏ thì hạ tầng tính toán đám mây là một hướng đi hợp lý. Tuy nhiên, triển khai các tính toán trên những hạ tầng như thế vẫn còn nhiều thách thức khoa học. Trong nghiên cứu của chúng tôi chỉ mới dừng lại đăng ký triển khai trên hạ tầng dựa trên MPI cơ bản.

Nhiều phương thức tính toán song song được đề xuất cho các thuật toán meta-heuristic và được chia thành 3 dạng. Cách đầu tiên là cố gắng thực hiện tính toán song song trong mỗi phương thức tìm kiếm. Cách thứ hai là thiết lập các luồng quyết định, mỗi luồng tối ưu một phần, và sau đó mở rộng cho các luồng khác. Cách cuối cùng là tạo nhiều luồng tìm kiếm để mở rộng không gian tìm kiếm. Dạng này có thể hiểu là Parallel Heterogeneous Metaheuristics. Trong các kiểu phương thức tìm kiếm này, kiểu thứ ba cần có việc giao tiếp giữa các luồng tìm kiếm. Mỗi luồng sử dụng một phương thức tìm kiếm khác nhau (về thuật toán, thông số khởi tạo…) và hoàn toàn độc lập với nhau. Vì vậy việc giao tiếp giữa các luồng thường chỉ diễn ra khi kết thúc pha tìm kiếm và trao đổi thông tin với pha tiếp theo.

Lưới tính toán được sử dụng rộng rãi để nâng cao hiệu quả tính toán vì nó cung cấp một nguồn lực tính toán mạnh mẽ bằng sự kết hợp các tài nguyên tính toán trên một phạm vi rộng. Tuy nhiên, điểm hạn chế là giao tiếp giữa các tài nguyên vì chúng thường kết nối với nhau qua một giao tiếp chậm (slow connection) trong khi việc giao tiếp trên mạng là đáng kể. Để tránh trường hợp này, giao tiếp trong khi tính toán sẽ bị giới hạn. Trong 3 kiểu phương thức tính toán song song, kiểu giao tiếp thứ 3 phù hợp nhất trong tính toán lưới.

Có nhiều thư viện, khung ứng dụng và công cụ hỗ trợ cho viết các ứng dụng lưới, như mô hình lập trình chuyển thông điệp (message passing programming model), dịch vụ lưới (grid services), CORBA, v…v... Trong đó, lập trình đối tượng song song (Parallel Object Programming) trong C++ (POP-C++) là lựa chọn phù hợp cho việc viết ứng dụng lưới. POP-C++ được nâng cao từ C++ với một số từ

khóa mới để tạo các đối tượng song song, và bao gồm môi trường POP-C++ runtime để phát triển và quản lý các đối tượng trên các bộ xử lý từ xa.

Tại Việt Nam nhìn chung là có rất nhiều nghiên cứu về meta-heuristic cũng như ứng dụng meta-heuristic cụ thể nào đó vào những ứng dụng cụ thể, trong xu thế tính toán mềm. Trong khuôn khổ của báo cáo này không thể liệt kê tất cả, hơn nữa cũng không có một khảo sát ở Việt Nam về các nghiên cứu này. Tuy nhiên, theo đánh giá của chúng tôi là chưa có nhóm nghiên cứu nào ở Việt Nam đã thực hiện (hoặc định hướng đến) các mục tiêu đề xuất trong đề tài này, đó là xây dựng một thư viện mạnh và linh hoạt để hỗ trợ các nhà nghiên cứu ứng dụng khi họ mong muốn dùng meta-heuristic để giải bài toán của họ.

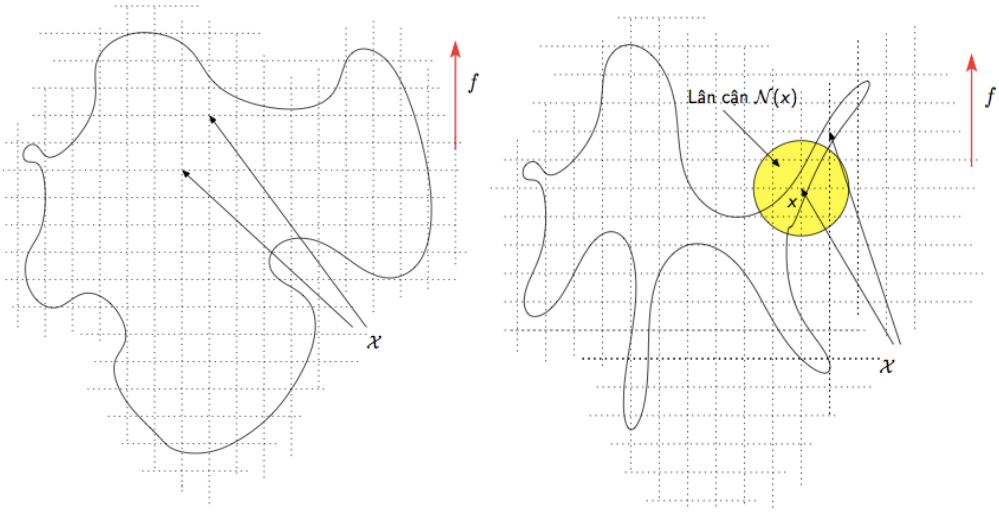
Một số nhóm nghiên cứu (và đề tài) hướng đến xây dựng các hạ tầng tính toán mạnh và những công cụ cấp thấp để quản lý những hạ tầng tính toán này (Supernode II tại ĐH Bách khoa TP.HCM, máy tính song song tại ĐH Tự nhiên Hà nội, ĐH Tự nhiên TP.HCM, EDAGrid [EDA08], VN-Grid. Cũng có những nghiên cứu ở mức cao hơn trên các hạ tầng này, nhưng mới chỉ dừng lại ở mức ngôn ngữ lập trình như POP-C++ [Ngo09]. Một hướng nghiên cứu khác ở Việt Nam là tập trung vào giải quyết các bài toán thực tế trong : sinh – tin học, lập lịch, thiết kế (trong kỹ thuật). Tuy nhiên, các hướng này chỉ mới dừng lại ở mức sử dụng một (hoặc một cách phối hợp cụ thể) các meta-heuristic nào đó vào giải quyết bài toán cụ thể.

Như đã nêu trên, đề tài muốn điền vào khoảng trống nghiên cứu ở Việt Nam trong việc tạo ra một công cụ mạnh hỗ trợ cho các nghiên cứu theo hướng meta-heuristic. Hơn nữa, mặc dù trên thế giới cũng có những nghiên cứu liên quan, nhưng nhóm nghiên cứu mong muốn thông qua đề tài để xây dựng được một công cụ tính toán của riêng mình. Thông qua đó, sự hiểu biết và cách giải quyết vấn đề trong khi thực hiện đề tài sẽ có ích rất nhiều cho nhóm trong những nghiên cứu sau này.

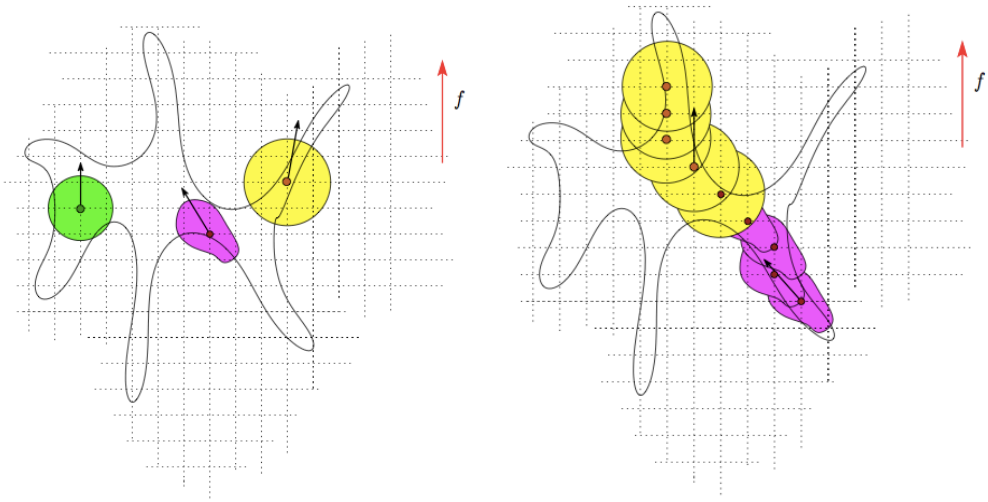
1.3 Ý NGHĨA VÀ TÍNH MỚI VỀ KHOA HỌC VÀ THỰC TIỄN

Như đã trình bày khá chi tiết trong phần trên, nhóm nghiên cứu đã xác định đây là một thách thức khoa học thú vị. Mặc dù cũng đã có một vài nghiên cứu gần đây, nhưng vẫn còn rất ít, ngay cả trên thế giới. Ngoài ra, nhóm cũng mong muốn bổ sung một số ý tưởng mới để đóng góp vào các nghiên cứu chung trên thế giới. **Một ý tưởng quan trọng nhất về mặt khoa học trong đề tài là sử dụng workflow để mô tả tiến trình tìm kiếm, hay nói cụ thể hơn là thông qua workflow để xây dựng một tìm kiếm cộng tác giữa các meta-heuristic khác nhau.**

Nếu chúng ta xét một bài toán tối ưu tổ hợp/rời rạc có các không gian nghiệm khác nhau trong 2 hình sau.



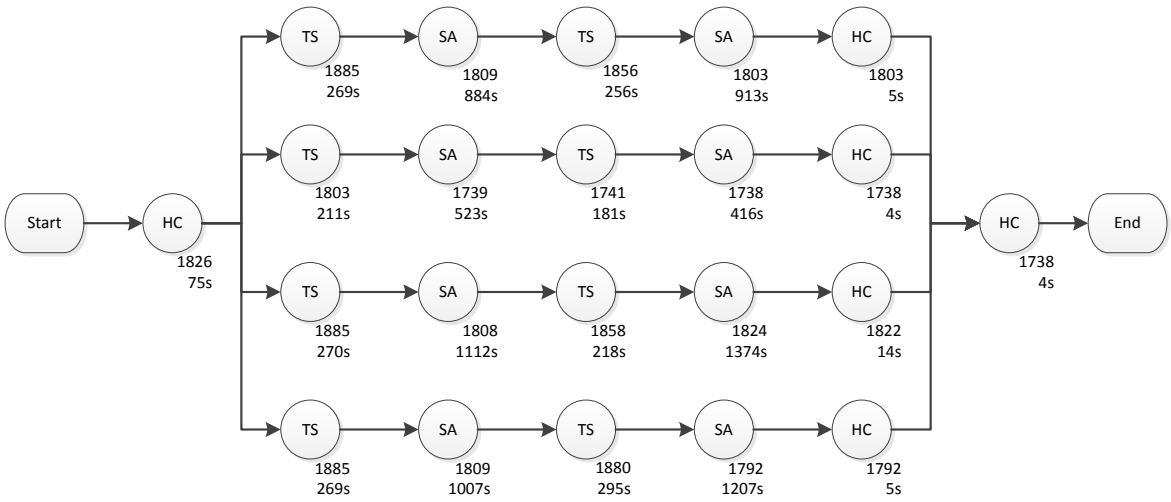
Hình 1. Ảnh hưởng của độ phức tạp không gian nghiệm đến quá trình tìm kiếm.



Hình 2. Ý tưởng phối hợp các quá trình tìm kiếm khác nhau.

Rõ ràng là với các ràng buộc phức tạp thì việc di chuyển trong không gian nghiệm trở nên khó khăn hơn rất nhiều. Như đã trình bày ở trên, rất nhiều nghiên cứu hiện nay dưới dạng phối hợp các meta-heuristic đều hướng đến chung một ý tưởng tương tự như 2 hình dưới đây.

Hình bên trái là đa dạng hóa tìm kiếm và hình bên phải là cộng tác các tìm kiếm. Định hướng chính và khác biệt quan trọng của đề tài là dùng workflow để mô tả các sự kết hợp này. Ví dụ như trong hình sau ta có một workflow mô tả việc giải bài toán tối ưu mạch luận lý tổ hợp, và kết quả tính toán rất khả quan so với các nghiên cứu đã được công bố trên thế giới [Hoai09].



Hình 3. Tìm kiếm dạng workflow trên 4 nhánh cho mạch pair.

Tuy nhiên, những kết quả ban đầu trên vẫn còn nhiều điểm cần được mở rộng. Nếu không xét về khía cạnh thiết kế vi mạch, mà chỉ xét về khía cạnh thư viện tối ưu tổ hợp, thì workflow còn được mô tả khá đơn giản. Cụ thể workflow chỉ mới dừng lại ở mức:

- Một đồ thị có hướng không vòng (directed acyclic graph),
- Hai cấu trúc chính: song song (rẽ nhánh) và tuần tự,
- Các thuật toán meta-heuristic hiện có chỉ dựa trên dịch chuyển trong không gian nghiệm. Vì thế việc lựa chọn nghiệm đầu vào của một nút tìm kiếm còn hạn chế (bỏ đi những nghiệm kém).
- Phương pháp đưa các giá trị tham số vào còn khá hạn chế.

Chúng tôi mong muốn sẽ thực hiện các bổ sung quan trọng cho những hạn chế trên trong khuôn khổ đề tài. Ngoài ra, đề tài cũng muốn hướng đến thử nghiệm thư viện vào hai bài toán có tính chất khá khác nhau:

- *Vận trù học:* bài toán định lộ trình cho xe chuyên chở chất thải rắn nguy hại từ các nhà máy đến các địa điểm tiêu hủy. Với nhóm bài toán này đã có một số đề tài cấp Sở KH&CN TP.HCM: Nghiên cứu mô hình tổ chức thu gom chất thải rắn đô thị tại TP.HCM (2009), Đánh giá hiện trạng thu gom, xử lý chất thải nguy hại trên địa bàn Tp. Hồ Chí Minh phục vụ cho việc xây dựng bãi chôn lấp an toàn (2010), Ứng dụng GIS trong quản lý chất thải rắn cấp Quận, thí điểm tại Q.10 (2009). Rõ ràng là các đề tài trên chỉ mới thực hiện ở mức phân tích nhu cầu, thách thức của quản lý, xây dựng hệ thống thông tin quản lý để theo dõi việc thu gom chất thải. Chúng tôi cũng đã thực hiện những tính toán thử nghiệm ban đầu về việc xây dựng lộ trình dựa trên các phương pháp xây dựng (constructive methods), nhưng nghiệm chưa đạt kết quả mong muốn (trong khuôn khổ một luận văn cao học). Các cải tiến cũng đã được thực hiện trong khuôn khổ của 2 luận văn cao học (2 đã bảo vệ, 1 đang thực hiện) cũng đã minh chứng là hướng tiếp cận của chúng tôi là có thể phát triển xa hơn được.
- *Kỹ thuật hàng không:* hiện nhóm đã và đang kết hợp với bộ môn hàng không của khoa giao thông, ĐH Bách khoa TP.HCM và nội dung kết hợp là sử dụng genetic algorithm kết hợp với tính toán số lưu chất trong bài toán thiết kế cánh máy bay (thuộc nhóm đề tài NAFOSTED). Thông qua việc kết hợp này, nhóm nghiên cứu muốn xây dựng các thuật toán dựa trên quần thể và đưa vào trong thư viện. Điều này cũng sẽ cải thiện rất nhiều việc sử dụng đầu vào của các meta-heuristic (không bỏ đi các nghiệm kém mà sử dụng chúng như các dân cư ban đầu của một dân số cho thuật toán dựa trên dân số). Hiện nay, nhóm nghiên cứu đã xây dựng một thuật toán GA có kết hợp với mạng neuron thần kinh để giảm thời gian mô phỏng dạng khí động. Kết quả ban đầu khá tiềm năng và đã được công bố trong [Khai11]. Thông qua kết quả

ban đầu này, nhóm muốn phát triển xa hơn để tích hợp GA, MA vào thư viện đang có với hy vọng cải thiện chất lượng nghiệm tối ưu.

CHƯƠNG 2. NỘI DUNG NGHIÊN CỨU

2.1 MỤC TIÊU NGHIÊN CỨU

Mục tiêu chính của nhóm nghiên cứu là phát triển một thư viện dựa trên tập hợp các thuật toán meta-heuristic, được chia ra nhiều thành phần có chức năng riêng biệt giúp người sử dụng dễ dàng hiểu về kiến trúc. Từ đó dễ dàng sử dụng các thuật toán để định nghĩa các bài toán qua giao diện được hỗ trợ nhờ vào cấu trúc rõ ràng và mềm dẻo. Tuy nhiên, việc xây dựng một môi trường tiện lợi cho người dùng không hoàn toàn đơn giản. Và để thực hiện được các điều nêu trên, đề tài hướng đến các nội dung nghiên cứu chính cần thực hiện như sau:

- Xây dựng một sườn chương trình dạng ngôn ngữ C++ để cho các nhà nghiên cứu ứng dụng triển khai nhanh các phương pháp dựa trên meta-heuristic để giải bài toán tối ưu.
- Hỗ trợ làm việc với máy tính cụm sử dụng MPI (trong khối computing resources của hình 2) (MPI – Message Passing Interface): hiện nay thư viện chỉ mới sử dụng POP-C++ để làm việc trên lưới tính toán (phát triển trong khuôn khổ 2 đề tài EDAGrid và POP-C++). Tuy nhiên, một môi trường tính toán mạnh thông dụng nhất là máy tính cụm sử dụng thư viện lập trình truyền thông điệp MPI. Vì thế đề tài muốn bổ sung phần này để tăng khả năng sử dụng của thư viện.
- Bổ sung genetic algorithm (GA) (trong phần meta-heuristic): xây dựng thêm cơ chế cho thư viện hỗ trợ các phương pháp dựa trên dân số và hiện thực thuật toán GA.
- Bổ sung các cấu trúc cho workflow (trong phần workflow engine): lặp (có giới hạn số lần lặp), chọn lựa nghiệm trong tất cả các nghiệm đầu vào (có thể phục vụ cho GA), cơ chế đa dạng hóa giá trị của thông số trên các nhánh song song.
- Thử nghiệm tính toán cho 3 bài toán (trong phần ứng dụng): bài toán người bán hàng (TSP), bài toán định lộ trình chuyên chở chất thải nguy hại (VRPTWC), bài toán thiết khí động lực học cánh máy bay (airfoil).

2.2 HƯỚNG TIẾP CẬN CỦA NGHIÊN CỨU

Để có thể đạt được kết quả nghiên cứu tốt cho các mục tiêu nêu trên, nhóm nghiên cứu đã đang thực hiện cách tiếp cận như sau:

- Tìm hiểu các nghiên cứu liên quan trong 2 nhóm chính:

  Phối hợp các meta-heuristic trong các ứng dụng, và tập trung vào nhóm xây dựng công cụ, môi trường hỗ trợ ứng dụng. Công trình đào sâu vào 2 ứng dụng như VRPTWC và AIRFOIL và có các kết quả khả quan. Trong những công trình nghiên cứu do nhóm công bố gần đây [Thai13a, Thai13b] việc phối hợp các giải thuật meta-heuristic đã đem lại chất lượng nghiệm tốt.

  Mô hình hóa các tính toán khoa học dạng workflow. Đây là một hướng nghiên cứu rất được quan tâm hiện nay do có sự thay đổi rất nhiều về hạ tầng tính toán: kiến trúc vi xử lý nhiều nhân, điện toán đám mây (trước đó là điện toán lưới). Trong khuôn khổ của báo cáo không thể đề cập hết các nghiên cứu này. Người đọc có thể xem trong [Fahr05, Hoff08, Truong11].

- Tìm hiểu các ứng dụng cụ thể (bài toán người bán hàng, vận chuyển chất thải, thiết kế cánh máy bay) từ đó xác định ra được các giao diện phát hiện để hỗ trợ nhiều loại ứng dụng.
- Bổ sung GA vào trong thư viện có sẵn, và quan trọng nhất là tận dụng được các meta-heuristic có sẵn để có thể xây dựng được một tập dân cư tốt phục vụ cho GA, MA (và ngược lại).
- Thông qua các nghiên cứu liên ngành để đánh giá lại các đặc tính của thư viện khi dùng vào giải quyết các bài toán nghiên cứu ở những ngành khác.
- Công bố các kết quả tại các hội nghị để nhận được những phản hồi, giúp cho việc định hướng nghiên cứu trong tương lai.

2.3 PHÂN TÍCH VÀ TÌM HIỂU CÁC VẤN ĐỀ LIÊN QUAN

Các phương pháp meta-heuristic là các phương pháp giải bài toán một cách gần đúng, sử dụng trong các bài toán tìm lời giải tối ưu có độ phức tạp cao. Với định

hướng không tìm nghiệm chính xác, và dựa trên các kinh nghiệm thì có hai dạng chính trong meta-heuristic là *move-based* (dịch chuyển) và *population-based* (quần thể). Với những dạng tìm kiếm gần đúng này, chất lượng nghiệm của bài toán sẽ phụ thuộc vào nhiều yếu tố chính sau:

- Khởi động tìm kiếm từ đâu,
- Cách tìm kiếm, cải thiện trong không gian nghiệm như thế nào,
- Cách thoát khỏi tối ưu cục bộ như thế nào.

Nhìn chung, một sườn chung cho nhóm thuật toán meta-heuristic dựa trên dịch chuyển sẽ như sau:

1. Khởi động nghiệm x
2. Lựa chọn cấu trúc lân cận $N \in \{N_1, N_2, \dots, N_q\}$
3. Lựa chọn ứng cử viên $C(x) \subseteq N(x)$
4. Ước lượng giá của các dịch chuyển từ nghiệm hiện tại đến các nghiệm mới $g(x, y), y \in C(x)$
5. Hiện thực sự dịch chuyển theo một định hướng tối ưu $\tilde{x} = \arg opt_y \{g(x, y), y \in C(x)\}$
6. Ước lượng (và điều chỉnh nghiệm hiện tại), điều chỉnh các tham số (nếu cần); $x = \tilde{x}$
7. Kiểm tra điều kiện dừng và nếu chưa thì quay lại bước 2 hoặc bước 3 (tùy theo cách tiếp cận)

Dưới đây là một số thuật toán meta-heuristic đã được đưa vào trong thư viện của đề tài, dựa trên khung chính được trình bày ở trên:

- **Leo đồi.** Giải thuật này thuộc nhóm dịch chuyển. Trong giải thuật leo đồi, $q = 1$ (tức là chỉ có 1 cấu trúc lân cận). $C(x) = N(x)$, hay nói cách khác là tất cả các nghiệm chấp nhận được trong lân cận của x đều được xem xét. Hàm mục tiêu của bài toán tối ưu được sử dụng để lượng giá các dịch chuyển từ nghiệm hiện tại đến nghiệm mới. Thuật toán sẽ dừng lại khi không tồn tại nghiệm nào trong lân cận của x có giá trị hàm mục tiêu tốt hơn (hoặc với một

điều kiện dừng bắt buộc nào đó của người dùng). Vấn đề lớn nhất mà giải thuật leo đồi mắc phải là rơi vào vùng tối ưu cục bộ. Để khắc phục vấn đề này, một số cải tiến cũng đã được đưa ra như: khi đến một đỉnh của tối ưu cục bộ, để tìm được lời giải tốt hơn nữa, giải thuật leo đồi có thể được thực hiện lại với nhiều điểm xuất phát khác nhau mà được chọn ngẫu nhiên và lưu lại kết quả tốt nhất ở mỗi lần lặp. Nếu số lần lặp đủ lớn, lời giải tối ưu toàn cục có thể được tìm thấy. Dựa trên quan điểm này, nhóm nghiên cứu mong muốn cung cấp thêm cho người dùng khả năng thực hiện nhiều HC cùng một lúc với các điểm bắt đầu khác nhau. Sự tiện lợi khi phát triển sẽ giúp cho việc nghiên cứu giải pháp dựa trên HC sẽ hiệu quả hơn.

- **Tìm kiếm Tabu.** Giải thuật này thuộc nhóm dịch chuyển, trong đó có 2 điểm quan trọng. Thứ nhất là có 3 cấu trúc lân cận tương ứng với 3 giai đoạn tìm kiếm: local search, intensification, diversification. Thứ 2 là thuật toán có 1 tham số quan trọng, gọi là *danh sách tabu*. Danh sách này lưu trữ các lời giải đã được sử dụng để đảm bảo rằng một lời giải nào đó sẽ không phải xét lại. Trong giải thuật tìm kiếm Tabu, khác với leo đồi, một số lời giải có chi phí thấp hơn với lời giải hiện tại cũng vẫn được chấp nhận chỉ cần nó không có trong danh sách các lời giải đã qua với mong muốn khắc phụ tối ưu cục bộ. Điều kiện dừng cho giải thuật Tabu là khi sau một số lần tìm kiếm đủ lớn, không thể tìm kiếm được một lời giải nào tốt hơn cho lời giải hiện tại giải thuật kết thúc và chọn lời giải tốt nhất làm lời giải cuối cùng. Trong giải thuật tìm kiếm Tabu, kích thước của danh sách chứa các lời giải đã qua rất quan trọng, ảnh hưởng đến hiệu quả của việc tìm kiếm.
- **Giải thuật mô phỏng luyện kim.** Giải thuật này thuộc nhóm dịch chuyển, dựa trên ý tưởng quá trình luyện kim trong thực tế. Hai điểm khác biệt so với leo đồi là: $C(x)$ không nhất thiết phải bằng $N(x)$, mà thay vào đó là sinh ngẫu nhiên một nghiệm thuộc $N(x)$; một cơ chế điều khiển bằng nhiệt để chấp nhận nghiệm có giá trị hàm mục tiêu kém hơn nghiệm hiện tại. Đây là cơ chế giúp giải thuật luyện kim tránh tối ưu cục bộ.

Sau đây là sườn cho thuật toán dựa trên quần thể và tiến hoá:

1. Khởi động tập nghiệm $S \in F$

2. Lựa chọn các toán tử biến đổi $M \subseteq \{m_1, \dots, m_q\}$
3. Thực hiện biến đổi tập nghiệm $C = \cup_{m \in M} \cup_{s \in S} m(s)$
4. Ước lượng giá trị của các nghiệm mới $x \in C$
5. Lựa chọn tập nghiệm mới theo một định hướng tối ưu $\tilde{S} \subseteq S \cup C$
6. Ước lượng (và điều chỉnh nghiệm hiện tại), điều chỉnh các tham số (nếu cần); $S = \tilde{S}$
7. Kiểm tra điều kiện dừng và nếu chưa thì quay lại bước 2 hoặc bước 3 (tùy theo cách tiếp cận)

Với nhóm thuật toán dựa trên quần thể, nhóm nghiên cứu đã triển khai thuật toán GA như sau:

- **Giải thuật di truyền.** Giải thuật này thuộc nhóm quần thể và tiến hoá, lấy ý tưởng từ nguyên lý tiến hóa di truyền trong tự nhiên. Giải thuật cần một cộng đồng với một số dân cư ban đầu cụ thể, sau đó tại mỗi vòng lặp, một số hoạt động di truyền như lai ghép (*cross-over*), chọn lọc (*selection*), đột biến (*mutation*) được áp vào các dân cư hiện tại để sinh ra những cư dân hay những cá thể mới cho thế hế hệ tiếp theo. Sau đó, các cá thể được đưa qua một hàm mục tiêu để đo lường và chỉ những cá thể nào đáp ứng đủ tiêu phát hiện mới được giữ lại trong cộng đồng. Giải thuật cứ thế tiếp tục cho đến khi gặp phải điều kiện dừng hoặc một kết quả tốt như mong muốn. Với nguyên lý trên thì giải thuật GA có một số khác biệt so với sườn thuật toán dịch chuyển ở những điểm sau:
    - Tại mỗi thời điểm không phải chỉ có một nghiệm, mà là một tập nghiệm $S \subseteq F$, trong đó F là tập nghiệm chấp nhận được của bài toán.

Các cấu trúc lân cận trong thuật toán dựa trên dịch chuyển không còn nữa, mà thay vào đó là các toán tử biến đổi nghiệm trong S. Chú ý là việc dịch chuyển cũng tương ứng như việc biến đổi, tuy nhiên không chỉ là từ một nghiệm sang một nghiệm mà có thể là từ nhiều nghiệm sang nhiều nghiệm. Cụ thể nếu $m(x)$ là hàm dịch chuyển trong thuật toán dịch chuyển thì $m: F \rightarrow F$. Trong khi đó, trong thuật toán quần thể thì $m: 2^F \rightarrow 2^F$.

Một ý tưởng quan trọng mà đề tài theo đuổi là các thuật toán meta-heuristic có cấu trúc khá giống nhau và dựa trên các sườn đã nêu trên. Vì thế có nhiều nghiên cứu cố gắng triển khai các phân tích tương tự nêu trên để thành các thư viện có thể sử dụng lại nhiều lần.

CHƯƠNG 3. NGUYÊN LÝ THIẾT KẾ CHUNG CỦA THƯ VIỆN

Meta-heuristic đã được sử dụng rộng rãi để giải quyết các vấn đề tối ưu vì nó hỗ trợ cho việc tìm kiếm với những kết quả chấp nhận được trong khoảng thời gian cho phép. Tuy nhiên chất lượng của việc tìm kiếm tùy thuộc nhiều vào không gian và chiến lược tìm kiếm. Có thể nâng cao hiệu quả tìm kiếm bằng cách thực hiện nhiều phương pháp tìm kiếm khác nhau trên các thông số khác nhau để có thể tìm ra chiến lược tìm kiếm hiệu quả hơn.

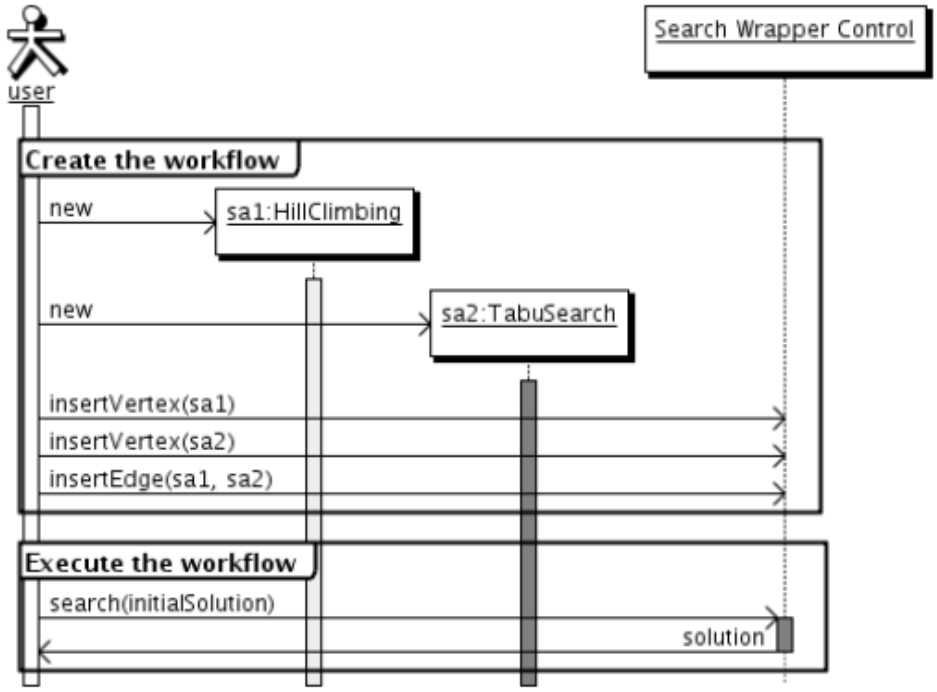
Thư viện cung cấp các giải thuật tìm kiếm meta-heuristic, cụ thể là HC (Hill Climbing), SA (Simulated Annealing), TS (Tabu Search), GA (Genetic Algorithm) và MA (Memetic Algorithm). Người dùng cũng có thể viết thêm các giải thuật khác dựa trên các interface cung cấp của thư viện nếu có nhu cầu.

Ngoài ra, để nâng cao hiệu quả tính toán, một trong những biện pháp cần thiết là tổ hợp các phương pháp tìm kiếm để tận dụng tối đa các ưu điểm của từng phương pháp, xây dựng một lưới tính toán phục vụ cho việc tìm kiếm, ứng dụng tính toán song song để nâng cao hiệu quả tìm kiếm. Có thể xây dựng các ứng dụng thực thi trên lưới bằng cách sử dụng các module đã được cung cấp được gọi là Workflow Engine. Các module này đáp ứng cho việc quản lý thực thi các lệnh và đưa ứng dụng lên môi trường tính toán hiệu năng cao. Workflow engine quản lý việc thực hiện các công việc tìm kiếm, và nó tiến hành thực thi tất cả các công việc đang ở trạng thái "*sẵn sàng*" trên nguồn tài nguyên được cung cấp lớn nhất có thể. Trước khi người sử dụng thực thi ứng dụng trên lưới, người sử dụng cần đăng ký lệnh thực thi với engine để tạo các quy trình. Quy trình là một đồ thị có chu trình có hướng tượng trưng cho thứ tự các công việc tìm kiếm và các dòng dữ liệu giữa chúng. Vậy để tiến hành xây dựng một chu trình cần tiến hành thực hiện các công việc sau:

- Xây dựng thuật toán tìm kiếm cho các công việc tìm kiếm.
- Xây dựng các dòng dữ liệu giữa các công việc, bao gồm việc kết nối các nhánh tuần tự, rẽ nhánh, lặp.

- Cung cấp khả năng giải độc lập cũng như kết hợp các giải thuật tìm kiếm Meta-heuristic trên sử dụng workflow. Người dùng sẽ đặc tả luồng thực thi tìm kiếm, sau đó thư viện sẽ thực thi, theo dõi và cho kết quả. Thư viện hiện tại hỗ trợ 2 dạng workflow: tuần tự và song song. Người lập trình có thể sử dụng workflow dạng tuần tự để thử nghiệm khi phát triển. Sau đó, các dạng workflow song song và tuần tự có thể kết hợp theo ý người sử dụng để thực thi thật sư trên môi trường phân bố.

Việc xây dựng dòng dữ liệu để xác định trình tự làm việc của các công việc như thứ tự thực hiện các công việc, các công việc sẽ được thực hiện song song đồng thời, các công việc nào thực hiện tuần tự, các dữ liệu vào ra của các công việc. Như hình 3.1, bài toán tuần tự đã sử dụng 2 thuật toán meta-heuristic phục vụ cho việc tìm kiếm là giải thuật leo đồi và tìm kiếm Tabu. Trong từng thuật toán, với những thông số phù hợp sẽ nâng cao hiệu quả tính toán. Vậy cần tiến hành xây dựng các đối tượng tìm kiếm tương ứng với từng thuật toán, mỗi công việc tìm kiếm của quy trình sẽ tương ứng là với một đối tượng tìm kiếm với thông số thích hợp.

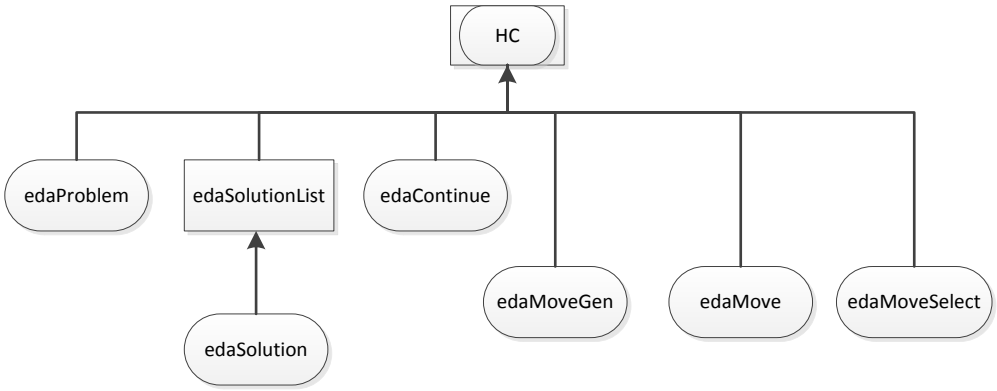


Hình 4. Sơ đồ các bước tự của chương trình sử dụng thư viện.

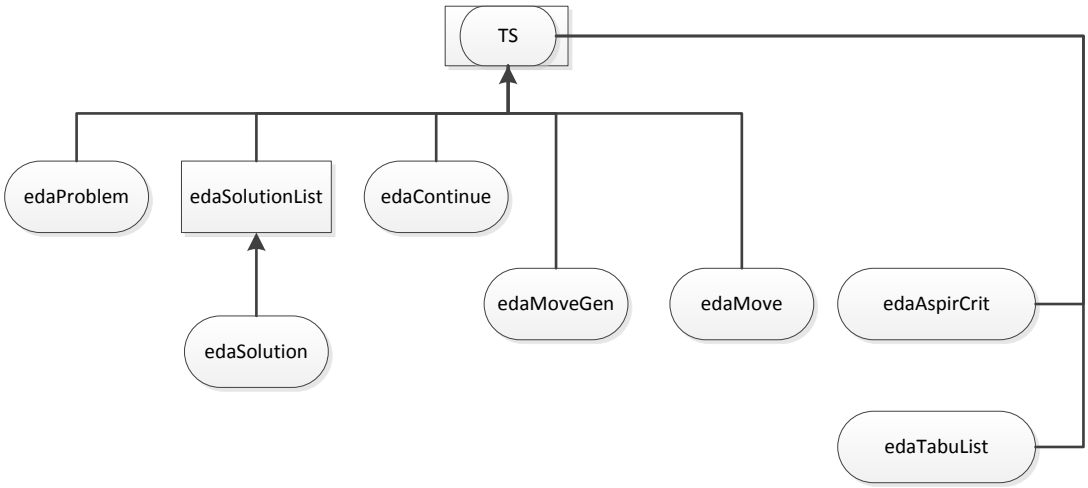
Thành phần chính do thư viện cung cấp bao gồm các giải thuật meta-heuristic và hiện thực hạ tầng song song. Đối với các lớp không đảm nhiệm những tác vụ này, chúng được thực hiện dưới dạng các giao diện (interface). Để sử dụng thư viện, người dùng cần phải tuân theo các nguyên tắc sau:

3.1 ĐẶC TẢ CÁC NỘI DUNG LIÊN QUAN ĐẾN BÀI TOÁN

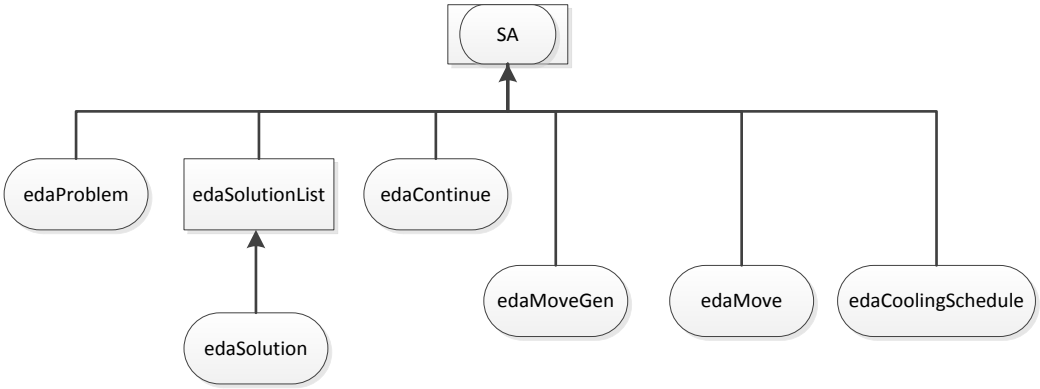
Người dùng thực hiện các interface của thư viện để mô tả bài toán của mình như nghiệm của bài toán, hàm lượng giá, các phép biến đổi nghiệm trong không gian (các bước chuyển), phương pháp mã hóa. Sau đây, sẽ là các lược đồ mô tả các lớp đối tượng cần được đặc tả ứng với từng loại giải thuật tối ưu.



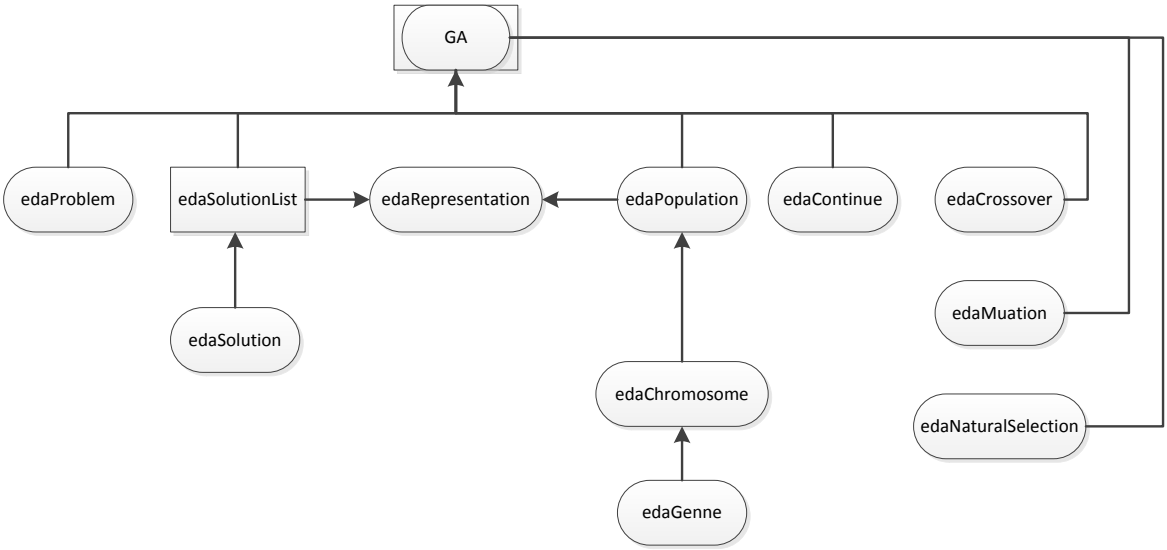
Hình 5. Các interface cần thực hiện đối với giải thuật HC.



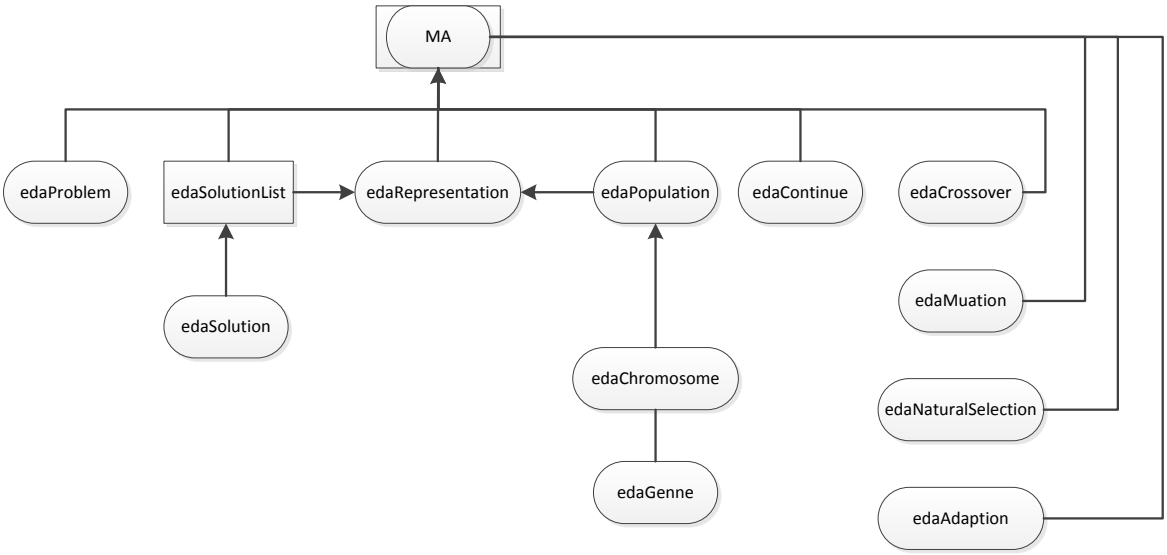
Hình 6. Các interface cần thực hiện đối với giải thuật TS.



Hình 7. Các interface cần thực hiện đối với giải thuật SA.



Hình 8. Các interface cần thực hiện đối với giải thuật GA.



Hình 9. Các interface cần thực hiện đối với giải thuật MA.

Biểu tượng oval ở trên các sơ đồ hình 3.4 đến 3.6 đại diện cho các interface (abstract class) ứng với các lớp đối tượng mà người sử dụng thư viện cần phải thực hiện. Biểu tượng hình chữ nhật đại diện cho lớp đối tượng đã được hỗ trợ sẵn trong thư viện. Các mối kết nối trong sơ đồ cho thấy liên hệ phụ thuộc giữa các đối tượng được kế thừa từ interface. Cụ thể như trong giải thuật HC chúng ta cần định nghĩa các interface: `edaProblem`, `edaSoltution`, `edaSolutionList`, `edaContinue`, `edaMove`, `edaMoveGen` và `edaMoveSelect`. Trong đó lớp edaSoltuionList trong thư viện sẽ được xây dựng từ lớp đối tượngkế thừa từ interface edaSolution do người dùng đặc tả.

Tùy theo yêu cầu của các loại giải thuật tối ưu được chọn mà người dùng phải cần đặc tả các lớp đối tượng liên quan. Tuy nhiên, vẫn có những lớp đối tượng đã được hỗ trợ sẵn bởi thư viện và điều này sẽ là rất tiện dụng cho người sử dụng thư viện. Các chi tiết về thông tin của các interface và các lớp được hiện thực kèm theo chúng sẽ được trình bày rõ ở các chương sau và trong tài liệu hướng dẫn kỹ thuật.

3.2 ĐÓNG GÓI VÀ TÁI TẠO THÔNG TIN

Để việc đóng gói và thực thi song song, người sử dụng thư viện cần chú ý:

- Các lớp muốn có thể truyền nhận giữa các đối tượng song song cần phải thừa kế từ lớp `edaSerialize` và viết lại phương thức `Serialize()`. Phương thức này sẽ làm nhiệm vụ đóng gói và khôi phục dữ liệu. Do đó, người dùng khi hiện thực lại các giao diện lập trinh cũng cần phải viết lại phương thức này trong trường hợp có chứa các thuộc tính mới
- Để giúp cho việc đóng gói dữ liệu, mỗi lớp cần phải có một định danh để giúp thư viện xác định kiểu dữ liệu ban đầu khi phục hồi dữ liệu. Thư viện đã cung cấp sẵn hàm `setClassID()` để gán định danh.
- Để có thể khôi phục lại dữ liệu, thư viện cần sử dụng hàm `userClassGernerate()`. Hàm này có nhiệm vụ tái tạo lại kiểu dữ liệu ban đầu dựa trên định danh của lớp. Đối với các lớp do người dùng mới định nghĩa, thư viện chưa biết được các định danh. Vì vậy, người dùng cần phải viết hàm này để thư viện có thể khôi phục lại kiểu dữ liệu ban đầu.

- Ngoài ra, các lớp nên hiện thực đầy đủ các constructor mặc định, copy constructor sao chép và phương thức `clone()` – dùng để tạo bản sao của một đối tượng.

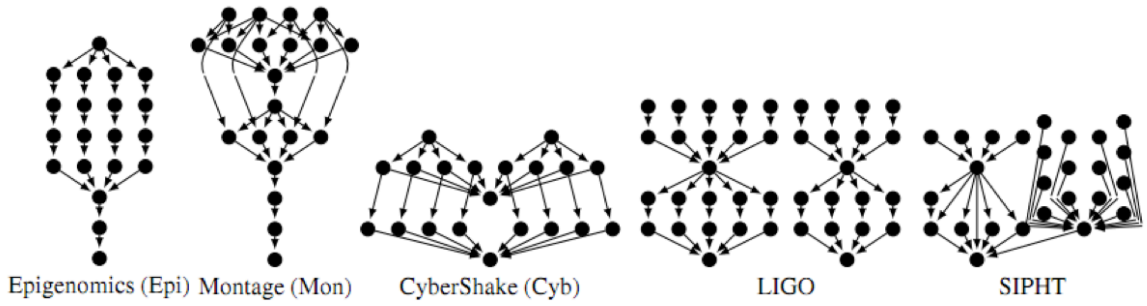
Chi tiết hơn của cách thực hiện các bước được yêu cầu sẽ được trình bày chi tiết trong phần ứng dụng minh hoạ.

3.3 LỰA CHỌN GIẢI THUẬT TÌM KIẾM

Người dùng sẽ lựa chọn giải thuật tìm kiếm thích hợp và kết hợp với các đặc tả để thành giải thuật hoàn chỉnh. Các giải thuật tìm kiếm có thể do thư viện cung cấp hoặc nếu cần thiết, người dùng có thể tạo ra các giải thuật tìm kiếm mới bằng cách sử dụng các interface cung cấp từ thư viện.

3.4 ĐẶC TẢ WORKFLOW

Sau khi đã có các giải thuật tìm kiếm cần thiết, người dùng sẽ tạo workflow cho chương trình. Workflow được định nghĩa dưới dạng đồ thị có hướng, trong đó, các giải thuật tìm kiếm sẽ là các đỉnh trong đồ thị, các cạnh của đồ thị mô tả sự phụ thuộc dữ liệu giữa các giải thuật tìm kiếm. Người dùng có thể lựa chon một trong hai dạng workflow đang có trong thư viện: dạng song song hoặc tuần tự. Chỉ với 2 cấu trúc này thì người dùng có thể tạo rất nhiều các dạng DAG khác nhau. Hình dưới đây cho thấy một số workflow được sử dụng trong các bài toán thực tế khác nhau đa số là dưới dạng DAG, sử dụng 2 cấu trúc tuần tự và song song.



Hình 10. Các cấu trúc workflow được sử dụng trong sinh học, thiên văn học, động đất, vật lý [Dong13].

Đối với workflow tuần tự, các tác vụ tìm kiếm sẽ thực thi tuần tự trên máy tính cục bộ như một ứng dụng đơn bình thường. Khi sử dụng workflow song song, các tác vụ tìm kiếm có thể thực thi đồng thời với nhau nếu có đồng thời nhiều tác vụ sẵn sang dữ liệu và có đủ tài nguyên tính toán. Các đối tượng song song cần được thực thi có thể được phân bố trên nhiều máy tính khác nhau. Lớp điều khiển workflow sẽ lần lượt thực thi các tác vụ tìm kiếm đang sẵn sàng. Tại một thời điểm, nếu có nhiều tác vụ tìm kiếm sẵn sàng, lớp điều khiển sẽ cố gắng tìm tài nguyên để thực thi toàn bộ các tác vụ đó.

Nghiệm ban đầu của một tác vụ tìm kiếm do lớp tìm kiếm lựa chọn. Nếu tác vụ không phụ thuộc dữ liệu vào các lớp khác, nó sẽ nhận nghiệm ban đầu do người dùng cung cấp. Nếu tác vụ phụ thuộc dữ liệu vào tác vụ khác, nó sẽ nhận một trong các nghiệm đó. Nếu có nhiều hơn một nghiệm ban đầu cho một tác vụ tìm kiếm, lớp điều khiển sẽ tổng hợp các nghiệm đó lại cho tác vụ tìm kiếm tiếp theo.

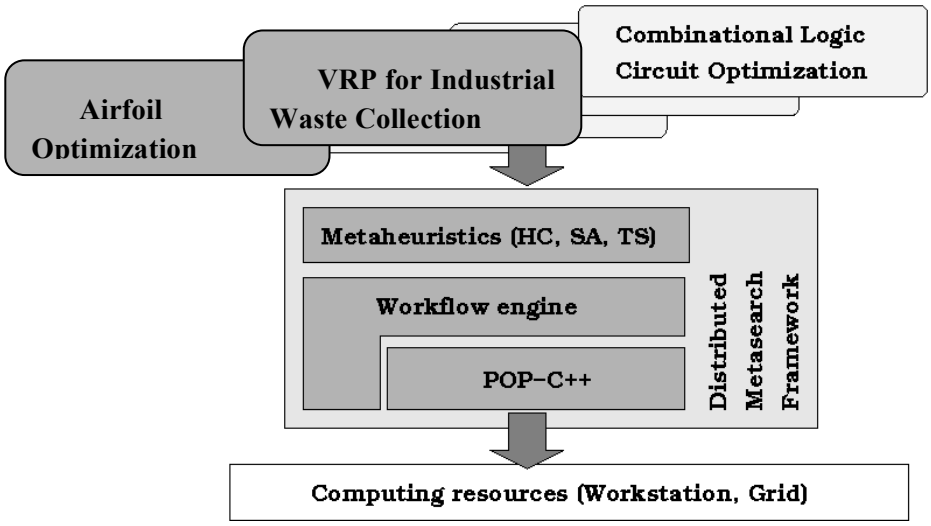
Để theo dõi các tác vụ tìm kiếm đã kết thúc hay chưa, thư viện sử dụng cơ chế polling để kiểm tra trạng thái các đối tượng tìm kiếm. Khi phát hiện một tác vụ hoàn thành, thư viện sẽ tiến hành lấy nghiệm về, giải phóng tài nguyên, sau đó kiểm tra lại sự phụ thuộc dữ liệu giữa các tác vụ để thực thi tác vụ tiếp theo. Trong trường hợp bị lỗi, lớp điều khiển sẽ cố gắng thử lại một số lần. Nếu lỗi không thể phục hồi được, tác vụ sẽ bị đánh dấu là thất bại. Các tác vụ phụ thuộc trên tác vụ này vẫn có thể thực thi nếu nó có những nghiệm khác.

3.5 THỰC THI TÌM KIẾM

Cuối cùng, người dùng sẽ gọi lệnh thực thi tìm kiếm. Thư viện sẽ dựa vào workflow do người dùng mô tả để điều khiển thứ tự thực thi. Nghiệm sẽ được trả về khi kết thúc việc tìm kiếm.

CHƯƠNG 4. KIẾN TRÚC CỦA THƯ VIỆN

Kiến trúc thư viện meta-heuristic cơ bản được chia ra ba tầng Search Algorithms, Workflow Engine, POPC++/MPI.



Hình 11. Kiến trúc của thư viện lập trình hỗ trợ tối ưu tổ hợp.

- *Tầng POPC++/MPI* quản lý tất cả các đối tượng chạy trong môi trường phân bố. Thư viện cũ tồn tại chứa các đối tượng chạy song song chạy trên lưới nên việc giao tiếp của thư viện thông qua hệ thống phần cứng dùng POPC++, một ngôn ngữ dùng cho lập trình song song trên lưới kế thừa từ C++. Thư viện hiện tại trong đề tài còn đưa thêm giao tiếp MPI vào tầng cuối cùng này để thư viện ngoài chạy trên hệ thống lưới máy tính còn có thể chạy trên các cụm máy tính hỗ trợ lập trình song song. Việc chạy song song các đối tượng trong một môi trường tính toán mạnh như lưới hoặc cụm máy tính cực kỳ cần thiết cho các bài toán tìm kiếm trong không gian nghiệm rất lớn, đẩy mạnh hiệu suất tính toán của thư viện.
- *Tầng Workflow Engine* điều khiển, quản lý các việc kết hợp các giải thuật tìm kiếm khi một workflow được tạo ra. Một workflow chứa các tập các đỉnh và các cạnh kết các đỉnh lại với nhau như trong hình 4.2 và 4.3. Các đỉnh đại diện cho các giải thuật tìm kiếm có thể và các cạnh đại diện cho mối quan hệ kết hợp của các giải thuật này. Những khối chức năng trong tầng này sẽ kiểm

tra tính khả thi của dữ liệu trước mỗi đỉnh, tức trước khi thực thi một giải thuật nào đó, và thực thi tất cả các giải thuật trong trường hợp nguồn tài nguyên cho phép và dữ liệu khả thi. Một workflow được tạo như thế nào là tốt nhất cho một bài toán cụ thể sẽ còn phụ thuộc vào kinh nghiệm của người giải quyết bài toán.

- *Tầng Search Algorithm* cung cấp các giải thuật tìm kiếm mà thư viện hỗ trợ và các lớp phụ trợ cho chúng. Lớp này cung cấp một chuỗi các giải thuật tìm kiếm mà thư viện hỗ trợ, cách thức định nghĩa bài toán, lời giải, cách xác định lân cận cũng như cách tính giá trị thích nghi cho bài toán. Đây là tầng mà người sử dụng thư viện cần nắm rõ để có thể tổ chức mạng lưới tính toán tối ưu một cách hiệu quả cho bài toán cần nghiên cứu.

4.1 TẦNG POPC++/MPI

Thư viện meta-heuristic song song sử dụng POPC++ runtime hoặc MPI hoặc tuần tự để quản lý các đối tượng thực thi trên môi trường phân tán. Tuy nhiên khung làm việc này chỉ làm việc với POPC++ runtime hoặc MPI hoặc tuần tự khi cần thiết. Trường hợp khi xây dựng các workflow engine tuần tự, workflow engine sẽ làm việc trực tiếp trên tài nguyên tính toán. Mặt khác, với workflow engine song song, cần được hỗ trợ môi trường POPC++ runtime hoặc MPI hoặc tuần tự vì cần phải tạo các đối tượng từ xa. Hai lớp này có thể làm việc cùng nhau hoặc có thể độc lập, đều có thể giao tiếp trực tiếp với các lớp thấp hơn. Trường hợp các thuật toán tìm kiếm có thể thực thi không cần workflow engine, lúc này chúng có thể thực thi tương tự như các lớp C++, nhưng khi cùng sử dụng 2 lớp, ứng dụng sẽ thực thi theo mô hình quy trình công việc.

Thực tế, thư viện meta-heuristic không viết lại các hàm liên lạc, mà sử dụng các thư viện truyền tin phổ biến sử dụng trên các hệ thống song song và phân bố. Cụ thể trong hai thư viện được sử dụng là POP-C++ (một ngôn ngữ dùng cho lập trình song song trên lưới kế thừa từ C++) và MPI. POP-C++ được đưa vào do một lý do lịch sử liên quan đến hạ tầng tính toán lưới (trong đề tài EDAGrid, và VNGrid), tuy nhiên thư viện này không được thông dụng lắm trong cộng đồng tính toán song song. Và đó cũng là lý do nhóm nghiên cứu đã thực hiện đưa MPI vào trong thư

viện. Với các thư viện lập trình song song khác nhau thì có các cách khác nhau để khởi động tính toán và truyền thông điệp.

4.2 TẦNG WORKFLOW ENGINE

Tầng này điều khiển thứ tự thực hiện của các task. Nó kiểm tra tình trạng dữ liệu trên mỗi task, và sẽ thực hiện chúng nếu tất cả dữ liệu đầy đủ. Workflow Engine giám sát quá trình thực hiện của các task cho đến khi kết thúc. Các công việc (task) là đối tượng thuộc các lớp tìm kiếm tương ứng. Có thể mô tả cơ chế hoạt động quy trình này như sau: Sau khi người sử dụng đăng ký ứng dụng với workflow engine, workflow engine sẽ khởi tạo ra quy trình tối ưu.

Với mô hình master – slave, master sẽ quản lý và điều khiển toàn bộ quá trình thực thi quy trình vừa tạo. Khi một task sẵn sàng, master sẽ chỉ định một slave thực hiện tính toán với dữ liệu đầu vào. Master sẽ truyền dữ liệu đến slave đối tượng có kiểu thuật toán tìm kiếm task và dữ liệu đầu vào, đồng thời master cũng truyền cho slave các đối tượng tham biến phục vụ cho tính toán tại slave. Trong quá trình tính toán với dữ liệu đầu vào, slave sử dụng các tham biến cần tham chiếu dữ liệu được lưu trữ tại master, slave và master sẽ trao đổi dữ liệu để phục vụ quá trình tính toán tại slave. Slave tính toán xong, dữ liệu được tính toán sẽ được trả về master, slave sẽ giải phóng tất cả bộ nhớ sẵn sàng nhận nhiệm vụ tính toán tiếp theo. Vì vậy với các lớp ứng dụng trên quy trình, cần có các hàm khởi tạo, hàm hủy.

Sau khi task thực hiện xong, các task khác được chuyển sang trạng thái sẵn sàng vì đã có đầy đủ dữ liệu. Nếu số bộ xử lý tham gia tính toán lớn hơn 1, master sẽ chỉ định các slave tham gia xử lý một cách song song. Quá trình trao đổi dữ liệu giữa master – slave và tính toán tại các slave song cũng tương tự như thực hiện tính toán ở các task tuần tự, tuy nhiên master sẽ quản lý và điều khiển từng task riêng biệt.

4.2.1 Cấu trúc của workflow

Một workflow chính là một đồ thị đơn có hướng không vòng (DAG – Directed Acyclic Graph), định nghĩa như sau:

$$DAG = (V, A)$$

Với *V* là tập các thuật toán tìm kiếm dạng meta-heuristic; *A* là tập các kết nối đầu ra/đầu vào của các meta-heuristic. Hai meta-heuristic được nối với nhau nghĩa là kết quả của meta-heuristic nguồn sẽ dùng làm nghiệm khởi động cho meta-heuristic đích. Trong hiện thực việc thiết lập DAG sẽ do các đối tượng kế thừa từ giao tiếp của `edaWrapperControl` thực hiện, người dùng khai báo các đối tượng trên và đưa các thông tin của đồ thị vào trình điều khiển `edaWrapperControl` qua các phương thức được hỗ trợ sẵn trong nó.

```
class edaWrapperControl
{
  public:
    edaWrapperControl();
    virtual ~edaWrapperControl();
    virtual int insertVertex(edaSearch *sa);
    virtual int insertVertex(edaSearch *sa,
                        const edaSelectionWrapper &slect);
    virtual int insertEdge(const int from, const int to);
    virtual int insertLoop(const int from, const int to,
                           const edaContinue &con);
    virtual bool search(edaSolutionList &list) = 0;

  protected:
    virtual vector<int> findReadyTask() const;
    virtual int checkTaskStatus( int taskID);
    virtual int checkLoopStatus( int taskID);
    virtual edaSolutionList* chooseSolution( int taskID,
                                             edaSolutionList &list ) ;
    bool allDone();
    map<int, edaSolutionList*> taskSolutionList;
    map<int, edaSelectionWrapper *> taskSlect;
    map<int, edaContinue*> loopCond;
    edaDAG<edaSearch*> taskDAG;
    map<int, int> taskStatus;
    map<int, int> loopStatus;

    int edgeNum;
    int vertexNum;
    int loopNum;
};
```

Các phương thức hỗ trợ bởi Workflow bao gồm: `insertVertex()`, `insertEdge()` và `insertLoop()`. Đối tượng được đưa vào hàm `insertVertex()` phải thuộc giao tiếp của `edaSearch` – tác vụ cần tối ưu. Ngoài ra người dùng còn có thể đưa vào các điều kiện chọn lựa lời giải. Lời giải được tổ chức trong thư viện là một danh sách các nghiệm khả kiến và các tác vụ sẽ làm việc trên danh sách các nghiệm khả kiến này. Với điều kiện chọn lựa lời giải người dùng có thể chọn một hoặc một nhóm các lời giải (tùy theo chiến lựa lựa chọn được thiết lập bởi người dùng) có tiềm năng để

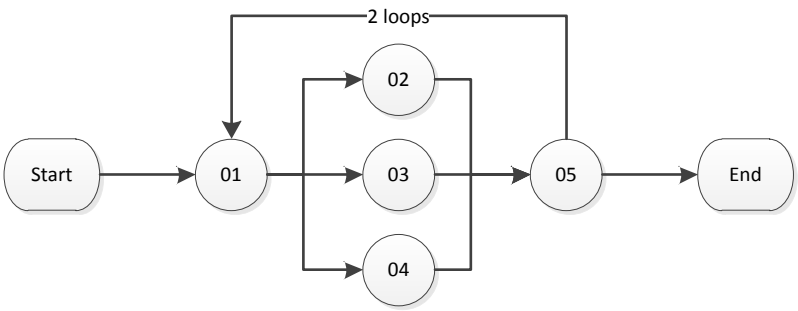
tiếp tục cải tiến bằng các giải thuật tối ưu khác. Tất nhiên người dùng vẫn có thể chọn lựa hết tất cả các lời giải khả kiến trong danh sách lời giải thông qua cơ chế lựa chọn `edaFullSelectWrapper` được hiện thực sẵn trong thư viện (đây cũng là cơ chế chọn lựa mặc định của thư viện nếu người dùng không có bất cứ tùy biến nào) hoặc tuyển và chọn lấy lời giải tốt nhất trong danh sách bởi cơ chế `edaBestSelectWrapper`.

Một trong những mục tiêu chính của đề tài là bổ sung chức năng lặp vào Workflow và phương thức `interLoop()` là hiện thực của khả năng năng này. Vòng lặp (loop) được lưu trữ trong đồ thị là một cạnh ngược hướng với đồ thị DAG cùng với điều kiện lặp được thực hiện bằng cách kế thừa từ giao tiếp của `edaContinue`. Điều kiện để vòng lặp khả dụng là các phần tử (các đỉnh) được lặp phải thuộc cùng một ***cluster***.

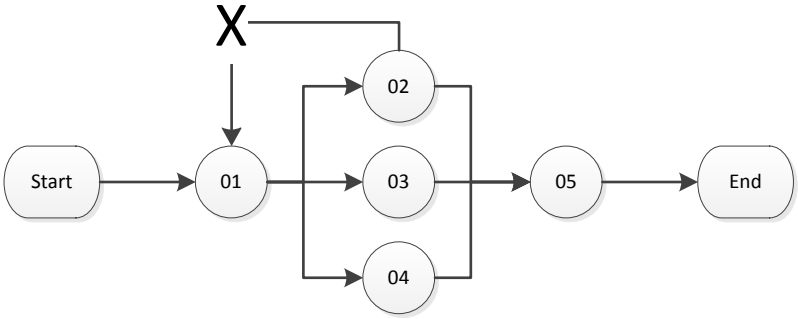
Khái niệm ***cluster*** được định nghĩa ở đây là một đồ thị con chứa một số các đỉnh của đồ thị DAG, sao cho các đỉnh trong cluster điều phải liên thông với nhau. Điều kiện để vòng lặp tạo thành cluster là:

- Cấp (level) của đỉnh đầu ra của vòng lặp phải là cấp lớn nhất và duy nhất trong cluster tính từ vị trí của đỉnh đầu vào.
- Cấp được xác định số các đỉnh đi qua nhiều nhất giữa đỉnh đầu vào đến vị trí cần xét.

Những yêu cầu trên đã bảo quá trình lập tương đối kép kín chỉ với duy nhất một đỉnh đầu vào và một đỉnh đầu ra tuy nhiên vẫn đảm bảo khả năng thực hiện được các vòng lặp lồng nhau cũng như việc thực hiện đồng thời các nhánh song song trong cluster. Trong hình 4.2 biểu diễn workflow với 5 tác vụ trong đó có 3 tác vụ có thể thực hiện song song. Một vòng lặp nối liền từ tác vụ 5 tới tác vụ 1 tạo thành cluster bên trong nó. Với workflow ở hình 4.3 ta thấy điều kiện lặp bị vi phạm do, các phần tử bên trong vòng lặp không thể tạo thành cluster do điều kiện liên thông giữa các phần tử không được thỏa.



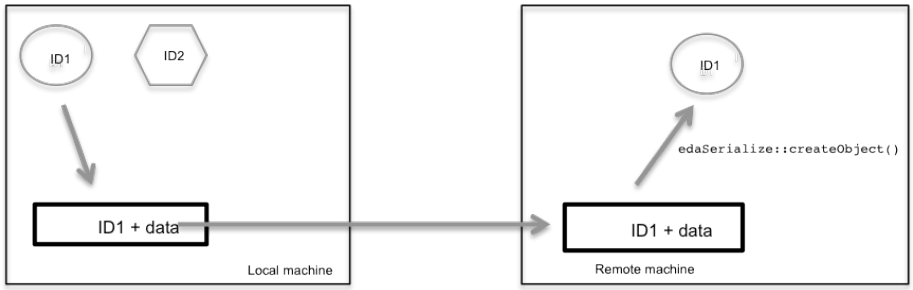
Hình 12. Workflow tối ưu với cấu trúc rẽ nhánh và vòng.



Hình 13. Workflow tối ưu với cạnh vòng vi phạm điều kiện lặp.

4.2.2 Các lớp đối tượng thích nghi dữ liệu

- `edaSerialize`: trong phần trên, người đọc đã thấy các đối tượng cơ bản đều kế thừa từ đối tượng này. Đây là đối tượng hỗ trợ đóng/mở gói dữ liệu và do đó các đối tượng nào mà người dùng muốn truyền qua máy khác đều phải kế thừa. Một việc quan trọng nữa của `edaSerialize` là giúp tái tạo lại đối tượng theo cơ chế như hình bên dưới:



Hình 14. Cơ chế tái tạo đối tượng dựa vào ID.

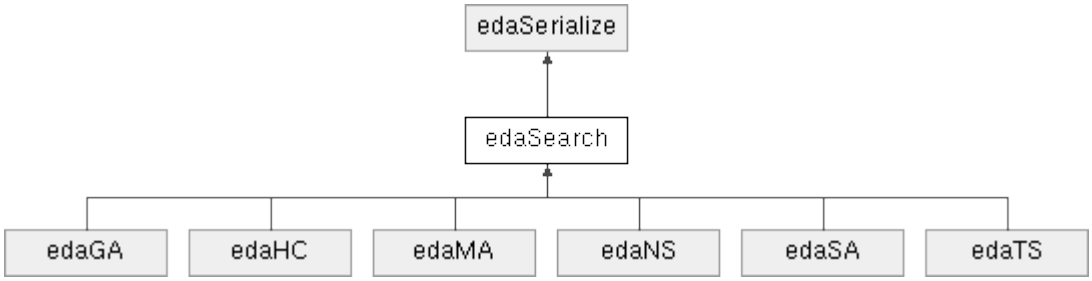
- *`edaBuffer`*: đây là gói dữ liệu được đóng gói ở tầng ứng dụng và sẽ được truyền qua lại giữa các máy.

4.3 TẦNG SEARCH ALGORITHM

Tầng Search Algorithm là nơi cung cấp một số thuật toán tìm kiếm meta-heuristic cơ bản. Các thuật toán này làm các khung nền, và người sử dụng phải định nghĩa các bài toán theo các khung nền để thực hiện. Tại tầng này, các đối tượng thuộc nó được chia ra thành các nhóm nhỏ hơn bao gồm:

- Nhóm các lớp giải thuật tìm kiếm bao gồm một loạt các giải thuật tìm kiếm mà thư viện hỗ trợ như HC, SA, TS, GA và MA. Chúng được kế thừa từ lớp edaSearch như trong hình 4.5 và đưa phân thành 2 nhóm nhỏ hơn bao gồm các giải thuật tối ưu thuộc nhóm dịch chuyển và các giải thuật tối ưu thuộc nhóm quần thể. Người sử dụng cũng có thể đưa vào thư viện một giải thuật tìm kiếm nào đó tại tầng ứng dụng và nó cũng cần phải được kế thừa từ lớp `edaSearch`.
- Nhóm các lớp liên quan đến bài toán và lời giải: Các lớp này ***buộc phải xây dựng*** bởi người sử dụng và ***phải kế thừa*** từ lớp `edaProblem` và `edaSolution`. Các lớp kế thừa từ `edaProblem` sẽ chứa các thông tin (phát biểu) của bài toán, thông tin này có thể được dùng để kiểm tra tính hợp lệ của lời giải (các điều kiện ràng buộc của bài toán) và hỗ trợ tính toán hàm thích nghi hoặc hàm mục tiêu (`evaluate()` hoặc giá trị thích nghi). Đối với các đối tượng kế thừa từ lớp `edaSolution`, chúng thực hiện việc mô tả lời giải nào là khả thi (feasible) và làm như thế nào để đánh giá được chúng. Việc tổ chức cấu trúc dữ liệu bên trong các lớp này hoàn toàn do người dùng tự quyết định và chỉ có người dùng mới có biết được việc tổ chức dữ liệu như thế nào để phù hợp với loại bài toán tối ưu mà họ đang quan tâm.
- Nhóm các lớp liên quan đến lân cận và cách thức tạo lân cận trong nhóm giải thuật dựa trên dịch chuyển: Các lớp này hỗ trợ việc thực hiện các bước chuyển trong các giải thuật tìm kiếm meta-heuristic sử dụng.
- Nhóm các lớp phụ trợ và toán tử trong giải thuật dựa trên quần thể: Khi hiện thực thư viện, nhóm nghiên cứu đã ***tách rời các giai đoạn trong các giải thuật dựa trên quần thể thành các toán tử riêng biệt***. Cách thức tổ chức

này sẽ rất tiện lợi, khi người sử dụng muốn tái sử dụng các thành phần và xây dựng một giải thuật tìm kiếm mới.

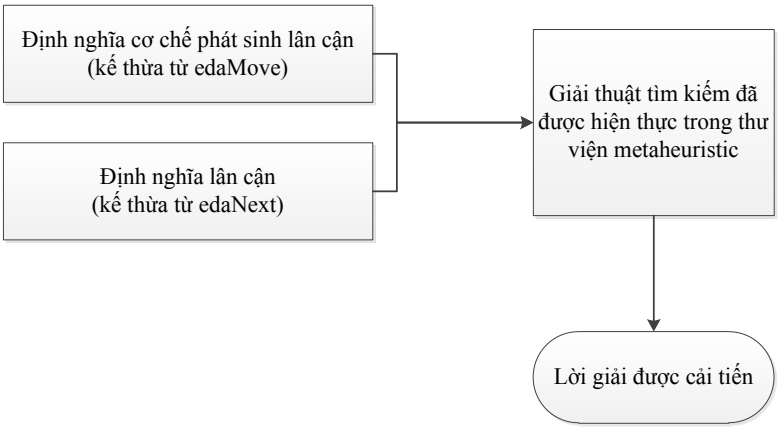


Hình 15. Các lớp giải thuật được hỗ trợ trong hệ thống thư viện.

4.3.1 Các giải thuật dựa trên dịch chuyển

Các phương pháp meta-heuristic là các phương pháp giải bài toán một cách gần đúng được sử dụng trong các bài toán tìm lời giải tối ưu có độ phức tạp cao. Thông thường có hai dạng chính trong meta-heuristic là *dịch chuyển* và *quần thể*. Trong dạng tìm kiếm gần đúng này, chất lượng nghiệm của bài toán sẽ phụ thuộc hoàn toàn vào cách tìm kiếm nghiệm trong không gian nghiệm như thế nào. Đối với nhóm giải thuật dựa trên dịch chuyển, đó chính là chiến lược chọn bước chuyển trong không gian tìm kiếm và sau đây chúng lần lược khảo sát qua các giải thuật thuộc nhóm này.

4.3.1.1 Giải thuật HC



Hình 16. Sơ đồ khối minh họa quá trình cải tiến lời giải ban đầu bằng các hàm sẵn có trong thư viện meta-heuristic.

Giải thuật HC chính là nền tảng cơ sở của các kỹ thuật tìm kiếm cục bộ. Mặc dù đây là giải thuật đơn giản nhưng nó lại rất mạnh và hiệu quả trong việc giải quyết các bài toán tối ưu tổ hợp lớn. Thuật ngữ "leo đồi" xuất phát từ cơ chế tham lam: ở mỗi bước của việc tìm kiếm, chúng ta sẽ chọn một lân cận mà nó cải thiện giá trị hàm mục tiêu để thực hiện tìm kiếm tiếp. Thông tin chi tiết của HC đã được mô tả ở trên. Trong phần này, báo cáo chỉ trình bày phần hiện thực liên quan đến thư viện.

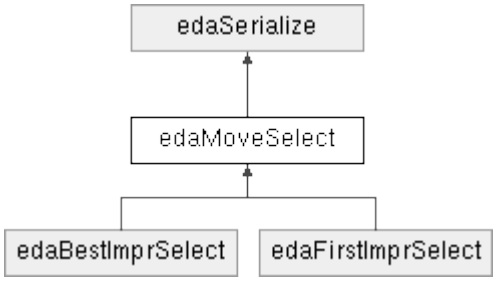
Trong hiện thực của thư viện, giải thuật HC nói riêng và các giải thuật meta-heuristic nói chung không chọn và thực hiện duy nhất một lời giải, thay vào đó giải thuật sẽ làm việc trên danh sách (tập hợp) các lời giải. Nhiệm vụ của người dùng cần phải tự xây dựng tập các lời giải khả thi, đưa chúng vào danh sách các lời giải (lớp `edaSolutionList`).

Để vận hành giải thuật, chúng ta cần có những lớp phụ trợ sau:

- Lân cận (`edaMove`): Người dùng buộc phải định nghĩa cấu trúc lớp lân cận cho từng bài toán cụ thể kế thừa từ giao diện `edaMove`. Cấu trúc các lân cận là khác nhau trong từng bài toán cụ thể và chỉ có người dùng mới có thể thực hiện được điều này.
- Cơ chế tạo lân cận (`edaMoveGen`): Tương tự như lớp `edaMove`, người sử dụng cần xây dựng cơ chế tạo lân cận bằng các kế thừa lớp `edaMoveGen` và điều này là bắt buộc để có thể sử dụng các giải thuật tìm kiếm dạng dịch chuyển. Thông thường, các láng giềng được tạo bởi lớp `edaMoveGen` trong giải thuật HC theo một trật tự nhất định và điều này khác với cơ chế tạo lân cận trong giải thuật SA, được trình bày bên dưới.
- Cơ chế chọn lân cận (`edaMoveSelect`): Sau khi tập các lời giải lân cận được sinh ra bằng cơ chế sinh các lời giải lân cận được lựa chọn và cần thiết có một cơ chế để chọn lời giải lân cận ứng viên cho lần tìm kiếm kế tiếp. Có 2 cơ chế để lựa chọn lời giải ứng viên giữa tập hợp các lời giải lân cận đối được hỗ trợ sẵn trong hệ thống thư viện là First Best và Global Best.

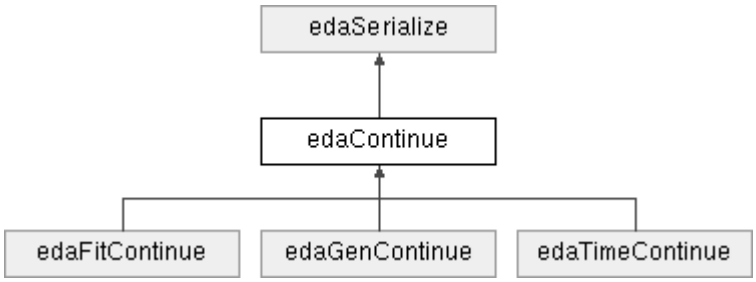
  Đối với cơ chế First Best, khi lân cận được tìm thấy có giá trị hàm mục tiêu tốt hơn lời giải hiện tại, thuật toán lập tức chuyển lời giải sang giá trị được

tìm thấy, trong khi đó với cơ chế Global Best, láng giềng được chọn phải thỏa hai yêu cầu đó là có giá trị hàm mục tiêu tốt hơn lời giải hiện tại và tốt nhất trong số tất cả láng giềng của lời giải hiện tại.



Hình 17. Các cơ chế lựa chọn lân cận trong thư viện.

- Điều kiện dừng (`edaContinue`): Trong thư viện, quá trình tìm kiếm không nhất định chỉ dừng khi tìm thấy lời giải tối ưu cục bộ. Linh hoạt hơn, người dùng có thể tự thiết lập các điều kiện dừng cho bài toán và khi các điều kiện này được thỏa, quá trình tìm kiếm sẽ lập tức được dừng lại. Hình bên dưới liệt kê 3 lớp điều kiện dừng được hỗ trợ sẵn trong hệ thống thư viện, nếu không có các yêu cầu đặc biệt người sử dụng có thể các lớp điều kiện dừng này trong việc thiết kế các giải thuật tối ưu riêng cho bài toán mình quan tâm.



Hình 18. Các kiểu điều kiện dừng được hỗ trợ trong thư viện.

Với lớp `edaGenContinue` người dùng có thể thiết lập điều kiện dừng thông qua số bước chuyển của lời giải trong không gian tìm kiếm, lớp `edaFitContinue` đưa ra điều kiện dừng thông giá trị cận dưới của hàm mục tiêu, lớp `edaTimeConitnue` thiết lập điều kiện thông qua việc giới hạn thời gian tìm kiếm lời giải trong không gian nghiệm.

4.3.1.2 Giải thuật SA

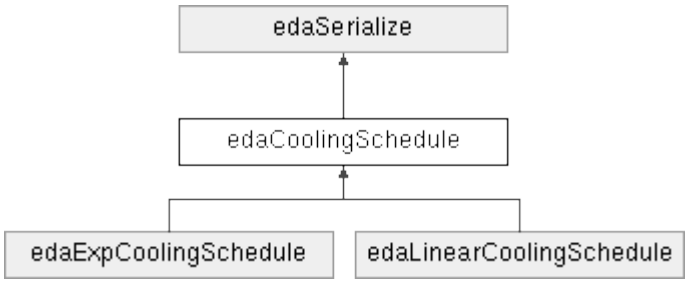
Khi sử dụng giải thuật mô phỏng luyện kim nó không tìm kiếm lời giải tốt nhất trong các lời giải lân cận của lời giải hiện tại. Thay vào đó giải thuật thực hiện việc sinh ngẫu nhiên một lời giải lân cận từ lời giải hiện tại. Nếu lời giải lân cận tốt hơn thì luôn luôn chấp nhận nó và xem nó như lời giải hiện tại mới. Nếu lời giải lân cận vừa sinh ra xấu hơn lời giải hiện tại, nó chỉ được phép chấp nhận với một xác suất xác định. Xác suất chấp nhận được xác định bằng "nhiệt độ" tuần tự giảm dần. Bên cạnh đó nó luôn luôn lưu trữ lời giải tốt nhất hiện tại nhằm đạt được kết quả tốt nhất.

Nếu ta gọi E là mức năng lượng của trạng thái S, E' là mức năng lượng của trạng thái S' và $\Delta E = E - E'$ là sự chênh lệch năng lượng giữa trạng thái S' và trạng thái S. Nếu $\Delta E \leq 0$ thì sự thay đổi trạng thái sẽ được chấp nhận với xác suất $p = \exp\left(-\frac{\Delta E}{k_B T}\right)$, trong đó k_B là hằng số Boltzmann (có giá trị $1.38.10^{-23}$ J/K trong hệ S.I.). SA sử dụng một biến điều khiển toàn cục là biến nhiệt độ T. Ban đầu T được thiết lập với giá trị cao và sau đó được giảm dần. Nhiệt độ T' ở trạng thái tiếp theo được xác định theo công thức sau:

$$T' = \frac{1}{F(T)} \exp\left(\frac{\Delta E}{k_B T}\right),$$

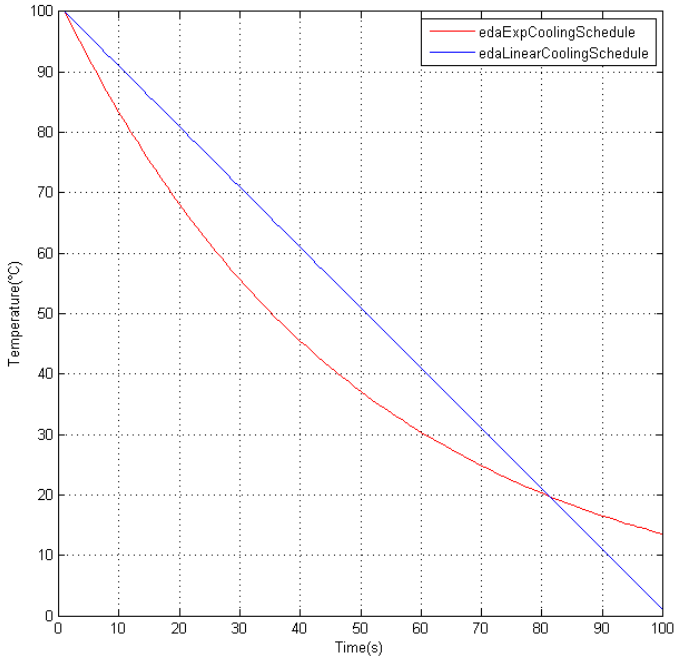
trong đó $F(T)$ là một hàm làm lạnh nhiệt độ.

Trong hiện thực của thư viện, trạng thái S chính là lời giải hiện tại và mức năng E được thiết lập chính bằng hàm mục tiêu của lời giải. Với hằng số Boltzmann, để linh hoạt, thư viện cho phép người dùng có quyền thiết lập giá trị cho chúng. Mặc định giá trị này được chọn là 1. Đối với hàm làm lạnh nhiệt độ, chúng được đưa vào SA dưới dạng các đối tượng. Tùy theo, yêu cầu của bài toán cũng như mục đích của người sử dụng mà cơ chế làm lạnh hoàn toàn có thể tự xây dựng hoặc sử dụng các lớp đã được định nghĩa sẵn trong thư viện.



Hình 19. Hai cơ chế làm lạnh nhiệt độ được hỗ trợ trong thư viện.

- Cơ chế làm lạnh (`edaCoolingSchedule`): Đối với giải thuật SA, cơ phân phối nhiệt độ giữ một vai trò quan trọng. Nhiệt độ của lời giải được cập nhật liên tục qua cơ chế làm lạnh đến khi chúng đạt được trạng thái bền vững. Trong thư viện 2 cơ chế làm lạnh được hỗ trợ là `edaLinearCoolingSchedule` và `edaExpCoolingSchedule`.



Hình 20. Các kiểu điều kiện dừng được hỗ trợ trong thư viện.

Để hiệu rõ cơ chế làm việc của hai lớp hàm làm lạnh nhiệt độ này, quan sát hình 4.10, hai lớp `edaExpCoolingSchedule` và `edaLinearCoolingSchedule` có cùng nhiệt dộ ban đầu là T = 100°C và nhiệt độ này sẽ giảm dần theo thời gian. Đối với lớp hàm `edaExpCoolingSchedule` trong hình vẽ tỷ số thay đổi nhiệt độ là 0.98, đối với hàm `edaLinearCoolingSchedule` lượng nhiệt đổi sau mỗi giây là 1°C. Theo thời gian nhiệt độ của hàm

`edaLinearCoolingSchedule` sẽ thay đổi một cách tuyến tính trong khi nhiệt độ của `edaExpCoolingSchedule` chỉ giảm mạnh khi nhiệt độ cao và dần dần giảm tốc độ giảm nhiệt khi nhiệt độ hạ thấp. Tùy theo mục tiêu của bài toán mình nghiên cứu mà người dùng có thể tùy chọn hàm làm lạnh nhiệt độ hoặc cũng có thể xây dựng hàm mới dựa trên việc kế thừa khung hàm `edaCoolingSchedule`.

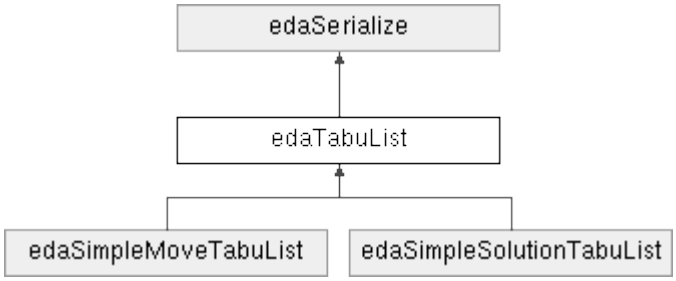
- Lân cận (`edaMove`): Tương tự như giải thuật HC người dùng buộc phải định nghĩa cấu trúc lân cận cho bài toán của mình, chi tiết cách xây dựng lân cận chúng ta có thể tham khảo tại các mục hướng dẫn tại các chương sau.
- Cơ chế tạo lân cận (`edaMoveGen`): Do đặc trưng của giải thuật SA, việc tạo ra các lân cận thường không theo một trật tự nhất định, người dùng nên xây dựng cơ chế tạo lân cận theo phương cách ngẫu nhiên.
- Điều kiện dừng (`edaContinue`): Chi tiết về điều kiện dừng người dùng có thể xem tại mục thuật toán HC.

4.3.1.3 Giải thuật TS

Đây cũng là một giải thuật này thuộc nhóm dịch chuyển, và ý tưởng trong tìm kiếm Tabu là một danh sách lưu trữ các lời giải đã qua được sử dụng để đảm bảo rằng một lời giải nào đó sẽ không phải xét lại. Trong giải thuật tìm kiếm Tabu, khác với leo đồi, một số lời giải có chi phí kém hơn với lời giải hiện tại cũng vẫn được chấp nhận chỉ cần nó không có trong danh sách các lời giải đã qua với mong muốn khắc phụ tối ưu cục bộ. Điều kiện dừng cho giải thuật Tabu là khi sau một số lần tìm kiếm đủ lớn, không thể tìm kiếm được một lời giải nào tốt hơn cho lời giải hiện tại giải thuật kết thúc và chọn lời giải tốt nhất làm lời giải cuối cùng. Trong giải thuật tìm kiếm Tabu, kích thước của danh sách chứa các lời giải đã qua rất quan trọng, ảnh hưởng đến hiệu quả của việc tìm kiếm. Chiều dài của danh sách Tabu phải đủ lớn để đảm bảo thoát ra được vùng tối ưu cục bộ nhưng cũng phải đủ ngắn để tránh bỏ qua các lời giải có khả năng. Đây là tham số ảnh hưởng đến hiệu quả của việc tìm kiếm và được thiết lập bởi người lập trình, vì vậy đối với mỗi bài toán cụ thể chúng ta cần trải qua thực nghiệm mới có thể đưa ra thông số tốt nhất cho

giải thuật. Phần tiếp theo, chúng ta sẽ khảo sát qua các lớp dối tượng (tham số) đầu vào cho giải thuật TS:

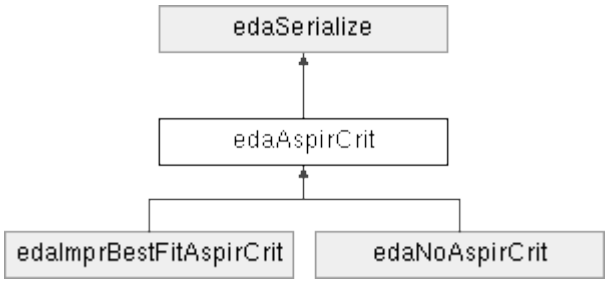
- Danh sách Tabu (`edaTabuList`): được tổ chức theo cấu trúc của hàng đợi First In First Out (FIFO) với kích thước của hàng đợi được xác định trước. Khi danh sách Tabu đầy, lời giải cuối danh sách (được chèn vào danh sách sớm nhất) sẽ được xóa khỏi danh sách và lời giải hiện tại sẽ được chèn vào đầu danh sách.



Hình 21. Các loại danh sách Tabu được hỗ trợ trong thư viện.

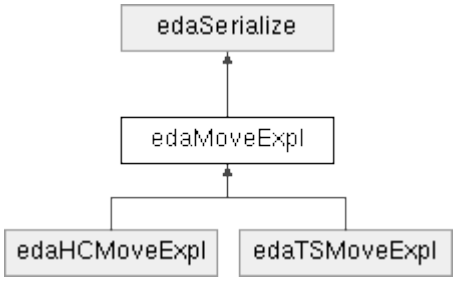
Được hỗ trợ sẵn trong thư viện là hai dạng danh sách Tabu cơ bản `edaSimpleMoveTabuList` và `edaSolutionMoveTabuList`. Đối với dạng danh sách Tabu thứ nhất đối tượng được chứa trong danh sách là các bước chuyển (move) với dạng thứ hai đối tượng được chứa lại là các lời giải (nghiệm) khả thi trong không gian tìm kiếm. Để xây dựng một loại danh sách Tabu mới người dùng phải kế thừa chúng từ lớp `edaTabuList`.

- Tiêu chí tuyển chọn (`edaAspirCrit`): Đây cũng là một thành phần không thể thiếu đối với giải thuật TS. Các lớp kế thừa từ `edaAspirCrit` giữ vai trò tuyển chọn các phần tử trong danh sách Tabu. Với các chiến lược tuyển chọn khác nhau ta sẽ xây dựng được các lớp đối tượng khác nhau. Quan sát lớp `edaImprBestAspriCrit`, ta sẽ nhận thấy chiến lược vẫn cho phép lựa chọn một phần tử đã nằm trong danh sách cấm Tabu nếu bước chuyển (hoặc lời giải) này thỏa tiêu chí có hàm lượng giá tốt nhất. Ngược lại, với chiến lược `edaNoAspirCrit` các bước chuyển hoặc lời giải đã chứa trong danh sách Tabu sẽ hoàn toàn bị ngăn cấm.



Hình 22. Các tiêu chí tuyển chọn thực hiện bước chuyển cho các đối tượng nằm trong danh sách Tabu.

- Cơ chế tạo lân cận (`edaMoveGen`) và điều kiện dừng (`edaContinue`): Đối hai lớp đối tượng này chúng ta đã được giới thiệu tại giải thuật HC và chi tiết xây dựng chúng như thế nào sẽ được trình bày ở các mục sau. Tuy nhiên, một lưu ý đối với người sử dụng thư viện đó là nếu không có nhu cầu đặc biệt chúng ta có thể tái sử dụng các lớp đã được định nghĩa sẵn trước đó hoặc sữ dụng trực tiếp các lớp hỗ trợ trong thư viện.



Hình 23. Các cơ chế khai phá không gian được hỗ trợ trong thư viện.

4.3.2 Các giải thuật dựa trên quần thể

Đây là các thuật toán tối ưu mà hoạt động của nó không cần phải tính toán giá trị đạo hàm. Thay vào đó, các giải thuật thuộc nhóm này dựa trên sự tồn tại của một quần thể và khả năng *tiến hóa*. Hai khái niệm quần thể và tiến hóa chính là đại điện cho lớp giải thuật này, trong đó việc lựa chọn các biến thiết kế và quá trình cải tiến lời giải là rất đặc thù với từng ứng dụng. Phương pháp này đối nghịch với các phương pháp tất định. Do sử dụng các thành phần ngẫu nhiên nên giải thuật tiến hóa có thể tránh bị kẹt ở các vùng tối ưu cục bộ thay vào đó phương pháp có thể tìm kiếm toàn cục trong không gian thiết kế.

Các ưu điểm của giải thuật dựa trên quần thể bao gồm:

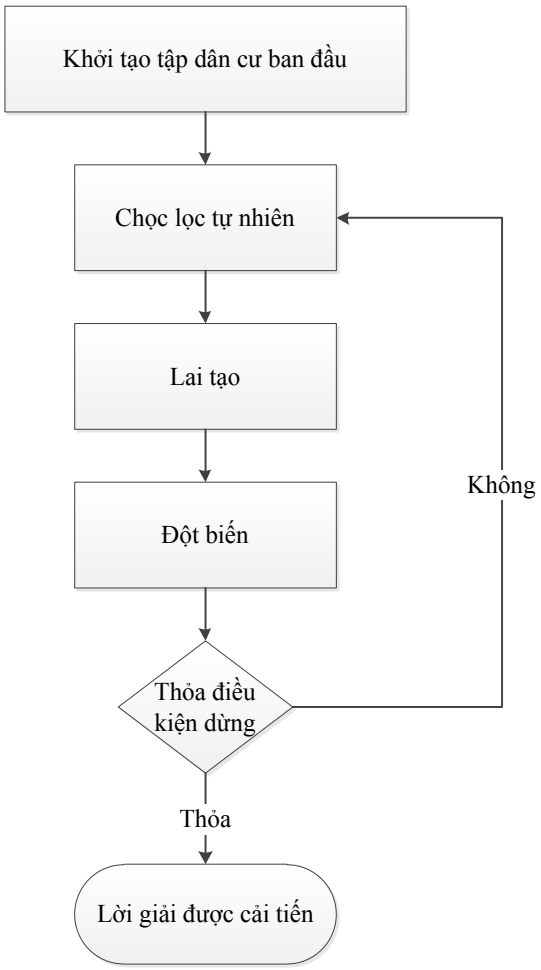
- Có khả năng tìm kiếm cho trên cả miền không gian liên tục hoặc rời rạc. Trong một số trường hợp, khi quy trình tham số hóa biên dạng cánh là khác nhau, thuật toán tỏ ra khá hữu hiệu và mềm dẻo. Thuật toán tỏ ra bền vững và hiệu quả cho các bài toán với không gian tìm kiếm có số chiều lớn.
- Không cần các thông tin về đạo hàm mà thông thường rất khó và chi phí rất lớn để tính được thông tin ở nhiều bậc khác nhau.
- Có khả năng khai phá đồng thời trên nhiều vùng của không gian tìm kiếm. Về mặt toán học điều này có nghĩa là nhóm thuật toán này có khả năng tránh bị kẹt vào các vị trí tối ưu cục bộ trong những không gian có bề mặt phức tạp. Về mặt kỹ thuật thuật toán rất thích hợp để bố trí làm việc trên các cụm máy tính song song.
- Cung cấp một danh sách các biến tối ưu, không phải duy nhất một biến như các giải thuật dựa trên nền tảng gradient.

Giống như các giải thuật tối ưu khác, các thuật toán dựa trên quần thể cần định nghĩa các biến thiết kế và hàm mục tiêu. Khi kết thúc, nó cũng hội tụ về kết quả như các thuật toán tối ưu khác. Tuy nhiên, các thành phần của thuật toán nhóm này lại rất khác biệt tùy theo mục đích sử dụng.

4.3.2.1 Giải thuật GA

Giải thuật này thuộc nhóm dựa trên quần thể, lấy ý tưởng từ nguyên lý tiến hóa di truyền trong tự nhiên. Giải thuật cần một cộng đồng với một số dân cư ban đầu cụ thể, sau đó tại mỗi vòng lặp, một số hoạt động di truyền như lai ghép, chọn lọc, đột biến được áp vào các dân cư hiện tại để sinh ra những cư dân hay những cá thể mới cho thế hệ tiếp theo. Sau đó, các cá thể được đưa qua một hàm mục tiêu để đo lường và chỉ những cá thể nào đáp ứng đủ tiêu phát hiện mới được giữ lại trong cộng đồng. Giải thuật cho phép một tập dân cư bao gồm nhiều cá thể được phát triển theo quy luật chọn lọc xác định và hướng đến trạng thái cực đại hóa hàm mục tiêu và cứ thế tiếp tục cho đến khi gặp phải điều kiện dừng hoặc một kết quả tốt như mong muốn.

Phương pháp này được phát triển bởi John Holland (1975) và các cộng sự của mình từ thập niên 60 đến 70. David Goldberg (1989), một trong những học trò của ông, đã ứng dụng thành công GA để giải bài toán khó liên quan đến việc điều kiển truyền tải trong đường ống khí đốt và ông cũng là người giúp thuật toán GA phổ biến trên khắp thế giới. Kể từ đó, nhiều phiên bản của GA được thử nghiệm với một mức độ thành công nhất định.



Hình 24. Sơ đồ khối của giải thuật GA.

Như được trình bày ở trên, giải thuật di truyền có những ưu điểm sau:

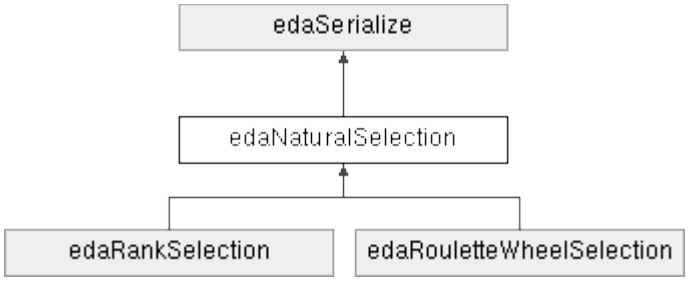
- Tối ưu hóa cho trên cả miền liên tục hoặc rời rạc.
- Không cần các thông tin về đạo hàm.
- Tìm kiếm đồng thời trên bề mặt chi phí.

- Làm việc với số lượng lớn các biến.
- Thích ứng tốt với hệ thống máy tính song song.
- Tối ưu hóa các biến trên các bề mặt phức tạp.
- Cung cấp một danh sách các biến tối ưu, không phải duy nhất một biến.

Với nhiều ưu điểm nổi trội, trong nhiều bài toán GA có thể hội tụ về các kết quả tốt trong khi phương pháp tối ưu truyền thống đi vào bế tắt. Tuy nhiên việc hiện thực thuật toán GA cho từng bài toán cụ thể không phải là công việc đơn giản bởi tính phức tạp của các thành phần có trong nó.

- Gen (`edaGenne`): Đây là thành phần cơ bản cấu thành nên nhiễm sắc thể, tùy theo đối tượng của bài toán tối ưu là tổ hợp hay liên tục mà gen được định nghĩa theo những cách khác nhau. Người dùng cần xác định miền trị cho lớp gen được định nghĩa cho bài toán của mình.
- Nhiễm sắc thể (`edaChromosome`): Trong thư viện, lớp nhiễm sắc thể được hiện thực dưới dạng một danh sách có thứ tự các gen. Nhiễm sắc thể ở đây sẽ đóng vai trò là toán hạng của các toán tử lai chéo và đột biến.
- Tập dân cư (`edaPopulation`): Thông thường GA bắt đầu với một nhóm các nhiễm sắc thể được biết như một tập dân cư. Một tập dân cư có với M nhiễm sắc thể sẽ có thể biểu diễn bằng một ma trận $M \times N$ biến (N là số lượng gen của mỗi nhiễm sắc thể) và thường thì các biến này được khởi tạo một cách ngẫu nhiên. Trong hiện thực của thư viện, tập dân cư là kết quả của việc mã hóa danh sách các lời giải được đưa vào giải thuật. Khi kết thúc quá trình tối ưu, lớp `edaRepresentation` sẽ có nhiệm vụ chuyển đổi tập dân cư trở thành danh sách lời giải và trã chúng về cho người dùng. Nói một cách khác, khi người dùng đã đặc tả chính xác cách thức mã hóa và giải mã cho nhiễm sách thể, việc xây tập dân cư và đánh giá chất lượng cho từng các thể sẽ được điều khiển bởi giải thuật và hoàn toàn tự động.
- Phương pháp mã hóa (`edaRepresentation`): Một thông tin đầu vào quan trọng khác trong giải thuật GA thuộc hệ thống thư viện là phương pháp mã hóa. Để có được các gen, nhiễm sắc thể và tập dân cư, người sử dụng thư viện phải chỉ rõ cách thức ánh xạ lời giải thành nhiễm sắc thể và ngược lại.

- Quá trình chọn lọc tự nhiên (`edaNaturalSelection`): Lớp đối tượng này có nhiệm vụ loại bỏ các cá thể không phù hợp trong GA. Nhiều phương pháp chọn lọc đã được phát triển trong GA. Trong hiện thực, một tỷ lệ được đề ra để xác định bao nhiêu phần trăm cá thể được giử lại sau mỗi thế hệ trong GA. Ngoài ra một cách tiếp cận khác trong chọn lọc tự nhiên là chọn ngưỡng. Trong cách tiếp cận này tất cả các nhiễm sắc thể được giữ lại phải có chi phí lớn hơn ngưỡng được chọn. Điểm nổi bật của phương pháp này là không cần phải tiến hành sắp xếp các cá thể trên tập dân cư.



Hình 25. Các tiêu chí chọn lọc tự nhiên trong thư viện.

Hiện tại, chỉ có hai lớp đối tượng cơ bản được hỗ trợ trong thư viện là `edaRankSelection` và `edaRouletteWheelSelection`. Đối với chọn lọc theo thứ hạng (`edaRankSelection`), tập dân cư sẽ được sắp sếp theo thứ tự của hàm mục tiêu và ứng với mỗi thứ hạng sẽ được gắn một con số xác xuất tương ứng theo công thức sau:

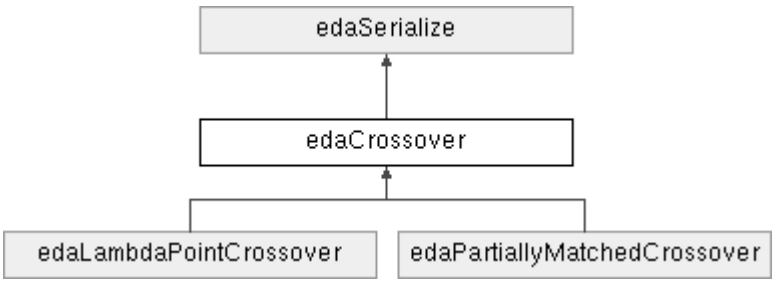
$$P_n = \frac{N_{keep} - n + 1}{\sum_{n=1}^{N_{keep}} n}$$

Trong đó: N_{keep} là số cá thể được giữ lại sau quá trình chọn lọc tự nhiên, n là thứ hạng của cá thể trong tập dân cư. Tương tự như chọn lọc theo thứ hạng, chọn lọc theo phương pháp quay số (`edaRouletteWheelSelection`) cần một tập dân cư đã được sắp xếp theo thứ tự hàm mục tiêu. Tuy nhiên xác xuất của mỗi nhiễm sắc thể không là một con số cố định theo từng thứ hạng mà phụ thuộc vào hàm mục tiêu, theo công thức sau:

$$P_n = \left| \frac{f_n - f_{N_{keep+1}}}{\sum_{m}^{N_{keep}} f_m - f_{N_{keep+1}}} \right|$$

Với f_n là giá trị hàm mục tiêu của cá thể có thứ hạng n.

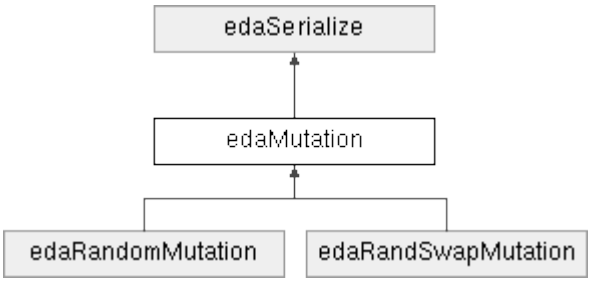
- Quá trình lai chéo (`edaCrossover`): Toán tử này sẽ tạo ra các nhiễm sắc thể mới bằng cách hoán đổi các gen giữa hai hay nhiều bộ nhiễm sắc thể và thay thế các nhiễm sắc thể bị loại bỏ trong quá trình chọn lọc tự nhiên. Đối với lớp `edaLambdaPointCrossover`, hai nhiễm sắc thể ban đầu sẽ bị phân chia thành một số đoạn nhất định (phụ thuộc vào số điểm Lambda) và thực hiện trao đổi giữa các đoạn này. Với lớp `edaPartiallyMatchedCrossover`, chúng được thiết kế cho việc trao đổi chéo ở các bài toán mà lời giải mang tính chất chu trình Hamilton (tối ưu tổ hợp), vì vậy ngoài cơ chế trao đổi chéo các gen như toán tử `edaLambdaPointCrossover` nó còn có pha cập nhật lại thông tin chuỗi nhiễm sắc thể nhằm khôi phục lại tính chất của chu trình Hamilton. Hai toán tử lai chéo thuộc thư viện này chỉ hỗ trợ cơ chế trao đổi chéo giữa hai nhiễm sắc thể, nếu muốn người dùng vẫn có thể định nghĩa chiến lược lai chéo mới.



Hình 26. Các cơ chế lai chéo được hỗ trợ trong thư viện.

- Quá trình đột biến (`edaMutation`): Đây là thành phần giúp GA có khả năng khai phá những vùng không gian chưa được xem xét. Quy luật chi phối của quá trình này hoàn toàn ngẫu nhiên. Trong tập dân cư chỉ có một số rất ít các cá thể chịu sự chi phối của quá trình đột biến vì nếu tỷ lệ đột biến lớn sẽ làm chậm quá trình hội tụ của tập dân cư. Toán tử đột biến được xây dựng trong nghiên cứu là đột biến cận (`edaBoundaryMutation`) và đột biến hoán đổi

(`edaRandSwapMutation`). Với chiến lược đột biến cận trong đó giá trị của gen được chọn sẽ được thay thế bằng một giá trị ngẫu nhiên nằm trong khoảng từ 0 đến 1. Với `edaRandSwapMutation`, thì hai gen bất kỳ trong nhiễm sắc thể sẽ được hoán đổi với nhau. Để khả năng đột biến hiệu quả và đa dạng hơn người sử dụng có thể tự xây dựng hàm đột biến riêng cho bài toán của mình thông qua việc định nghĩa các lớp đột biến mới (kế thừa từ lớp `edaMutation`) tại tầng ứng dụng.



Hình 27. Các cơ chế đột biến được hỗ trợ trong thư viện.

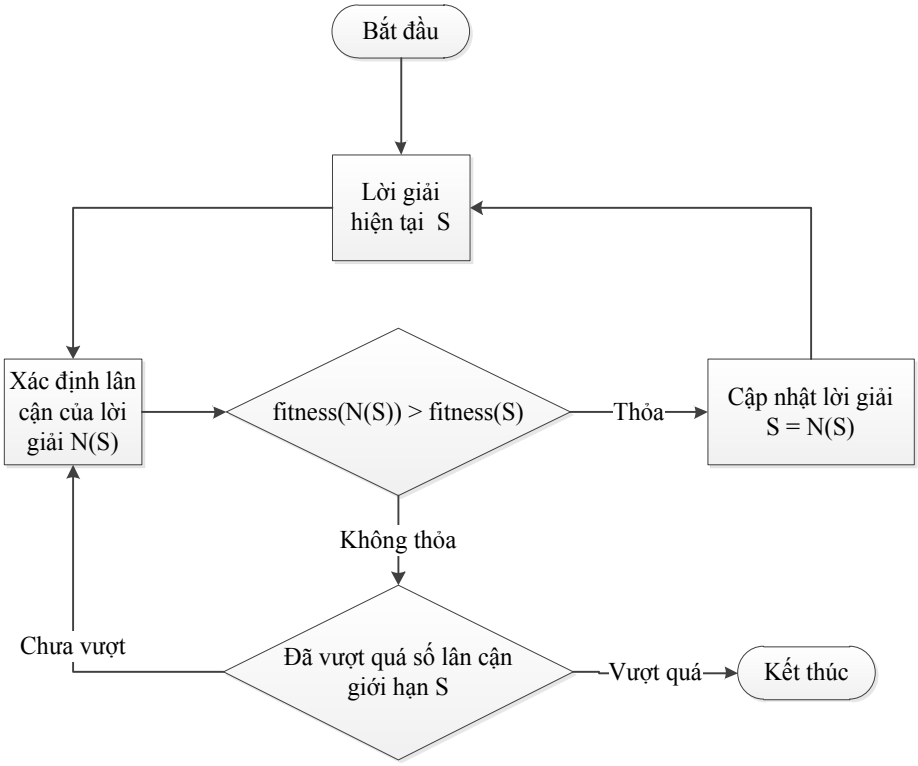
- Điều kiện dừng (`edaContinue`): Chi tiết về hiện thực điều kiện dừng cho các thuật toán dựa trên quần thể người dùng có thể xem tại mục thuật toán HC.

Báo cáo này không thể đưa hết các kiến thức về GA cũng như các thuật toán tiến hóa dạng sinh học. Người đọc có thể tham khảo thêm các tài liệu như [Yang13, Thuc01] để có thêm thông tin chi tiết.

4.3.2.2 Giải thuật MA

Thuật toán Memetic được đề xuất bởi Moscato và Norman năm 1992, trong đó các giải thuật tìm kiếm cục bộ (thông thường là giải thuật leo đồi) được lồng ghép trong quá trình tiến hóa trong giải thuật GA nhằm cải tiến chất lượng của tập dân cư qua từng thế hệ với hy vọng sẽ đạt được lời giải có hiệu suất cao hơn. Về mặt kỹ thuật giải thuật MA là cải tiến của GA trong đó sau pha đột biến, các nhiễm sắc thể được trải qua quá trình cải tiến bằng các thuật toán tìm kiếm cục bộ trước khi bước qua giai đoạn chọn lọc tự nhiên ở thế hệ tiếp theo. Về mặt tư tưởng, quá trình cải tiến (hoặc thích nghi – adaptation) là giai đoạn phát triển về mặt xã hội bên cạnh giai đoạn tiến hóa sinh học của các cá thể.

Thực tế cho thấy, yếu tố khai phá trong giải thuật GA là rất mạnh, giải thuật là một quá trình liên kết của các cách tiếp cận ngẫu nhiên bao gồm quá trình khởi tạo tập dân cư ban đầu, quá trình ghép cặp và quá trình đột biến, đó chính là những công cụ giúp GA khai phá không gian tìm kiếm. Kết quả của quá trình khai phá nếu có chất lượng tốt sẽ được giữ lại trong quần thể. Tuy nhiên sự khắt khe của quá trình chọn lọc tự nhiên có thể buộc các cá thể có tiềm năng chưa được khai thác bị đã đào thải ngay sau khi chúng được tìm ra. Trong khi đó về mặt này giải thuật dựa trên dịch chuyển lại thực hiện tốt hơn hẳn, một lời giải được sinh ra tiếp tục được khai thác cho đến không thể nào cải tiến được nữa và khi đó quá trình mới dừng lại. Việc kết hợp hai giải thuật này trong giải thuật Memetic sẽ mang lại cơ hội cho các phần tử được khai phá có điều kiện tốt hơn để phát triển trước khi bước vào giai đoạn chọn lọc tự nhiên tiếp theo.



Hình 28. Quy trình thực hiện cơ chế cải tiến lời giải theo MA.

Điều này cũng đúng với thực tế cuộc sống chúng ta, nếu hàm mục tiêu trong cuộc sống là vị trí của một người trong xã hội và nếu quá trình tiến hóa chỉ đơn giản là dựa vào đánh giá vị trí ban đầu khi một người được sinh ra thì chọn lọc tự nhiên sẽ đào thải rất nhiều cá nhân xuất sắc. Nhiều nhà khoa học lớn của nhân loại sẽ nằm

trong nhóm bị đào thải này, bởi lẽ ngay từ đầu họ không được sinh ra trong các gia đình có vị trí xã hội tốt, vị trí mà họ có được là do quá trình học tập và lao động. Như vậy các giải thuật dựa trên dịch chuyển sẽ đóng vai trò như quá trình lao động và học tập trong thực tế giúp mỗi cá thể có cơ hội để phát triển hoàn thiện bản thân.

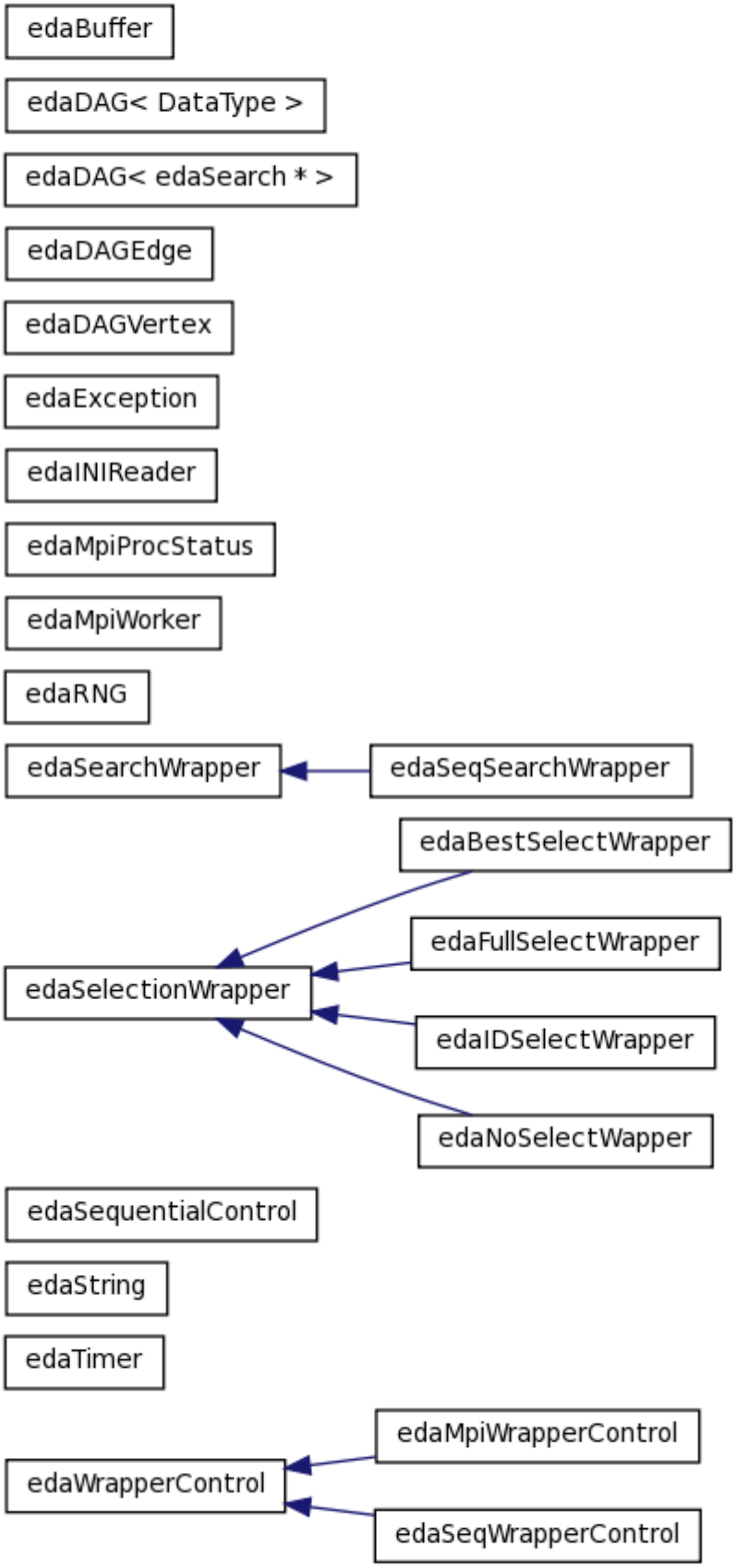
Các thành phần của giải thuật MA bao gồm khởi tạo tập dân cư ban đầu, quá trình chọn lọc tự nhiên, quá trình chọn cặp và lai chéo – gọi chung là quá trình tái tổ hợp, quá trình đột biến và quá trình cải tiến. Nhìn chung, hoạt động tìm kiếm của giải thuật MA rất tương tự giải thuật di truyền ngoại trừ vòng lặp bên trong là quy trình cải tiến các cá thể bằng giải thuật tìm kiếm cục bộ. Phần trình bày tiếp theo, chúng ta sẽ xem xét sâu hơn về quá trình cải tiến trong giải thuật MA.

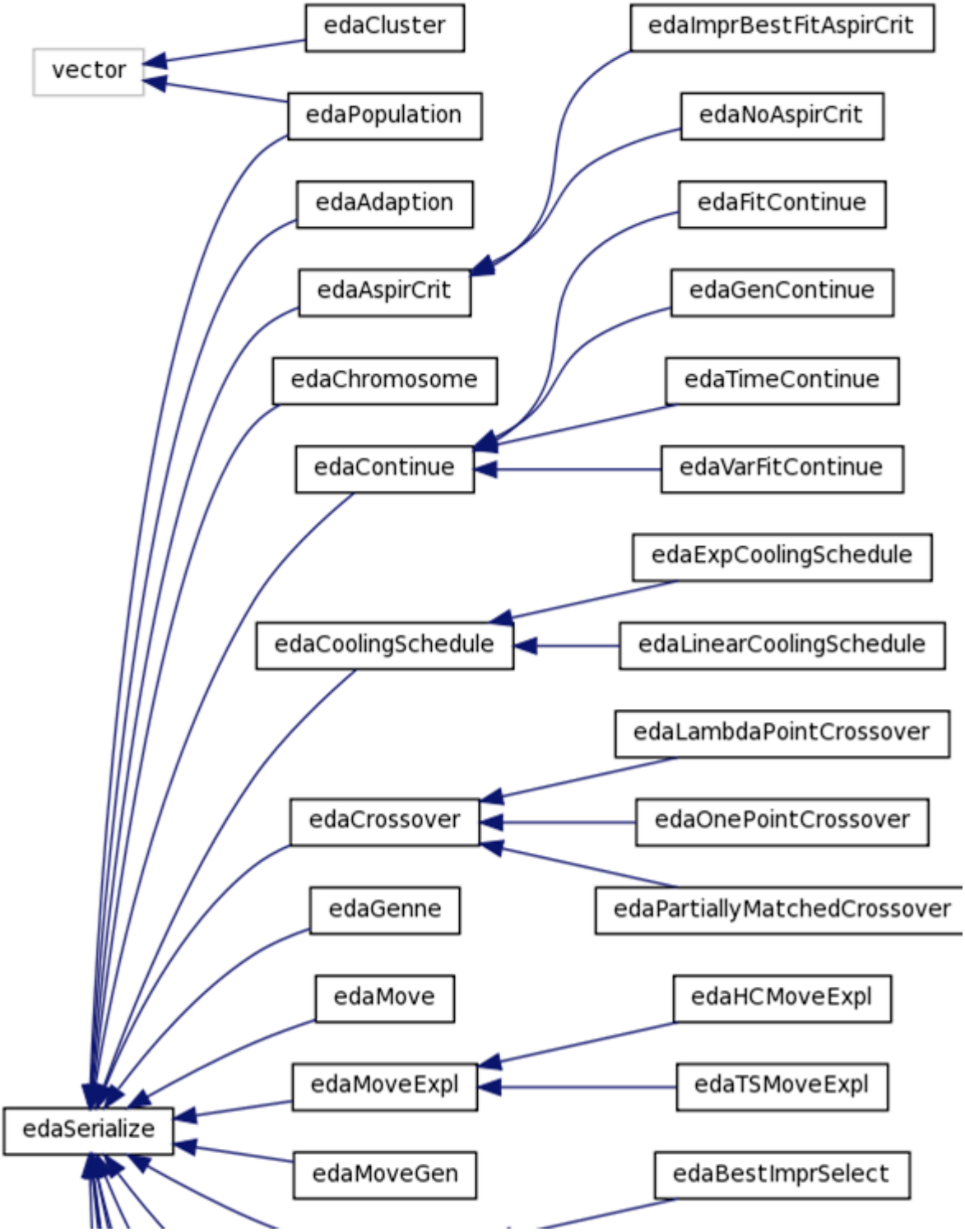
- Quá trình cải tiến (`edaAdaption`): Toán tử cải tiến là một công trình giúp nâng cao khả năng khai khác của giải thuật Memetic so với GA. Trong hiện thực của đề tài, các giải thuật tìm kiếm cục bộ như HC, SA và TS được sử dụng trong giai đoạn cải tiến của quy trình tối ưu hóa. Như vậy việc định nghĩa lân cận, cơ chế tạo lân cận là cần thiết đối với giải thuật MA. Với từng giải thuật dựa trên dịch chuyển khác nhau chiến lược tìm kiếm trong không gian nghiệm và kết quả của chất lượng nghiệm đầu ra là khác nhau.

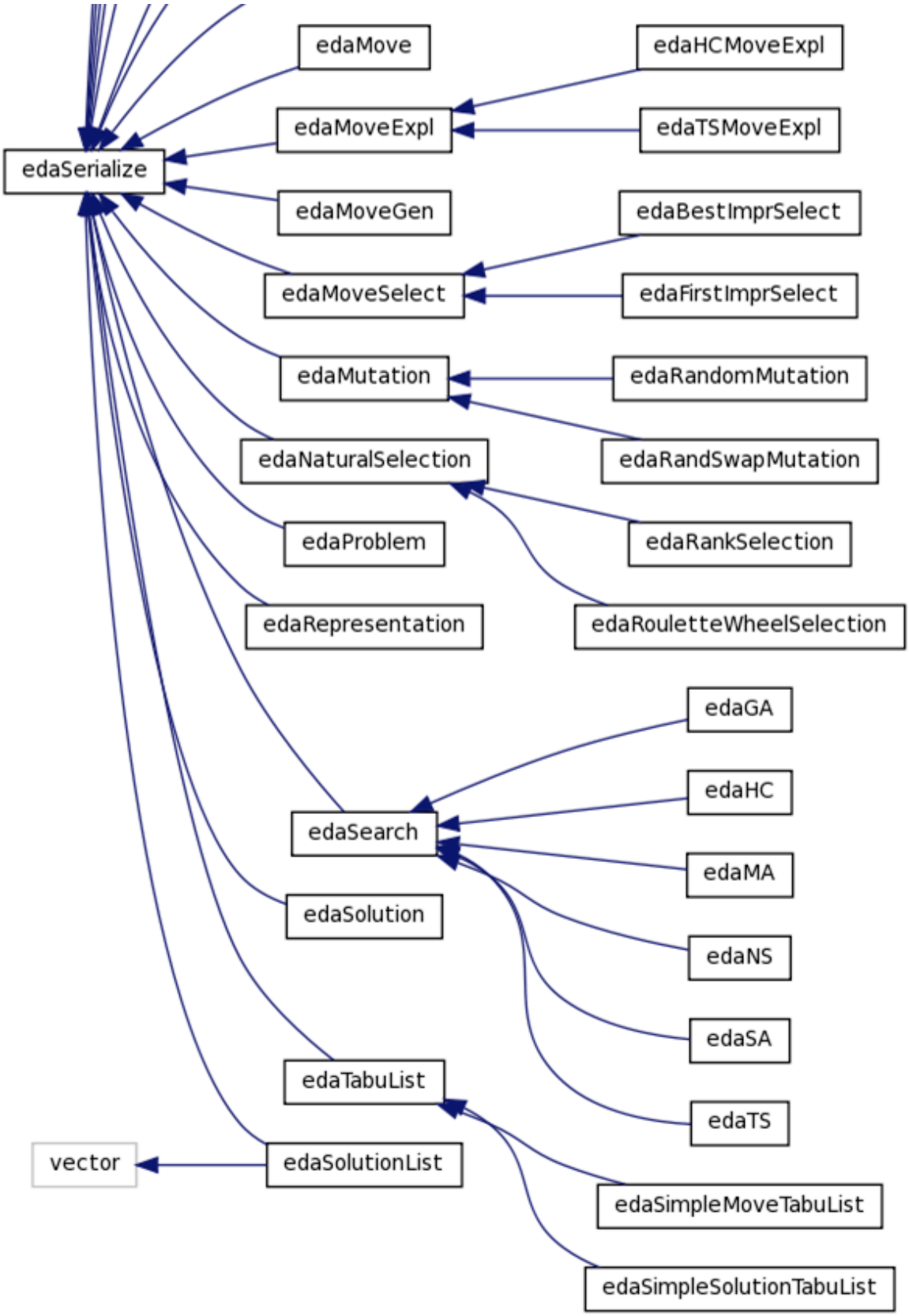
4.4 KIẾN TRÚC LỚP TRONG THƯ VIỆN

Ở chương này chúng ta sẽ trình bày sơ đồ kiến trúc các lớp hiện có trong thư viện và mối liên hệ giữa chúng. Đối với các đối tượng cần gửi đi bởi máy chủ, chúng được xây dựng bằng cách kế thừa lớp edaSearilize. Một số hiện thực kế thừa từ các lớp trừu tượng cũng được xây dựng sẵn trong thư viện nhằm hỗ trợ người dùng sử dụng thư viện tối ưu hiệu quả hơn.

Tổng số các lớp đối tượng được xây dựng riêng cho thư viện là 72 với gần một ngàn phương thức tương được xây dựng. Phần 4.5 sẽ trình một cách tóm lược chức năng của từng lớp đối tượng trong thư viện, chi tiết chức năng các phương thức ứng với chúng sẽ được trình bày rõ trong phần tài liệu hướng dẫn sử dụng đính kèm theo.


edaBuffer
edaDAG< DataType >
edaDAG< edaSearch * >
edaDAGEdge
edaDAGVertex
edaException
edaINIReader
edaMpiProcStatus
edaMpiWorker
edaRNG
edaSearchWrapper
edaSeqSearchWrapper
edaBestSelectWrapper
edaFullSelectWrapper
edaSelectionWrapper
edaIDSelectWrapper
edaNoSelectWapper
edaSequentialControl
edaString
edaTimer
edaMpiWrapperControl
edaWrapperControl
edaSeqWrapperControl


vector
edaCluster
edaPopulation
edaAdaption
edaAspirCrit
edaChromosome
edaContinue
edaImprBestFitAspirCrit
edaNoAspirCrit
edaFitContinue
edaGenContinue
edaTimeContinue
edaVarFitContinue
edaExpCoolingSchedule
edaCoolingSchedule
edaLinearCoolingSchedule
edaLambdaPointCrossover
edaCrossover
edaOnePointCrossover
edaPartiallyMatchedCrossover
edaGenne
edaMove
edaHCMoveExpl
edaMoveExpl
edaTSMoveExpl
edaSerialize
edaMoveGen
edaBestImprSelect


edaMove
edaHCMoveExpl
edaMoveExpl
edaTSMoveExpl
edaSerialize
edaMoveGen
edaBestImprSelect
edaMoveSelect
edaFirstImprSelect
edaMutation
edaRandomMutation
edaNaturalSelection
edaRandSwapMutation
edaProblem
edaRankSelection
edaRepresentation
edaRouletteWheelSelection
edaGA
edaHC
edaSearch
edaMA
edaSolution
edaNS
edaSA
edaTabuList
edaTS
vector
edaSolutionList
edaSimpleMoveTabuList
edaSimpleSolutionTabuList

4.5 CẤU TRÚC CƠ SỞ DỮ LIỆU TRONG THƯ VIỆN

edaAdaption	Lớp **edaAdaption** thực hiện việc cải tiến trên danh sách lời giải đầu vào dựa trên các giải thuật tìm kiếm cục bộ theo một tỷ lệ cho trước
edaAspirCrit	Lớp trừu tượng **edaAspirCrit** thực hiện việc kiểm tra sự hợp lệ của các bước chuyển ứng viên
edaBestImprSelect	Lớp **edaBestImprSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược Global Best
edaBestSelectWrapper	Lớp **edaBestSelectWrapper** thực hiện việc chọn lời giải có hàm lượng giá tốt nhất trong tập lời giải được đưa vào tại mỗi nút tìm kiếm
edaBuffer	Lớp **edaBuffer** thực việc đóng gói (pack) và mở gói (unpack) các loại dữ liệu cơ bản
edaChromosome	Lớp **edaChromosome** chứa thông tin về nhiễm sắc thể và các phương thức tương ứng
edaCluster	Lớp **edaCluster** chứa thông tin vì các cụm node tìm kiếm và các thao tác trên chúng
edaContinue	Lớp trừu tượng **edaContinue** mô tả các thông tin về điều kiện dừng trong thư viện
edaCoolingSchedule	Lớp trừu tượng **edaCoolingSchedule** mô tả các thông tin về chiến lược làm lạnh nhiệt độ
edaCrossover	Lớp trừu tượng **edaCrossover** mô tả các thông tin về chiến lược lai chéo giữa các nhiệm sắc thể
edaDAG	Lớp **edaDAG** hiện thực đồ thị có hướng không vòng với kiểu dữ liệu tùy biến
edaDAGEdge	Lớp **edaDAGEdge** là hiện thực của cạnh đồ thị DAG
edaDAGVertex	Lớp **edaDAGEdge** là hiện thực của đỉnh đồ thị DAG
edaException	Lớp **edaException** hiện thực việc xử lý ngoại lệ trong thư viện

edaExpCoolingSchedule	Lớp **edaExpCoolingSchedule** là hiện thực của chiến lược làm lạnh nhiệt độ theo hàm mũ
edaFirstImprSelect	Lớp **edaBestImprSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược First Best
edaFitContinue	Lớp **edaFitContinue** hiện thực điều kiện dừng theo hàm lượng giá
edaFullSelectWrapper	Lớp **edaFullSelectWrapper** thực hiện việc chọn tất cả các lời giải trong danh sách lời giải và đưa chúng vào nút tìm kiếm
edaGA	Lớp **edaGA** thực hiện việc tối ưu theo giải thuật di truyền
edaGenContinue	Lớp **edaGenContinue** hiện thực điều kiện dừng theo theo số lần lặp
edaGenne	Lớp **edaGenne** là hiện thực đối tượng gen trong giải thuật tiến hóa
edaHC	Lớp **edaHC** thực hiện việc tối ưu theo giải thuật leo đồi
edaHCMoveExpl	Lớp **edaHCMoveExpl** hiện thực chiến lược khai phá không gian ứng với giải thuật HC
edaIDSelectWrapper	Lớp **edaIDSelectWrapper** thực hiện việc chọn lời giải theo thứ tự của chúng trong sanh sách lời giải
edaImprBestFitAspirCrit	Lớp **edaImprBestFitAspirCrit** thực hiện việc lấy bước chuyển ứng viên có hàm lượng giá tốt nhất
edaLambdaPointCrossover	Lớp **edaLambdaPointCrossover** hiện thực chiến lược lai chéo giữa các nhiệm sắc thể với số diểm lai chéo là tùy chọn
edaLinearCoolingSchedule	Lớp **edaExpCoolingSchedule** là hiện thực của chiến lược làm lạnh nhiệt độ theo hàm tuyến tính
edaMA	Lớp **edaMA** thực hiện việc tối ưu theo giải thuật Memetic

edaMove	Lớp trừu tượng **edaMove** mô tả các thông tin về chiến lược duy chuyển trong không gian tìm kiếm của các giải thuật
edaMoveExpl	Lớp trừu tượng **edaMoveExpl** hiện thực chiến lược khai phá không gian ứng với từng giải thuật
edaMoveGen	Lớp trừu tượng **edaMoveGen** mô tả các thông tin về thức tạo các bước chuyển trong không gian tìm kiếm của các giải thuật
edaMoveSelect	Lớp trừu tượng **edaMoveSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược tương ứng
edaMpiProcStatus	Cấu trúc **edaMpiProcStatus** chứa thông tin tra trạng thái của các node tìm kiếm
edaMpiWorker	Lớp **edaMpiWorker** thực thi việc phân chia tác vụ giữa giữa máy chủ và các máy trạm
edaMpiWrapperControl	Lớp **edaMpiWrapperControl** điều khiển quá trình tối ưu bằng cơ chế MPI
edaMutation	Lớp trừ tượng **edaMutation** hiện thực các chiến lược đột biến
edaNaturalSelection	Lớp trừ tượng **edaNaturalSelection** hiện thực các chiến lược chọn lọc tự nhiên
edaNoAspirCrit	Lớp **edaImprBestFitAspirCrit** không thực hiện việc thay đổi danh sách Tabu
edaNoSelectWapper	Lớp **edaNoSelectWapper** thực hiện việc loại bỏ tất cả các lời giải trong sanh sách lời giải
edaNS	Lớp **edaNS** là một chiến lược tối ưu rỗng (không thực hiện quá trình tối ưu)
edaOnePointCrossover	Lớp **edaOnePointCrossover** hiện thực chiến lược lai chéo giữa các nhiệm sắc thể với một diểm lai chéo với các mode lai khác nhau
edaPartiallyMatchedCrossover	Lớp **edaPartiallyMatchedCrossover** hiện thực

	chiến lược lai chéo PMX giữa các nhiệm sắc thể
edaPopulation	Lớp **edaPopulation** hiện thực đối tượng quần thể trong giải thuật tiến hóa
edaProblem	Lớp trừu tượng **edaProblem** chứa thông tin bài toán cho từng vấn đề tối ưu cụ thể
edaRandomMutation	Lớp **edaRandomMutation** hiện thực chiến lược đột biến phát sinh ngẫu nhiên
edaRandSwapMutation	Lớp **edaRandSwapMutation** hiện thực chiến lược đột biến hoán đổi gen
edaRankSelection	Lớp **edaRankSelection** hiện thực chiến lược chọn lọc tự nhiên theo thứ hạng
edaRepresentation	Lớp trừu tượng **edaRepresentation** hiện thực việc chứa phương pháp mã hóa từ lời giải thành nhiệm sắc thể
edaRNG	Lớp **edaRNG** hiện thực cơ chế tạo các số ngẫu nhiên
edaRouletteWheelSelection	Lớp **edaRouletteWheelSelection** hiện thực chiến lược chọn lọc tự nhiên theo trọng số
edaSA	Lớp **edaSA** thực hiện việc tối ưu theo giải thuật mô phỏng luyện kim
edaSearch	Lớp trừu tượng **edaSearch** thực hiện việc tối ưu với các chiến lược hỗ trợ trong thư viện hoặc do người dùng xây dựng
edaSearchWrapper	Lớp **edaSearchWrapper** hỗ trợ cơ chế tối ưu với thông tin tìm kiếm đã được đóng gói
edaSelectionWrapper	Lớp trừu tượng **edaSelectionWrapper** thực hiện việc chọn lời giải với các chiến lược hỗ trợ trong thư viện hoặc do người dùng xây dựng
edaSeqSearchWrapper	Lớp trừu tượng **edaSeqSearchWrapper** điều khiển quá trình tối ưu bằng cơ chế tuần tự

edaSequentialControl	Lớp trừu tượng **edaSequentialControl** điều khiển quá trình thiết lập workflow tuần tự
edaSeqWrapperControl	Lớp trừu tượng **edaWrapperControl** điều khiển quá trình tối ưu tuần tự
edaSerialize	Lớp **edaSerialize** hiện thực cơ chế gói và giải gói cho các đối tượng phức tạp
edaSimpleMoveTabuList	Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo lớp **edaSolution**
edaSimpleSolutionTabuList	Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo lớp **edaMove**
edaSolution	Lớp trừu tượng **edaSolution** hiện thực lời giải của bài toán
edaSolutionList	Lớp trừu tượng **edaSolutionList** hiện thực danh sách lời giải của bài toán
edaString	Lớp **edaString** hiện thực việc hỗ trợ xử lý chuỗi trong thư viện
edaTabuList	Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo các đối tượng khác nhau
edaTimeContinue	Lớp **edaTimeContinue** hiện thực điều kiện dừng theo theo thời gian chạy
edaTimer	Lớp **edaTimer** hiện thực việc hỗ trợ tính toán thời gian trong thư viện
edaTS	Lớp trừu tượng **edaTS** thực hiện việc tối ưu theo giải thuật tìm kiếm Tabu
edaTSMoveExpl	Lớp **edaTSMoveExpl** hiện thực chiến lược khai phá không gian ứng với giải thuật TS
edaVarFitContinue	Lớp **edaVarFitContinue** hiện thực điều kiện dừng theo theo hàm lượng giá có khả năng thay đổi
edaWrapperControl	Lớp trừu tượng **edaWrapperControl** điều khiển quá trình tối ưu

CHƯƠNG 5. CÁC CÔNG CỤ PHỤ TRỢ KHÁC

5.1 PHÁT HIỆN LỖI

Phát hiện lỗi (debugging) là một phần công việc rất quan trọng và không thể thiếu trong quá trình lập trình. Quá trình phát triển của các công cụ phát hiện lỗi trong những năm qua đã làm cho công việc phát hiện lỗi trở nên dễ dàng hơn và đỡ mất thời gian hơn. Đối với quá trình lập trình song song, vai trò của công cụ phát hiện lỗi càng quan trọng hơn bởi khả năng xảy ra lỗi trong quá trình lập trình song song thường cao hơn nhiều so với việc lập trình tuần tự. Hơn thế nữa, các lỗi xảy ra trong lập trình song song cũng phức tạp hơn. Do vậy, sự trợ giúp của thư viện đối với việc phát hiện lỗi là rất cần thiết và sẽ giúp công việc của người lập trình sẽ trở nên thuận lợi hơn.

Đối với một chương trình thực thi song song bao gồm nhiều tiến trình cùng thực thi song song trên nhiều đơn vị tính toán và liên tục trao đổi thông tin với nhau trong suốt quá trình hoạt động nhằm đem lại kết quả và hiệu năng tốt hơn so với quá trình tính toán tuần tự. Tuy nhiên, do chính các đặc điểm này đã gây ra những khó khăn cho quá trình phát hiện lỗi:

- Chương trình song song không phải bao gồm một luồng công việc được thực hiện một cách có thứ tự như trong chương trình tuần tự. Chúng bao gồm nhiều luồng chương trình chạy song song và đồng thời trong các ràng buộc về thời gian và dữ liệu.
- Trong quá trình hoạt động, các luồng chương trình song song này quan hệ với nhau rất chặt chẽ thông qua việc trao đổi thông điệp.
- Khái niệm trạng thái của chương trình song song có nhiều khác biệt so với chương trình tuần tự. Tại một thời điểm, trạng thái của chương trình song song bao gồm một tập các trạng thái của các tiến trình trên các nút tính toán. Tập các trạng thái này thay đổi với mỗi lần chạy chương trình và gây khó khăn cho người lập trình khi bắt trạng thái để gỡ lỗi.

Việc thiết kế tổ chức chương trình song song là không hề đơn giản. Một chút sai sót rất nhỏ (chẳng hạn quên không đồng bộ giữa các tiến trình sẽ dễ dàng dẫn đến lỗi deadlock) có thể gây ra các lỗi ngầm bên trong và rất khó phát hiện. Với hệ thống thư viện meta-heuristic, người sử dụng có thể tránh được điều này bởi nhiệm vụ thực thi song song của chương trình hoàn toàn được đảm nhiệm bởi tầng Workflow Engine. Tuy nhiên điều này không có nghĩa là người dùng có thể hoàn toàn tránh được các lỗi phát sinh trong quá trình lập trình ví dụ như xung đột của các tiến trình trong workflow tính toán do người dùng tự định nghĩa, hoặc quá trình đóng/mở gói và gửi dữ liệu.

Để giải quyết khó khăn trong quá trình phát hiện lỗi cho lập trình song song, hiện nay trên thế giới có hai hướng tiếp cận chính: theo mô hình phát hiện lỗi tuần tự truyền thống và theo hướng mô hình thông điệp. Đối với mô hình theo hướng tuần tự, công việc phát hiện lỗi được tiến hành theo hai pha. Trong đó pha thứ nhất sẽ tiến hành gỡ rối cho từng đơn vị tính toán như trình phát hiện lỗi truyền thống và pha thứ hai sẽ tổng hợp các kết quả từ các nút tính toán và đưa ra cái nhìn tổng quát của cả hệ thống. Đây chính là thành phần quan trọng tạo nên sự khác nhau giữa việc phát hiện lỗi tuần tự và phát hiện lỗi song song. Đối với hướng mô hình truyền thông điệp, công việc gỡ rối chú trọng theo dõi quá trình truyền thông điệp giữa các tiến trình. Quy trình này xuất phát từ quan điểm các lỗi phổ biến và khó phát hiện trong chương trình song song thường là các lỗi của quá trình truyền thông. Sử dụng trình gỡ rối theo hướng này, người lập trình sẽ dễ dàng hơn trong việc phát hiện các lỗi đặc trưng của các tiến trình song song. Tuy nhiên, nhược điểm của công cụ này là không cho phép người sử dụng tương tác với chương trình, người dùng chỉ được phép theo dõi quá trình truyền thông chứ không được can thiệp và quá trình đó.

Trong hai hướng tiếp cận, hướng thứ hai tạo cho người sử dụng khả năng chủ động trong việc phát hiện lỗi lập trình. Người sử dụng có thể chạy, dừng và theo dõi cũng như thay đổi các giá trị các biến để kiểm tra. Tuy nhiên, việc hiện thực công cụ có những khả năng như vậy không hề đơn giản, việc tổng hợp các kết quả từ các tiến trình đơn lẽ để đưa ra khung nhìn tổng quát và trực quan đòi hỏi sự phức tạp cao và cần nhiều thời gian và công sức. Ngược lại, công cụ hỗ trợ theo dõi việc

truyền thông điệp cho phép người sử dụng quan sát một cách trực quan quá trình hoạt động của các tiến trình song song và sẽ thuận lợi hơn cho người lập trình cho việc phát hiện các lỗi do truyền thông. Đây chính là hướng tiếp cận hiện tại của cộng cụ phát hiện lỗi thuộc hệ thống thư viện meta-heuristic, bởi lẽ các lỗi thường gặp của người dùng phát sinh quá trình gói và truyền dữ liệu do người dùng thực hiện ở tầng ứng dụng. Việc phát hiện lỗi và sửa chữa các lỗi sâu thuộc về kiến trúc thư viện là không cần thiết đối với các người dùng cuối, bởi đó thuộc về trách nhiệm của nhưng người phát triển.

Hỗ trợ của thư viện trong việc phát hiện lỗi là giúp người dùng xác định được lỗi xảy ra ở tầng nào, tác vụ vụ tối ưu nào, đơn vị xử lý nào. Và thông tin của chúng sẽ được xuất ra màn hình khi quá trình tối ưu được thực hiện. Để khích hoạt chức năng trên, khi biên dịch thư viện người dùng phải bật thông số:

```
-D_DEBUG_ = [level]
```

trong đó level là một con số nguyên không âm thể hiện cấp độ debug.

Hiện tại thư viện chỉ hỗ trợ 3 cấp độ debug khác nhau, mỗi cấp độ sau sẽ bao gồm các thông tin được cung cấp độ bởi trước.

- Mức 1: Đây là mức thấp nhất giúp hiện thị thông tin, hiện tại tác vụ nào đang thực thi (ID), meta-heuristic đang thực thi có tên là gì, và kết quả giá trị hàm mục tiêu của mỗi lần lặp (hoặc mỗi thế hệ) trong quá trình tối ưu bởi giải thuật đó.
- Mức 2: Mức cao hơn này giúp người dùng có thể quan sát được các hoạt động bên trong của từng quá trình tối ưu trong các giải thuật, tùy theo chi tiết hiện thực của giải thuật tối ưu mà thông tin được xuất ra là khác nhau. Qua những thông tin này, người có thể xác định được các lỗi nếu có trong hiện thực các toán tử bên trong của các giải thuật.
- Mức 3: Đây là mức hỗ trợ cao nhất hiện tại được hỗ trợ trong thư viện, nhiệm vụ của tần này giúp cho người thực hiện quan sát các thông tin diễn ra trong quá trình song song hóa bao gồm việc truyền nhận dữ liệu giữa các đơn vị xử lý, các trạng thái được thiết lập trong quá trình xử lý song song.

5.2 NHẬT KÝ VÀ BIỂU ĐỒ

Ngoài việc hỗ trợ cơ chế phát hiện lỗi, thư viện còn hỗ trợ chức năng ghi vết lại hoạt động tìm kiếm qua file nhật ký. Mỗi khi quá trình tìm kiếm được thực thi, chức năng ghi nhật ký sẽ được kích hoạt và ghi lại hoạt động tìm kiếm của mỗi quá trình trên từng đơn vị xử lý.

Định dạng của tập tin nhật ký trong thư viện là .csv (Comma-Separated Values) một định dạng text phổ biến trong các nền tảng hệ điều hành gồm cả Linux, Unix, Window và MacOS. Người có thể quan sát các kết quả từ file để có được cái nhìn tổng quan từ quá trình tối ưu. Các thông tin được ghi lại trong file bao gồm: loại thuật toán tối ưu đang thực thi, ID (số thứ tự) của đỉnh, ID (số thứ tự) của đơn vị xử lý, trạng thái đang xử lý của quá trình tối ưu, ID (số thứ tự) của lời giải khả kiến trong danh sách lời giải đang tối ưu, thời điểm xuất ra thông tin (đơn vị là giây tính từ quá trình bắt đầu tối ưu), giá trị hàm mục tiêu của lời giải, số lần lặp hoặc số thế hệ (với các giải thuật dựa trên quần thể), số lời giải trong danh sách, lời giải tốt nhất trong danh sách, lời giải xấu nhất, độ lệch phát hiện và thể hiện của lời giải hoặc nhiễm sắc thể.

Bảng 1. Log file được ghi lại trong quá trình tối ưu của giải thuật GA.

Heuristic	Task	Proc	Status	ID	Time	Fitness	Loop/Ger	Count	Best	Worst	Mean	Std	Sol/Chro
GA	1	1	S	17	90.9318	0.972137	0	20	0.972137	-0.97186	-0.06867	0.623577	7.91E+35
GA	1	1	P	0	353.035	0.972137	0	20	0.972137	-0.99802	0.178316	0.671512	7.91E+35
GA	1	1	F	0	357.554	0.972137	1	20	0.972137	-0.99802	0.178316	0.671512	7.91E+35
GA	1	1	S	0	361.81	0.972137	0	20	0.972137	-0.99802	0.178316	0.671512	7.91E+35
GA	1	1	P	0	606.952	0.972137	0	20	0.972137	0.069391	0.611027	0.353509	7.91E+35
GA	1	1	F	0	613.596	0.972137	1	20	0.972137	0.069391	0.611027	0.353509	7.91E+35

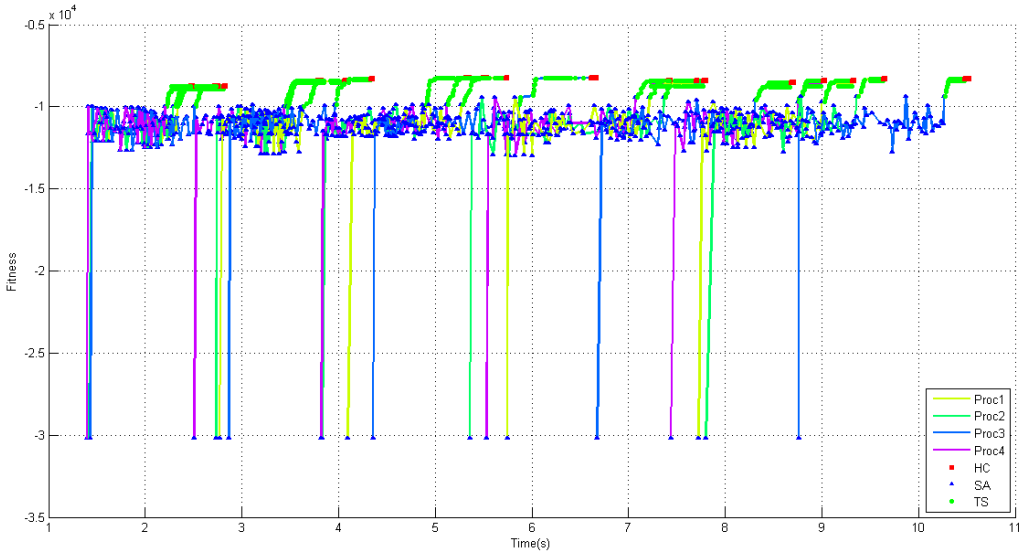
Để trực quan hóa thông tin của tập tin nhật ký, nhóm phát triển thư viện cũng đã hiện thực một script file nhỏ theo phát hiện của MATLAB (người dùng linux có thể sử dụng với Octave). Cú pháp để hiện thị thông tin của tập tin nhật ký là:

```
plotLogFile("Log_File_Name")
```

với giá trị `Log_File_Name` là tên của tập tin nhật ký (bao gồm cả phần mở rộng của tên file). Hình 5.2 là biểu đồ kết quả hiện thực của một quy trình tối ưu hóa với nhiều tác vụ tối ưu lồng ghép và số vi xử lý được cấp phát là 5 (1 master và 4

slave). Biểu đồ của tập tin nhật ký với trục tung hiển thị giá trị hàm mục tiêu, trục hoành biểu thị thời gian tính toán, tên của giải thuật meta-heuristic là được ký hiệu bởi các biểu tượng hình vuông, tròn và tam giác. Màu của các đường biểu thị tiến trình hoạt động của các đơn vị xử lý khác nhau.

Việc trực quan hóa thông tin quá trình tối ưu thông qua đồ thị sẽ giúp người sử dụng thư viện dễ dàng nắm bắt các thông tin quan trọng. Và qua việc phân tích các thông tin trong nó người dùng đưa ra các quyết định hiệu chỉnh hợp lý hơn cho quy trình tối ưu kế tiếp. Tuy nhiên ở phần nội dung này, chúng ta sẽ không đi sâu về việc phân tích sâu về nội dung đồ thị, chúng sẽ được trình bày chi tiết ở những phần sau.



Hình 29. Biểu đồ hiển thị tập tin nhật ký cúa quá trình tối ưu với các tác vụ tối HC, SA, TS với 4 đơn vị tính toán song song.

CHƯƠNG 6. THỬ NGHIỆM ĐÁNH GIÁ HIỆU NĂNG CỦA THƯ VIỆN

Trong phần thử nghiệm, đánh giá hiệu năng cũng như tính đúng đắn của thư viện, chúng tôi chọn và thực hiện trên bài toán TSP (Travelling salesman problem). Đây là một dạng bài toán kinh điển thường được dùng làm ví dụ trong lĩnh vực tối ưu với độ phức tạp là *NP-hard*. Với độ phức tạp này, chưa có một giải thuật nào tìm được nghiệm tối ưu toàn cục cho bài toán một cách hiệu quả (trong thời gian đa thức). Tuy nhiên có thể áp dụng các phương pháp tìm kiếm meta-heuristic cho bài toán này, các nghiệm thu được có thể chấp nhận được trong khoảng thời gian tìm kiếm cho phép. Trong phần nội dung này, chúng ta sẽ được hướng dẫn một cách chi tiết cách thức thực thi quá trình tối ưu với sự hỗ trợ của thư viện meta-heuristics.

6.1 PHÁT BIỂU CỦA BÀI TOÁN

Cho một tập các thành phố và khoảng cách giữa các thành phố. Người bán hàng xuất phát từ một thành phố, đi qua tất cả các thành phố, mỗi thành phố đi đúng một lần, và sau đó quay về thành phố xuất phát. Mục tiêu của bài toán là tìm thứ tự các thành phố mà người bán hàng đi qua sao cho tổng quãng đường đi là ngắn nhất.

Như vậy những thông tin cần cho bài toán chúng ta là:

- Số lượng thành phố cần viếng thăm,
- Tọa độ các thành phố (xét trong không gian Euclidean 2 chiều),
- Khoảng cách giữa hai thành phố bất kỳ (trong không gian bất kỳ thì đại lượng này là có thể phải cung cấp riêng, nhưng trong bài toán TSP cơ bản thì có thể tính được dùng khoảng cách Eucliedan).

Tổ chức của bài toán TSP tương ứng trong thư viện là một lớp `Graph` với các các thuộc tính thành phần như sau:

```
unsigned int numVert;       // Số đỉnh của bài toán
double *vectCoord;          // Mãng tọa độ X, Y của các đỉnh
double *dist;               // Mãng khoảng cách giữa các đỉnh
```

Một lớp đối tượng đầu tiên người dùng phải hiện thực là dùng để mô tả bài toán, và vì thế phải thừa kế từ lớp `edaProblem`. Do đó lớp `Graph` (thừa kế từ `edaProblem`) buộc phải hiện thực các phương thức:

```
virtual ~ edaProblem();
virtual edaProblem *clone() const = 0;
virtual edaProblem& operator = (const edaProblem &pro) = 0;
virtual void printOn(ostream &os) const;
virtual void load(const char *fileName);
void Serialize(edaBuffer &buf, bool pack) = 0;
```

trong đó phương thức `clone()` dùng để nhân bản chính bản thân đối tượng, phương thức `Serialize()` dùng để đóng gói dữ liệu và tái tạo lại đối tượng khi truyền nhận giữa các quá trình song song, toán tử `operator = ()` là hiện thực của phép gán, phương thức `printOn()` có chức năng xuất kết quả ra thiết bị xuất chuẩn và tác vụ `load()` dùng để đọc các thông tin của bài toán và lưu trữ chúng vào các thuộc tính của đối tượng.

Ngoài ra, trong đối tượng `Graph` còn có phải thực hiện các công việc sau đây để hỗ trợ cho việc đóng gói dữ liệu:

- Hiện thực đầy đủ việc khởi tạo mặc định đối tượng, khởi tạo đối tượng qua cơ chế sao chép và hủy đối tượng.
- Khai báo class ID: trong đặc tả của lớp mới, người dùng gọi hàm `setClassID()` để gán một định danh cho lớp đó.

Sau đây chúng ta sẽ xem xét cách thức gói và tái tạo dữ liệu trong phức `Serialize()` của lớp `Graph` được hiện thực trong thư viện:

```
void Graph::Serialize(edaBuffer &buf, bool pack)
{
  if (pack) {
    // First, pack the numVert
    buf.Pack(&numVert, 1);

    // And the vector coordinates
    buf.Pack(vectCoord, numVert * 2);

    // Then pack the dist vector (of vector)
    buf.Pack(dist, numVert * numVert);
  }
  else {
    easer();
    // First, unpack the numVert
    buf.UnPack(&numVert, 1);
```

```
      // And the vectCoord
      vectCoord = new double[numVert * 2];
      buf.UnPack(vectCoord, numVert * 2);

      // Then unpack the dist vector
      dist = new double[numVert * numVert];
      buf.UnPack(dist, numVert * numVert);
    }
  }
```

Trong quá trình đóng gói và tái tạo các thuộc tính, trật tự giữa các biến là rất quan trọng. Nếu trật tự này bị đảo lộn việc tái tạo cấu trúc các biến trong bộ nhớ sẽ không trọn vẹn và điều này sẽ gây ra lỗi trong quá trình truyền gửi thông điệp giữa các đơn vị xử lý khác nhau. Đối tượng `buf` thuộc lớp `edaBuffer` có nhiệm vụ đóng gói các kiểu dữ liệu cơ bản, đối với kiểu dữ liệu cần đóng/mở gói hoặc làm việc với dạng mãng thông tin người dùng khi chỉ định cho phương thức `Pack()` và `UnPack()` là kích thước của mãng. Như vậy thông tin về kích thước mãng nên được truyền đi trước khi các thông tin về mãng được truyền đi. Phương thức `easer()` được thực thi trước quá trình giải gói dữ liệu nhằm đảm bảo rằng các thông tin có sẵn trong đối tượng phải được xóa trước khi quá trình tái tạo thông tin diễn ra. Và điều này sẽ tránh các xung đột về kiểu dữ liệu con trỏ có thể diễn ra.

Để định danh một lớp mới và giúp thư viện xác định kiểu dữ liệu ban đầu khi phục hồi dữ liệu. Các đối tượng trong thư viện cần phải được khai báo class ID, để thực hiện việc này thư viện hỗ trợ cung cấp sẵn hàm `setClassID()` để gán giá trị định danh:

```
setClassID(_USERCLASSID_ + _CLSID_GRAPH_);
```

Class ID của các đối tượng `Graph` được cấu thành bởi 2 thành phần là giá trị `_USERCLASSID_` và `_CLSID_GRAPH_`. Đối với mọi lớp do người dùng định nghĩa tiền số `_USERCLASSID_` được thêm vào để phân biệt với hệ thống các lớp đã được định danh sẵn trong thư viện. `_CLSID_GRAPH_` chính là giá trị ID của đối tượng `Graph`, giá trị này phải là duy nhất đối với một lớp và phải là giá trị nguyên. Để tiện cho việc quản lý, người dùng nên chủ động tập hợp các giá trị ID của tất cả các lớp được định nghĩa vào cùng một header file như trường hợp được minh họa sau:

```
#ifndef _tspDefine_h_
#define _tspDefine_h_
```

```
#include "../lib/eda.h"

#define _CLSID_TWO_OPT_                 1
#define _CLSID_TWO_OPT_NEXT_            2
#define _CLSID_TWO_OPT_MOVE_RANDOM_     3
#define _CLSID_K_OPT_                   4
#define _CLSID_K_OPT_NEXT_              5
#define _CLSID_K_OPT_RAND_              6
#define _CLSID_OR_OPT_                  7
#define _CLSID_OR_OPT_NEXT_             8
#define _CLSID_OR_OPT_RAND_             9

#define _CLSID_TSP_SOLUTION_           10
#define _CLSID_GRAPH_                  11
#define _CLSID_TSP_REPRESENTATION_     12
#define _CLSID_TSP_GENNE_              13

#endif
```

Trên đây là nội dung của file `tspDefine.h`, nơi chứa thông tin về ID của các lớp do người dùng tự định nghĩa trong bài toàn TSP. Tiếp theo ta quan sát hiện thực của lớp `Graph` được kế thừa từ giao tiếp của `edaProblem`:

```
class Graph : public edaProblem {
public:
  Graph();
  Graph(const char *fileName);
  Graph(const Graph &g);
  ~Graph();

  Graph* clone() const;
  void load(const char *fileName);
  float distance(unsigned int from, unsigned int to);
  unsigned int size() const;
  Graph& operator = (const edaProblem &pro);
  void Serialize(edaBuffer &buf, bool pack);
  void printOn(ostream &os) const;
  setClassID(_USERCLASSID_ + _CLSID_GRAPH_);

private:
  void computeDistances();
  void easer();

  unsigned int numVert;
  double *vectCoord;
  double *dist;

};
```

Ngoài phương thức `easer()` đã kể trên, hai phương thức mới không được kế thừa từ giao tiếp của `edaProblem` là `distance()` và `computeDistances()`. Chức năng của phức thức `computeDistances()` là xác định khoảng cách giữa cách thành phố từ tọa độ của chúng và lưu trữ vào đối tượng mãng hai chiều `dist`. Quá trình tính toán

này diễn ra ngay khi việc đọc thông tin bài toán từ các file. Phương thức `distance()` giúp lấy thông tin khoảng cách giữa hai thành phố bất kỳ trong bài toán từ và điều này rất cần thiết để tính toán xác định chiều dài của lộ trình cũng như lượng giá cho các bước chuyển.

Qua những phân tích trên, ta đã phần nào thấy được cách thức xây lớp `tspProblem` kế thừa từ `edaProblem` của thư viện. Những phân tích tiếp theo trong quá trình xây dựng quy trình tối ưu cho bài toán TSP sẽ giúp người đọc nắm rõ hơn nữa tầm quan trọng của chúng trong quá trình tối ưu.

6.2 ĐẶC TẢ NGHIỆM

Nghiệm của bài toán là thứ tự các thành phố mà người bán hàng phải đi qua. Chúng ta sẽ biểu diễn nghiệm dưới dạng một mảng các số nguyên các số này sẽ giúp ta định danh các thành phố. Để biểu diễn nghiệm trong thư viện, ta tạo một lớp mới kế thừa từ lớp `edaSolution`. Khi thừa kế, lớp này cần phải có thuộc tính biểu diễn nghiệm và cần phải hiện thực các phương thức được yêu cầu từ `edaSolution` như sau:

```
virtual ~edaSolution();
virtual edaSolution* clone() const = 0;
virtual void init();
virtual double evaluate() = 0;
virtual double getCost() const = 0;
virtual void setCost(double value) {}
virtual edaSolution& operator = (const edaSolution &sol) = 0;
virtual bool operator == (const edaSolution &sol) const = 0;
virtual void printOn(ostream &os) const;
virtual void Serialize(edaBuffer &buf, bool pack) = 0;
```

trong đó, chức năng của từng phương thức là:

- `clone()`: nhân bản đối tượng.
- `init()`: khởi tạo (ngẫu nhiên) lời giải (nghiệm) ban đầu.
- `evaluate()`: lượng giá nghiệm, tính toán giá trị hàm mục tiêu (hoặc độ thích nghi).
- `getCost()`: lấy giá trị hàm mục tiêu dựa trên lần lượng giá trước đó.
- `setCost()`: thiết lập trực tiếp giá trị hàm mục tiêu.
- `operator = ()`: toán tử gán của đối tượng.

- `operator == ()`: toán tử so sánh bằng của đối tượng.
- `printOn()`: xuất kết quả ra output stream.
- `Serialize()`: dùng để đóng gói dữ liệu và tái tạo lại đối tượng trong truyền nhận dữ liệu song song.

Hiện thực của lớp đối tượng `tspSolution` của nó được xây dựng như sau:

```
class tspSolution: public vector<unsigned int>, public edaSolution
{
 public:
  tspSolution();
  tspSolution(const Graph &_graph);
  tspSolution(const tspSolution &sol);
  virtual ~tspSolution();
  edaSolution* clone() const;
  double evaluate();
  double getCost() const;
  void init();
  void setCost(double value);
  void Serialize( edaBuffer &buf, bool pack );
  void printOn(ostream &os) const;
  edaSolution& operator = (const edaSolution &sol);
  bool operator == (const edaSolution &sol) const;
  setClassID(_USERCLASSID_ + _CLSID_TSP_SOLUTION_);

private:
  void easer();
  Graph *graph;
  double cost;
};
```

Lời giải được hiện thực ở đây là một vector (danh sách) các biến có kiểu unsigned int, cấu trúc dữ liệu này sẽ được chức thành một chuỗi các thành phố được viếng thăm trong lộ trình. Hai thuộc tính private của lớp đối tượng `tspSolution` là giá trị `cost` – một biến số thuộc miền số thực đại diện cho giá trị hàm mục tiêu của lời giải – và đối tượng graph thuộc lớp `edaProblem`. Thông thường với các đối tượng được kế thừa từ `edaSolution`, `edaMove`, `edaMoveGen`, `edaRepresent`... thường mang có thuộc tính `edaProblem`, bởi lẽ thông tin bài toán luôn cần thiết để tính toán giá trị hàm mục tiêu. Ngoài các phương thức buộc phải viết lại các phương thức từ lớp `edaSolution`, lớp `tspSolution` còn được hiện thực thêm hai phương thức phụ trợ mới là easer(). Phương thức này giúp hủy các thuộc tính kiểu con trỏ và gán giá trị chúng là NULL.

Một lần nữa ta hãy xem xét quá trình hiện thực phương thức `Serialize()`, ở đây đối tượng cần gói không đơn thuần chỉ có kiểu dữ liệu cơ bản đã được hỗ trợ trong lớp `edaBuffer` của thư viện.

```
void tspSolution::Serialize( edaBuffer &buf, bool pack ) {
  if (pack) {
    // Pack the Graph
    graph->doSerialize(buf, pack);

    // Then pack the vector
    unsigned _size = this->size();
    buf.Pack(&_size, 1);

    vector<unsigned int>::iterator iter;
    for (iter = this->begin(); iter != this->end(); iter++) {
      buf.Pack((unsigned int *) &(*iter), 1);
    }

    //Pack the Cost
    buf.Pack(&cost, 1);
  }
  else {
    easer();

    // Unpack the Graph
    graph = (Graph*) classGenerateFromBuffer(buf);

    // then unpack the vector
    unsigned int _size;
    buf.UnPack(&_size, 1);
    this->resize(_size);

    unsigned int atom;
    for (unsigned int i = 0; i < _size; i++) {
      buf.UnPack((unsigned int *) &atom, 1);
      (*this)[i] = atom;
    }

    //UnPack the Cost
    buf.UnPack(&cost, 1);
  }
}
```

Việc đóng gói đối tượng `graph` được thực hiện qua phương thức `doSerialize()` với tham trị là đối tượng `buf` thuộc lớp `edaBuffer`. Khi được thực thi, phương thức `doSerialize()` có hai nhiệm vụ, một là nó sẽ xác định và gói thông tin Class ID của đối tượng (dựa vào thông tin do hàm `setClassID()` của chính lớp đó cung cấp) và hai là chúng sẽ gọi đến chính phương thức `Serialize()` của đối tượng và thực thi các lệnh có trong đó. Tất cả các thông tin này sẽ được đưa vào đối tượng `buf` theo trật tự người dùng thiết lập.

Tiếp theo, chúng ta sẽ xét đến quá trình phục hồi đối tượng. Cũng như phương thức `doSerialize()`, phương thức `classGenerateFromBuffer()` được định nghĩa sẵn trong lớp `edaSerialize`. Hàm này có nhiệm vụ chuyển các thông tin được gói từ các đối tượng của lớp `edaBuffer` thành các lớp đối tượng tương ứng. Trước tiên thông tin về Class ID được chuyển ra từ buffer (bộ đệm), giúp thư viện định danh được lớp nào đang nằm trong buffer. Hàm `classGenerate()` với đối số là Class ID trong phương thức `classGenerateFromBuffer()` được gọi thực thi nhằm tạo ra đối tượng tương ứng. Ngay sau đó, đối tượng được tạo ra này sẽ gọi thực hiện hàm `Serialize()` với thiếp lập giải gói nhằm trích rút các thông tin trong bộ đệm. Như vậy thông tin của đối tượng graph được gói trước dây hoàn toàn được phục hồi, tuy nhiên kiểu của đối tượng được phục hồi chỉ ở dạng lớp đối tượng tổng quát là `edaSerialize`. Do vậy để phép gán con trỏ có hiệu lực ta cần thực hiện việc ép kiểu đối tượng.

Tuy việc thực thi các quá trình gói, giải gói và khôi phục dữ liệu bên dưới tầng ứng dụng là khá phức tạp, nhưng người sử dụng thư viện hoàn toàn có thể yên tâm, bởi lẽ, ứng dụng của họ hoàn toàn được tách biệt khỏi các tầng bên dưới. Điều quan trọng là họ cần thực hiện đúng và đầy đủ các quy trình được yêu cầu bởi thư viện. Một quy tắc xuyên suốt, nếu người dùng muốn tận dụng khả năng song song hóa của thư viện là "***Khi các biến được khai báo, chúng cần được đóng gói và giải gói theo cùng một trật tự***".

6.3 ĐẶC TẢ LÂN CẬN

Các nghiệm khả thi của bài toán có thể được xây dựng bằng cách hoán vị các thành phố của lời giải ban dầu. Để có thể sử dụng các thuật toán tối ưu dạng dịch chuyển, chúng ta cần đặc tả các lân cận thông qua các phép hoán vị. Lớp thông tin đặc tả các lân cận sẽ thừa kế từ giao diện `edaMove`. Các phương thức cần phải hiện thực của giao tiếp này bao gồm:

```
virtual edaMove* clone() const = 0;
virtual void init(const edaSolution &sol) = 0;
virtual double incrEval(const edaSolution &sol) const = 0;
virtual void update( edaSolution &sol ) const = 0;
virtual edaMove& operator = (const edaMove& _move) = 0;
virtual void Serialize( edaBuffer &buf, bool pack ) = 0;
```

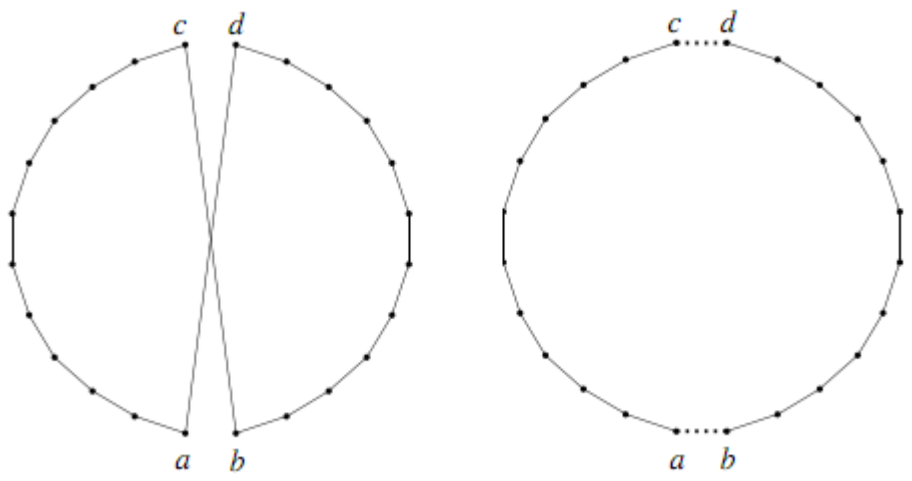
```
virtual bool operator == (const edaMove &_move) const = 0;
virtual void printOn( ostream &os ) const;
```

Ngoài các phương thức điển hình của một đối tượng kế thừa từ lớp `edaSerialize`, chúng ta có những phương thức riêng đối với các đối tượng thuộc lớp lân cận như sau:

- `incrEval()`: hàm lượng giá, được tính dựa trên sự thay đổi khi áp dụng phép biến đổi hiện tại lên nghiệm. Hàm này thường được sử dụng để tính nhanh giá trị của hàm mục tiêu, cũng như trong các trường hợp giá trị hàm mục tiêu có thể tính theo dạng tăng dần (incrementally).
- `update()`: áp dụng bước chuyển hiện tại lên lời giải.
- `init()`: khởi tạo lân cận ban đầu.

Trong hiện thực cho TSP, từ giao tiếp của `edaMove`, chúng tôi xây dựng hai dạng lân cận cho bài toán bài toán này, bao gồm lân cận dạng 2-Opt cổ điển và phức tạp hơn là dạng lân cận dạng Or-Opt với tham số λ là tùy chọn.

6.3.1 *Lân cận 2-Opt*



Hình 30. Cơ chế tạo lời giải khả thi của *2-Opt.*

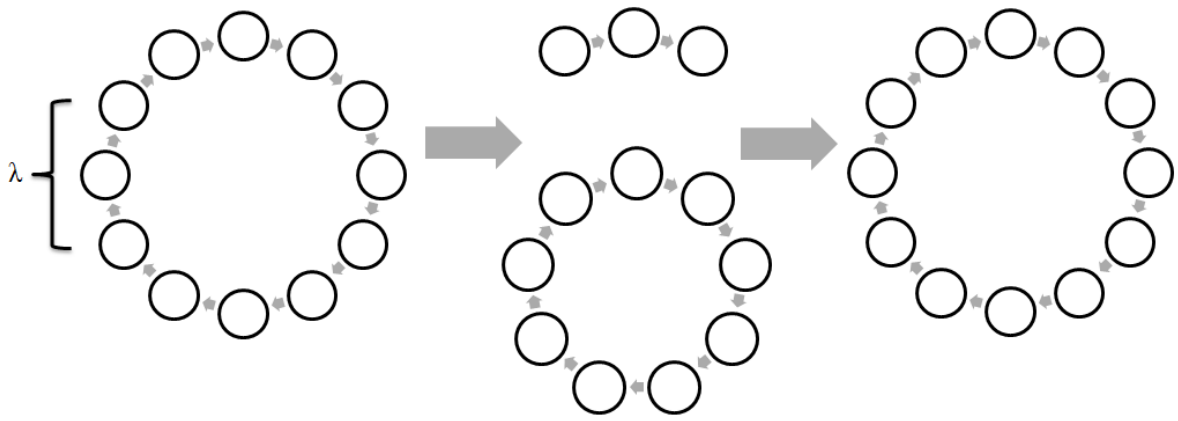
Đối với lân cận dạng truyền thống là 2-Opt, cấu trúc dữ liệu dạng `pair` thuộc thư viện `<utility>` được dùng để xây dựng cấu trúc của bước chuyển. Thông tin cần lưu trữ ở đây chỉ đơn giản là vị trí 2 điểm (ID) trong lộ trình mà tại đó chúng được tách

rời và ghép nối chéo từng cặp lại với nhau. Cách thức thực hiện toán tử này chi tiết người đọc quan tâm có thể tìm xem tại các tài liệu về TSP.

```
class tspTwoOpt : public pair<unsigned int, unsigned int>, public edaMove
{
 public:
  tspTwoOpt();
  tspTwoOpt(const tspTwoOpt &move);
  tspTwoOpt(const Graph &g);
  ~tspTwoOpt();
  edaMove* clone() const;
  void init(const edaSolution &sol);
  double incrEval(const edaSolution &sol) const;
  void update( edaSolution &sol ) const;
  void Serialize( edaBuffer &buf, bool pack );
  edaMove& operator = (const edaMove &_move);
  bool operator == (const edaMove &_move) const;
  void printOn( ostream &os ) const;
  setClassID(_USERCLASSID_ + _CLSID_TWO_OPT_);

 private:
  void easer();
  Graph *graph;
};
```

Ở đây thuộc tính `graph` được sử dụng để hỗ trợ hàm `incrEval()` tính toán giá trị hàm mục tiêu của bài toán.

6.3.2 Lân cận Or-Opt



Hình 31. Cơ chế tạo lời giải khả thi của *Or-Opt.*

Bên cạnh dạng lân cận truyền thống, một dạng lân cận khác, linh động hơn được xây dựng cho bài toán TSP là Or-Opt. Từ lời giải khả thi, một tập các đỉnh trong lộ trình được tách khỏi lời giải lời ban đầu và được chèn trở lại vào các vị trí mới của lộ trình.

```
class tspOrOpt : public edaMove
{
 public:
  tspOrOpt(unsigned int lambda = 2);
  tspOrOpt(const tspOrOpt &move);
  tspOrOpt(const Graph &g, unsigned int lambda = 2);
  virtual ~tspOrOpt();
  edaMove* clone() const;
  void init(const edaSolution &sol);
  double incrEval(const edaSolution &sol) const;
  void update( edaSolution &sol ) const;
  void Serialize( edaBuffer &buf, bool pack );
  edaMove& operator = (const edaMove &_move);
  bool operator == (const edaMove &_move) const;
  void printOn( ostream &os ) const;
  setClassID(_USERCLASSID_ + _CLSID_OR_OPT_);

//Properties
  unsigned int LAMBDA;
  unsigned int FIRST;
  unsigned int SECOND;

private:
  void easer();
  Graph *graph;

};
```

Hệ số λ (LAMBDA) được chọn chính là số tập các đỉnh được tách khỏi lộ trình. Người dùng có thể thiết lập giá trị cho thuộc tính này thông qua hàm khởi tạo (mặc định là 2). Hai vị trí lấy và chèn các node trong lộ trình tương ứng với hai thuộc tính FRIST và SECOND được định nghĩa trong lớp `tspOrOpt`. Tùy theo chiến lược của thuật toán các thuộc tính FRIST và SECOND sẽ duyệt qua một cách tuần tự hoặc ngẫu nhiên.

6.4 CƠ CHẾ PHÁT SINH LÂN CẬN

Như đề cập trong phần lý thuyết, các thuật toán dựa trên meta-heuristic cần có một định nghĩa về cấu trúc lân cận. Trong triển khai thuật toán bằng thư viện, các nghiệm lân cận sẽ được sinh bằng cách áp dụng một phép biến đổi nào đó lên nghiệm hiện tại. Để đặc tả nghiệm lân cận này, ta tạo một lớp thừa kế từ lớp `edaMoveGen`. Trong lớp này, người dùng cần phải hiện thực phương thức `generate()`. Phương thức này sẽ tạo ra các phép biến đổi tiếp theo để sinh ra nghiệm lân cận.

Trong hiện thực của bài toán TSP ứng với 2 loại lân cận 2-Opt và Or-Opt là 4 cơ chế tạo lân cận `tspTwoOptNext`, `tspTwoOptRand`, `tspOrOptNext`, `tspOrOptRand`. Với 2 cơ chế `tspTwoOptNext` và `tspOrOptNext`, các lớp này sẽ lần lượt tạo ra các lân cận của lời giải hiện tại thông qua phương thức `generate()`. Phương thức này phải được định nghĩa trả về giá trị là FASLE khi toàn bộ bước chuyển trong không gian lân cận đã được quét qua.

```
class tspTwoOptNext: public edaMoveGen
{
 public:
  tspTwoOptNext();
  tspTwoOptNext(const Graph &g);
  tspTwoOptNext(const tspTwoOptNext &m);
  ~tspTwoOptNext();
  edaMoveGen *clone() const;
  virtual bool generate( edaMove *move, const edaSolution &sol );
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_TWO_OPT_NEXT_);

 private:
  void easer();
  Graph *graph;
};

class tspOrOptNext: public edaMoveGen
{
 public:
  tspOrOptNext();
  tspOrOptNext(const Graph &g);
  tspOrOptNext(const tspOrOptNext &m);
  ~tspOrOptNext();
  edaMoveGen *clone() const;
  virtual bool generate( edaMove *move, const edaSolution &sol );
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_OR_OPT_NEXT_);

 private:
  void easer();
  Graph *graph;
};
```

Ngược lại, trong cơ chế phát sinh lân cận dạng ngẫu nhiên (`tspTwoOptRand` và `tspOrOptRand`), số bước chuyển sẽ được chọn một cách ngẫu nhiên trong không gian lân cận. Vì quá trình phát sinh lân cận là ngẫu nhiên nên sẽ có những trường hợp bước chuyển được sẽ trùng lắp (người dùng có thể định nghĩa các cơ chế của riêng mình để tránh chuyện này, hoặc sử dụng TB).

```
class tspTwoOptMoveRandom: public edaMoveGen
{
```

```
 public:
  tspTwoOptMoveRandom();
  tspTwoOptMoveRandom(Graph &g);
  tspTwoOptMoveRandom(const tspTwoOptMoveRandom &m);

  virtual ~tspTwoOptMoveRandom();
  edaMoveGen *clone() const;
  virtual bool generate( edaMove *move, const edaSolution &sol );
  setClassID(_USERCLASSID_ + _CLSID_TWO_OPT_MOVE_RANDOM_);
  void Serialize(edaBuffer &buf, bool pack);

 private:
  void easer();
  Graph *graph;
};

class tspOrOptRand: public edaMoveGen
{
 public:
  tspOrOptRand();
  tspOrOptRand(const Graph &g);
  tspOrOptRand(const tspOrOptRand &m);
  ~tspOrOptRand();
  edaMoveGen *clone() const;
  virtual bool generate( edaMove *move, const edaSolution &sol );
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_OR_OPT_RAND_);

 private:
  void easer();
  Graph *graph;
};
```

Như vậy với hai kiểu cơ chế phát sinh trên sẽ phục vụ cho quá trình tối ưu theo cơ chế của HC, TS và SA. Chúng ta hoàn toàn có thể tự do đặc tả các cơ chế phát sinh lân cận mới miễn là chúng tuân theo các nguyên tắc trong giao tiếp của `edaMoveGen`.

6.5 ĐẶC TẢ GEN VÀ CƠ CHẾ MÃ HÓA

Đối với các giải thuật tiến hóa theo quy luật di truyền thì gen là thành phần cơ bản để cấu thành nên lời giải. Các nhiễm sắc thể (chromosome) được hình thành trên cơ sở tổ hợp các gen theo một trật tự nhất định và trên cơ sở đó quần thể (population) sẽ được hình thành.

Tùy theo dạng của bài toán tối ưu và phương pháp mã hóa được lựa chọn mà người dùng sẽ lựa chọn kiểu dữ liệu cấu thành gen. Trong bài toán TSP – một dạng của bài toán tối ưu tổ hợp, nhóm hiện thực chọn lựa kiểu dữ liệu cấu thành gen là số nguyên dương tương ứng với ID của thành phố. Như vậy, biến thành phần private

dùng để chứa kiểu dữ liệu cho lớp gen trong bài toán của chúng ta là một số nguyên không âm (`unsigned int`).

```
class tspGenne: public edaGenne {
public:
    tspGenne();
    tspGenne(unsigned int value);
    tspGenne(const tspGenne& genne);
    virtual ~tspGenne();
    tspGenne *clone() const;
    double getValue() const;
    void swap(edaGenne *genne);
    void printOn(ostream& os) const;
    void Serialize(edaBuffer& buf,bool pack);
    tspGenne& operator = (const edaGenne &genne);
    bool operator == (const edaGenne &genne) const;
    setClassID(_USERCLASSID_ + _CLSID_TSP_GENNE_);
private:
    unsigned int value;
};
```

Một phương thức mới cần định nghĩa trong bài toán TSP là `swap()` với tham số là một đối tượng gen khác. Khi phương thức này được thực thi thì giá trị của 2 gen sẽ được hoán đổi cho nhau và điều này là rất cần thiết cho các biến đổi ở cấp độ nhiễm sắc thể khi ràng buộc lời giải là mỗi thành phố chỉ được viếng thăm duy nhất 1 lần.

6.6 CƠ CHẾ MÃ HÓA

Làm thế nào để chuyển đổi một chuỗi các thành phố viếng thăm thành một chuỗi lời giải hay tổng quát hơn là làm thế nào để chuyển đổi một lời giải thành một nhiễm sắc thể ? Đây là câu hỏi mà những ai muốn sử dụng thuật toán GA.

Để trả lời cho câu hỏi trên thực sự là điều không đơn giản và cũng không có một phương pháp nào đủ tổng quát để biến đổi các lời giải cho từng bài toán riêng biệt. Trong bài toán của chúng ta nhận thấy chuỗi tuần tự các thành phố trong lời giải cũng chính là chuỗi tuần tự các gen và việc chuyển đổi lời giải thành các gen hoàn toàn không mất quá nhiều công sức. Tuy nhiên đối với các dạng bài toán phức tạp hơn điều này không hoàn là như vậy, và người dùng phải định nghĩa (đúng như ý đồ xây dựng thư viện). Ta hãy quan sát cấu trúc của lớp `tspRepresentation` được xây dựng trong bài toán:

```
class tspRepresentation: public edaRepresentation {
```

```
public:
  tspRepresentation();
  tspRepresentation(const Graph &gra);
  tspRepresentation(const tspRepresentation& repre);
  virtual ~tspRepresentation();

  tspRepresentation* clone() const;
  void init(const edaSolutionList &list, edaPopulation &pop) const;
  void decode(const edaPopulation &pop, edaSolutionList &list) const;
  void encode(const edaSolutionList &list, edaPopulation &pop) const;
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_TSP_REPRESENTATION_);

private:
  void decode(const edaChromosome *chro, edaSolution *sol) const;
  void encode(const edaSolution *sol, edaChromosome *chro) const;
  void easer();

  Graph *gra;
};
```

Hai phương thức `decode()` và `encode()` được khai báo private có chức năng trái ngược nhau. Một nhận nhiễm sắc thể ở đầu vào và chuyển thành lời giải, và một nhận lời giải và chuyển thành nhiễm sắc thể. Dựa trên các phương thức này, hai phương thức `decode()` và `encode()` tổng quát hơn (được khai báo public) sẽ nhận các thành phần đầu vào là `edaSolutionList` và `edaPopulation`. Chúng có chức năng chuyển một tập các lời giải thành một tập dân cư và ngược lại. Xem xét các thành phần của phương thức này ta thấy quá trình đặc tả việc biến đổi chỉ đơn giản bao gồm một vòng lặp tuần tự duyệt qua các thành phần của thông tin đầu vào thành các thành phần của thông tin đầu ra. Cuối cùng là bước thiết lập giá trị hàm mục tiêu cho giá trị đầu ra.

```
void tspRepresentation::decode(const edaChromosome *chro, edaSolution
*sol) const {
    tspSolution* tspSol = (tspSolution*)sol;
    for(unsigned int i = 0; i < chro->getLength() ; i++ ) {
      tspGenne *genne = (tspGenne*)chro->getGenne(i);
      tspSol->push_back(genne->getValue());
    }
    tspSol->setCost(chro->getFitness());
}

void tspRepresentation::encode(const edaSolution *sol, edaChromosome
*chro) const {
    tspSolution *tspSol = (tspSolution*)sol;
    for(unsigned int i = 0; i < tspSol->size() ; i++) {
      tspGenne genne = (double)tspSol->at(i);
      chro->addGenne(genne);
    }
    chro->setFitness (tspSol->evaluate());
}
```

6.7 XÂY DỰNG WORKFLOW TỐI ƯU

Sau khi đã mô tả bài toán, người dùng sẽ tạo các tác vụ tìm kiếm cần thiết. Hiện tại thư viện hỗ trợ HC, SA, TS, GA hay MA. Sau khi chọn các giải thuật cần thiết, để tạo Workflow, người sử dụng dùng phương thức `insertVertex()` để thêm tác vụ tìm kiếm vào trong đồ thị. Phương thức này trả về ID của các tác vụ tìm kiếm đó. Sau đây ta có 3 ví dụ đưa các giải thuật tối ưu SA, TS và HC vào trong Workflow tìm kiếm:

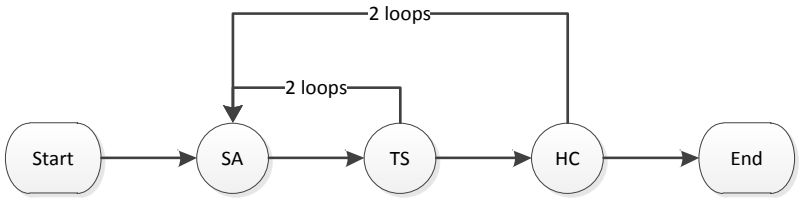
```
int said0 = wfControl.insertVertex (&saSearch);
int tsid1 = wfControl.insertVertex (&tsSearch);
int hcid2 = sfControl.insertVertex (&hcSearch);
```

Sau đó, phương thức `insertEdge()` được sử dụng để mô tả sự phụ thuộc dữ liệu giữa các tác vụ:

```
wfControl.insertEdge(said0, tsid1);
wfControl.insertEdge(tsid1, hcid2);
```

trong đó tác vụ SA được thực hiện đầu tiên, kết quả từ đầu ra của giải thuật này sẽ được tiếp tục tối ưu hóa bởi giải thuật TS và kế tiếp đầu ra từ quá trình tối ưu TS lại được nhận làm đầu vào của quá trình tối ưu bởi HC. Chúng ta có thể bổ sung thêm các quá trình lặp như sau:

```
edaGenContinue nloop(2);
wfControl.insertLoop(tsid1, said0, nloop);
wfControl.insertLoop(hcid2, said0, nloop);
```



Hình 32. Workflow tính toán với ngữ cảnh lặp lồng.

Ở đây, chúng ta sử dùng cùng một điều kiện lặp cho cả 2 quá trình. Vòng lặp thứ 1 sẽ thực thi quá trình tối ưu từ TS đến SA, sau đó lời giải từ SA lại được gửi trở lại giải thuật TS và quá trình tìm kiếm trên sẽ được thực thi lại đến khi điều kiện lặp được thỏa (điều kiện lặp ở đây là quá trình được thực thi 2 lần). Đối với vòng lặp

thứ 2, đây là vòng lặp bao lấy vòng lặp thứ nhất, do vậy cơ chế hoạt động của nó sẽ phức tạp hơn.

Sau khi kết thúc vòng lặp thứ 1, lời giải được tối ưu sẽ được gửi đến giải thuật HC để tối ưu, và bước tiếp theo nó sẽ được gửi lại giải thuật SA do điều kiện lặp được thiết lập trước đó. Những vòng lặp lồng được tái khởi động lại và như vậy lời giải sẽ được tối ưu 2 lần theo quy trình SA đến TS do vòng lặp thứ 1 xác lập (xem hình 5.2).

Một thành phần quan trọng khác cần được định nghĩa trong quá trình tìm kiếm, đó chính là phương thức `userClassGenerate()`, chúng giúp tái tạo các đối tượng được gởi đi trong quá trình truyền thông điệp trong thư viện. Đối với các đối tượng thuộc tầng Search Algorithm, việc định nghĩa phương thức `userClassGenerate()` đã được thực hiện trong lớp `edaSerialize`, tuy nhiên đối với những lớp do người dùng tự định nghĩa tại tầng Application, chúng ta cần phải định nghĩa cách thức khởi tạo tại hàm `userClassGenerate()` ở chương trình chính.

```
edaSerialize* userClassGenerate (int clsid)
{
  switch (clsid)
  {
    case _USERCLASSID_ + _CLSID_TWO_OPT_:
      return new tspTwoOpt;

    case _USERCLASSID_ + _CLSID_TWO_OPT_NEXT_:
      return new tspTwoOptNext;

    case _USERCLASSID_ + _CLSID_TSP_SOLUTION_:
      return new tspSolution;

    case _USERCLASSID_ + _CLSID_TWO_OPT_MOVE_RANDOM_:
      return new tspTwoOptRand;

    case _USERCLASSID_ + _CLSID_GRAPH_:
      return new Graph;

    case _USERCLASSID_ + _CLSID_TSP_REPRESENTATION_:
      return new tspRepresentation;

    case _USERCLASSID_ + _CLSID_TSP_GENNE_:
      return new tspGenne;

    default:
      std::
        cerr << "Unknown classId " << clsid << " for object
generation!!!"
             << endl;
```

```
        exit (-1);
    }
}
```

Phương thức bên trên định nghĩa lại các thức khởi tạo các lớp đối tượng `tspTwoOpt`, `tspTwoOptNext`, `tspSolution`, `tspTwoOptRand`, `tspGenne`, `Graph`, `tspRepresentation`. Đây là các lớp đặc thù cho bài toán TSP đã được định nghĩa ở bên trên.

Khi đã xây dựng xong bài toán, ta gọi phương thức `search()` của workflow để thực thi việc tìm kiếm. Ta cần phải cung cấp nghiệm ban đầu cho phương thức, và khi kết thúc, biến này sẽ chứa nghiệm đã được cải thiện thông qua các thuật toán tối ưu.

6.8 MÔI TRƯỜNG THỬ NGHIỆM

Để thử nghiệm môi trường tính toán song song cho thư viện, nhóm nghiên cứu cần một máy tính song song đã có thiết lập chạy được thư viện MPI. Do hệ thống supernode2 [Nam07] đã cũ, và không thể vận hành trơn tru (có thể thực hiện tính toán trong thời gian lớn), nên nhóm nghiên cứu đã chọn một máy chủ tính toán dạng rack có cấu hình như sau:

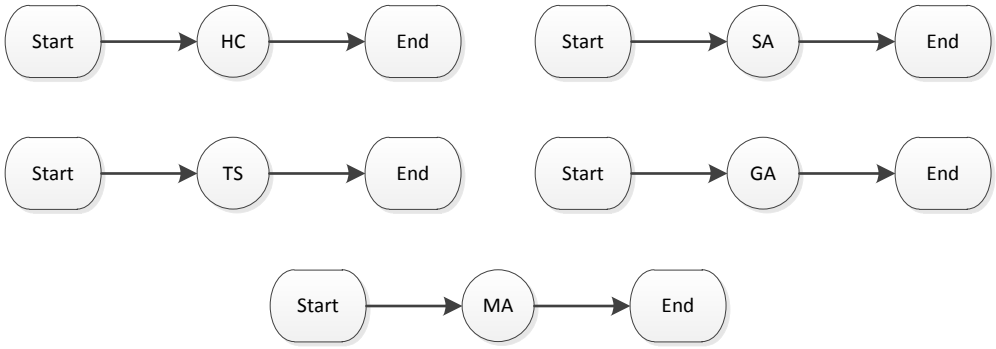
- Bộ xử lý: 2 x Xeon® E5-2660 2.20GHz, 20M Cache, 8.0GT/s QPI, Turbo, 8C, 95W, Max Mem 1600MHz,
- Bộ nhớ: 8 x 16GB RDIMM, 1600 MHz, Standard Volt, Dual Rank,
- Đĩa cứng: RAID 1 (2 HDDs), 600GB 15K RPM SAS 6Gbps 3.5in Hot-plug HDD,
- Giao tiếp mạng: Network Controller: 01 x Gigabit Server Adapter, 01 x 10Gigabit Server Adapter,
- Bộ xử lý đồ hoạ (không được sử dụng): 1 x NVIDIA® TESLA™ M2090 GPU Computing Module PCIe.

Với cấu hình như trên, chúng tôi có thể chạy 32 quá trình MPI đồng thời (nếu không có vấn đề gì về bộ nhớ).

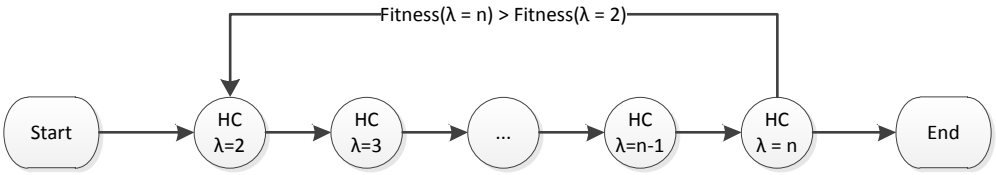
6.9 KẾT QUẢ THỬ NGHIỆM VÀ HIỆU NĂNG

Để đánh giá các hiệu năng của thư viện chúng tôi chọn và thử nghiệm 16 sơ đồ tính toán khác nhau bao gồm cả các dạng sơ đồ tính toán song song và tuần tự. Trong đó gồm 6 sơ đồ tuần tự với các giải thuật meta-heuristic đơn lẽ như được trình bày ở Hình 33, 10 sơ đồ còn lại (từ Hình 34 đến Hình 39) là sự kết nối các giải thuật meta-heuristic theo các chiến lược khác nhau. Mục đích việc đưa các sơ đồ tính toán nhằm minh họa sức mạnh và khả năng tùy biến của thư viện.

Bộ dữ liệu thử nghiệm (benchmark) được lấy từ nguồn thuộc Đại học Waterloo[1], Canada. Đây là dữ liệu được sử dụng trong nhiều nghiên cứu kinh điển về bài toán *NP* và đặc biệt là bài toán TSP. Kết quả thử nghiệm với các workflow tính toán của nhóm nghiên cứu sẽ so sánh với các kết quả tối ưu được công bố trên trang của Đại học Waterloo. Hiện tại nhóm đã hoàn tất thử nghiệm trên các benchmark sau berlin52, eil101 và ali535.

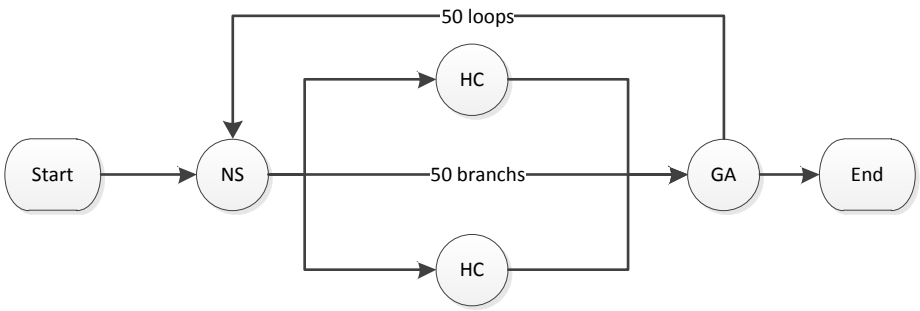


Hình 33. Các sơ đồ tối ưu đối với từng loại giải thuật meta-heuristic đơn lẽ.

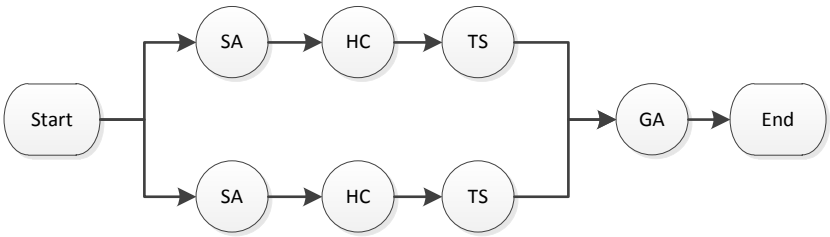


Hình 34. Hiện thực giải thuật VNS bằng giải thuật HC kết hợp với sự thay thông số λ của toán tử Or-Opt.

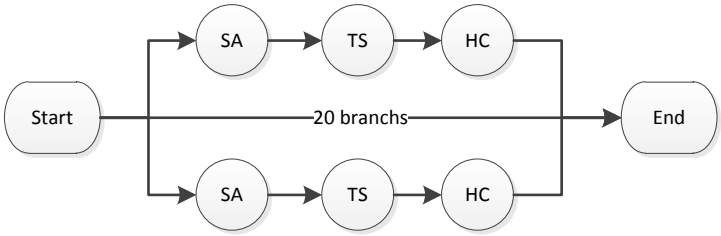
[1] http://www.math.uwaterloo.ca/tsp/concorde/benchmarks/bench.html



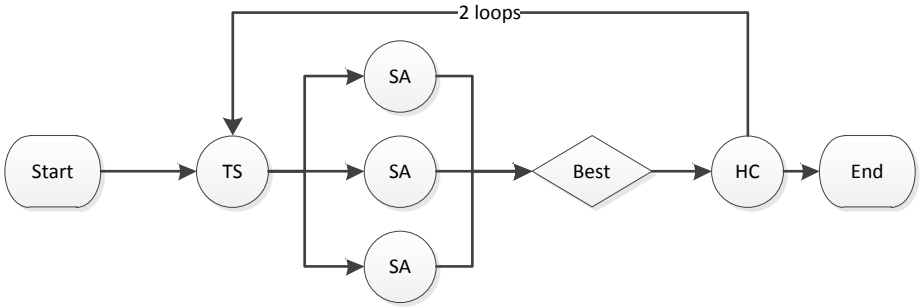
Hình 35. Hiện thực giải thuật MA song song (MPIMA) bằng workflow trên nền tảng kết hợp giải thuật HC và GA.



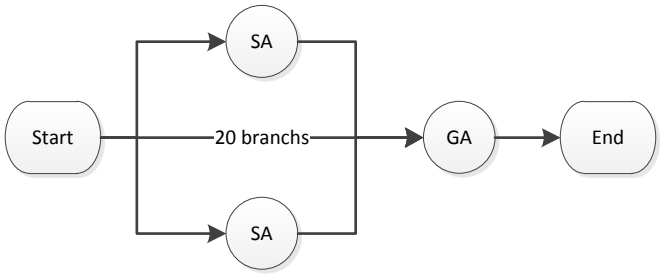
Hình 36. Workflow của SEQ01 & MPI01.



Hình 37. Workflow của SEQ02 & MPI02.



Hình 38. Workflow của SEQ03 & MPI03.



Hình 39. Workflow của SEQ04 & MPI04.

Việc lựa chọn các tham số thử nghiệm cho từng loại thuật toán sẽ có ảnh hưởng quan trọng đến độ hiệu quả của từng thuật toán đồng thời cũng ảnh hưởng đến kết quả tối ưu toàn cục trên workflow tính toán. Qua thực thi và chọn lựa các cấu hình phù hợp, nhóm nghiên cứu đã đưa ra các thông số cho từng loại giải thuật meta-heuristic và các workflow tính toán như được trình trong Bảng 2.

Bảng 2. Các thông số được thiết lập cho các workflow tính toán.

	Số lời giải	Số lần lặp	Chiến lược
HC	40	1500	First Best + 2-Opt hoặc Or-Opt
SA	40	1500	ExpCooling(0.01, 0.98) + 2-Opt + T(50°)
TS	40	1500	Global Best + 2-Opt
GA	40	1500	Roulette Wheel (0.5) + PMX + Swap(0.05)
MA	40	80	Roulette Wheel (0.5) + PMX + Swap(0.05) + HC(2-Opt)
MPIMA	1/40	80	Roulette Wheel (0.5) + PMX + Swap(0.05) + HC(2-Opt) + 20 Slave
VNS	40	Var	HC + 5 x Or-Opt
SEQ01	40	1	SA + TS + HC + GA
MPI01	40	1	SA + TS + HC + GA + 2 Slave
SEQ02	20	1	SA + TS + HC
MPI02	1/20	1	SA + TS + HC + 20 Slave
SEQ03	1	2	TS + SA + HC
MPI03	1	2	TS + SA + HC + 3 Slave
SEQ04	40	1	SA + GA
MPI04	1/40	1	SA + GA + 20 Slave

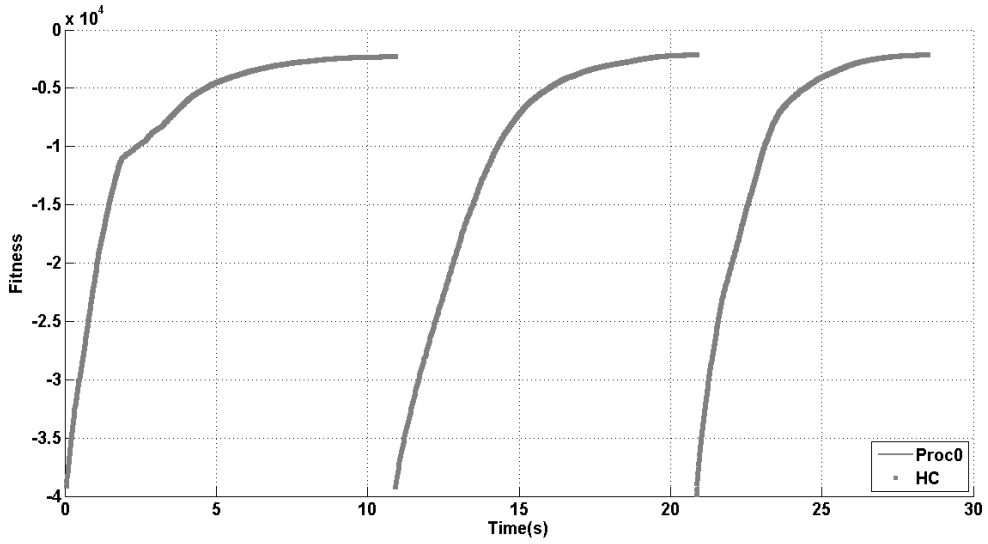
Số lời giải được thiết lập cho từng sơ đồ tối ưu thông thường được chọn là 40. Với sự lựa chọn này, các kết quả thông kê cho những giải thuật tối ưu sẽ rơi vào phân bố chuẩn ($n \geq 30$) theo định lý giới hạn trung tâm. Và như vậy giá trị trung bình và ước lượng không chệch của độ lệch chuẩn của hàm fitness sẽ có khả năng

đại diện cho kỳ vọng và độ lệch chuẩn của giải thuật. Trong mỗi sơ đồ tính toán khi hoàn tất điều được ghi lại trong log (nhật ký) các thông tin thống kê cho toàn danh sách lời giải được tối ưu bao gồm: lời giải tốt nhất, lời giải kém nhất và giá trị trung bình, thời gian chạy.

Điều kiện dừng (số lần lặp) được chọn với các giải thuật dựa trên dịch chuyển (HC, SA, TS), là 1500 lần. Riêng 2 giải thuật dựa trên sự dịch chuyển của quần thể là GA và MA có số thế hệ được chọn là 40. Với các workflow tính toán dạng tổ hợp điều kiện dừng cho từng thành phần meta-heuristic bên trong nó có cùng điều kiện dừng bên trên. Tuy nhiên, tùy theo sơ đồ mà chúng còn có điều kiện lặp riêng cho từng thành phần ghép nối như trong sơ đồ VNS (Hình 34), MPIMA(Hình 35), SEQ03 và MPI03 (Hình 38).

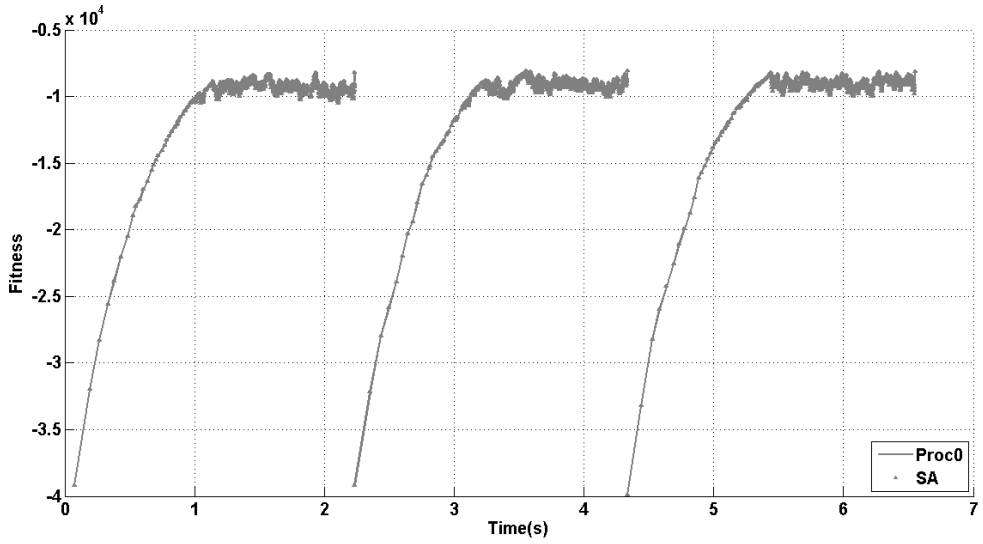
Bảng 3. Kết quả tối ưu với các workflow tính toán khác nhau trên dữ liệu berlin52.

Workflow	Thời gian tính (giây)	Giá trị hàm mục tiêu				
		Tốt nhất	Xấu nhất	Trung bình	Độ lệch chuẩn	Tối ưu
HC 2-Opt	0.25	-7935	-9284	-8545	333	-7544
HC Or-Opt	0.21	-9356	-12680	-10816	812	
SA	4.85	-7692	-9144	-8210	361	
TS	4.95	-7875	-9429	-8540	396	
GA	0.82	-10978	-13985	-11126	569	
MA	0.55	-7544	-7544	-7544	0	
MPIMA	1.86	-7544	-10721	-7703	595	
VNS	0.20	-8686	-11344	-9676	539	
SEQ01	9.96	-7544	-8111	-7559	88	
MPI01	7.27	-7545	-8126	-7559	91	
SEQ02	4.35	-7634	-9218	-8231	366	
MPI02	2.02	-7952	-8375	-8062	131	
SEQ03	19.06	-7544	-7544	-7544	0	
MPI03	10.97	-7544	-7544	-7544	0	
SEQ04	2.64	-7544	-7544	-7544	0	
MPI04	1.81	-7926	-11047	-8261	843	

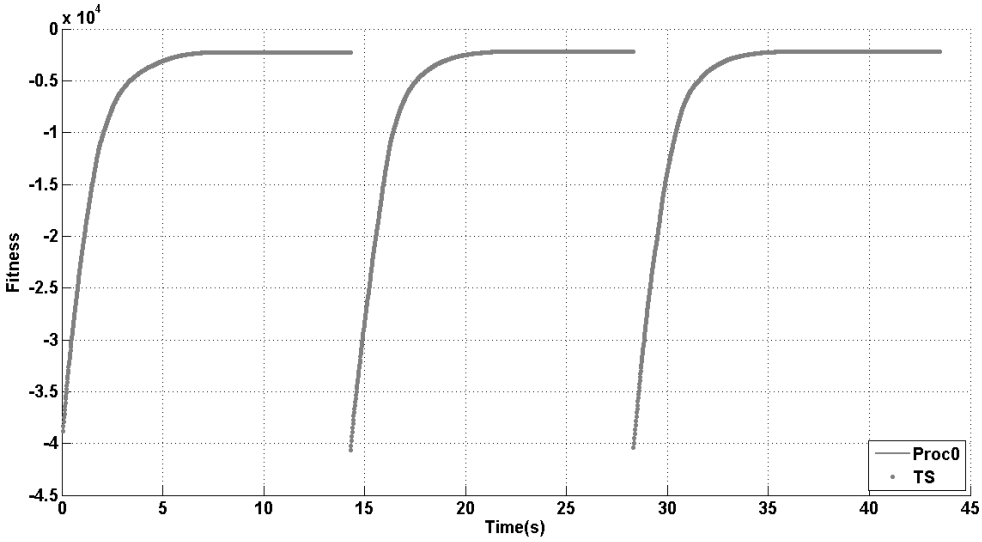


Hình 40. Quá trình tối ưu của giải thuật HC (2-Opt) với tập benchmark ali535.

Với giải thuật HC, chiến lược First Best được chọn cho các sơ đồ tối ưu. Cả hai loại lân cận 2-Opt và Or-Opt điều được chọn để thử nghiệm. So sánh các kết quả tính toán trên các tập benchmark đều cho thấy hiện thực bài toán TSP từ toán tử 2-Opt luôn đạt kết quả tốt hơn Or-Opt (trong cả 3 trường hợp giá trị trung bình của hàm mục tiêu của 2-Opt luôn cao hơn Or-Opt – chi tiết từ Bảng 3 đến Bảng 5).

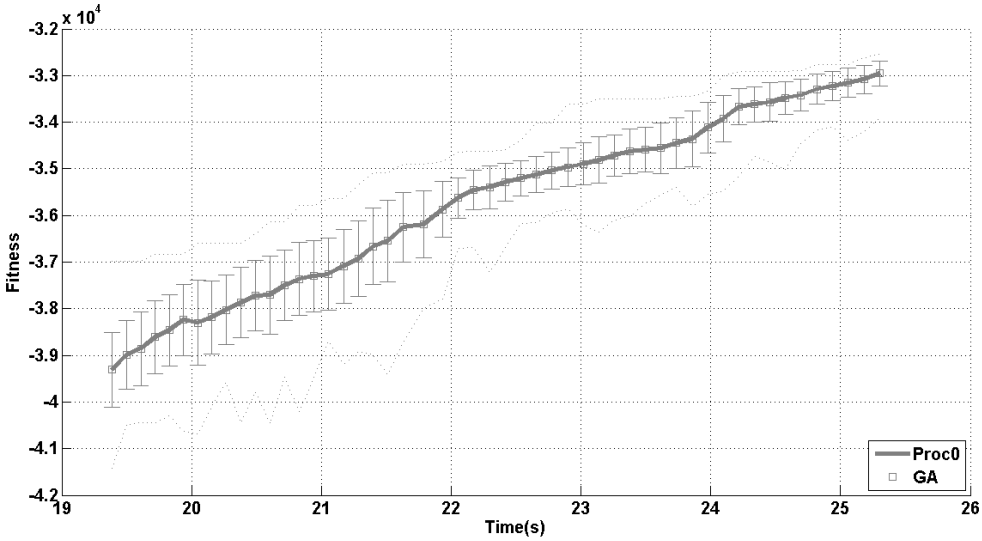


Hình 41. Quá trình tối ưu của giải thuật SA cho tập benchmark ali535.



Hình 42. Quá trình tối ưu của giải thuật TS cho tập benchmark ali535.

Các thông số thiết lập cho sơ đồ TS và SA tương tự như giải thuật HC. Tuy nhiên chiến lược chọn lân cận trong TS là Global Best, điều này đồng nghĩa với việc giải thuật phải quét qua toàn bộ không gian lân cận mới có thể xác định được bước chuyển nào là tốt nhất. Còn với SA, chiến lược giảm nhiệt độ được lựa chọn là ExpCooling.



Hình 43. Quá trình tối ưu của giải thuật GA cho tập benchmark ali535.

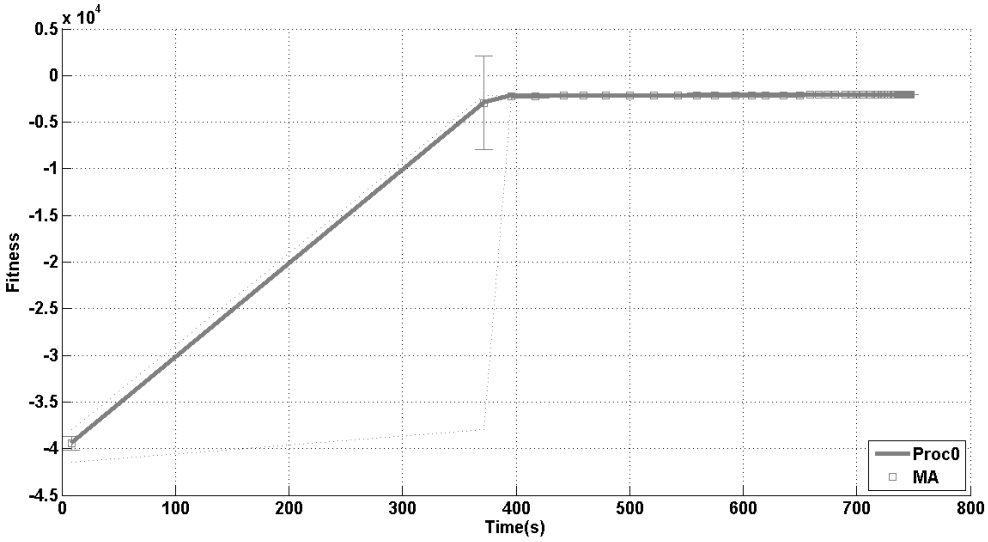
Hai giải thuật mới được hiện thực trong phạm vi nghiên cứu của đề tài là GA và MA.Về kết quả tính toán, kết quả từ GA vẫn còn nhiều hạn chế so với các giải thuật khác, tuy nhiên MA lại là một điểm sáng của đề tài. Giải thuật memetic trong bài toán TSP hoàn toàn tương tự GA ngoài việc thêm vào toán tử 2-Opt được lồng bên trong quá trình cải tiến lời giải. Trong tất cả các trường hợp, kết quả tối ưu do MA thực hiện rất hiện quả và tiến rất sát đến lời giải tối ưu toàn cục. Đặc biệt hơn, việc hiện thuật giải thuật MA bằng sơ đồ song song (Hình 45) sẽ rút ngắn thời gian tính toán một cách đáng kể và điều này thực sự rất có tiềm năng đối các bài toán lớn khi hàm lượng giá có giá trị trọng số lớn trong bài toán.

Bảng 4. Kết quả tối ưu với các workflow tính toán khác nhau trên dữ liệu eil101.

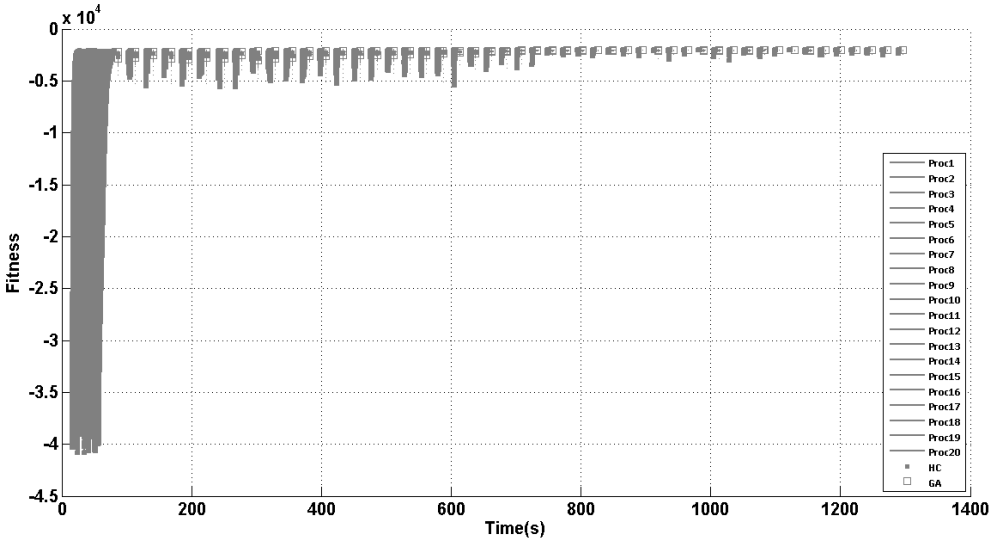
Workflow	Thời gian tính (giây)	Giá trị hàm mục tiêu				
		Tốt nhất	Xấu nhất	Trung bình	Độ lệch chuẩn	Tối ưu
HC 2-Opt	1.15	-671	-1597	-744	173	-641
HC Or-Opt	0.82	-834	-1806	-929	165	
SA	4.85	-7692	-9144	-8210	361	
TS	17.26	-668	-757	-716	20	
GA	2.63	-1354	-1502	-1365	29	
MA	3.78	-644	-654	-646	3	
MPIMA	2.32	-648	-702	-650	9	
VNS	1.80	-729	-948	-827	49	
SEQ01	35.00	-673	-876	-687	40	
MPI01	17.92	-669	-791	-673	20	
SEQ02	24.51	-687	-739	-716	16	
MPI02	3.64	-685	-726	-705	12	
SEQ03	61.87	-683	-683	-683	0	
MPI03	29.82	-679	-679	-679	0	
SEQ04	7.39	-1044	-1156	-1050	24	
MPI04	3.22	-1070	-1070	-1070	0	

Workflow VNS là sự kết hợp tuần tự của các giải thuật HC với chiến lược thay đổi (biến đổi) kiểu láng giềng một cách tuần tự, như trong Hình 34. Trong hiện thực của toán tử Or-Opt, thông số λ có thể thay đổi tùy theo các thiết lập ban đầu của người dùng. Và đó là lý do Or-Opt được chọn để thực hiện giải thuật VNS. Chiến lược thay đổi một cách liên tục việc chọn lựa tập lân cận giúp quá trình tối ưu có thể

vượt qua các vùng tối ưu cục bộ. Kết quả này được thể hiện trong các từ trong các Bảng 3 đến Bảng 5. Ở cả ba trường hợp thử nghiệm, hiệu suất tối ưu trung bình đều được cải thiện một cách rõ rệt.



Hình 44. Quá trình tối ưu của MA cho tập benchmark ali535.



Hình 45. Quá trình tối ưu của MPIMA cho tập benchmark ali535.

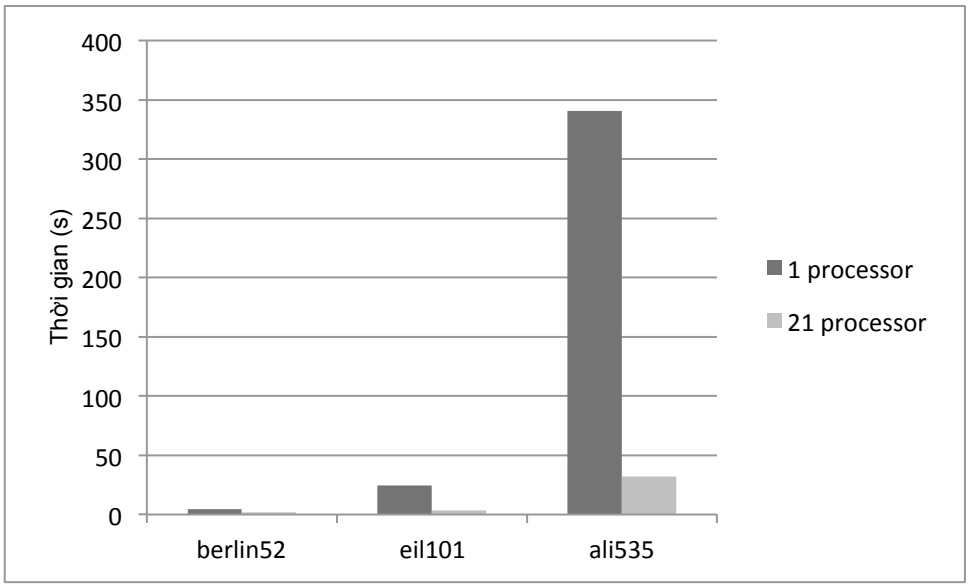
Việc kết hợp các giải thuật HC để tạo ra thuật toán VNS cho thấy khả năng hiện thực các giải thuật mới, hiệu quả hơn trên hệ thống thư viện là rất đơn giản. Đó đơn thuần chỉ là việc ghép các phần tử sẵn có và thay đổi các thông số của các thành phần trong giải thuật một cách phù hợp.

Bảng 5. Kết quả tối ưu với các workflow tính toán khác nhau trên dữ liệu ali535.

Workflow	Thời gian tính (giây)	Giá trị hàm mục tiêu				
		Tốt nhất	Xấu nhất	Trung bình	Độ lệch chuẩn	Tối ưu
HC 2-Opt	16	-11493	-39363	-21548	7405	-2023
HC Or-Opt	2	-17832	-38374	-24335	5273	
SA	28	-7670	-8412	-8042	155	
TS	415	-2131	-2457	-2227	58	
GA	42	-21663	-22010	-21678	62	
MA	1070	-2036	-2817	-2117	151	
MPIMA	125	-2060	-2088	-2061	6	
VNS	394	-2693	-3579	-3101	202	
SEQ01	832	-2123	-2601	-2183	148	
MPI01	288	-2160	-3297	-2222	230	
SEQ02	341	-2162	-2391	-2239	55	
MPI02	32	-2297	-2410	-2356	28	
SEQ03	776	-2129	-2129	-2129	0	
MPI03	448	-2156	-2156	-2156	0	
SEQ04	58	-7353	-8231	-7417	207	
MPI04	35	-7401	-7600	-7411	43	

Với giải thuật SA, thông số được thiết lập cho giải thuật SA bao gồm: toán tử 2-Opt, nhiệt độ ban đầu là 50°, hàm tăng nhiệt độ dạng mũ với tỷ lệ giảm nhiệt độ là 0.98 và ngưỡng là 0.01. Giá trị nhiệt độ được là không quá cao để tránh sự không ổn định (dao động mạnh) trong quá trính tối ưu.

Hai giải thuật mới được hiện thực trong phạm vi nghiên cứu của đề tài là GA và MA. Về kết quả tính toán, kết quả từ GA vẫn còn nhiều hạn chế so với các giải thuật khác, tuy nhiên MA lại là một điểm nổi bật của đề tài. Giải thuật MA trong bài toán TSP hoàn toàn tương tự GA ngoài việc thêm vào toán tử 2-Opt được lồng bên trong quá trình cải tiến lời giải. Trong tất cả các trường hợp, kết quả tối ưu do MA thực hiện rất hiện quả và tiến rất sát đến lời giải tối ưu toàn cục. Đặc biệt hơn, việc hiện thực giải thuật MA bằng sơ đồ song song trong Hình 35 sẽ rút ngắn thời gian tính toán một cách đáng kể và điều này thực sự rất có tiềm năng đối các bài toán lớn khi hàm lượng giá trọng số tính toán lớn trong bài toán.



Hình 46. So sánh thời gian tối ưu giữa workflow SEQ02 (1 processor) và MPI02 (21 processors) với 3 tập benchmark berlin52, eil101 và ali535.

Ngoài các sơ đồ tối ưu đơn lẻ cho từng loại giải thuật, trong bài toán TSP, nhóm nghiên cứu còn xây dựng 4 workflow tối ưu như Hình 36, Hình 37, Hình 38 và Hình 39. Chúng là sự kết hợp của các giải thuật meta-heuristic riêng lẽ cùng với các điều kiện chọn và vòng lặp để tạo nên các chiến lược tối ưu mới với mục tiêu nâng cao hiệu quả của quá trình tìm kiếm. Kết quả tối ưu của các lược đồ này được liệt kê trong Bảng 3, Bảng 4 và Bảng 5. Các kết quả trên là các minh chứng rõ ràng nhất về hiệu quả mà các lược đồ tối ưu mang lại về cả chất lượng nghiệm và thời gian tính toán (Hình 46).

CHƯƠNG 7: ỨNG DỤNG TRONG TỐI ƯU THỰC TẾ

Ở phần nội dung này chúng ta sẽ trình bày các kết quả ứng dụng thư viện vào các bài toán tối ưu thực tế với không gian tìm kiếm có độ phức tạp cao cũng như thảo luận sâu hơn về khả năng tùy biến của các thành phần trong thư viện. Hai ứng dụng được quan tâm là "*Bài toán chuyên chở chất thải nguy hại*" và "*Bài toán tối ưu hóa biên dạng cánh*".

Bước đầu thử nghiệm, cả hai bài toán đã cho ra một kết quả đáng khích lệ, chứng minh hướng tiếp cận này là khả thi. Kết quả áp dụng GA, MA cho bài toán "chuyên chở chất thải có nguy hại" đã được thực hiện thành công và đã được công bố trên tạp chí LNCS số 7971. Đối với bài toán "biên dạng cánh" , các kết quả công bố trong luận văn, tuy vẫn còn rất đơn giản (chỉ dừng lại ở mức sử dụng các giải thuật meta-heuristic một cách đơn lẽ trong thư viện vào quy trình tối ưu) nhưng vẫn đạt được các kết quả rất khả quan (so với bài báo đã công bố [Khai11]). Tuy nhiên để minh họa được sự linh động cũng như sức mạnh tính toán của thư viện, nhóm tiếp tục thực các nghiên cứu chuyên sâu hơn về việc sử dụng workflow kết hợp các meta-heuristic lại với nhau nhằm cải thiện hơn nữa hiệu năng của quy trình tối ưu.

7.1 HỖ TRỢ CỦA THƯ VIỆN

Trong thư viện meta-heuristic, hiện tại các giải thuật HC, SA, TS, GA, MA đã được hỗ trợ. Các giải thuật có thể kết hợp với nhau thông qua workflow. Hiện tại ngoài 2 cấu trúc chính là song song và nối tiếp, thư viện cũng đã tích hợp thêm một số cấu trúc phức tạp hơn như cấu trúc lặp và điều kiện lặp nhằm gia tăng khả năng tùy biến giúp người dùng có thể xây dựng các tiến trình tối ưu mong muốn. Hơn nữa, thư viện cũng đã cho phép các thành phần tối ưu lựa chọn một hoặc nhiều đầu vào (kể cả tùy chỉnh theo mong muốn), và đây cũng là thêm một cách để người dùng định hướng tìm kiếm nghiệm.

Thư viện hiện tại không chỉ chạy được trên hệ thống tính toán lưới với POP-C++ (hiện nay không còn môi trường để kiểm chứng) mà còn có thể chạy trên hệ thống máy tính cụm sử dụng thư viện lập trình truyền thông điệp MPI. Thư viện có thể

chạy trên các hệ thống tính toán mạnh, dùng xử lý song song cho việc tìm kiếm lời giải tối ưu trong một không gian nghiệm lớn là điều hết sức cần thiết, giúp tăng tốc đáng kể thời gian chạy giải thuật.

7.2 BÀI TOÁN VRPTWC

Thành phố Hồ Chí Minh gồm hơn 10 khu công nghiệp với hơn 100 nhà máy khác nhau là chủ nguồn thải các chất thải nguy hại (CTNH). Mỗi nhà máy có một khung thời gian là thời gian sớm nhất và muộn nhất mà phương tiện có thể bắt đầu thu gom tại nhà máy. Nếu phương tiện đến sớm hơn thì có thể chờ cho đến lúc có thể bắt đầu thu gom. Thời gian để hoàn tất việc thu gom cũng được biết trước dựa trên từng nhà máy khác nhau. Mỗi nhà máy có thể là chủ của một hoặc nhiều CTNH khác nhau. Các chất thải khác nhau có thể được chở chung trên cùng một phương tiện hoặc không thể chở chung theo quy định của việc vận chuyển và thu gom CTNH của viện tài nguyên và môi trường TPHCM.

Việc thu gom và vận chuyển CTNH do công ty vận chuyển CTNH thực hiện để mang CTNH về các trạm xử lý. Công ty vận chuyển có một địa điểm xuất phát cho tất cả các phương tiện vận chuyển gọi là điểm xuất phát. Điểm xuất phát cũng là điểm mà tất cả các phương tiện phải trở về vào cuối ngày sau khi thu gom. Phương tiện vận chuyển chỉ trở về điểm xuất phát sau khi tất cả các CTNH trên xe đã được mang đến và bốc dỡ tại các trạm xử lí. Điểm xuất phát cũng có khung thời gian là thời gian sớm nhất mà các phương tiện có thể xuất phát và thời gian trễ nhất mà các phương tiện có thể quay về điểm xuất phát.

Các trạm xử lí là các địa điểm mà các công ty kinh doanh CTNH thực hiện việc tái chế, xử lí, tái sử dụng các CTNH. Nhiệm vụ các phương tiện vận chuyển là mang các CTNH đến các trạm xử lí và bốc dỡ CTNH. Các trạm xử lí cũng có khung thời gian cho phép các phương tiện vào bốc dỡ CTNH tương tự như các nhà máy.

Các phương tiện vận chuyển do các công ty vận chuyển sở hữu. Mỗi phương tiện có khối lượng tối đa CTNH mà phương tiện có thể chở. CTNH là do các nhà máy thải ra theo từng ngày với khối lượng có thể khác nhau trên từng loại chất thải.

Các chất thải được phân loại để có thể chuyên chở trên cùng một phương tiện hoặc khác phương tiện theo quy định của việc thu gom và vận chuyển CTNH.

Vào đầu mỗi ngày, các phương tiện xuất phát từ điểm xuất phát và tiến hành thu gom các chất thải từ các nguồn thải, sau khi các phương tiện không thể thêm do giới hạn về tổng khối lượng chất thải có thể chở, các phương tiện ghé qua trạm xử lí, tiến hành bốc dỡ CTNH và tiếp tục thu gom. Việc thu gom và vận chuyển các chất thải nguy hại phải đảm bảo được trong khung thời gian của các nhà máy, các trạm xử lí và của điểm xuất phát. Đồng thời, các CTNH được chở trên cùng một phương tiện phải theo đúng quy định của việc thu gom và vận chuyển CTNH.

7.2.1 Phát biểu của bài toán:

Yêu cầu của bài toán là lập lộ trình vận chuyển cho các phương tiện trong một ngày để đáp ứng nhu cầu của các chủ nguồn thải, thỏa mãn các ràng buộc và quy định vận chuyển nhằm tối thiểu hóa số phương tiện sử dụng và khoảng cách di chuyển của các phương tiện.

Các ràng buộc chính khi tạo ra một lộ trình cho phương tiện di chuyển là:

- *Ràng buộc trên phương tiện*: là tổng khối lượng CTNH tối đa mà phương tiện có thể chở. Sau khi không thể chở thêm, phương tiện phải ghé các trạm xử lí để bốc dỡ CTNH trước khi tiếp tục thu gom.
- *Ràng buộc lộ trình*: là tổng chiều dài tối đa mà một phương tiện có thể di chuyển trong ngày. Ràng buộc này nhằm hạn chế khoảng cách di chuyển của phương tiện, đáp ứng nhu cầu về nhiên liệu trong ngày.
- *Ràng buộc trên CTNH*: là sự xung đột của các CTNH khi chở chung trên cùng một phương tiện. Do đặc trưng của việc vận chuyển CTNH, không phải tất cả các CTNH có thể được chở chung trên cùng một phương tiện. Đây chính là điểm mới của bài so với các bài toán đã nghiên cứu.
- *Ràng buộc về khung thời gian*: là do khung thời gian quy định tại các nhà máy, các trạm xử cũng như thời gian sớm nhất mà muộn nhất mà các phương tiện có thể xuất phát và trở về điểm xuất phát vào cuối ngày.

Mục tiêu chính của bài toán theo thứ tự ưu tiên là:

- Tối thiểu hóa tổng khoảng cách và thời gian di chuyển của các phương tiện.
- Tối thiểu tổng hóa thời gian chờ của các phương tiện.
- Tối thiểu hóa số phương tiện sử dụng.
- Phương tiện không nên chở số lượng CTNH lớn để đi thu gom với khoảng cách xa.

Với các mục tiêu trên cùng với các ràng buộc đặc trưng cho bài toán vận chuyển chất thải nguy hại trong công nghiệp, lớp đối tượng `vrpProblem` được xây dựng để chứa được các thông tin này. Các thuộc tính `_num_stops`, `_speed`, `_capacity_veh`, `_max_total_dist`, `_max_total_stop`, `_conflict_group` chứa các thông tin về ràng buộc của bài toán bao gồm số lượng nhà máy cần thu gom, tốc độ di chuyển của phương tiện, quãng đường tối đa mà phương tiện có thể di chuyển, số lượng tối đa các nhà máy mà mỗi phương tiện có thể ghé qua và ma trận thông tin về sự xung đột chất thải công nghiệp. Đối với thuộc tính `_dist`, đây là một ma trận vuông có kích cỡ tương ứng với số đỉnh của đồ thị và chứa thông tin khoảng cách Manhattan giữa 2 đỉnh bất kỳ. Thuộc tính `_list_stops` thuộc lớp đối tượng `vrpStop` sẽ chứa thông tin các trạm được đọc từ tập tin dữ liệu ban đầu. Qua hai minh họa được trình bày, ta có thể nhận thấy ***các lớp được kết thừa từ kiểu trừu tượng `edaProblem` có nhiệm vụ đúc kết nguồn thông tin từ tập tin dữ liệu đầu vào của bài toán***.

```
class vrpProblem: public edaProblem {
public:
    vrpProblem();
    vrpProblem(const vrpProblem &problem);
    vrpProblem(const char *filename);
    virtual ~vrpProblem();
    vrpProblem* clone() const;
    void load(const char* filename);
    vrpProblem& operator = (const edaProblem &pro);
    vector<vrpStop*> getLandfill() const;
    vector<vrpStop*> getRegularStop() const;
    vrpStop* getDepot() const;
    vrpStop* getStop(const unsigned int id) const;
    unsigned int getNumStops() const;
    double getMaxCapacity() const;
    double getMaxTotalDist() const;
    double getDist(const unsigned int id_1, const unsigned int id_2)
const;
    double getTime(const unsigned int id_1, const unsigned int id_2)
const;
    bool isConflict(const unsigned int group,
                    const vector<unsigned int>& groups) const;
    bool isConflict(const vector<unsigned int>& groups) const;
    void printOn(ostream& os) const;
```

```
    void printTable(const char* file_name);
    void Serialize(edaBuffer &buf, bool pack);
    setClassID(_USERCLASSID_ + _CLSID_VRP_PROBLEM_);

private:
    unsigned int _num_stops;
    double _speed;
    double _capacity_veh; //the capacity of the vehicle
    double _max_total_dist;
    double _max_total_stop;
    double *_dist; //the distance between 2 stops
    vrpGroupConflict *_conflict_group; // the conflict group in problem
    vrpStop *_list_stops; //list stops have num_stops

    void computeDistances();
    double getMinutes(const double hour_minute) const;
    void easer();
};
```

Một trong những tổ chức dữ liệu cần quan tâm trong bài toán VRP là hai lớp đối tượng `vrpStop` và `vrpGroupConflict`. Không đơn giản chỉ chứa thông tên về ID và toạ độ các đỉnh như bài toán TSP, thông tin của mỗi đỉnh trong bài toán VRP được nghiên cứu ở đây còn có loại chất thải, khối lượng, thời gian quy định mà phương tiện cần đến để thu gom chất thải cũng như các tính chất riêng biệt của từng trạm mà phương tiện ghé qua (bãi chứa phương tiện, nhà máy xử lý chất thải và nơi thu gom chất thải). Do vậy, lớp đối tượng `vrpStop` được tổ chức để gom nhóm các thông tin trên. Những thông tin cần đối với một đối tượng kiểu bao gồm ID, tọa độ địa lý (X, Y), lượng chất thải cần thu gom, khung thời gian có thể thu gom, nhóm chất thải và tính chất của trạm. Các thông tin này được truy xuất trực tiếp do được khai báo với tầm vực public và điều này sẽ mang đến những tiện lợi nhất định trong quá trình mã hóa và giải mã sẽ được trình bày ở phần sau.

```
class vrpStop : public edaSerialize {
public:
  vrpStop();
  vrpStop(const vrpStop& stop);
  vrpStop *clone() const;
  virtual ~vrpStop();
  void printOn(ostream& os) const;
  void debug(ostream& os) const;
  vrpStop& operator =(const vrpStop &stop);
  bool operator ==(const vrpStop &stop) const;
  void Serialize(edaBuffer &buf, bool pack);
  setClassID(_USERCLASSID_ + _CLSID_VRP_STOP_);

public:
  unsigned int ID; //index in original stop
  double X; // X coordinate
  double Y; // Y coordinate
```

```
  double Load; // amount waste at stop
  double EarlyTime; // early time window
  double LateTime; // late time window
  double ServiceTime; //time for unload and load waste
  unsigned int Type; //type of stop ( regular_stop , landfill, depot)
  unsigned int Group; //group of regular stop
};
```

Đối với tổ chức của `vrpGroupConflict`, các thuộc tính bên dưới của lớp này chỉ đơn giản là biến `_size` chứa số loại chất thải và ma trận vuông `_matrix` chứa thông tin các loại chất thải xung đột. Đối tượng này sẽ hỗ trợ việc kiểm tra chất thải đang quan tâm có xung đột hay không với nhóm chất thải hiện đang chuyên chở trên phương tiện thông qua phương thức `isConflict()`.

```
class vrpGroupConflict: public edaSerialize  {
public:
    vrpGroupConflict();
    vrpGroupConflict(const vrpGroupConflict& group);
    vrpGroupConflict(const unsigned int size);
    vrpGroupConflict *clone() const;
    virtual ~vrpGroupConflict();
    void pushConflictGroup(unsigned int group_A,
                           unsigned int group_B);
    bool isConflict(const unsigned int group,
                    const vector<unsigned int>& groups) const;
    bool isConflict(const vector<unsigned int>& groups) const;
    void prinOn(ostream& os) const;
    void Serialize(edaBuffer &buf, bool pack);
    setClassID(_USERCLASSID_ + _CLSID_VRP_GROUP_CONFLICT_);

private:
    void easer();
    unsigned int _size;
    bool *_matrix;

};
```

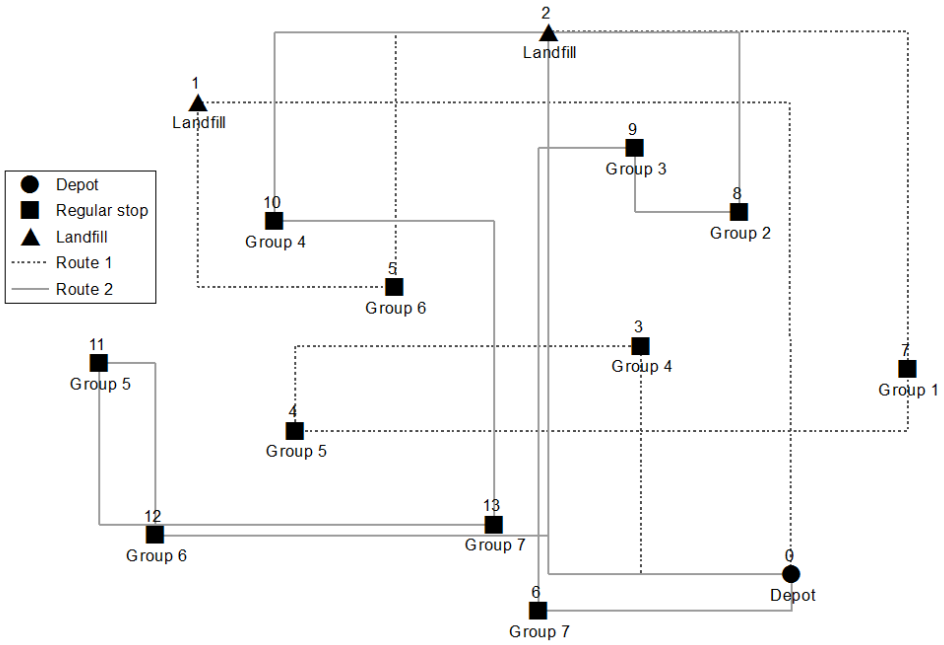
7.2.2 Đặc tả nghiệm:

Như đã trình bày ở trên, bài toán thu gom chất thải nguy hại là bài toán VRP mở rộng (và VRP là bài toán mở rộng của TSP) với nhiều ràng buộc kèm theo, giải bằng phương pháp chính xác một cách hiệu quả là điều không thể. Phần công việc này tập trung giải bài toán bằng phương pháp áp dụng meta-heuristic từ thư viện đang xây dựng của đề tài này: xây dựng lời giải khả thi ban đầu, sau đó đưa ra các cấu trúc lời giải lân cận để giới hạn không gian tìm kiếm, nhằm cải thiện lời giải ban đầu.

Lời giải của bài toán được xem như tập hợp gồm nhiều lộ trình chính, mỗi lộ trình chính bao gồm một hoặc nhiều lộ trình con. Mỗi lộ trình con bắt đầu là điểm xuất phát hoặc các trạm xử lí và kết thúc cũng là điểm xuất phát hoặc các trạm xử lí. Trong mỗi lộ trình con, các CTNH không được mâu thuẩn nhau về loại CTNH đã được định nghĩa trước và có tổng khối lượng chất thải trong lộ trình con không được vượt quá tổng khối lượng tối đa mà phương tiện được phép chở.

Trong hầu hết các phương pháp giải bài toán VRP bằng phương pháp meta-heurictic, xây dựng lời giải khả thi ban đầu đóng vai trò quan trong trong việc quyết định chất lượng lời giải và thời gian đề tìm ra lời giải cuối cùng. Trong các phương pháp xây dựng lời giải khả thi ban đầu cho họ các bài toán VRP, phương pháp chèn của Solomo được đánh giá là một trong những phương pháp tốt nhất. Phần công việc này cũng sử dụng phương pháp chèn của Solomo mở rộng, kết hợp với các đặc điểm và ràng buộc đặc trưng của bài toán thu gom vận chuyển CTNH để xây dựng lời giải khả thi ban đầu cho bài toán.

Trong hiện thực thuộc đề tài lời giải của bài toán VRP được chia thành 3 cấp độ khác nhau, lời giải `vrpSubRoute`, lời giải `vrpRoute` và lời giải `vrpSolution`. Tất cả các lớp này điều được kế thừa từ `edaSolution` thuộc thư viện. Về mặt ý tưởng 3 đối tượng trên tương ứng với 3 cấp độ cấu thành lời giải của bài toán VRP. Tập hợp các subroute sẽ thạo thành route và tập hợp các route sẽ tạo thành lời giải. Subroute hay lộ trình con là tập các trạm thu gom xuất phát từ nơi lưu trữ phương tiện hoặc xử lý chất thải và kết thúc tại nơi xử lý chất thải hoặc nơi lưu trữ phương tiện, các chất thải trong lộ trình con phải đáp ứng được yêu cầu về thời gian thu gom và không vi phạm ràng buộc xung đột giữa các loại chất thải. Route là tập hợp các subroute trong đó lộ trình đầu tiên và cuối cùng phải xuất phát và kết thúc tại nơi lưu trữ phương tiện. Ngoài ra các subroute trong nó còn phải thỏa các điều kiện ràng buộc về tổng quảng đường duy chuyển. Số lượng các route tương ứng (bằng hoặc nhỏ hơn) với số lượng phương tiện sẽ cấu thành lời giải cho bài toán VRPTWC. Hình 47 minh họa một lời giải cho bài toán thu gom chất thải công nghiệp, trong đó các phương tiện tạo thành một lộ trình kép kín đi qua tất cả các trạm và trở về depot ban đầu.



Hình 47. Một lộ trình thu gom chất thải nguy hại thực giữa các điểm trong thành phố.

Các thông chi tiết về nền tảng toán học của lời giải người đọc có thể tham khảo tại tài liệu [Thai13b]. Ở đây chúng ta sẽ đi vào chi tiết tổ chức của lời giải bài toán VRP. Đối với lớp đối tượng subroute, chúng được xây dựng từ việc kế thừa từ `edaSolution` và cấu trúc dữ liệu `vector<unsigned int>`. Với lớp đối tượng route, cấu trúc dữ liệu `vector <vrpSubRoute*>` được sử dụng và lớp đối tượng `vrpSolution` là `vector <vrpRoute*>`. Việc kế thừa từ `edaSolution` hướng đến việc tối ưu cho các thành phần trong lời giải.

```
class vrpSubRoute: public vector<unsigned int>, public edaSolution {
public:
  vrpSubRoute();
  vrpSubRoute(double total_res_dist, double start_time);
  vrpSubRoute(const vrpSubRoute& sub);
  vrpSubRoute *clone() const;
  virtual ~vrpSubRoute();

  void init() {}
  double evaluate();

  // nhóm hàm biến đổi
  ...
  void update();
  ...

  // nhóm hàm truy xuất thông tin
  ...
```

```
  // nhóm hàm gán, so sánh, đóng gói cần có cho thư viện
  ...
  void printOn(ostream &os) const;
  void debug(ostream &os);
  edaSolution& operator = (const edaSolution &sol);
  vrpSubRoute& operator = (const vrpSubRoute &subroute);
  bool operator == (const edaSolution &sol) const;
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_VRP_SUBROUTE_);

private:
  vector<double> _start;
  vector<double> _waiting;
  vector<double> _loaded;
  vector<double> _dist;
  double _init_res_dist;
  double _total_service_time;
  double _total_move_dist;
  double _total_move_time;
  double _total_wait_time;
  double _total_load;
  double _coming_time;
  double _cost;
  double _x_center;
  double _y_center;
  vector<unsigned int> _group_list;
  vrpProblem* _problem;
  vrpConst* _const;
  void addGroup(vrpStop* stop);
};

class vrpRoute: public vector<vrpSubRoute*>, public edaSolution {
public:
  ...
  vrpRoute *clone() const;
  double evaluate() ;
  void update();
  void init();

  // nhóm hàm truy xuất thông tin
  ...

  // nhóm hàm yêu cầu bởi thư viện
  void printOn(ostream &os) const;
  void debug(ostream &os) const;
  edaSolution& operator = (const edaSolution &sol);
  vrpRoute& operator = (const vrpRoute &route);
  bool operator == (const edaSolution &sol) const;
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_VRP_ROUTE_);
};


class vrpSolution: public edaSolution, public vector<vrpRoute*> {
public:
  ...
  vrpSolution* clone() const;

  void init();
  double evaluate() ;

  // nhóm hàm truy xuất thông tin
```

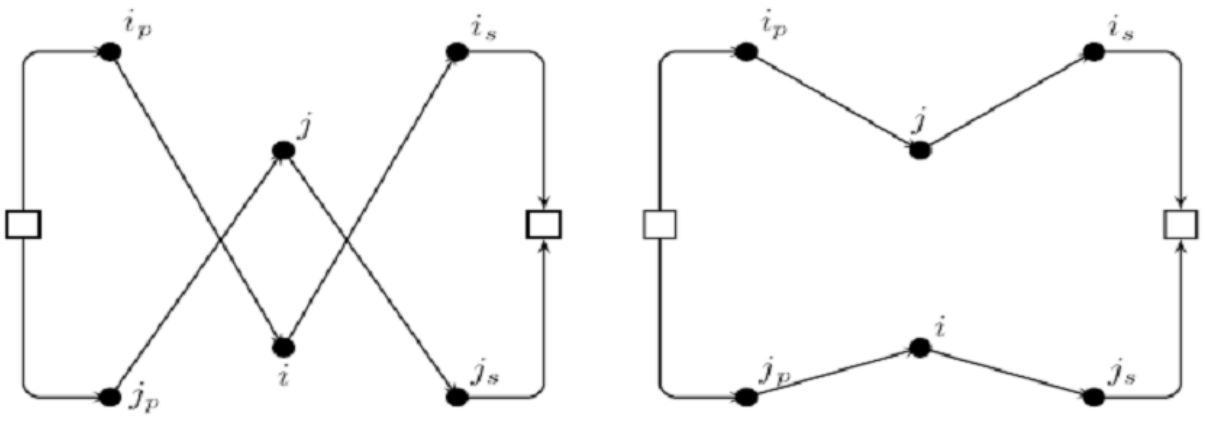
```
  ...
  void update();
  edaSolution& operator = (const edaSolution &sol);
  vrpSolution& operator = (const vrpSolution &sol);
  bool operator == (const edaSolution &sol) const;
  void improveLandfill();
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_VRP_SOLUTION_);

private:
  vrpProblem *_problem;
  vrpConst *_const;
  bool *_status;

  double _total_cost,
         _total_dist,
         _total_wait_time,
         _total_move_time,
         _total_service_time,
         _total_use_time,
         _total_load,
         _shape_metric;
  unsigned int _num_route,
               _num_sub_route;
};
```

Bước tiếp theo người dùng cần phải xây dựng các phương thức được yêu cầu bởi interface edaSolution. Những phương thức quan trọng mà chúng ta cần quan tâm ở đây bao gồm `init()`, `update()`, `evaluate()`. Trong bài toán VRP, phương thức `init()` có trách nhiệm khởi tạo lời giải khả thi theo chiến thuật tham lam, phương thức `update()` có nhiệm vụ cập nhật lại toàn bộ thông tin lời giải khi có các thay đổi diễn ra. Và hàm `evaluate()` giúp xác định giá trị hàm mục tiêu của bài toán.

7.2.3 Cơ chế phát sinh lân cận

Sau khi tìm được lời giải khả thi ban đầu cho bài toán, bước tiếp theo là lựa chọn một cơ chế sinh ra các lời giải lân cận và cách tìm kiếm và lựa chọn các lời giải lân cận nhằm mục tiêu cải tiến lời giải hiện tại. Bài toán này đã thử nghiệm các phương pháp tìm lời giải lân cận là λ-Interchange và λ-Composite Cyclic Exchange.

Trong cơ chế phát sinh lân cận λ-Interchange, toán tử Interchange được xây dựng để dịch chuyển chéo hai đoạn lộ trình có chiều dài là λ (hoặc nhỏ hơn) từ hai subroute khác nhau. Hình 48 minh họa việc dịch chuyển hai lộ trình khác nhau giữa hai subroute, các thông tin cần cho việc hoán đổi bao gồm route nào, subroute nào và vị trí cần hoán đổi thông tin.



Hình 48. Cơ chế tạo lời giải khả thi của toán tử Interchange với $\lambda = 1$.

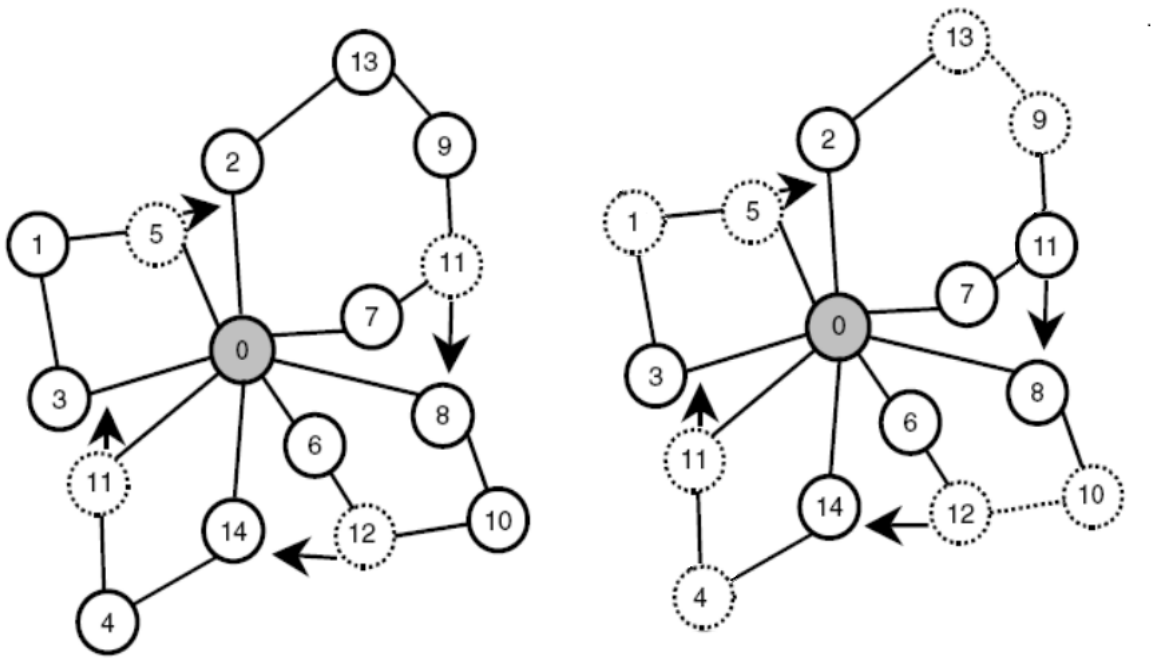
Trong hiện thực, bước chuyển `vrpInterchangeMove` sẽ bao gồm 9 thuộc tính, 6 thuộc tính ban đầu giữa thông tin route, subroute và vị trí của 2 lộ trình cần hóa đổi. Ba thuộc tính còn lại đùng để chứa thông tin về mục tiêu tối ưu của bài toán bao gồm thời gian đợi, quãng đường và thời gian di chuyển.

```
class vrpInterchangeMove: public edaMove {
public:
  vrpInterchangeMove();
  vrpInterchangeMove(const vrpInterchangeMove &move);
  ~vrpInterchangeMove();
  edaMove* clone() const;
  void init(const edaSolution &sol);
  double incrEval(const edaSolution &sol) const;
  void update(edaSolution &sol) const;
  void Serialize( edaBuffer &buf, bool pack );
  edaMove& operator = (const edaMove &_move);
  bool operator == (const edaMove &_move) const;
  setClassID(_USERCLASSID_ + _CLSID_VRP_INTERCHANGE_MOVE_);
  void printOn(ostream& os) const;

//Properties
  unsigned int ID_ROUTE_1;
  unsigned int ID_ROUTE_2;
  unsigned int ID_SUB_1;
  unsigned int ID_SUB_2;
  unsigned int NUM_OPT_1;
  unsigned int NUM_OPT_2;
  unsigned int POS_OPT_1;
  unsigned int POS_OPT_2;

  double INC_DIST;
  double INC_WAIT_TIME;
  double INC_MOVE_TIME;
};

class vrpCyclicMove : public edaMove {
public:
    vrpCyclicMove();
```

```
    vrpCyclicMove(const vrpCyclicMove &move);
    ~vrpCyclicMove();
    edaMove* clone() const;
    void init(const edaSolution &sol);
    double incrEval(const edaSolution &sol) const;
    void update( edaSolution &sol ) const;
    void Serialize( edaBuffer &buf, bool pack );
    edaMove& operator = (const edaMove &_move);
    bool operator == (const edaMove &_move) const;
    setClassID(_USERCLASSID_ + _CLSID_VRP_CYCLIC_MOVE_);
    void printOn(ostream& os) const;

//Properties
    vector<unsigned int> NUM_OPT;
    vector<unsigned int> POS_OPT;
    vector<unsigned int> ID_SUB;

    double INC_DIST;
    double INC_MOVE_TIME;
    double INC_WAIT_TIME;

private:
  void easer();
};
```

Đối với bước chuyển `vrpCyclicMove`, do đặc trưng của cơ chế này là hoàn đổi vòng giữa các lộ trình trong các subroute thuộc mỗi route, nên ta cần các ma trận với kích thước tương ứng với số phương tiện hay số lộ trình. Các thuộc tính khác lớp `vrpCyclicMove` đều tương tự như các thành phần thuộc bước chuyển `vrpInterchangeMove`.

Như vậy, nét quan trọng trong quá trình cải tiến lời giải khả thi ban đầu là việc sử dụng lại các thành phần sẵn có trong thư viện meta-heuristic. Cơ chế phát sinh lân cận và định nghĩa của lân cận của các toán tử của λ-Interchange, λ-Composite Cyclic Exchange được định nghĩa bởi người dùng qua việc kế thừa từ hàm `edaMove` và `edaMoveNext` đã được hiện thực sẵn trong thư viện. Bước tiếp theo người dùng chỉ đơn giản là sử dụng các giải thuật meta-heuristic trong thư viện. Quá trình cải tiến lời giải là quá trình tìm kiếm lời giải của giải thuật meta-heuristic (người dùng có thể lựa chọn nhiều phương pháp giúp cải tiến lời giải ban đầu bằng một trong những giải thuật như HC, SA, TS đã hiện thực sẵn trong thư viện).



Hình 49. Cơ chế tạo lời giải khả thi của toán tử Composite Cyclic với λ = 1 và 2.

7.2.4 Cơ chế giải mã và mã hóa:

Một hiện thực khác mà người dùng cần phải xây dựng để có thể sử dụng được các giải thuật tối ưu thuộc nhóm dựa trên quần thể là cơ chế mã hóa và giải mã. Trong bài toán VRP, nếu giữ nguyên chuỗi lộ trình các stop và chuyển chúng trực tiếp thành chuỗi các gen sẽ gây ra rất nhiều khó khăn cho việc tận dụng các toán tử Crossover được xây dựng sẵn trong thư viện.

Do cấu trúc lời giải bao gồm nhiều lộ trình (route), mỗi lộ trình xuất phát từ depot và quay về depot, mỗi lộ trình ấy lại là tập hợp của các lộ trình con với trạm landfill (nơi xử lý chất thải) nằm ở cuối lộ trình. Khi đó việc tách lộ trình và hoán đổi rất dễ dàng đi đến kết quả lộ trình mới được sinh ra sẽ mất thông tin về hoặc landfill hoặc depot. Để tránh hiện tượng trên, cơ chế mã hóa được xây dựng với khả năng tách các trạm phụ trợ (depot và landfill) ra khỏi lời giải và cơ chế giải mã ngược lại sẽ khôi phục các thông tin này trở lại lời giải. Nói một cách khác chuỗi gen bài toán VRP sẽ không bao gồm các thông tin về trạm xử lý và nơi chứa phương tiện. Tổ chức của lớp đối tượng này sẽ có dạng như sau:

```
class vrpRepresentation: public edaRepresentation {
public:
```

```
  vrpRepresentation();
  vrpRepresentation(const vrpProblem &pro, const vrpConst &con);
  vrpRepresentation(const vrpRepresentation& orig);
  vrpRepresentation* clone() const;

  void init(const edaSolutionList &list, edaPopulation &pop) const;
  void decode(const edaPopulation &pop, edaSolutionList &list) const;
  void encode(const edaSolutionList &list, edaPopulation &pop) const;

  virtual ~vrpRepresentation();
  void Serialize( edaBuffer &buf, bool pack );
  setClassID(_USERCLASSID_ + _CLSID_VRP_REPRESENTATION_);

private:
  vrpSubRoute* createSubRouteFormList(vrpStop* ref, double rest_dist,
                                      double coming_time,
                                      vector<unsigned int>& list,
                                      vrpProblem* _problem,
                                      vrpConst* _const) const;
  void decode(const edaChromosome *chro, edaSolution *sol) const;
  void encode(const edaSolution *sol, edaChromosome *chro) const;
  void easer();
  vrpProblem* pro;
  vrpConst* con;
};
```

7.2.5 Chiến lược tối ưu:

Sau khi xây dựng kiến trúc cho lời giải, lân cận, cơ chế mã hóa và giải mã, công việc tiếp theo là xác định chiến lược tìm kiếm. Trong công trình [Thai03b], nhóm nghiên cứu đã chọn và thực hiện các chiến lược cho bài toán VRPTWC với các chiến lược HC, SA, TS, GA và MA.

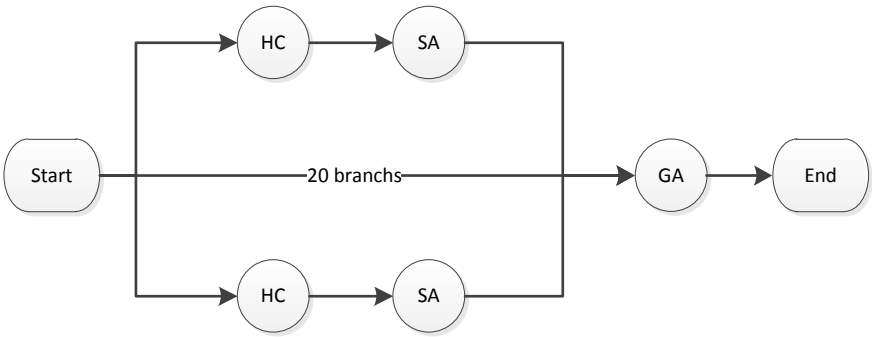
Đến thời điểm hiện tại, các thành viên trong nhóm đã hoàn thành việc tối ưu thông qua workflow với sự kết nối các các giải thuật cơ bản trên. Bài toán VRPTWC đã đưa vào chạy thử nghiệm với cơ chế song song trong thư viện với mục tiêu đẩy nhanh tốc độ cải tiến lời giải cho từng giải thuật. Xa hơn nữa, nhóm áp dụng quy trình song song hóa cho giải thuật MA với hai mục tiêu chính là giảm thời gian tối ưu đồng thời tăng chất lượng của lời giải đầu ra.

Các sơ đồ trong Hình 33, Hình 34 và Hình 35 sẽ được dùng trở lại trong các tính toán thuộc bài toàn VRPTWC. Ngoài ra chúng tôi còn đưa thêm 2 sơ đồ mới là SEQ và MPI vào quy trình tính toán như hình Hình 50. Quan sát Hình 50 ta thấy rõ chiến lược tối ưu cho trong workflow là sử dụng các giải thuật tìm kiếm cục bộ cố gắng thực hiện việc cải tiến tập dân cư đầu vào trước quá trình tối ưu bằng giải thuật

GA diễn ra. Như vậy dựa vào thư viện, các kết quả đầu ra từ các giải thuật tìm kiếm cục bộ (HC, SA, TS…) có thể ghép nối để trở thành hạt giống cho tập dân cư ban đầu của giải thuật GA và ngược lại kết quả đầu ra của giải thuật GA vẫn có thể tiếp tục cải tiến bằng các giải thuật tìm kiếm cục bộ.

Bảng 6. Các thông số được thiết lập cho bài toán VRPTWC.

	Số lời giải	Số lần lặp	Chiến lược
HC	40	1500	Global + Interchange.
SA	40	1500	ExpCooling(0.01, 0.98) + Interchange + T(5°)
TS	40	1500	Global Best + Interchange
GA	40	80	Roulette Wheel (0.5) + PMX + Swap(0.05)
MA	40	80	Roulette Wheel (0.5) + PMX + Swap(0.05) + HC(Interchange, 1500)



Hình 50. Workflow tuần tự SEQ và song song MPI được thiết lập cho bài toán vận chuyển chất thải nguy hại.

Việc thiết lập các thông số cho các sơ đồ được trình bày trong Bảng 6. Điều kiện lặp và số lời giải tối ưu là như nhau cho toàn bộ các giải thuật nhằm so sánh hiệu quả của chúng khi cùng làm việc với một bài toán với cùng điều kiện và tài nguyên tính toán. Ngoài tập lân cận Interchange, nhóm nghiên cứu cũng đã đưa vào một hiện thực lân cận khác là Cyclic Exchange. Tuy nhiên, khi được đưa vào thử nghiệm, các lân cận này tỏ ra không hiệu quả bởi cần lượng thời gian lớn, và do đó chúng không được đưa vào phần trình bày kết quả.

7.2.6 Kết quả thực nghiệm của bài toán VRP

Không chỉ đơn thuần là các ràng buộc về trị trí các trạm như bài toán TSP, bài toán VRPTWC được nghiên cứu trong đề tài có một hệ thống ràng buộc phức tạp. Như vậy, hình dạng của không gian nghiệm sẽ rất phức tạp và việc di chuyển trong không gian nghiệm thường dẫn đến nhiều vi phạm với các ràng buộc nếu không được xử lý tốt.

Mục đích hiện thực của nhóm nghiên cứu trong bài toán VRP là tìm ra các sơ đồ tối ưu mà lời giải của chúng là tốt hơn so với các kết quả đã dược công bố tại công trình [Thai13b]. Để làm được điều này, tập dữ liệu được chọn trong nghiên cứu hoàn toàn tương ứng với tập dữ liệu được công bố trong công trình trên tại website http://www.cse.hcmut.edu.vn/~hoai/VRPTWC/benchmark/. Các dữ liệu được chọn thử nghiệm bao gồm 10 tập dữ liệu mang thông tin xung đột là 102_stop, 277_stop, 335_stop , 444_stop.

Hai giải thuật TS, SA cũng được đưa vào trong quy trình tính toán. Ở đây, nhiệt độ của bài toán được hạ thấp để tránh vi phạm các ràng buộc mạnh bởi không gian nghiên cứu có không gian di chuyển hẹp. Nhiệt độ ban đầu được chọn trong nghiên cứu là 5°, các thông số còn lại được thiết lập như được trình bày trong Bảng 6.

Thực hiện việc cải tiến tập dân cư đang hỗ trợ cùng với các cải tiến có thể để tăng tốc độ tìm kiếm cũng như độ tối ưu tốt nhất có thể. Bài toán được chạy một cách đơn lẽ với từng giải thuật tìm kiếm, và đã có một số thử nghiệm nhằm kết hợp các giải thuật lại với nhau để cho ra một kết quả tốt hơn. Hiện tại nhóm nghiên cứu đã hoàn thành việc kết hợp giữa các giải thuật tìm kiếm cục bộ với giải thuật di truyền.

Với giải thuật SA, do phải khai phá trong gian hẹp nên chiến lược phát sinh lời giải ngẫu nhiên thường vi phạm các ràng buộc. Quá trình tối ưu của SA trong bài toán VRP với ràng buộc khung thời gian và xung đột thường dao động bên dưới lời giải cần được cải tiến. Hơn thế nữa lời giải tốt nhất được SA đưa ra thường mất rất nhiều thời gian tính toán và cũng không thể cạnh tranh với được với các giải thuật tối ưu khác. Những nhận xét này có thể thấy được từ các Bảng 7, Bảng 8, Bảng 9 và

Bảng 10. Có khả năng ***chiến lược phát sinh lời giải ngẫu nhiên không quan tâm đến các ràng buộc không tỏ ra hiệu quả trong bài toán có nhiều ràng buộc và xung đột***. Để có được một chiến lược phù hợp hơn cho SA đòi hỏi cần thêm nhiều nghiên cứu chi tiết hơn để cải thiện điều này.

Với giải thuật GA các thông số được thiết lập cho bài toán TSP hoàn toàn được sử dụng lại cho bài VRP, và trong trường hợp này thuật toán có hiệu quả tốt. Hãy xét lại vấn đề xây dựng lời giải cho bài toán VRP [Thai13b], quá trình khởi tạo lời giải mới dựa trên các heuristic và kết quả là chúng ta có được một tập dân cư có chất lượng tốt bởi các cá thể trong được xây dựng trên chiến lược tham lam. Và điều này cũng tương tự như việc cải tiến lời giải ban đầu của tập dân cư đầu vào như đã trình bày trong hai sơ đồ SEQ04 và MPI04 thuộc bài toán TSP. Với MA, việc ứng dụng GA sẽ tiết kiệm rất nhiều thời gian tính toán và đây cũng là giải thuật có thời gian tính toán thấp nhất nếu được so sánh với các giải thuật khác trong cùng một tập dữ liệu và cùng số thế hệ (số lần lặp). Lần thử nghiệm này so với công trình [Thai13b], chúng ta thu được các kết quả tốt hơn trong các tập dữ liệu có không gian tìm kiếm lớn. Điểm khác biệt của lần thử nghiệm này so với công trình chúng tôi công bố trước đây là tập dân cư ở lần nghiên cứu này được mở rộng lên con số 80 cá thể so với con số 50 trước đây. Lý do này có thể giải thích phần nào tại sao giải thuật lại hiệu quả hơn đối với việc tìm kiếm trên các không gian lớn.

Một nhận xét chung đối với các sơ đồ tối ưu hóa với chỉ một giải thuật meta-heuristic riêng rẽ là quy trình tối ưu hóa dựa trên giải thuật MA thường vượt trội so với các quy trình tối hóa khác. Trong cả 4 tập dữ liệu nghiên cứu, MA luôn tìm được lời giải tốt nhất đối với dạng bài toán vận chuyển chất nguy hại có ràng buộc. Một hạn chế lớn đối với giải thuật MA là sự tốn hao tài nguyên tính toán lớn, nhất là về mặt thời gian. Chênh lệch giữa thời gian tính toán giữa MA và GA từ 8 đến 14 lần và điều này càng chứng tỏ việc cần thiết phải áp dụng sơ đồ song song. So với các kết quả công bố trước đây, trong trường hợp của dữ liệu 227_stop, 335_stop và 444_stop của giải thuật MA được thực hiện với 80 cá thể đạt được giá trị fintess tốt hơn. Đây cũng là điểm mạnh của giải thuật MA, với dân cư càng đông thì mức hiệu quả của giải thuật càng tăng (điều này cũng đồng nghĩa với thời gian tối ưu tăng).

Bảng 7. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 102_stop.

Workflow	Thời gian tính (giây)	Giá trị hàm mục tiêu			
		Tốt nhất	Xấu nhất	Trung bình	[Thai13b]
HC	12.1334	-297.884	-379.165	-330.672	-307
SA	692.532	-300.821	-451.946	-378.730	
TS	43.1195	-329.892	-373.748	-352.684	
GA	3.61939	-301.313	-499.060	-314.411	-294
MA	50.7744	-297.986	-939.378	-653.985	-271
MPIMA	42.092	-299.396	-1550.400	-710.483	
VNS	23.6174	-297.282	-356.660	-322.970	
SEQ	5545.24	-297.541	-485.294	-321.198	
MPI	213.079	-364.171	-437.976	-405.502	

Chúng ta quan tâm hơn các lược đồ tính toán kết hợp giữa các tác vụ với nhau. Chiến lược VNS một lần nữa lại được thực hiện, và lần này được áp dụng với bài toán VRP và toán tử Interchange với thông số λ thay đổi trong khoảng từ 2 đến 5. Các kết quả đầu ra cho sơ đồ này trong các Bảng 7 đến Bảng 10 đều cho kết quả tốt hơn giải thuật HC. Việc chuyển đổi chiến lược chọn lận cận tại các điểm tối ưu cục bộ phát huy một ý nghĩa tích cực giúp trình tối ưu hóa vượt qua các điểm cực trị địa phương bằng một lối đi (lên) khác. Khi việc tối ưu đạt gần đến mức cực trị toàn cục, các giải thuật thường đạt trạng thái bảo hòa và ít có những cải tiến đáng kể trong giai đoạn này. Trong sơ đồ VNS, khi đạt đến ngưỡng bảo hòa này, giải pháp thay đổi bước chuyển giúp lời giải tiếp tục được cải tiến. Do vậy quá trình hiện thực các bước chuyển hiệu quả trong giải thuật này sẽ hiệu quả hơn so với các giải thuật dịch chuyển cùng cấp. Quan sát thời gian tính toán của giải các giải thuật, ta có thể nhận thấy chi phí tính toán của sơ đồ VNS thường ở mức gấp hai lần so với giải thuật HC, nếu ước lượng mức độ hiệu quả và thời gian tính thì giải thuật VNS vẫn ở mức tích cực nếu so với giải thuật SA.

Chúng ta bàn đến giải thuật TS, trong phần lớn trường hợp, giải thuật TS thường có thời gian tính thấp và cũng đồng thời chất lượng lời giải ở mức xấp xỉ giải thuật HC. Ngay khi tìm được lời giải tốt nhất giải thuật TS thường dao động quanh nó và cố gắng tìm kiếm một bước chuyển thích hợp để vượt qua những vùng tối ưu cục

bộ. Với chiến lược chọn trong danh sách Tabu đơn giản và toàn giống các chiến lược trong bài toàn TSP, giải thuật Tabu không thể phát huy hết khả năng của mình trong bài toán có nhiều ràng buộc của chúng ta. Vì đây không là giải thuật trọng tâm của nghiên cứu này nên chúng tôi chưa đề xuất ra một chiến lược xây dựng danh sách cấm tốt hơn để có khả năng vượt qua các cái bẫy tối ưu cục bộ mà giải thuật này thường phạm phải. Trong trường hợp dữ liệu 444_stop, giải thuật có một quy trình tối ưu hiệu quả và đưa ra lời giải có chất lượng rất tốt vượt mức lời giải do giải thuật MA đưa ra trong công trình [Thai13b].

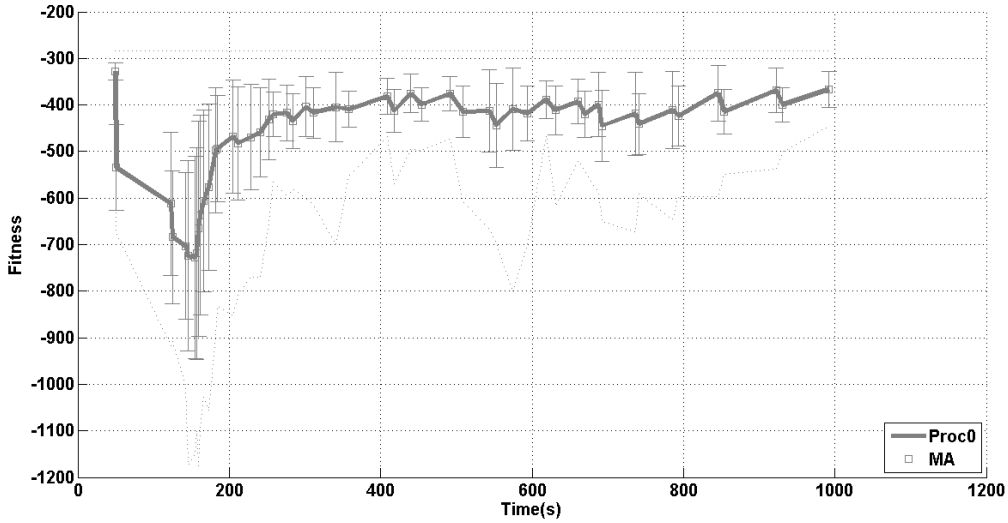
Bảng 8. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 277_stop.

Workflow	Thời gian tính (giây)	Giá trị hàm mục tiêu			
		Tốt nhất	Xấu nhất	Trung bình	[Thai13b]
HC	540.969	-551.364	-554.732	-553.359	-511
SA	10328.4	-563.356	-581.603	-571.909	
TS	780.106	-529.809	-555.986	-546.804	
GA	38.3054	-516.846	-661.261	-521.702	-527
MA	555.48	-514.665	-1560.33	-960.614	-521
MPIMA	210.43	-520.029	-1467.12	-984.552	
VNS	1027.44	-548.67	-554.303	-551.664	
SEQ	8512.05	-505.111	-610.134	-533.735	
MPI	484.809	-565.884	-662.884	-619.554	

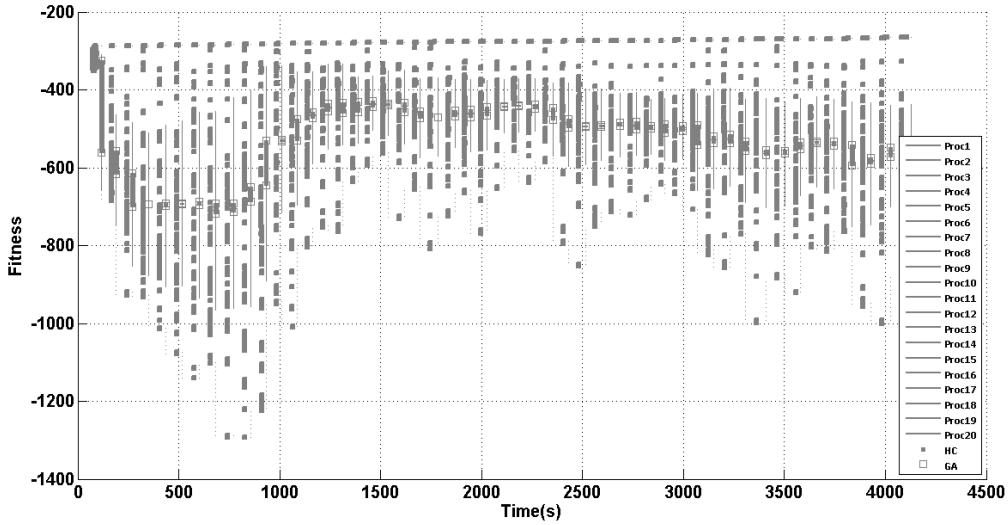
Một hiện thực khác của giải thuật MA là MPIMA, lược đồ này song song hóa quy trình cải tiến trong giải thuật MA bởi HC được hiện thực bằng các tiến trình song song. Với kích thước bài toán nhỏ, các cải tiến về thời gian chưa đạt được ý nghĩa quan trọng bởi trọng số của việc trao đổi thông tin (liên lạc) giữa các bộ xử lý là cao.

Hai workflow tính toán khác được thực thi ở đây là SEQ và MPI. Sơ đồ ghép nối của chúng được trình bày trong hình Hình 50. Trọng tâm của sơ đồ tối ưu này chính là giải thuật GA ở trung tâm kết hợp với khả năng đa dạng hóa lời giải bằng giải thuật SA. Quan sát các kết quả tính toán từ các 2 đồ tính toán các Bảng 7 đến Bảng 10, ta nhận thấy sự chênh lệch về thời gian tính toán giữa các sơ đồ thường lớn hơn giá trị 21 lần (đây chính là giá trị của số lượng vi xử lý được sử dụng cho

quá trình tối ưu hóa). Xét về giá trị hàm mục tiêu 2 sơ đồ này đã đồng thời mang lại các kết quả tối ưu tốt so với các giải thuật như HC, SA, TS và HC ở đa số các trường hợp. Ở ba tập dữ liệu 227_stop, 335_stop và 444_stop, hai sơ đồ này đồng thời đem lại các kết quả tối ưu tốt hơn công trình [Thai13b].



Hình 51. Quá trình tối ưu của sơ đồ MA cho tập benchmark 335_stop.



Hình 52. Quá trình tối ưu của sơ đồ MPIMA cho tập benchmark 335_stop.

Qua các thử nghiệm trên ta đã thấy rõ hiệu quả của việc sử dụng thư viện trong vai trò lập sơ đồ và xử lý tiến trình tối ưu hóa cho các dạng bài toán tối ưu tổ hợp (rời rạc). Khi áp dụng vào một bài toán với một hệ ràng buộc chặt, yêu cầu được đặt

ra để có được một kết quả tối ưu tốt là đòi hỏi người thực hiện phải hiểu rõ về bài toán mình đang tối ưu và đưa ra các định nghĩa lân cận hiệu quả. Đáp ứng được yêu cầu này thư viện sẽ là một công cụ đắc lực hỗ trợ cho người sử dụng. Đối với dạng bài toán tối ưu liên tục, như dạng bài toán Airfoil, kết quả việc thử nghiệm thư sẽ được trình bày trong phần sau.

Bảng 9. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 335_stop.

Workflow	Thời gian tính (giây)	Giá trị hàm mục tiêu			
		Tốt nhất	Xấu nhất	Trung bình	[Thai13b]
HC	1244.08	-272.887	-291.124	-282.83	-269
SA	10648.1	-302.594	-339.751	-325.929	
TS	1235.03	-272.887	-291.124	-282.83	
GA	116.794	-284.521	-413.376	-369.401	-263
MA	992.348	-284.521	-445.037	-366.306	-249
MPIMA	126.27	-265.816	-904.156	-601.018	
VNS	2322.37	-269.694	-287.044	-279.796	
SEQ	53672.7	-261.445	-390.402	-342.154	
MPI	1604.13	-279.152	-375.844	-342.022	

Bảng 10. Kết quả tối ưu cho bài toán VRPTWC trên dữ liệu 444_stop.

Workflow	Thời gian tính (giây)	Giá trị hàm mục tiêu			
		Tốt nhất	Xấu nhất	Trung bình	[Thai13b]
HC	2225.04	-59.1754	-65.1501	-62.4727	-86
SA	4996.63	-62.0169	-66.0327	-63.3789	
TS	2687.34	-58.4243	-63.1647	-60.9619	
GA	206.64	-55.8963	-75.2503	-58.1707	-91
MA	2586.79	-55.7559	-83.0377	-63.2629	-56
MPIMA	631.66	-57.6123	-77.0034	-61.3022	
VNS	5375.4	-59.1352	-64.0358	-61.5275	
SEQ	44593.6	-54.7762	-77.7698	-62.3105	
MPI	1545.49	-59.061	-74.2125	-63.755	

7.3 BÀI TOÁN AIRFOIL

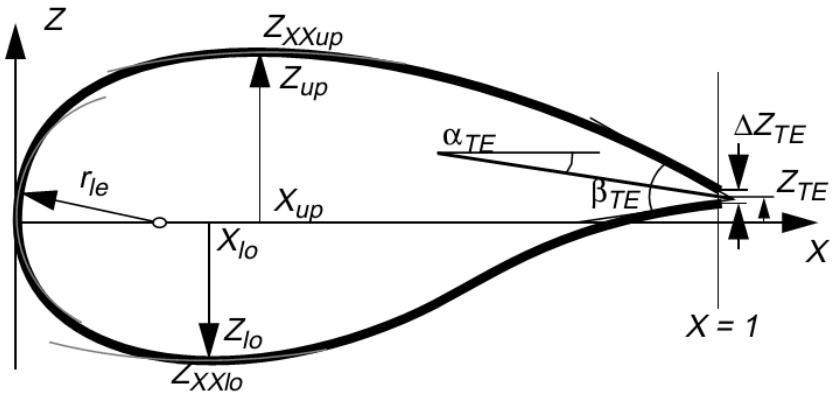
Bài toán "tối ưu hóa dạng khí động" là một đề tài đang thu hút được nhiều sự quan tâm trong ngành Cơ học lưu chất ngày nay. Phương pháp toán của bài toán này là "tính toán số lưu chất" (Computational Fluid Dynamics – CFD), trong đó máy tính được sử dụng để giải các phương trình vi phân đạo hàm riêng phi tuyến nhằm phân tích các hiện tượng vật lí xảy ra khi dòng chất lưu di chuyển theo một bề mặt nào đó. Các phương trình này gồm hệ phương trình Reynold cho chuyển động rối và hệ phương trình Navier-Stokes cho chuyển động có nhớt. Trong bài toán "tối ưu hóa dạng khí động", dòng chất lưu chỉ bao gồm chất khí. Do số lượng tính toán rất lớn nên dù có sự trợ giúp từ sức mạnh tính toán của siêu máy tính, phương pháp tính toán này vẫn cần phải được kết hợp với các giải thuật tối ưu hóa hiệu quả trong tính toán số thì mới mong tìm ra được lời giải chất lượng trong thời gian hợp lý.

Mặc dù việc kết hợp CFD với các phương pháp tối ưu hóa đã được ứng dụng vào nhiều bài toán thực tế và đạt được nhiều thành công, nhưng việc ứng dụng vào bài toán tối ưu thiết kế khí động học vẫn tồn tại nhiều thách thức, cả trong nghiên cứu và trong công nghiệp, vì các lý do sau:

- Hàm mục tiêu trong tối ưu khí động học tổng quát là hàm phi tuyến, đa cực trị và không lồi (nonconvex); nghĩa là có độ phức tạp rất cao, vì mô hình toán của dòng chất lưu là một hệ phương trình vi phân đạo hàm riêng phi tuyến.
- Việc xác định giá trị hàm mục tiêu bằng các phép tính tích phân số phương trình Reynold Navier-Stokes (RAS) hoặc Euler đòi hỏi tài nguyên tính toán lớn, tiêu tốn nhiều thời gian, đặc biệt là đối với các mô phỏng hình dạng không gian ba chiều (3D).
- Số lượng hình dạng mô phỏng được các giải thuật tối ưu hóa sinh ra trong một quy trình thiết kế tự động có thể là rất lớn (ví dụ GA), khiến việc tính toán giá trị hàm mục tiêu phải được thực hiện rất nhiều lần.

Bằng mô tả toán học, mẫu thiết kế (biên dạng cánh máy bay) được xác định qua một hay nhiều biến thiết kế. Hình 53 là mẫu thiết kế với 11 thông số biến thiết kế theo phương pháp PARSEC. Mô hình này giúp xác định chính xác hình dạng của

mẫu thiết kế biên dạng cánh và đó chính là nội dung của phương pháp tham số hóa. Nhiều nghiên cứu liên quan đến việc so sánh các kỹ thuật tham số hóa kể trên bằng các thuật toán tối ưu dựa trên phương pháp gradient người đọc quan tâm có thể tìm kiếm thêm thông tin tại tài liệu [Thai13a].



Hình 53. Biên dạng cánh PARSEC với 11 thông số hình học cơ bản [Sobiec99].

Trọng tâm của nghiên cứu này là dùng các giải thuật trong thư viện meta-heuristic kết hợp với CFD nhằm mục tiêu tối ưu khí động học trong một quy trình thiết kế cánh máy bay tự động. Mục tiêu hiệu quả và bền vững thể hiện ở cố gắng tạo ra hình dạng thiết kế tốt nhất cho mặt cắt ngang của cánh máy bay mà cụ thể là tăng tối đa hệ số giữa lực nâng và lực kéo (lift/drag), đồng thời đảm bảo phải cho kết quả trong thời gian hợp lý. Trong nghiên cứu của chúng tôi đã sử dụng giải thuật di truyền kết hợp với các giải thuật tìm kiếm cục bộ trong thư viện meta-heuristic. Song song hóa với sự hỗ trợ của thư viện là chiến lược tiên quyết để đạt được thời gian tính toán hợp lý trong quá trình tối ưu.

7.3.1 Đặc tả bài toán Airfoil

So với các bài toán tối ưu tổ hợp được trình bày, tổ chức của bài toán Airfoil có đặc thù là một quy trình thiết kế tự động và khép kín. Những dữ liệu đầu vào cho bài toán bao gồm thông tin của mô hình tính toán với các thiết lập của các hệ số vật lý, các tham số của quá trình tối ưu (giúp trình điều khiển có khả năng tối ưu hóa một cách tự động) và các thông tin hình học giúp xây dựng lưới tính toán 2 chiều cho bài toán.

```
[Model]
RASModel = kEpsilon
Mach Number = [0.73 0.0 0.0]
Static Pressure = 108987
Inlet Temperature = 255
Turbulent Kinetic Energy = 0.125
Turbulent Dissipation Rate = 7.32
Viscosity = 1.983e-05 ;
Conductivity = 0.0242
Specific Heat at Constant Volume = 720.43
Molecular Weight = 28.966

[Control]
Starting Time = 0.00
Time Step = 1e-05
Writing Time = 1e-02
End Time = 1e-01
Num Processors = 1

[Mesh]
Geometric chord length = 1.0 ; in meter
Angle of attack = 2.31 ; in deg
Scaling factor = 1.0
Half height of channel = 8.0
Half depth of foil = 0.5
Length of downstream section = 16.0
Number of interpolation points along the foil = 200
Number of mesh cells along the foil = 125
Number of cells in the downstream direction = 75
Number of cells the transverse direction = 50
Number of cells in the y direction = 1
Expansion rate in transverse direction = 500
Expansion rate in the downstream direction = 100
Expansion rate along the inlet arc = 50
```

Bên trên là trích dẫn của tập tin KEpsilon.air thiết lập cho quy trình tối ưu hóa cho bài toán biên dạng cánh. Thông tin từ tập tin sẽ được đưa vào các đối tượng tương ứng trong lớp `airProblem`. Từ phương thức `load()`, các đối tượng thuộc lớp `airControl`, `airSolver`, `airMesh` sẽ nhận (lưu trữ) các thông tin từ tập tin dữ liệu đầu vào.

```
class airProblem: public edaProblem {
public:
    airProblem();
    airProblem(const airProblem& orig);
    airProblem(const char* filename);
    airProblem *clone() const;
    virtual ~airProblem();
    airFoil* createFoil(double* foil = NULL) const;
    airProblem& operator = (const edaProblem &pro);
    double run(const airFoil& foil);
    void Serialize(edaBuffer &buf, bool pack);
    setClassID(_USERCLASSID_ + _CLSID_AIR_PROBLEM_);

private:
    void earse();
    double readForceCoeffsFile(const airFoil& foil);
    void createData(const airFoil& foil);
    airControl *control;
```

```
    airSolver* solver;
    airMesh* mesh;

};
```

7.3.2 Đặc tả nghiệm:

Để bắt đầu quy trình tối ưu cho bài toán biên dạng cánh cần phải định nghĩa xác định số biến thiết kế và miền trị của chúng. Phương pháp được sử dụng để mô tả các biến thiết kế cho cánh máy bay trong nghiên cứu này là PARSEC, sử dụng bộ gồm 11 biến thiết kế thực là r_{le}, X_{up}, Z_{up}, X_{lo}, Z_{lo}, Z_{XXup}, Z_{XXlo}, Z_{TE}, ΔZ_{TE}, α_{TE} và β_{TE}. Tuy nhiên, nghiên cứu này chỉ tập trung vào các dạng biên dạng cánh đuôi nhọn và điều này giúp thuận tiện hơn rất nhiều trong việc xây dựng hệ lưới tính tự động cho quá trình phân tích CFD. Do đó biến thiết kế độ dày mép đuôi ΔZ_{TE} sẽ thiết lập với giá trị cố định là 0 và được lược bỏ khỏi quá trình tối ưu. Ngoài ra để biểu diễn được các dạng biên khác bằng phương pháp PARSEC cần tách giá trị bán kính mép đầu r_{le} thành hai thành phần là bán kính mép đầu trên rle_{up} và bán kính mép đầu dưới rle_{lo}.

Các thông tin về biên dạng cánh trên sẽ được lưu trữ trong lớp đối tượng `airFoil`. Một ánh xạ đơn giản sẽ chuyển thông tin các biến của biên dạng cánh thành các hệ số của đa thức mô tả tọa độ hình học của biên dạng cánh và thông tin này sẽ được lưu trữ dưới dạng ma trận trong đối tượng `airFoil` và đối tượng này sẽ là một thành phần (quan trọng) trong lớp đối tượng `airSolution`.

```
class airFoil: public edaSerialize, public vector<double> {
public:
    airFoil();
    airFoil(const airFoil& foil);
    airFoil(double foil[]);
    airFoil(const char* foilname);
    virtual ~airFoil();
    airFoil* clone() const;
    void update();
    bool incrEval(unsigned int index, unsigned int sign);
    double z_up(double x) const;
    double z_lo(double x) const;
    double norm(unsigned int index) const;
    void denorm(unsigned int index, const double &value);
    unsigned int size() const;
    bool check() const;
    airFoil& operator = (const airFoil &foil);
    bool operator == (const airFoil &foil);
    void printOn(ostream& os) const;
    string detail() const;
```

```
    string name() const;
    void Serialize(edaBuffer &buf, bool pack);
    static airFoil* createFoil(double* foil = NULL);
    setClassID(_USERCLASSID_ + _CLSID_AIR_FOIL_);

private:
    double a[6], b[6];
};
```

Bảng 11 quy định miền trị của các biến thiết kế được chọn, số chữ số có nghĩa và số phần tử khả dĩ đối với từng biến thiết kế. Đối từng biến thiết kế, số phần tử thuộc miền trị là xác định và như vậy miền trị của bài toán được thực hiện trong nghiên cứu là hữu hạn với 6.1504E+29 bước chuyển trong không gian tìm kiếm.

Bảng 11. Miền trị được thiết lập đối với từng biến thiết kế.

	rle_{up}	rle_{lo}	X_{up}	Z_{up}	Z_{XXup}	X_{lo}	Z_{lo}	Z_{XXlo}	Z_{TE}	α_{TE}	β_{TE}
Cận trên	0.099	0.099	0.99	0.299	9.99	0.99	0.009	9.99	0.099	49.99	39.99
Cận dưới	0.000	0.000	0.00	-0.010	-10.00	0.00	-0.300	-10.00	-0.100	-50.00	-40.00
Số chữ số biểu diễn	4	4	3	4	4	3	4	4	4	4	4
Số chữ số có nghĩa	2	2	2	3	4	2	3	4	3	4	4
Số chữ số thập phân	3	3	2	3	2	2	3	2	3	2	2
Số phần tử thuộc miền trị	100	100	100	310	2000	100	310	2000	200	10000	8000

Chúng ta quan tâm đến lớp đối tượng `airSolution` và phương thức quan trọng trong nó là `evaluate()`. Để có được giá trị hàm mục tiêu cho lời giải, trình giải CFD sẽ được gọi thực thi và trả về tỷ số giữa lực nâng và lực đẩy của biên dạng cánh.

```
class airSolution: public edaSolution {
public:
    airSolution();
    airSolution(const airProblem &pro, airFoil* foil = NULL);
    airSolution(const airSolution& orig);
    airSolution* clone() const;
    double evaluate();
    double getCost() const;
    void setCost(double value);
    airFoil* getFoil() const;
    string name() const;
    string detail() const;
    airSolution& operator = (const edaSolution &sol);
    bool operator == (const edaSolution &sol) const;
    void printOn(ostream &os) const;
    void Serialize(edaBuffer &buf, bool pack);
    setClassID(_USERCLASSID_ + _CLSID_AIR_SOLUTION_);
    ~airSolution();
```

```
private:
    airProblem *pro;
    airFoil *foil;
    double cost;
    void easer();
};
```

7.3.3 Phương pháp mã hóa

Để bắt đầu giải thuật tiến hóa, nhóm nghiên cứu phải định nghĩa các gen và nhiễm sắc thể. Ta có thể giữ các biến thiết kế như là các gen đầu vào cho MA và GA. Như vậy mỗi tập các biến thiết kế sẽ ứng với một nhiễm sắc thể. Tuy nhiên, điều này sẽ gây ra khó khăn cho việc hoán đổi các gen bởi miền trị và số chữ có nghĩa cho mỗi biến thiết kế là khác nhau. Để tránh sự nhập nhằng này, các gen đầu vào của giải thuật MA cần được chuẩn hóa. Trong trường hợp này là 11 biến thiết kế theo phương pháp PARSEC sẽ được chuẩn hóa theo công thức sau để giá trị chúng nằm trong miền trị là [0, 1].

$$g_i = \frac{v_i - lb_i}{ub_i - lb_i}$$

Trong đó, g_i là giá trị của gen thứ i sau khi đã được chuẩn hóa, v_i là giá trị của biến thiết kế thứ i, ub_i và lb_i là các giá trị cận trên và dưới của biến thiết kế thứ i được tham khảo từ Bảng 11.

Đối với từng bộ biến thiết kế của biên dạng cánh, để thuận tiện và tránh nhầm lẫn cho việc truy xuất, nhóm nghiên cứu đề xuất phương pháp định danh bằng mã ASCII. Các ký tự ASCII được sử dụng trong phương pháp này là các ký tự từ "0" đến "9". Với mỗi biến thiết kế, số ký tự được dùng để định danh bằng với số chữ số có nghĩa tương ứng. Tên của các biến thiết kế sẽ xác định bằng công thức sau:

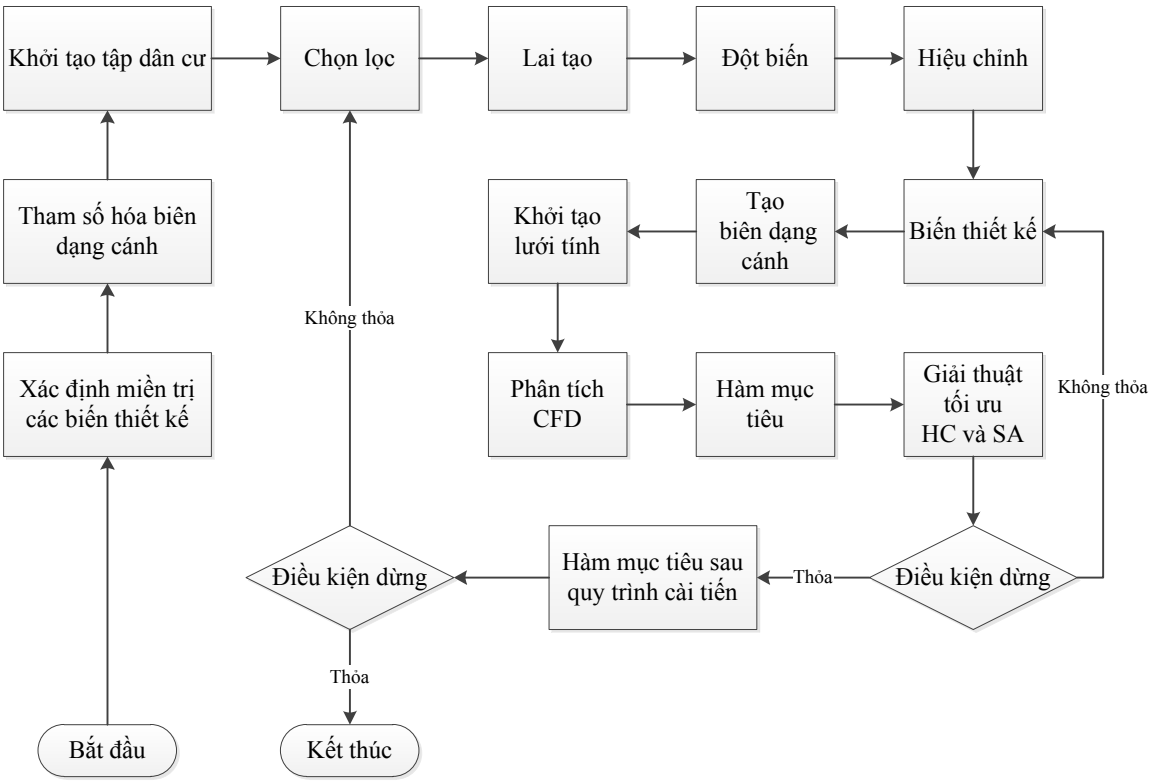
$$n_i = \text{NUM2ASCII}\left[(v_i - lb_i)\times 10^{d_i}\right] \quad (1)$$

n_i là tên của biến thiết kế thứ i, NUM2ASCII là hàm chuyển đổi từ số nguyên sang ký tự, v_i là giá trị biến, lb_i là cận dưới và d_i là số chữ số thập phân tương ứng với biến thứ i được tham khảo từ Bảng 11. Với các định danh này, tên của mỗi tập biến thiết kế sẽ dài 36 ký tự (3 ký tự '0' sẽ được mặc định gán cho biến thiết kế ΔZ_{TE}) và

tên này sẽ là duy nhất trong không gian tìm kiếm, lấy ví dụ với dạng biên dạng cánh RAE282 sẽ có tên mã là 094307309603524110731001004245491 0. Các vấn đề xử lý liên quan đến từng biên dạng cánh bao gồm quá trình tạo lưới tính, quá trình phân tích CFD, xác định giá trị hàm mục tiêu sẽ được lưu trữ trong thư mục tương ứng với tên mã của biên dạng cánh đó.

7.3.4 Quy trình tối ưu cho bài toán Airfoil

Cơ chế tối ưu trong bài toán tối ưu trong thiết kế khí động học thuộc đề tài được trình bày chi tiết trong hình Hình 54. Nền tảng chính của quy trình là sự kết hợp của giải thuật tối ưu là GA và cùng với các quá trình xử lý đặc thù trong thiết kế khí động học. Mỗi thành phần trong quy trình điều mang những nét riêng biệt, xong khi chúng được kết nối lại trong quy trình thì được xử lý một cách tự động. Các quyết định trong quy trình thiết kế được đưa ra dựa trên dữ liệu đầu ra từ các quá trình thực hiện trước đó và các tiêu chí thiết kế định sẵn ban đầu.



Hình 54 Quy trình giải thuật tối ưu tự động cho bài toán tối ưu biên dạng cánh.

Quá trình hiện thực bài toán airfoil mang những đặc thù riêng biệt, tuy nhiên nếu nhìn một cách tổng quát thì đây vẫn là một bài toán tối ưu tổ hợp. Các quá trình

crossover, selection, mutation được sử dụng từ hệ thống thư viện, và nhóm nghiên cứu chỉ tập trung định nghĩa cách biểu diễn biên dạng cánh và cách đánh giá lời giải để có thể đưa và quy trình tối ưu theo giải thuật GA. Việc biểu diễn biên dạng cánh theo phương pháp PARSEC với 13 biến và đánh giá chất lượng biên dạng cánh thông qua quá trình mô phỏng CFD đã được trình bày ở trên. ***Quy trình tối ưu của giải thuật GA tương tự như trong bài toán VRPTWC và điều này chứng tỏ lợi điểm của việc sử dụng hệ thống thư viện của đề tài.***

Như vậy nhờ hệ thống thư viện xây dựng sẵn, người nghiên cứu chỉ tập trung vào chuyên môn vào quá trình xây dựng lời giải cũng như quá trình đánh giá lời giải. Các hiện thực về quy trình tối ưu hóa được giảm nhẹ một cách đáng kể và trong một số trường hợp người thực hiện không cần qua tâm, chỉ đơn giản là gọi hàm và kết quả sẽ được trã về một cách tự động tùy theo việc thiết lập thông số của người sử dụng và đây chính là hướng đi của các hệ thống thư viện của các hãng lập trình và các tổ chức ngoài nước.

```
int main(int argc, char** argv) {
  EDAMetasearchStart (argc, argv);
  if (argc != 2)
  {
    std::cerr << "Usage : ./" <<  __progname
              << " [instance]" << std::endl;
  }
  else
  {
    airProblem pro(argv[1]);
    unsigned int n = 20;
    airFoil* *foil = new airFoil* [n];
    foil[0] = new airFoil("09094307309603524110731001004245491 0");
    foil[1] = new airFoil("09094307309603524110731001004245491 1");
    foil[2] = new airFoil("22083408909361726610671001004522550 8");
    foil[3] = new airFoil("23083608909371726610671011004523550 8");
    foil[4] = new airFoil("20103209009371726810681001004522550 0");
    foil[5] = new airFoil("20103209009381526710691001004522550 0");
    foil[6] = new airFoil("16163007009553023910451011005000473 6");
    foil[7] = new airFoil("16163007109552923910451011005000473 6");
    foil[8] = new airFoil("14152507109542924110461011005000473 0");
    foil[9] = new airFoil("14152507009542924110461011005000473 0");
    foil[10] = new airFoil("15162907109553124110461011005000472 5");
    foil[11] = new airFoil("15162907109553124110461021005000472 5");
    foil[12] = new airFoil("23373608009745219512420881004483506 7");
    foil[13] = new airFoil("62465411509112721811810781002985611 9");
    foil[14] =  foil[15] =  foil[16] = foil[17] = foil[18]
                                     = foil[19] = NULL;

    edaSolutionList list;
    for(unsigned int i = 0; i < n; i++) {
```

```
      airSolution* sol = new airSolution(pro, foil[i]);
      list.push_back(sol);
    }

    airNeighbourMove move;
    airNeighbourRand next;
    edaFirstImprSelect moveSelect;
    edaGenContinue cont;
    edaHC hcSearch (&move, &next, &moveSelect, &cont);

    airRepresentation repre(pro);
    edaRankSelection rank(0.8);
    airCrossover cross(0.8);
    edaRandomMutation mute(0.1);
    edaGenContinue gens(1);
    edaGA gaSearch(&gens, &repre, &rank, &cross, &mute);

    // Use the declaration below to run on parallel machine:
    edaMpiWrapperControl mfControl;
    edaNS nullSearch;
    int nullid, hcid[n], gaid;
    nullid = mfControl.insertVertex (&nullSearch);
    for(unsigned int i = 0; i < n; i++) {
      edaIDSelectWrapper slect(i);
      hcid[i] = mfControl.insertVertex (&hcSearch, slect);
    }
    gaid = mfControl.insertVertex (&gaSearch);
    for(unsigned int i = 0; i < n; i++) {
      mfControl.insertEdge(nullid, hcid[i]);
      mfControl.insertEdge(hcid[i], gaid);
    }
    mfControl.insertEdge(gaid, nullid);
    edaGenContinue loop(50);
    mfControl.insertLoop(gaid, nullid, loop);

    if (!mfControl.search (list))
    {
      std::cerr << "Error: Cannot execute search" << std::endl;
    }
    else
    {
        airSolution* sol = (airSolution*)list.getBest();
        cout << "The best solution: " << sol->name() << endl;
        cout << "Objective value: " << sol->getCost() << endl;
        cout << sol->detail() << endl;
    }
  }
  EDAMetasearchStop ();
  return 0;
}
```

Quan sát mã lập trình tại chương trình chính của bài toán Airfoil được hổ trợ song song hóa bởi thư viện hỗ trợ tối ưu, người đọc sẽ thấy ban đầu đối tượng airProblem sẽ đọc thông tin file thiết lập ban đầu. Tiếp đến tập dân cư được khởi tạo với các dạng biên dạng cánh sẵn có. Sau đó các thuật toán meta-heuristic được khởi động cùng với các thông số đầu vào. Workflow được chọn trong bài toán là dạng

song song với 20 task HC nối tiếp với task GA, quá trình này sẽ được lặp lại trong 20 thế hệ như thể hiện tại sơ đồ khối của Hình 35.

Với tiến trình thực hiện sơ đồ tối ưu hóa tuần tự (được hỗ trợ song song hóa ở mức solver openFoam) đã được trình bày trong [Thai13a], thời gian thực hiện của giải thuật lên đến 13 giờ 51 phút với số lượng biên dạng cánh mới (hợp lệ) được tạo ra là 151. Những con số trên thực sự khá ấn tượng nếu so với công trình [Khai11] đã thực hiện trước đó và đó cũng chính là minh chứng cho sự hiệu quả của việc ứng dụng thư viện như một công cụ chính cho quá trình tối ưu. Tuy nhiên chúng ta sẽ chờ đợi các kết quả tốt hơn (về mặt thời gian) khi áp dụng một sơ đồ tối ưu song song cho giải thuật MA và các kết quả thử nghiệm này sẽ được trình bày chi tiết trong phần tiếp theo.

7.3.5 Kết quả thực nghiệm của bài toán Airfoil

Chúng ta xét đến kết quả thử nghiệm của bài toán tối ưu cho biên dạng cánh, dạng bài toán tối ưu phi tuyến với hàm lượng giá có chi phí tính toán rất cao. Để thử nghiệm, nhóm nghiên cứu xây dựng duy nhất một sơ đồ giải thuật MA với 20 nhánh song song. Số vi xử lý sử dụng cho toàn bộ quá trình là 21 với 1 master và 20 slave. Toàn thời gian chạy cho quá trình là 21 giờ với 20 thế hệ lời giải. Số biên dạng cánh được tạo ra trong quá trình tối ưu là 6579 và kích thước đĩa cứng sử dụng là 9.6 GB.

Một điểm cần chú ý về phương pháp luận giải bài toán airfoil là nhóm nghiên cứu hướng đến sử dụng phương pháp không đạo hàm. Bài toán airfoil là một bài toán tối ưu với miền nghiệm là liên lục, nhưng không lồi. Do đó, một trong các phương pháp tiếp cận là sử dụng các kỹ thuật của tối ưu tổ hợp để khám phá không gian nghiệm. Một cách hay sử dụng nhất hiện nay là dùng các phương pháp dựa trên quần thể và sự phối hợp tổ hợp các cá thể trong quần thể có thể giúp tìm được nghiệm tốt. Đây cũng là cách mà nhóm nghiên cứu theo đuổi (chung với bộ môn Hàng không, khoa Giao thông của ĐH Bách khoa TP.HCM). Dĩ nhiên, đó cũng là lý do mà nhóm chỉ thử nghiệm phương pháp dựa trên MA.

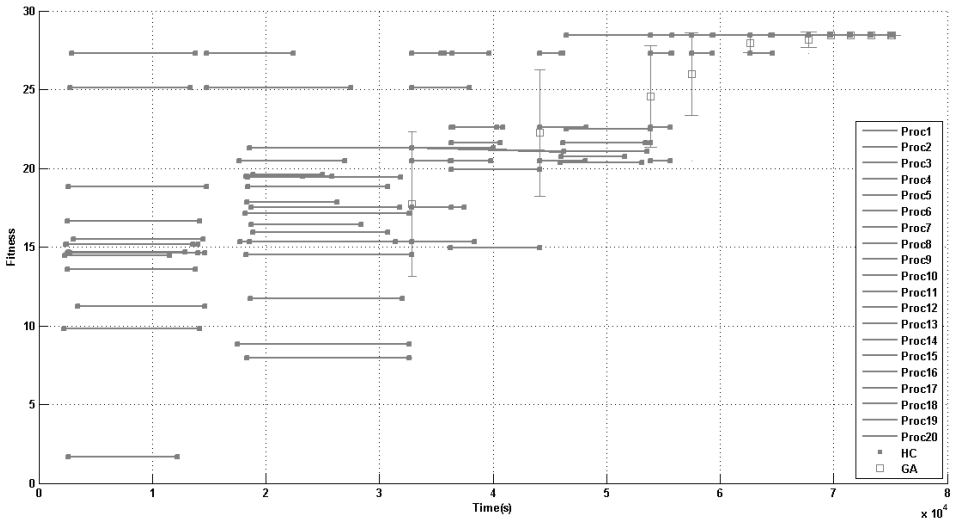
Bảng 12. Tập các biến thiết kế đầu ra từ quá trình tối ưu.

Tên biến các thiết kế	MA	HC
rle_{up}	0.01	0.009
rle_{lo}	0.01	0.009
X_{up}	0.43	0.43
Z_{up}	0.063	0.063
Z_{XXup}	-0.4	-0.4
X_{lo}	0.35	0.35
Z_{lo}	-0.059	-0.059
Z_{XXlo}	0.72	0.73
Z_{TE}	0	0
ΔZ_{TE}	0	0
α_{TE}	-7.55	-7.55
β_{TE}	9.1	9.11
Giá trị hàm mục tiêu	29.2676	29.0998

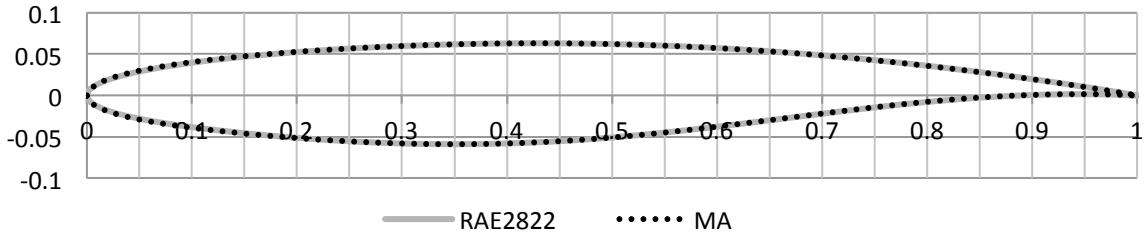
So với công trình được công bố trước đây trong luận vặn Thạc sỹ về "*Áp dụng thuật toán Memetic cho bài toán tối ưu biên dạng cánh*" [Thai13a], thời gian tìm ra lời giải tối ưu trong quá trình thiết kế là 12.7 giờ ngắn hơn hơn nữa giờ so với con số 13.51 giờ được công bố trước đây. Lược đồ hình 7.34 cho thấy quá trình tối ưu của giá thuật MA, quá trình chọn lọc sinh tồn sẽ đào các cá thể không phù hợp. Song song với các quá trình đó là việc lai chéo, đột biến và hiệu chỉnh trong mỗi thế hệ sẽ cải thiện chất lượng của lời giải. Về mức độ ảnh hưởng của tập dân cư đầu vào, với một quần thể tốt (các kinh nghiệm thiết kế tốt được đưa vào giải thuật) sẽ giúp quá trình tối ưu có được sự ổn định cần thiết và như vậy giải thuật có nhiều thời gian hơn để khai phá các vùng tiềm năng. Nếu không có kinh nghiệm thiết kế trước đó, quá trình tìm kiếm sẽ khó đạt được sự ổn định và biên dạng cánh đầu ra cũng có hình dáng và hiệu suất không như mong đợi. Và điều này hoàn toàn phù hợp với thực tế, các biên dạng cánh hiện đại hiện nay đều được chế tạo dựa trên các cải tiến của thiết kế trước đó, ngay cả những dạng máy bay bay đầu tiên do anh em nhà Wright chế tạo cũng dựa vào việc bắt chước hình dáng của các loài chim và đây chính là các biên dạng cánh trong tự nhiên. Đây cũng chính là ý tưởng của nhóm

nghiên cứu muốn đem một thư viện tiện lợi đến cho các nhà thiết kế để có thể tận dụng các kinh nghiệm của mình.

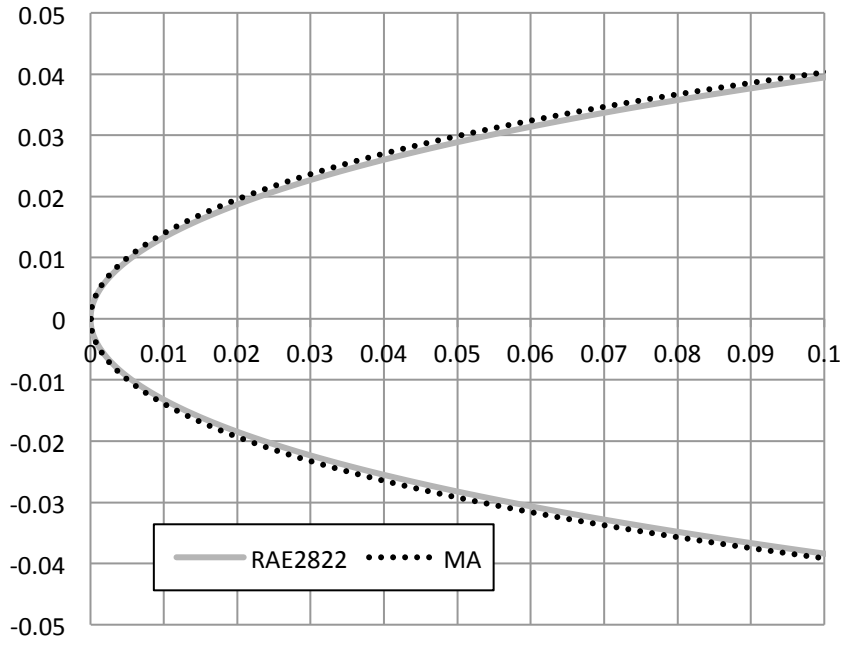
Các đặc tính của biên dạng cánh được tìm ra trong quá trình thử nghiệm được trình bày trong Bảng 12. Khi sơ đồ song song hóa được áp dụng cho quy trình tối ưu, tính hiệu quả về mặt thời gian của thuật toán tăng lên đáng kể. Sơ đồ giải thuật MA song song đạt được hiệu suất cao nhất có khả năng nâng hiệu suất biên dạng cánh RAE2822 lên mức 19.62% so với con số đưa ra của giải thuật bởi HC (khi cải tiến cho biên dạng cánh RAE2822) là 18.93%. Kết quả từ giải thuật MA với sơ đồ song song hóa thuộc thư viện rất hứa hẹn về cả chất lượng lẫn thời gian tính toán. Đồng thời cũng qua thử nghiệm này, ta thấy được tính trọn vẹn của thư viện trong vai trò của một công cụ hiệu quả phục vụ cho quy trình tối ưu các bài toán liên tục cũng như rời rạc.



Hình 55. Đồ thị của quá trình tối ưu của sơ đồ MPI MA cho bài toán tối ưu hóa biên dạng cánh 2D trong chuyển động cận thanh.



Hình 56. So sánh giữa biên dạng RAE2822 và kết quả tối ưu của quy trình MA.



Hình 57. Phần đầu của biên dạng RAE2822 và kết quả tối ưu của quy trình MA.

Như vậy, chương 7 của báo cáo đã ghi nhận lại các kết quả tính toán thử nghiệm cho 2 bài toán có xuất phát điểm từ các ngành khoa học khác nhau. Như đã đề xuất từ ban đầu, nhóm nghiên cứu tập trung chính vào việc xây dựng thư viện trong giới hạn của đề tài, và các ứng dụng chỉ mang tính minh họa. Tuy nhiên, người đọc có thể nhận thấy là với sự hỗ trợ của thư viện này, nhóm nghiên cứu đã có thể đào sâu chuyên môn vào để giải quyết thách thức từ ứng dụng. Đây là những minh chứng rõ ràng cho tính hiệu quả và linh động của thư viện.

CHƯƠNG 8. KẾT LUẬN VÀ ĐỀ NGHỊ

Nhìn lại thời gian thực hiện đề tài "*Xây dựng thư viện lập trình hỗ trợ tối ưu tổ hợp trên môi trường tính toán song song và phân bố*", nhóm nghiên cứu đề tài có một số nhận định chủ quan sau về những ưu và khuyết điểm như sau:

8.1 NHỮNG ƯU ĐIỂM

- Với yêu cầu về nội dung được đặt ra trong đề cương đã duyệt và hợp đồng nghiên cứu đã ký kết, nhóm nghiên cứu đã hoàn thành tất cả với một chất lượng chấp nhận được. Lĩnh vực mà đề tài hướng đến khá rộng, là cầu nối giữa nhiều hướng trong tính toán khoa học, và để hoàn thành tốt thì nhóm nghiên cứu đã phải nổ lực rất lớn để có được công bố tầm cỡ quốc tế.

- Thư viện được hoàn thiện trong đề tài đã hơn hẳn so với phiên bản trước về tính linh hoạt cho người dùng. Ngoài việc bổ sung 2 meta-heuristic thông dụng dạng quần thể, nhóm nghiên cứu đã đưa vào các cú pháp cho phép xây dựng các dạng workflow khác nhau. Một minh chứng mạnh mẽ nhất cho sự linh hoạt của thư viện là nhóm đã sử dụng các cú pháp định nghĩa workflow này để tạo ra những meta-heuristic phổ biến khác (như VNS, MA), mà không cần phải viết lại từ đầu dưới dạng C++.

- Hạ tầng tính toán song song dùng MPI đã được hỗ trợ bởi thư viện, và người dùng có thể triển khai tính toán với chỉ một vài bước bổ sung. Một điểm nhấn nữa trong kết quả chuyên môn là với công cụ hiện tại, nhóm có thể thực hiện song song các meta-heuristic dạng quần thể mà cần các phiên bản ước lượng giá trị hàm mục tiêu dạng song song. Đặc điểm này rất có lợi cho những bài toán mô phỏng khí động như tối ưu biên dạng cánh máy bay.

- Về mặt nhân sự, qua thời gian thực hiện đề tài nhóm đã đào tạo rất cơ bản các học viên cao học. Thông qua đó, nhóm đã gửi 2 học viên đi nước ngoài làm thực tập sinh, nghiên cứu sinh. Ngoài ra, với hiểu biết khá sâu sắc về meta-heuristic, nhóm đã tham gia đào tạo và viết bài báo chung với các nhóm khác về các bài toán ứng

dụng liên quan, như xếp lịch trong điện toán đám mây [Nguyen13, Dong13], xếp thời khóa biểu, hậu cần (logistics). Không chỉ dừng lại ở đó, nhóm đang dự định đến các bài báo về tối ưu trong các ứng dụng kỹ thuật sử dụng chính công cụ đã phát triển. Tuy nhiên, nhân sự cũng chính là vấn đề dẫn đến những khuyết điểm sẽ trình bày trong phần sau.

8.2 NHỮNG KHUYẾT ĐIỂM

- Thay vì thực hiện đề tài trong vòng 18 tháng như đề ra, nhóm đã phải xin gia han đến 2 lần để có thể hoàn thành các nội dung trong đề cương đã duyệt. Dù với bất kỳ lý do gì thì đây là một hạn chế mà nhóm ghi nhận và tìm những cách tiếp cận tốt hơn để có thể hoàn thành một đề tài đúng tiến độ.

- Như dự đoán từ ban đầu, kết quả nghiên cứu trong 2 ứng dụng là những điểm nhấn khoa học quan trọng nhất giúp cho việc công bố. Tuy nhiên, do những lý do khách quan và chủ quan nhóm đã mất khá nhiều thời gian để có thể tạo được kết quả khoa học tốt có thể công bố quốc tế được. Với bài toán tối ưu biện dạng cánh máy bay thì một công bố [Khai11] đã không thể nằm trong thời gian thực hiện đề tài. Với bài toán xây dựng lộ trình chuyên chở chất thải thì nhóm đã mất khá nhiều thời gian vào mô hình toán học hoàn thiện cho bài toán. Tuy nhiên, cho đến bây giờ thì nhóm đã có những nghiên cứu tốt để có thể công bố ở mức thư viện hỗ trợ, cũng như cải tiến phương pháp giải 2 bài toán ứng dụng được đề cập.

- Biến động nhân lực thực hiện là một khó khăn mà nhóm không nhìn thấy trước được. Như đề cập ở phần ưu điểm, nhân sự không nhiều trong khi gửi đi học nhiều nên đã mất nhiều thời gian hơn để xây dựng lại. Tuy nhiên, trong hoàn cảnh Việt Nam, việc biến động này là không thể tránh khỏi và nhóm xem đây là một khuyết điểm cần khắc phục để hoạt động ổn định hơn.

- Về mặt chuyên môn trong giới hạn về việc xây dựng thư viện thì một số khuyết điểm nên xem xét điều chỉnh trong tương lai:

  - Việc sử dụng bộ nhớ trong việc truyền nhận trên môi trường bộ nhớ phân tán còn chưa được hiệu quả, nhất là quá trình đóng gói/mở gói dữ liệu.

Chưa làm rõ được dữ liệu toàn cục/cục bộ nên dẫn đến dữ liệu nào cũng đóng/mở và truyền.

- Cơ chế master-slave với chỉ một master làm giảm khá nhiều khả năng mở rộng (scalability) của tính toán. Ngoài ra, do slave là không có trạng thái (stateless) nên có khá nhiều truyền nhận được thực hiện giữa master và slave. Nếu có cơ chế cho phép gửi nhiều công việc tuần tự đến một slave thì có thể giảm được truyền nhận trên mạng.
- Các meta-heuristic trong thư viện mới chỉ là những dạng cơ bản (mặc dù đã có 2 nhóm). Các meta-heuristic mới nhất nên được đưa thêm vào thì mới được công đồng khoa học dễ dàng tiếp nhận.

Ngoài ra, nhóm nghiên cứu đề tài cũng tự đề ra những mục tiêu lớn trong các nghiên cứu tiếp theo.

8.3 NHỮNG DỰ KIẾN VÀ ĐỀ NGHỊ

- Điện toán đám mây là một xu thế không thể tránh khỏi trong ngành công nghiệp máy tính [Truong11]. Cộng đồng nghiên cứu khoa học nói chung, và cộng đồng tính toán nói riêng cũng không thể tách khỏi xu thế này. Việc sử dụng hạ tầng tính toán mạnh sử dụng MPI chỉ có thể hiệu quả với các nước có tiềm lực kinh tế lớn, và thật sự có nhiều bài toán, nhiều nhu cầu tính toán. Qua thời gian dài tham gia vào cộng đồng tính toán hiệu năng cao, các thành viên của nhóm nhận thấy việc tự xây dựng các máy tính lớn là khó khả thi với kinh phí đầu tư hạn hẹp như hiện nay. Thay vào đó, việc sử dụng các dịch vụ của điện toán đám mây là một hướng đi tốt. Hơn nữa, các công ty cung cấp dịch vụ này đã và đang rất sẵn sàng hỗ trợ tính toán hiệu năng cao (như Amazon Web Service, Microsoft Azure). Một định hướng mà nhóm sẽ thực hiện là bổ sung việc sử dụng hạ tầng điện toán đám mây vào tầng thấp nhất của thư viện.

- Sử dụng thư viện để định nghĩa các meta-heuristic mới để gia tăng tiện tích cho thư viện, qua đó lôi kéo được người dùng. Ngoài ra, nhóm cũng sẽ đưa thư viện vào sử dụng nhiều hơn để thu thập thêm phản hồi của các nhà khoa học hướng đến một sản phẩm hoàn thiện.

Ngoài ra, qua thời gian thực hiện đề tài, nhóm có một đề nghị nhằm làm tốt hơn việc nghiên cứu của chính mình. Việc cố định các nội dung nghiên cứu (nhất là với nghiên cứu cơ bản) đã gây ra một số khó khăn cho nhóm (như đã đề cập ở trên). Thay vào đó, nhóm đề xuất quy định nên chỉ giới hạn kết quả phải tầm quốc tế, không đi sâu quá chi tiết vào nội dung để các ý tưởng nghiên cứu có thể dễ dàng được phát hiện và theo đuổi hơn.

Bài toán tối ưu lộ trình chuyên chở chất thải có xuất phát điểm là từ vấn đề quản lý của Việt Nam. Tuy nhiên, nhóm nghiên cứu đề tài đã phải sử dụng nhiều dữ liệu của quốc tế và kết quả cũng không áp dụng vào thực tế, làm hạn chế rất nhiều tính ứng dụng của nghiên cứu lý thuyết. Nhóm nghiên cứu đề tài rất mong muốn phía quản lý đồng ý và hỗ trợ để có thể thử nghiệm giải các bài toán tối ưu đang ngày càng là thách thức của các thành phố lớn như TP. Hồ Chí Minh.

TÀI LIỆU THAM KHẢO

[Alba05] E. Alba, ``*Parallel Meta-heuristics: A New Class of Algorithms*'', John Wiley & Sons, 2005, ISBN-0-471-67806-6.

[Anh07] N.T. Anh, N.C. Dat, N.Q. Hung, T.N. Minh, T. Nam, D.T. Nghia, N.T. Son, ''*EDAGrid@HCMUT : A Campus Grid Infrastructure for Service-Centric Grid Applications*'', Journal of Science and Technology Development, Vol. 10 (No. 13), pp. 52-59, 2007.

[Bian09] L. Bianchi, M. Dorigo, L.M. Gambardella, W.J. Gutjahr, ``*A survey on meta-heuristics for stochastic combinatorial optimization*'', Natural Computing, Vol. 8 (No. 2), pp. 239–287, 2009.

[Bleu03] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, ``*PISA - A Platform and Programming Language Independent Interface for Search Algorithms*'', LNCS ``Evolutionary Multi-Criterion Optimization (EMO 2003)'', pp. 494-508, Springer, 2003 .

[Caho04] S. Cahon, N. Melab, E.G. Talbi , `` *ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Meta-heuristics*'', Journal of Heuristics, Vol. 10 (No. 3), pp. 357–380, 2004.

[Deel05] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta1, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, Daniel S. Katz, "*Pegasus: A framework for mapping complex scientific workflows onto distributed systems*", Scientific Programming, Vol. 13, No. 3, pp. 1058-9244, 2005.

[Dom13] Julian Dominguez and Enrique Alba, "HydroCM: A Hybrid Parallel Search Model for Heterogeneous Platforms", Book chapter of "Hybrid Metaheuristics", Studies in Computational Intelligence, Vol. 434, pp. 219-235, 2013.

[Dong13] Pham Hoang Vu Dong, Tuan Tran Phuong, Thanh Van Le, Nguyen Huynh Tuong, Van Hoai Tran, *"Scheduling workflow applications in private cloud considering network bandwidth"*, Journal of Science and Technology, Vol. 51, No. 4B, pp. 20-29, 2013.

[Duri06] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, E. Alba, ``jMetal: A Java Framework for Developing Multi-Objective Optimization

Meta-heuristics'', Technical report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computacion, University of Malaga, 2006.

[EDA08] T.V. Hoai ''Phát triển tính toán lưới và ứng dụng vào bài toán khai phá dữ liệu và thiết kế vi mạch'' (EDAGrid), đề tài Ươm tạo Công nghệ 2005-2008.

[Fahr05] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr., H.L. Truong, ``*ASKALON: a tool set for cluster and Grid computing*'', Concurrency - Practice and Experience, Vol. 17 (No. 2-4), 2005.

[Hoai09] T.V. Hoai, T.D. Khoa, N..V Trong, ''*A Meta-Heuristic Based Grid-Enabled Solver for Logic Circuit Optimization'',* trình bày tại hội nghị FAIR 2009.

[Hoff08] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, K. Berriman, J. Good, "*On the Use of Cloud Computing for Scientific Workflows*", 8th IEEE Conference on eScience, pp. 640-645, 2008. ISBN-978-1-4244-3380-3.

[Igel08] C. Igel, V. Heidrich-Meisner, T. Glasmachers, ``*Shark*'', Journal of Machine Learning Research, Vol. 9, pp. 993-996, 2008.

[Khai11] T.Q. Khai, T.V. Hoai, N.A. Thi, "*Application of Online-Trained ANN and Parallel Computing in 2D Airfoil Optimization*", Journal of Science and Technology Development, Vol. 49, No. 4A, pp. 225-234, 2011.

[Kim05] B.I. Kim, S. Kimb, S. Sahoo, "*Waste collection vehicle routing problem with time windows*". Computers & Operations Research 3, 3624 – 3642, 2005.

[Lour08] P. Louridas, ``*Orchestrating Web Services with BPEL*'', IEEE Software, Vol. 25 (No. 2), pp. 85-87, 2008.

[Nam07] T. Nam, T.D. Khoa, ''*Supernode II : A Grid Node in EDAGrid*'', Journal of Science and Technology Development, Vol. 10 (No. 13), pp. 52-68, 2007.

[Ngo09] T.N. Ngo, N.T. Anh, T.V. Hoai, ''*Extending POP-C++ to Integrate Web Services into Parallel Objects*'', Proceedings of ICPADS 2009 – ''IEEE 15th International Conference on Parallel and Distributed Systems'', pp. 882-885, 2009.

[Nguyen13] Q.T. Nguyen, N. Quang-Hung, N.H. Tuong, T.V. Hoai, N. Thoai, "*Virtual Machine Allocation in Cloud Computing for Minimizing Total Execution Time on Each Machine*", Proceedings of Commantel'13 – International Conference on Computing, Management and Telecommunications, IEEE Computer Society, pp. 241-245, 2013.

[Pede10] M.E.H. Pedersen,; A.J. Chipperfield,. ``*Simplifying particle swarm optimization*''. Applied Soft Computing, Vol. 10, pp. 618–628, 2010.

[Sobiec99] Sobieczky, H. *Parametric Airfoils and Wings*, Notes on Numerical Fluid Mechanics, Vol. 68, pp. 71–88, 1999.

[Solo88] M.M. Solomon, M. Desrochers, "*Time windows constrained routing and scheduling problem*". Transportation Science 22, 1 – 13, 1988.

[Tant06] A.A. Tantar, N. Melab, E.G. Talbi, B. Toursel, ``*Solving the Protein Folding Problem with a Bicriterion Genetic Algorithm on the Grid*'', in Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW'06), 2006.

[Talb07] E.-G. Talbi, S. Cahona, N. Melab, "*Designing cellular networks using a parallel hybrid meta-heuristic on the computational grid*'', Computer Communications, Vol. 30 (No. 4), pp. 698-713, 2007.

[Thai13a] Thái Tiểu Minh, "*Áp dụng thuật toán Memetic cho bài toán tối ưu biên dạng cánh*", Luận văn Thạc sỹ Khoa học Máy tính, ĐH Bách khoa TP.HCM, 2013.

[Thai13b] T.T. Minh, T.V. Hoai, T.T.N. Nguyet,"*A Memetic Algorithm for Waste Collection Vehicle Routing Problem with Time Window and Conflicts*", ICCSA'2013 – International Conference on Computational Science and Its Applications, Part I, LNCS 7971, pp. 485-499, 2013.

[Thuc01] N.D. Thuc, D.T. Van, T.T. Huong, H.D. Hai, "*Lập trình tiến hoá: Cấu trúc dữ liệu + Thuật giải di truyền = Chương trình tiến hoá*", NXB Giáo dục, 2001.

[Truong11] Hong-Linh Truong, Schahram Dustdar, "*Cloud computing for small research groups in computational science and engineering: current status and outlook*", Journal Computing, Vol. 91, No. 1, pp. 75-91, 2011.