ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
-----ooOoo-----

Tài liệu kỹ thuật:

Hệ thống thư viện tối ưu dạng meta-heuristic

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

TS. Trần Văn Hoài
ThS. Thái Tiểu Minh
ThS. Trần Thị Như Nguyệt

Tháng 5/2014

Cơ quan chủ trì đề tài:
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Cơ quan quản lý đề tài:
SỞ KHOA HỌC VÀ CÔNG NGHỆ TP. HCM

Chủ nhiệm đề tài:
TS. Trần Văn Hoài
Trưởng PTN Tính toán Khoa học, Khoa KH&KT MT, ĐH Bách Khoa

LỜI NÓI ĐẦU

Meta-heuristic đã được sử dụng rộng rãi trong các nghiên cứu hàn lâm và cũng như trong thực tế để giải quyết các vấn đề tối ưu vì nó có nhiều ưu điểm so với các phương pháp tìm lời giải chính xác. Với nhóm bài toán NP-Hard và với những kích thước đầu vào lớn thì các thuật toán thuộc nhóm này đang được quan tâm rất lớn. Mặc dù chúng không thể cung cấp lời giải chính xác, cũng như không thể xác định được khoảng cách với nghiệm tối ưu (khi dừng), nhưng rất hữu ích khi đưa ra những lời giải tốt ban đầu. Hơn nữa, vì đây chủ yếu là các phương pháp kinh nghiệm nên việc dừng thuật toán ở một điều kiện người dùng là có thể và thông qua đó có thể thực thi thuật toán trong khoảng thời gian cho phép. Điểm yếu của nhóm thuật toán này là độ hiệu quả của một thuật toán tùy thuộc nhiều vào không gian và chiến lược tìm kiếm. Việc nâng cao hiệu quả của các thuật toán trong nhóm này sẽ có những nhóm phương pháp chính sau:

- **Nhóm 1:** Hiệu chỉnh các meta-heuristic sẵn có (về mặt cấu trúc, hoặc tham số điều khiển) để giải hiệu quả các bài toán đặc thù.
- **Nhóm 2:** Sử dụng các meta-heuristic mới hoặc nâng cao được phát triển trong thời gian gần đây nhất với mong muốn quét các vùng nghiệm chưa bao giờ đạt đến.
- **Nhóm 3:** Xây dựng các phương pháp lai ghép, hoặc cộng tác để bổ trợ cho nhau trong quá trình tìm kiếm nghiệm.
- **Nhóm 4:** Sử dụng các hạ tầng tính toán mạnh để khai phá nhiều hơn không gian nghiệm.
- **Nhóm 5:** Kết hợp các thuật toán dạng meta-heuristic với các nhóm thuật toán chính xác với mục đích giảm không gian tìm kiếm khi tìm (hoặc chứng minh) nghiệm tối ưu.
- **Nhóm 6:** Một số các nghiên cứu đi sâu vào bản chất của các meta-heuristic. Chẳng hạn như phân tích tính ổn định của các thuật toán meta-heuristic, nhất là nhóm các meta-heuristic dựa trên các quá trình ngẫu nhiên.

Thông thường với phương pháp tiếp cận dựa trên meta-heuristic thì các nghiên cứu đầu tiên thường tiếp cận các nhóm phương pháp (theo thứ tự): nhóm 1, nhóm 2, nhóm 3, và nhóm 4. Hai nhóm cuối cùng thường chỉ với các nghiên cứu chuyên sâu, hẹp và có mong muốn giải quyết triệt để các bài toán thực tế. Với nhận định như trên thì việc có một thư viện hỗ trợ tốt việc sử dụng meta-heuristic là một nhu cầu cần thiết. Thư viện meta-heuristic được phát triển với mong muốn cung cấp cho người dùng có thể nhanh chóng và hiệu quả giải quyết bài toán của mình nếu lựa chọn phương pháp thuộc ba nhóm 1, 3 và 4. Trong quá trình nghiên cứu của chính mình, nhà nghiên cứu có thể phát triển thêm các meta-heuristic của riêng mình và qua đó đóng góp thêm cho cộng đồng các meta-heuristic để phục vụ cho phương pháp thuộc nhóm 2.

Tài liệu sử dụng này được viết thành 4 chương. Chương 1 sẽ đề cập đến nguyên lý trong việc phát triển thư viện. Người đọc có thể bỏ qua chương này nếu đã có kiến thức nền tảng về meta-heuristic. Tuy nhiên, việc đọc chương này sẽ giúp hiểu rõ hơn cơ chế làm việc của thư viện để từ đó sử dụng hiệu quả và phát triển các meta-heuristic cho riêng mình. Sau đó, các tính năng của thư viện, cũng như các lớp đối tượng và giao diện sẽ được trình bày trong chương 2. Ngoài ra, chương này cũng sẽ giới thiệu khái quát các bước chính để giải quyết bài toán sử dụng meta-heuristic với sự hỗ trợ của thư viện. Để có thể hiểu và dùng được thì cách tiếp cận thông thường nhất là qua một ví dụ minh hoạ, và đó chính là nội dung chính của chương 3. Chương 4 sẽ cung cấp cho người đọc các thông tin khác như cách tạo tập tin nhật ký (log file) để phân tích, công cụ hiển thị trực quan tập tin nhật ký.

MỤC LỤC

LỜI NÓI ĐẦU **1**
MỤC LỤC **3**
DANH MỤC HÌNH **13**
DANH MỤC BẢNG **14**
CHƯƠNG 1: KIẾN TRÚC CỦA THƯ VIỆN **15**
1.1. Meta-heuristic 15
1.2. Kiến trúc của thư viện 17
1.3. Cấu trúc của workflow 18
CHƯƠNG 2: CHỨC NĂNG CỦA THƯ VIỆN **22**
2.1. Giới thiệu chung 22
2.2. Đặc tả các nội dung liên quan đến bài toán 24
2.3. Đặc tả workflow 27
2.4. Đóng gói và tái tạo thông tin 28
CHƯƠNG 3: VÍ DỤ MINH HOẠ **30**
3.1. Bài toán TSP 30
3.2. Đặc tả bài toán 30
3.3. Đặc tả nghiệm 35
3.4. Đặc tả lân cận 39
3.5. Cơ chế phát sinh lân cận 42
3.6. Đặc tả gen và cơ chế mã hóa 44
3.7. Lựa chọn các giải thuật tìm kiếm và xây dựng Workflow: 47
3.8. Thực thi tìm kiếm 50
CHƯƠNG 4: CÁC TÍNH TOÁN THỬ NGHIỆM VÀ TIỆN ÍCH HỖ TRỢ **51**
4.1. Tính toán thử nghiệm 51
4.2. Nhật ký và biểu đồ 56
PHỤ LỤC A. THÔNG TIN VỀ CẤU TRÚC CƠ SỞ DỮ LIỆU **58**
edaAdaption Class Tham chiếu 58
Các hàm thành viên Public *58*
Mô tả chi tiết *58*
Thông tin về Constructor và Destructor *59*
Thông tin về hàm thành viên *59*
edaAspirCrit Class Tham chiếu 59
Các hàm thành viên Public *60*
Mô tả chi tiết *60*
Thông tin về hàm thành viên *60*
edaBestImprSelect Class Tham chiếu 61
Các hàm thành viên Public *62*

Additional Inherited Members *62*
Mô tả chi tiết *62*
Thông tin về hàm thành viên *62*
edaBestSelectWrapper Class Tham chiếu 63
Các hàm thành viên Public *63*
Mô tả chi tiết *63*
Thông tin về hàm thành viên *64*
edaBuffer Class Tham chiếu 64
Các hàm thành viên Public *64*
Các hàm thành viên Protected *65*
các thuộc tính Protected *65*
Mô tả chi tiết *65*
Thông tin về hàm thành viên *65*
edaChromosome Class Tham chiếu 66
Các hàm thành viên Public *66*
Các hàm thành viên Protected *67*
các thuộc tính Protected *67*
Mô tả chi tiết *67*
Thông tin về Constructor và Destructor *67*
Thông tin về hàm thành viên *67*
edaCluster Class Tham chiếu 68
Các hàm thành viên Public *69*
Mô tả chi tiết *69*
edaContinue Class Tham chiếu 69
Các hàm thành viên Public *70*
Mô tả chi tiết *70*
Thông tin về hàm thành viên *70*
edaCoolingSchedule Class Tham chiếu 70
Các hàm thành viên Public *71*
Mô tả chi tiết *71*
Thông tin về hàm thành viên *71*
edaCrossover Class Tham chiếu 72
Các hàm thành viên Public *72*
Mô tả chi tiết *72*
Thông tin về hàm thành viên *73*
edaDAG< DataType > Class Template Tham chiếu 73
Các hàm thành viên Public *73*
Mô tả chi tiết *74*
Thông tin về hàm thành viên *74*

edaDAGEdge Class Tham chiếu 76
Các hàm thành viên Public *76*
Mô tả chi tiết *77*
edaDAGVertex Class Tham chiếu 77
Các hàm thành viên Public *77*
Mô tả chi tiết *77*
Thông tin về hàm thành viên *77*
edaException Class Tham chiếu 78
Các hàm thành viên Public *78*
Mô tả chi tiết *78*
Thông tin về Constructor và Destructor *78*
Thông tin về hàm thành viên *78*
edaExpCoolingSchedule Class Tham chiếu 78
Các hàm thành viên Public *79*
Mô tả chi tiết *79*
Thông tin về Constructor và Destructor *79*
Thông tin về hàm thành viên *80*
edaFirstImprSelect Class Tham chiếu 80
Các hàm thành viên Public *81*
Additional Inherited Members *81*
Mô tả chi tiết *81*
Thông tin về hàm thành viên *81*
edaFitContinue Class Tham chiếu 82
Các hàm thành viên Public *83*
Mô tả chi tiết *83*
Thông tin về hàm thành viên *83*
edaFullSelectWrapper Class Tham chiếu 83
Các hàm thành viên Public *84*
Mô tả chi tiết *84*
Thông tin về hàm thành viên *84*
edaGA Class Tham chiếu 85
Các hàm thành viên Public *85*
Additional Inherited Members *86*
Mô tả chi tiết *86*
Thông tin về Constructor và Destructor *86*
Thông tin về hàm thành viên *86*
edaGenContinue Class Tham chiếu 87
Các hàm thành viên Public *88*
Mô tả chi tiết *88*

Thông tin về Constructor và Destructor *88*
Thông tin về hàm thành viên *88*
edaGenne Class Tham chiếu 89
Các hàm thành viên Public *89*
Mô tả chi tiết *90*
Thông tin về Constructor và Destructor *90*
Thông tin về hàm thành viên *90*
edaHC Class Tham chiếu 91
Các hàm thành viên Public *91*
Additional Inherited Members *92*
Mô tả chi tiết *92*
Thông tin về Constructor và Destructor *92*
Thông tin về hàm thành viên *93*
edaHCMoveExpl Class Tham chiếu 93
Các hàm thành viên Public *94*
Mô tả chi tiết *94*
Thông tin về Constructor và Destructor *94*
Thông tin về hàm thành viên *95*
edaIDSelectWrapper Class Tham chiếu 95
Các hàm thành viên Public *96*
Mô tả chi tiết *96*
Thông tin về Constructor và Destructor *96*
Thông tin về hàm thành viên *96*
edaImprBestFitAspirCrit Class Tham chiếu 97
Các hàm thành viên Public *97*
Mô tả chi tiết *98*
Thông tin về Constructor và Destructor *98*
Thông tin về hàm thành viên *98*
edaINIReader Class Tham chiếu 99
Các hàm thành viên Public *99*
Mô tả chi tiết *99*
Thông tin về Constructor và Destructor *99*
Thông tin về hàm thành viên *99*
edaLambdaPointCrossover Class Tham chiếu 101
Các hàm thành viên Public *101*
các thuộc tính Protected *102*
Mô tả chi tiết *102*
Thông tin về Constructor và Destructor *102*
Thông tin về hàm thành viên *102*

edaLinearCoolingSchedule Class Tham chiếu 103
Các hàm thành viên Public *103*
Mô tả chi tiết *104*
Thông tin về Constructor và Destructor *104*
Thông tin về hàm thành viên *104*
edaMA Class Tham chiếu 105
Các hàm thành viên Public *105*
Additional Inherited Members *106*
Mô tả chi tiết *106*
Thông tin về Constructor và Destructor *106*
Thông tin về hàm thành viên *106*
edaMove Class Tham chiếu 107
Các hàm thành viên Public *107*
Mô tả chi tiết *108*
Thông tin về hàm thành viên *108*
edaMoveExpl Class Tham chiếu 109
Các hàm thành viên Public *109*
Mô tả chi tiết *109*
Thông tin về hàm thành viên *110*
edaMoveGen Class Tham chiếu 110
Các hàm thành viên Public *110*
Mô tả chi tiết *111*
Thông tin về hàm thành viên *111*
edaMoveSelect Class Tham chiếu 111
Các hàm thành viên Public *112*
các thuộc tính Protected *112*
Mô tả chi tiết *112*
Thông tin về hàm thành viên *112*
edaMpiProcStatus Struct Tham chiếu 113
các trường dữ liệu *113*
Mô tả chi tiết *113*
edaMpiWorker Class Tham chiếu 114
Các hàm thành viên Public *114*
Mô tả chi tiết *114*
Thông tin về Constructor và Destructor *114*
edaMpiWrapperControl Class Tham chiếu 114
Các hàm thành viên Public *115*
Additional Inherited Members *115*
Mô tả chi tiết *115*

Thông tin về hàm thành viên *115*
edaMutation Class Tham chiếu 116
Các hàm thành viên Public *116*
Mô tả chi tiết *116*
Thông tin về hàm thành viên *117*
edaNaturalSelection Class Tham chiếu 117
Các hàm thành viên Public *118*
Mô tả chi tiết *118*
Thông tin về Constructor và Destructor *118*
Thông tin về hàm thành viên *118*
edaNoAspirCrit Class Tham chiếu 119
Các hàm thành viên Public *119*
Mô tả chi tiết *120*
Thông tin về Constructor và Destructor *120*
Thông tin về hàm thành viên *120*
edaNoSelectWapper Class Tham chiếu 121
Các hàm thành viên Public *121*
Mô tả chi tiết *121*
Thông tin về hàm thành viên *121*
edaNS Class Tham chiếu 122
Các hàm thành viên Public *123*
Additional Inherited Members *123*
Mô tả chi tiết *123*
Thông tin về Constructor và Destructor *123*
Thông tin về hàm thành viên *123*
edaOnePointCrossover Class Tham chiếu 124
Public Types *125*
Các hàm thành viên Public *125*
các thuộc tính Protected *125*
Mô tả chi tiết *125*
Thông tin về Member Enumeration *125*
Thông tin về Constructor và Destructor *126*
Thông tin về hàm thành viên *126*
edaPartiallyMatchedCrossover Class Tham chiếu 126
Các hàm thành viên Public *127*
Mô tả chi tiết *127*
Thông tin về Constructor và Destructor *128*
Thông tin về hàm thành viên *128*
edaPopulation Class Tham chiếu 129

Các hàm thành viên Public *129*
Mô tả chi tiết *129*
Thông tin về Constructor và Destructor *130*
Thông tin về hàm thành viên *130*
edaProblem Class Tham chiếu 131
Các hàm thành viên Public *131*
Mô tả chi tiết *132*
Thông tin về Constructor và Destructor *132*
Thông tin về hàm thành viên *132*
edaRandomMutation Class Tham chiếu 132
Các hàm thành viên Public *133*
các thuộc tính Protected *133*
Mô tả chi tiết *133*
Thông tin về Constructor và Destructor *133*
Thông tin về hàm thành viên *134*
edaRandSwapMutation Class Tham chiếu 134
Các hàm thành viên Public *135*
Các hàm thành viên Protected *135*
các thuộc tính Protected *135*
Mô tả chi tiết *135*
Thông tin về Constructor và Destructor *136*
Thông tin về hàm thành viên *136*
edaRankSelection Class Tham chiếu 136
Các hàm thành viên Public *137*
Mô tả chi tiết *137*
Thông tin về Constructor và Destructor *137*
Thông tin về hàm thành viên *138*
edaRepresentation Class Tham chiếu 138
Các hàm thành viên Public *139*
Mô tả chi tiết *139*
Thông tin về Constructor và Destructor *139*
Thông tin về hàm thành viên *140*
edaRNG Class Tham chiếu 141
Các hàm thành viên Public *141*
Mô tả chi tiết *141*
edaRouletteWheelSelection Class Tham chiếu 142
Các hàm thành viên Public *142*
Mô tả chi tiết *143*
Thông tin về Constructor và Destructor *143*

Thông tin về hàm thành viên *143*
edaSA Class Tham chiếu 144
Các hàm thành viên Public *145*
Additional Inherited Members *145*
Mô tả chi tiết *145*
Thông tin về Constructor và Destructor *145*
Thông tin về hàm thành viên *146*
edaSearch Class Tham chiếu 146
Các hàm thành viên Public *147*
các trường đữ liệu *147*
Mô tả chi tiết *147*
Thông tin về Constructor và Destructor *148*
Thông tin về hàm thành viên *148*
edaSearchWrapper Class Tham chiếu 149
Các hàm thành viên Public *150*
Các hàm thành viên Protected *150*
các thuộc tính Protected *150*
Mô tả chi tiết *150*
Thông tin về Constructor và Destructor *150*
Thông tin về hàm thành viên *150*
edaSelectionWrapper Class Tham chiếu 150
Các hàm thành viên Public *151*
Mô tả chi tiết *151*
Thông tin về Constructor và Destructor *151*
Thông tin về hàm thành viên *151*
edaSeqSearchWrapper Class Tham chiếu 152
Các hàm thành viên Public *153*
Additional Inherited Members *153*
Mô tả chi tiết *153*
Thông tin về Constructor và Destructor *153*
Thông tin về hàm thành viên *153*
edaSequentialControl Class Tham chiếu 153
Các hàm thành viên Public *153*
Mô tả chi tiết *154*
Thông tin về hàm thành viên *154*
edaSeqWrapperControl Class Tham chiếu 154
Các hàm thành viên Public *155*
Additional Inherited Members *155*
Mô tả chi tiết *155*

Thông tin về hàm thành viên *155*
edaSerialize Class Tham chiếu 156
Các hàm thành viên Public *158*
Mô tả chi tiết *158*
Thông tin về hàm thành viên *158*
edaSimpleMoveTabuList Class Tham chiếu 159
Các hàm thành viên Public *159*
Mô tả chi tiết *160*
Thông tin về Constructor và Destructor *160*
Thông tin về hàm thành viên *160*
edaSimpleSolutionTabuList Class Tham chiếu 161
Các hàm thành viên Public *161*
Mô tả chi tiết *162*
Thông tin về Constructor và Destructor *162*
Thông tin về hàm thành viên *162*
Các hàm thành viên Public *163*
Mô tả chi tiết *164*
Thông tin về hàm thành viên *164*
edaSolutionList Class Tham chiếu 165
Các hàm thành viên Public *165*
Mô tả chi tiết *166*
Thông tin về hàm thành viên *166*
edaString Class Tham chiếu 167
Các hàm thành viên Public *167*
Mô tả chi tiết *168*
Thông tin về Constructor và Destructor *168*
edaTabuList Class Tham chiếu 168
Các hàm thành viên Public *168*
Mô tả chi tiết *169*
Thông tin về hàm thành viên *169*
edaTimeContinue Class Tham chiếu 170
Các hàm thành viên Public *170*
Mô tả chi tiết *171*
Thông tin về Constructor và Destructor *171*
Thông tin về hàm thành viên *171*
edaTimer Class Tham chiếu 171
Các hàm thành viên Public *171*
Mô tả chi tiết *172*
Thông tin về hàm thành viên *172*

edaTS Class Tham chiếu 172
Các hàm thành viên Public *173*
Additional Inherited Members *173*
Mô tả chi tiết *174*
Thông tin về Constructor và Destructor *174*
Thông tin về hàm thành viên *175*
edaTSMoveExpl Class Tham chiếu 175
Các hàm thành viên Public *176*
Các hàm thành viên Protected *176*
các thuộc tính Protected *177*
Mô tả chi tiết *177*
Thông tin về Constructor và Destructor *177*
Thông tin về hàm thành viên *177*
edaVarFitContinue Class Tham chiếu 178
Các hàm thành viên Public *178*
Mô tả chi tiết *179*
Thông tin về hàm thành viên *179*
edaWrapperControl Class Tham chiếu 179
Các hàm thành viên Public *180*
Các hàm thành viên Protected *180*
các thuộc tính Protected *180*
Mô tả chi tiết *181*
Thông tin về hàm thành viên *181*
edaWriter Class Tham chiếu 182
Các hàm thành viên Public *182*
Mô tả chi tiết *182*
Thông tin về Constructor và Destructor *182*
TÀI LIỆU THAM KHẢO 183

DANH MỤC HÌNH

Hình 1. Kiến trúc của thư viện lập trình hỗ trợ tối ưu tổ hợp..... 17
Hình 2. Workflow với cấu trúc rẽ nhánh và vòng. 20
Hình 3. Workflow với cạnh vòng vi phạm điều kiện lặp. 21
Hình 4. Sơ đồ các bước tuần tự của chương trình sử dụng thư viện. 23
Hình 5. Các lớp đối tượng cần thực hiện đối với giải thuật TS..... 24
Hình 6. Các lớp đối tượng cần thực hiện đối với giải thuật HC..... 25
Hình 8. Các lớp đối tượng cần thực hiện đối với giải thuật SA. 25
Hình 9. Các lớp đối tượng cần thực hiện đối với giải thuật GA. 25
Hình 10. Các lớp đối tượng cần thực hiện đối với giải thuật MA. 26
Hình 11. Cơ chế tạo lời giải khả thi của 2-Opt. 40
Hình 12. Cơ chế tạo lời giải khả thi của Or-Opt..... 41
Hình 13. Workflow tính toán với ngữ cảnh lặp lồng nhau. 48
Hình 14. Các sơ đồ tối ưu đối với từng loại giải thuật meta-heuristic đơn lẽ. 52
Hình 15. Hiện thực giải thuật MA song song (MPIMA) bằng workflow trên nền tảng kết hợp giải thuật HC và GA..... 53
Hình 16. Workflow của SEQ01 & MPI01. 53
Hình 17. Workflow của SEQ02 & MPI02. 53
Hình 18. Workflow của SEQ03 & MPI03. 53
Hình 19. Workflow của SEQ04 & MPI04. 54
Hình 20. Tập tin nhật ký được ghi lại trong quá trình tối ưu của giải thuật GA. ... 56
Hình 21. Biểu đồ hiển thị log file cúa quá trình tối ưu với các tác vụ tối HC, SA, TS với 4 đơn vị tính toán song song. 57

DANH MỤC BẢNG

Bảng 1. Các thông số chi từng loại giải thuật meta-heuristic đơn lẽ. 52
Bảng 2. Kết quả tối ưu với các workflow tính toán trên benchmark berlin52. 54
Bảng 3. Kết quả tối ưu với các workflow tính toán trên benchmark eil101. 54
Bảng 4. Kết quả tối ưu với các workflow tính toán trên benchmark ali535. 55

CHƯƠNG 1: KIẾN TRÚC CỦA THƯ VIỆN

1.1. Meta-heuristic

Tài liệu hướng dẫn sử dụng này không là một cuốn sách tham khảo về meta-heuristic, nên vì thế chỉ cung cấp các kiến thức để người đọc có thể theo dõi trong các phần sau. Nếu người đọc quan tâm đến meta-heuristic thì có nhiều tài liệu rất tốt hiện nay, một vài trong số chúng có thể là [Michel10], [Sean13, Fred03, El09, Thuc01].

Thư viện meta-heuristic quan tâm đến 2 nhóm thuật toán meta-heuristic sau:

- Nhóm dựa trên dịch chuyển (move-based meta-heuristic): ý tưởng của các thuật toán trong nhóm này là xây dựng các cơ chế dịch chuyển vị trí hiện tại (một nghiệm) trong không gian nghiệm sang vị trí mới với định hướng đến nghiệm tối ưu toàn cục. Ví dụ điển hình của meta-heuristic thuộc nhóm này là leo đồi (hill climbing), luyện kim (simulated annealing), tìm kiếm tabu (tabu search). Nhìn chung, cấu trúc của một thuật toán dựa trên dịch chuyển là như dưới đây:

Cấu trúc thuật toán dựa trên dịch chuyển
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 được thì quay lại bước 2 hoặc

bước 3 (tuỳ theo cách tiếp cận)

Và cấu trúc trên được dùng để hiện thực 3 meta-heuristic dựa trên dịch chuyển cơ bản trong thư viện meta-heuristic: leo đồi, mô phỏng luyện kim, tìm kiếm leo đồi. Mô tả chi tiết của các thuật toán trên có thể tìm kiếm trong các tài liệu [Michel10], [Sean13], [Fred03], [El09], hoặc trong tài liệu báo cáo của đề tài "*Xây dựng thư viện lập trình hỗ trợ tối ưu tổ hợp trên môi trường tính toán song song và phân bố*". Trong chương 3, thông qua ví dụ minh hoạ thì người đọc sẽ hiểu các khái niệm cơ bản của các thuật toán này.

- Nhóm dựa trên quần thể (population-based meta-heuristic): thay vì chỉ sử dụng một vị trí trong không gian nghiệm, nhóm thuật toán này luôn giữ một tập các nghiệm. Thông qua việc "kết hợp" dưới nhiều dạng biến đổi khác nhau (bằng các toán tử) thì tập nghiệm này sẽ phát triển qua các thế hệ với định hướng đến hàm mục tiêu của bài toán tối ưu. Cấu trúc của nhóm thuật toán này là như dưới đây:

Cấu trúc thuật toán dựa trên quần thể

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 (tuỳ theo cách tiếp cận)

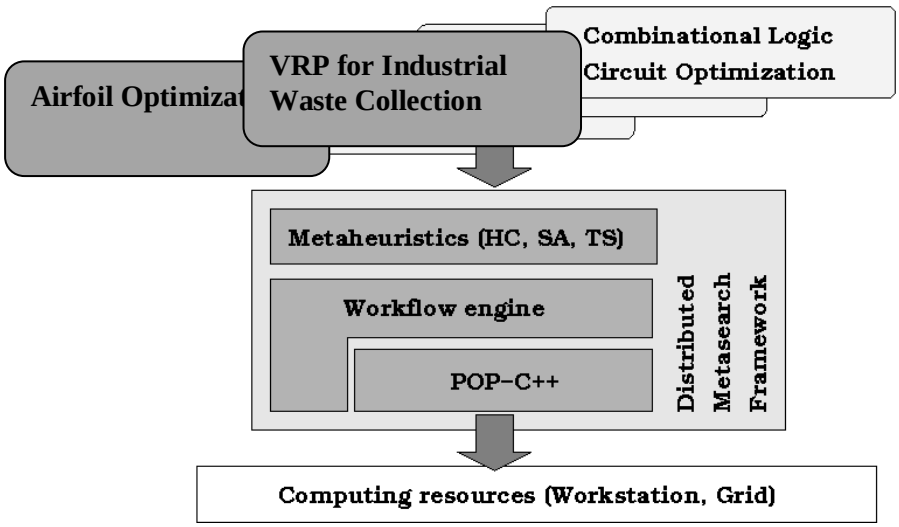
Cấu trúc cơ bản nêu trên đã được sử dụng để xây dựng 2 thuật toán chính trong thư viện meta-heuristic: thuật toán di truyền (Genetic Algorithm), thuật toán Memetic (Memetic Algorithm). Trong chương 4, thông qua ví dụ minh hoạ thì người đọc sẽ hiểu các khái niệm cơ bản của các thuật toán này.

Cho đến hiện nay thì có rất nhiều thuật toán dạng meta-heuristic, và mục tiêu của nhóm tác giả là đưa tất cả các dạng giải thuật meta-heuristic phổ biến vào trong thư viện meta-heuristic. Tuy nhiên, trong phiên bản đầu tiên, chỉ mới có 5 thuật toán cơ bản được đưa vào. Nói như vậy không đồng nghĩa với việc thư viện bị giới hạn, mà thay vào đó một bộ công cụ rất mạnh cho phép kết hợp các meta-heuristic đã được xây dựng trong thư viện. Với nó, người dùng có thể tạo thêm các meta-heuristic cho riêng mình. Các công cụ kết hợp sẽ được mô tả chi tiết hơn trong phần đặc tả workflow trong chương 3.

Trong phần sau, người đọc sẽ được giới thiệu kiến trúc bên trong của thư viện meta-heuristic để hiểu rõ cơ chế hoạt động.

1.2. Kiến trúc của thư viện

Kiến trúc thư viện meta-heuristic cơ bản được chia ra ba tầng Search Algorithm (các thuật toán tìm kiếm), Workflow Engine (Động cơ xử lý workflow), POP-C++/MPI (các thư viện truyền nhận thông điệp). Tài liệu này sẽ giới thiệu các tầng từ thấp đến cao.



Hình 1. Kiến trúc của thư viện lập trình hỗ trợ tối ưu tổ hợp.

- *Tầng POP-C++/MPI:* quản lý tất cả các đối tượng chạy trong môi trường phân bố. Do các môi trường phân bố khác nhau có kiến trúc và thư viện lập trình khác biệt nên đây là tầng thích nghi với các hệ thống phân bố. Thư

viện meta-heuristic được thiết kế cho các hệ thống có bộ nhớ phân tán, và 2 thư viện POP-C++ và MPI là những thư viện lập trình cho các hệ thống này. POP-C++ được phát triển bởi một nhóm nghiên cứu về tính toán lưới tại trường đại học Fribourg, Thuỵ Sỹ [POP12]. Tuy nhiên, do tính phổ biến không cao nên người dùng có thể sử dụng các hạ tầng có MPI. Rõ ràng MPI là một thư viện không thể thiếu được cho những máy tính song song lớn ngày nay, và vì thế người dùng có thể ứng dụng thư viện dễ dàng.

- *Tầng Workflow Engine:* đây là tầng điều khiển thứ tự thực hiện của các khối tìm kiếm meta-heuristic. Với một workflow của người dùng dưới dạng một đồ thị thì tầng này sẽ chịu trách nhiệm xác định những khối nào đang sẵn sàng tính toán và nếu có tài nguyên tính toán rãnh (từ tầng POP-C++/MPI cung cấp) sẽ thực hiện tính ngay khi có thể.

  Mô hình hoạt động hiện tại được sử dụng cho tầng workflow engine là master-slave. Trong đó, với một lượng tài nguyên tính toán cho trước thì thư viện sẽ dành riêng một đơn vị xử lý để làm master. Khi thấy một khối tìm kiếm đã có đầy đủ dữ liệu đầu vào, master sẽ gửi khối tìm kiếm đó, cùng với các dữ liệu liên quan đến một slave rãnh (nếu có). Chi tiết hoạt động có thể tham khảo thêm trong [Hoai14].
- *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 giá trị fitness cho bài toán. Đây là tầng mà người sử dụng thư viện cần nắm rõ để có thể tổ chức các sơ đồ tính toán (workflow) tối ưu một cách hiệu quả cho bài toán cần nghiên cứu.

1.3. Cấu trúc của workflow

Về cơ bản, khái niệm workflow được sử dụng trong thư viện 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 với đầy đủ các tham số điều khiển của chúng,

- *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 trong tập nghiệm khởi động cho meta-heuristic đích.

Trong hiện thực việc thiết lập DAG sẽ do các lớp kế thừa từ giao tiếp của lớp `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;
};
```

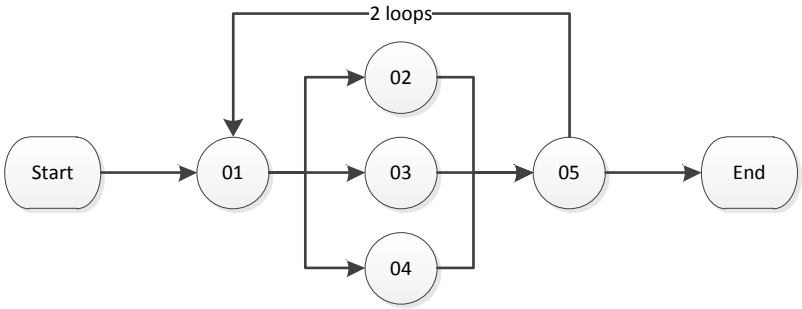
Quan sát giao tiếp có thể gọi của `edaWrapperControl` thì có thể thấy có đầy đủ các phương thức để xây dựng một DAG, bao gồm: `insertVertex()`, `insertEdge()` và `insertLoop()`. Cụ thể để xây dựng một DAG thì các tác vụ chính phải được thực hiện như sau:

- Đưa các đối tượng tìm kiếm dạng meta-heuristic: đưa vào bởi hàm `insertVertex()` và chú ý là đối tượng đưa vào phải thuộc lớp thừa kế từ `edaSearch` – và đây là tác vụ cần tối ưu.

- Kết nối các đối tượng: sử dụng hàm `insertEdge()`. Trong khi đưa thêm cạnh thì người dùng còn có thể đưa vào các điều kiện chọn lựa lời giải. Các điều kiện này giúp sàng lọc các lời giải trước khi đưa vào bên trong thuật

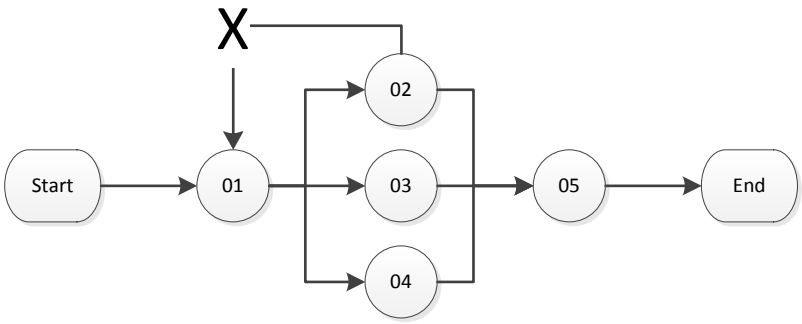
toán meta-heuristic. Thư viện đã hiện thực sẵn 2 khả năng: `edaFullSelectWrapper` - chọn lựa hết tất cả các lời giải khả kiến (đây cũng là cơ chế chọn lựa mặc định của thư viện nếu người dùng không có bất cứ tùy biến nào); `edaBestSelectWrapper` - chọn lấy lời giải tốt nhất trong danh sách.

- Đưa vào vòng lặp và điều kiện: thông qua phương thức `interLoop()`. 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* (mô tả chi tiết trong [Hoai14]). Một cách dễ hiểu là vòng lặp phải không được phép tạo ra các điểm bẻ gãy (vào hoặc ra) ở trung gian so với 2 điểm mà nó kết nối. Tuy nhiên, yêu cầu trên 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.



Hình 2. Workflow với cấu trúc rẽ nhánh và vòng.

Hình 2 biểu diễn một workflow với 5 tác vụ dạng meta-heuristic, trong đó có 3 tác vụ có thể thực hiện song song. Một vòng lặp nối liền từ tác vụ 5 đến tác vụ 1 tạo thành cluster bên trong nó. Với workflow ở hình 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 3. Workflow với cạnh vòng vi phạm điều kiện lặp.

Trong chương 2, các ví dụ minh hoạ sẽ giúp người đọc hiểu rõ và cụ thể hơn các bước trong lập trình để xây dựng workflow cho mình.

CHƯƠNG 2: CHỨC NĂNG CỦA THƯ VIỆN

2.1. Giới thiệu chung

Nhằm mục đích giúp cho người dùng dễ tiếp cận với thư viện, tài liệu này phân chia các chức năng theo nhóm người dùng (được mô tả trong phần giới thiệu):

Nhóm 1:

- 5 meta-heuristic thông dụng nhất hiện nay trong việc giải các bài toán tối ưu: leo đồi, mô phỏng luyện kim, tìm kiếm tabu, thuật toán gen, thuật toán memetic.
- Các toán tử cơ bản để hiệu chỉnh nghiệm, điều khiển tìm kiếm, điều khiển dừng được dùng trong 5 thuật toán nêu trên.
- Các tham số điều khiển các thuật toán nêu trên để người dùng có thể thích nghi với ứng dụng cụ thể.

Nhóm 3:

- Lớp đối tượng giúp định nghĩa workflow dạng một DAG.
- Các cú pháp mở rộng để cho phép tạo vòng lặp, tạo nhánh có điều kiện, tạo sự lựa chọn nghiệm giúp cho việc đa dạng hoá workflow.

Nhóm 4:

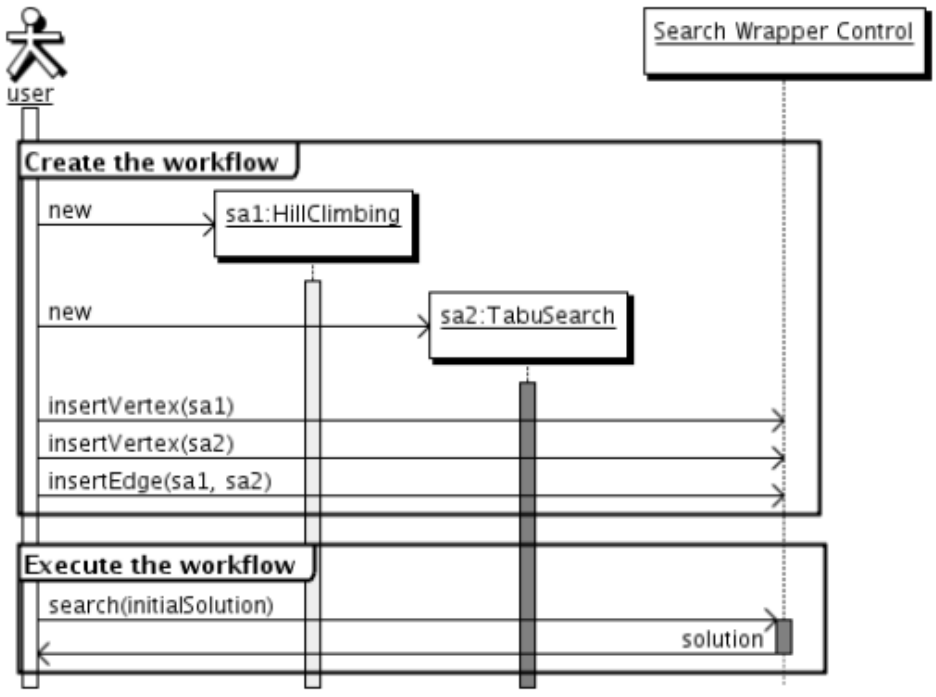
- Các đối tượng giúp đóng/mở gói dữ liệu khi muốn triển khai tính toán lớn.
- Triển khai tính toán trên máy đơn và trên máy tính song song hỗ trợ POP-C++ hoặc MPI.

Tất cả các thành phần trên của thư viện có thể biên dịch trước thành mã thực thi và có thể liên kết với mã ứng dụng của người dùng mà không cần phải biên dịch lại toàn bộ thư viện.

Ngoài ra, người dùng cũng có thể viết thêm các giải thuật khác dựa trên các lớp đối tượng và hàm cung cấp bởi thư viện nếu có nhu cầu. Các giải thuật viết thêm này giúp cho người dùng xây dựng thư viện của riêng mình, cũng như đóng góp cho cộng đồng nghiên cứu nếu tích hợp vào thư viện meta-heuristic.

Để giải quyết một bài toán tối ưu sử dụng thư viện meta-heuristic, người dùng cần tiến hành thực hiện các công việc sau:

- Xây dựng thuật toán meta-heuristic cho các công việc tìm kiếm, dựa trên thư viện.
- 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 sự trên môi trường phân bố.

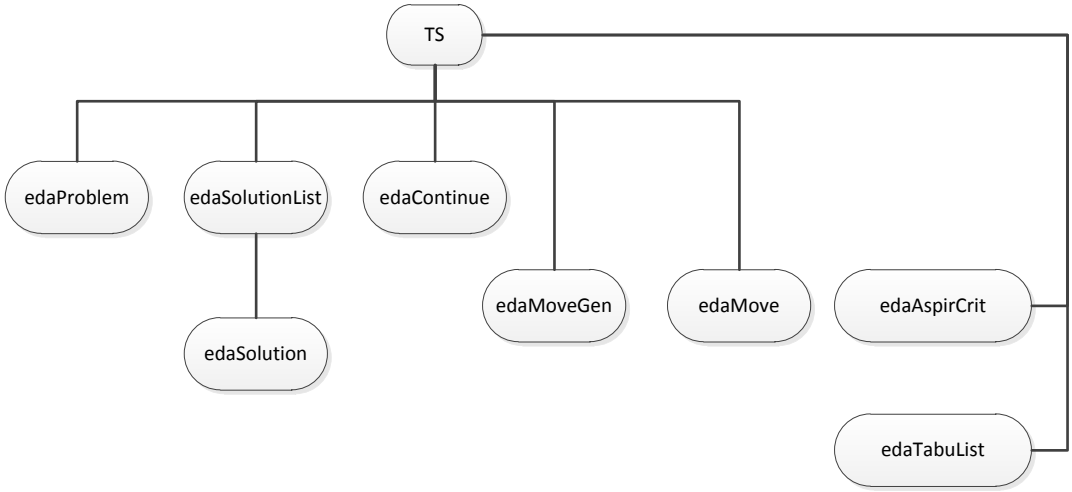


Hình 4. Sơ đồ các bước tuần tự của chương trình sử dụng thư viện.

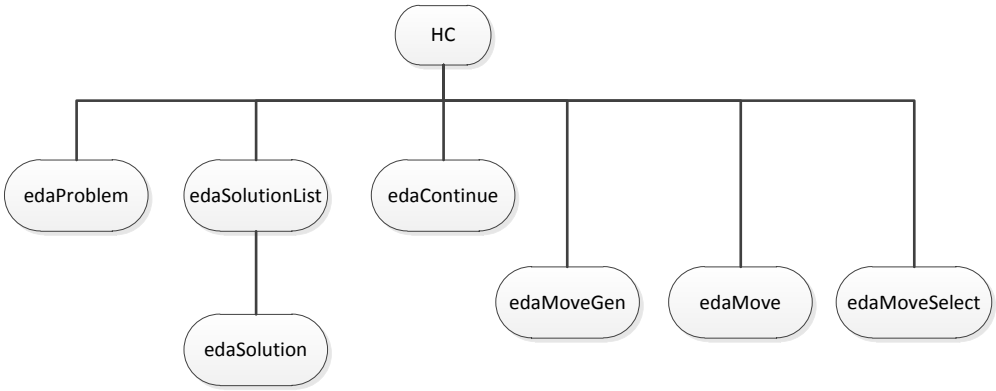
Hình 4 mô tả các bước chính như nêu ở trên. Bài toán tuần tự đã sử dụng 2 thuật toán meta-heuristic phục vụ cho việc tìm kiếm là giải thuật leo đồi và tìm kiếm tabu. Trong từng thuật toán, với những thông số phù hợp sẽ nâng cao hiệu quả tính toán. Vậy cần tiến hành xây dựng các đối tượng tìm kiếm tương 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. Nếu người dùng cảm thấy có thành phần nào của thư viện chưa phù hợp với bài toán của mình thì có thể phát triển thêm, thừa kế hoặc thay thế các giao tiếp sẵn có của thư viện.

2.2. Đặc tả các nội dung liên quan đến bài toán

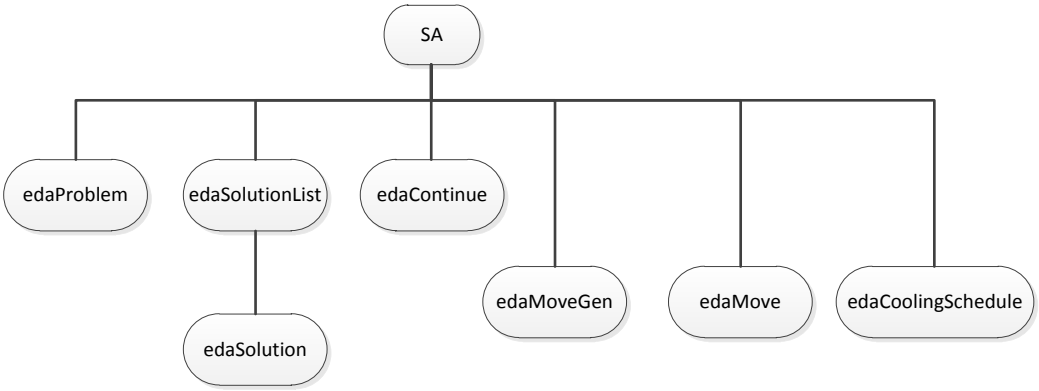
Người dùng sử dụng các giao tiếp của thư viện để mô tả bài toán của mình như nghiệm của bài toán, hàm lượng giá, các phép biến đổi nghiệm trong không gian (các bước chuyển), phương pháp mã hóa. Dưới đây là các lược đồ mô tả các lớp đối tượng cần được đặc tả ứng với từng loại giải thuật tối ưu:



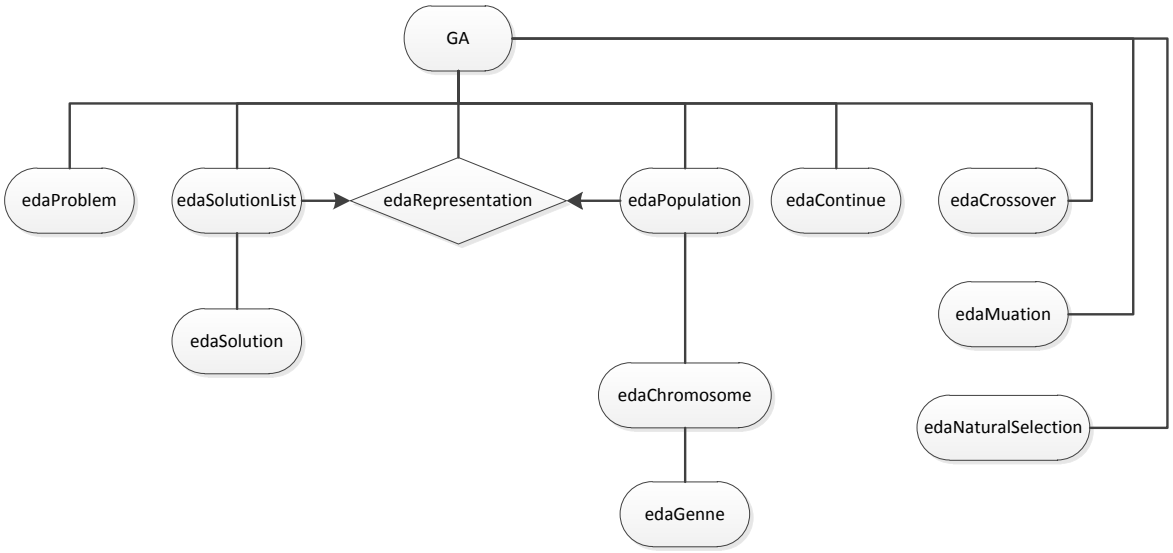
Hình 5. Các lớp đối tượng cần thực hiện đối với giải thuật TS.



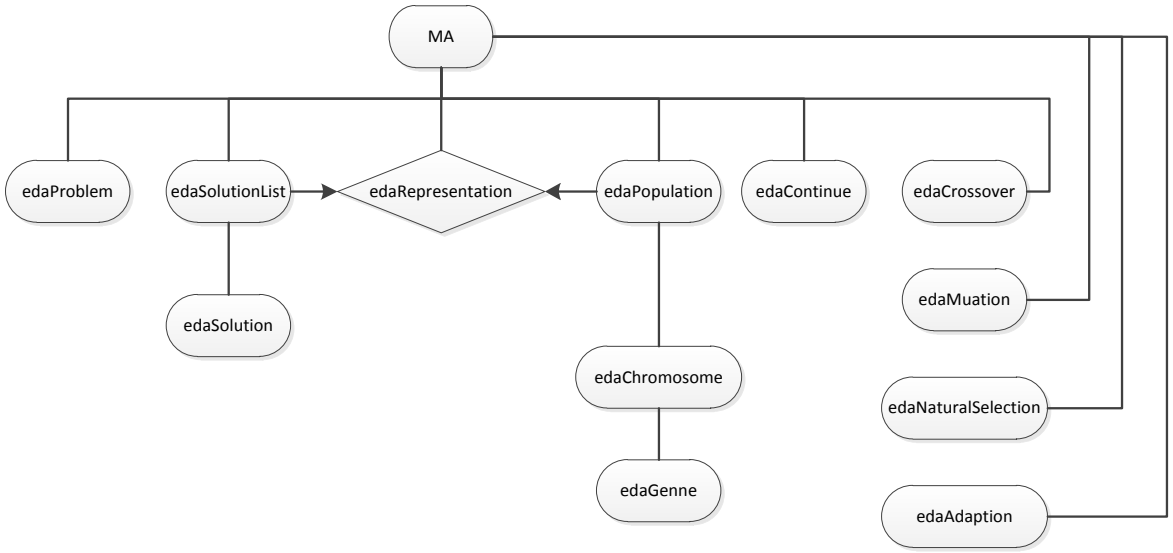
Hình 6. Các lớp đối tượng cần thực hiện đối với giải thuật HC.



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



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



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

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.

Một số điểm chú ý:

- Nhóm đối tượng dùng chung cho tất cả giải thuật:
    - `edaProblem`: mô tả bài toán và giúp chứa tất cả các dữ liệu (toàn cục) cần thiết khi giải bài toán.
    - `edaSolution`: mô tả một nghiệm đúng của bài toán.
    - `edaSolutionList`: một danh sách các nghiệm đúng của bài toán (thuộc lớp `edaSolution`).
    - `edaContinue`: mô tả điều kiện dừng của thuật toán meta-heuristic.
- Nhóm đối tượng dùng chung cho các meta-heuristic dạng dịch chuyển:
    - `edaMove`: đối tượng thể hiện một việc dịch chuyển trong không gian nghiệm.
    - `edaMoveGen`: đối tượng tạo ra các dịch chuyển theo chiến lược của người dùng. Đối tượng này (cùng với `edaMove`) giúp người dùng định nghĩa một cấu trúc lân cận đặc thù.

  - `edaMoveSelect`: đối tượng giúp hạn chế các lân cận có kích thước quá lớn, hoặc theo một định hướng chọn lựa khác với fitness.
- Nhóm đối tượng dùng chung cho các meta-heuristic dạng quần thể:
  - `edaGenne` và `edaChromosome`: các đối tượng phục vụ cho nhóm thuật toán tiến hoá.
  - `edaRepresentation`: đối tượng hổ trợ mã hoá nghiệm dưới dạng nhiễm sắc thể giúp dễ dàng và hiệu quả làm việc với quần thể. Thực chất `edaRepresentation` chính là đoạn mã mô tả các thức chuyển `edaSolution` thành `edaChromosome` và ngược lại.
  - `edaPopulation`: là một tập các nghiệm đã được mã hoá dưới dạng `edaRepresentation`. Hơn nữa, khi làm việc với các thuật toán dạng tiến hoá, thì đây là đối tượng làm việc chính. Chúng chính là danh sách các `edaChromosome`.
- Nhóm đối tượng đặc thù cho từng loại meta-heuristic. Dựa vào cây thừa kế và tương tác giữa các đối tượng, người dùng có thể phát triển thêm các phương pháp giải của mình. Chẳng hạn như nếu muốn điều chỉnh thuật toán làm nguội khác 2 thuật toán cung cấp bởi thư viện thì có thể thừa kế từ `edaCoolingSchedule`.

2.3. Đặc tả workflow

Sau khi đã có các giải thuật tìm kiếm cần thiết, người dùng sẽ tạo workflow cho chương trình. Như đã trình bày trong phần 1.3 thì việc xây dựng workflow chủ yếu là thông qua các hàm thành phần của lớp `edaWrapperControl`. Dựa trên lớp đối tượng trừu tượng này, thư viện meta-heuristic đã phát triển 3 lớp đối tượng phục vụ cho các hạ tầng tính toán khác nhau:

- `edaSeqWrapperControl`: là đối tượng giúp triển khai tính toán trên máy đơn, không có yêu cầu gì bổ sung về thư viện truyền nhận thông điệp (ví dụ MPI) cũng như không cần phải đóng gói dữ liệu.
- `edaParWrapperControl`: là đối tượng giúp triển khai tính toán trên hạ tầng tính toán lưới có cài đặt POP-C++.

- `edaMpiWrapperControl`: là đối tượng giúp triển khai tính toán trên hạ tầng tính toán có cài đặt MPI.

Với cùng một workflow, nếu người dùng sử dụng `edaSeqWrapperControl` thì tính toán sẽ thực hiện tuần tự, nhưng nếu dùng 2 loại đối tượng kia thì thư viện tự động chuyển đổi sang tính toán dạng song song/phân bố. Điều này đem lại sự tiện lợi rất nhiều cho người dùng. 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 sàng 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.

Ngoài ra, 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.
Sử dụng các đối tượng đặc tả workflow như thế nào sẽ được mô tả chi tiết trong ví dụ minh hoạ.

Tuy nhiên, để giúp cho thư viện triển khai tính toán được trên hạ tầng tính toán có bộ nhớ phân tán, người dùng phải biết và thực hiện thêm một số công việc trình bày trong phần sau.

2.4. Đóng gói và tái tạo thông tin

Để có thể thực thi song song, người sử dụng thư viện cần chú ý:

- Để có thể truyền nhận các đối tượng trong môi trường song song thì các lớp 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 trình 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. Người đọc có thể tìm hiểu thêm trong [Hoai14] để biết thêm nguyên lý làm việc của thư viện trên một hệ thống bộ nhớ phân tán.

CHƯƠNG 3: VÍ DỤ MINH HOẠ

Trong chương này, tài liệu kỹ thuật sẽ hướng dẫn một cách chi tiết cách thức thực thi quá trình tối ưu cho các dạng bài toán tối ưu tổ hợp và tối ưu liên tục. Tùy dạng bài toán chúng ta cần quan tâm mà việc ghép nối các chi tiết trong thư viện là khác nhau. Để nắm rõ quy trình xây dựng cho bài toán tối ưu mình quan tâm với sự hỗ trợ từ thư viện, người dùng sẽ được giới thiệu thông qua một bài toán cụ thể.

3.1. Bài toán TSP

TSP (Travelling Salesman Problem) là một 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. Người đọc có thể tìm thấy các tài liệu khác liên quan đến bài toán TSP, chẳng hạn [David06] và [Donald10].

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.

Sau đây, tài liệu hướng dẫn sẽ đi qua các phần cần thiết để giúp người dùng hiểu làm thế nào để dùng thư viện meta-heuristic để giải bài toán TSP.

3.2. Đặc tả bài toán

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 `Graph` 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.

3.3. 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 fitness.
- `getCost()`: lấy giá trị fitness dựa trên lần lượng giá trước đó.
- `setCost()`: thiết lập trực tiếp giá trị fitness.
- `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 fitness của lời giải – và đối tượng graph thuộc lớp `edaProblem`.

Thông thường với các đối tượng được kế thừa từ `edaSolution`, `edaMove`, `edaMoveGen`, `edaRepresentation`... thường mang có thuộc tính `edaProblem`, bởi lẽ thông tin bài toán luôn cần thiết để tính toán giá trị fitness. 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. Phương thức `classGenerateFromBuffer()` được định nghĩa sẵn trong lớp `edaSerialize` cũng như phương thức `doSerialize()`. 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***".

3.4. 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:

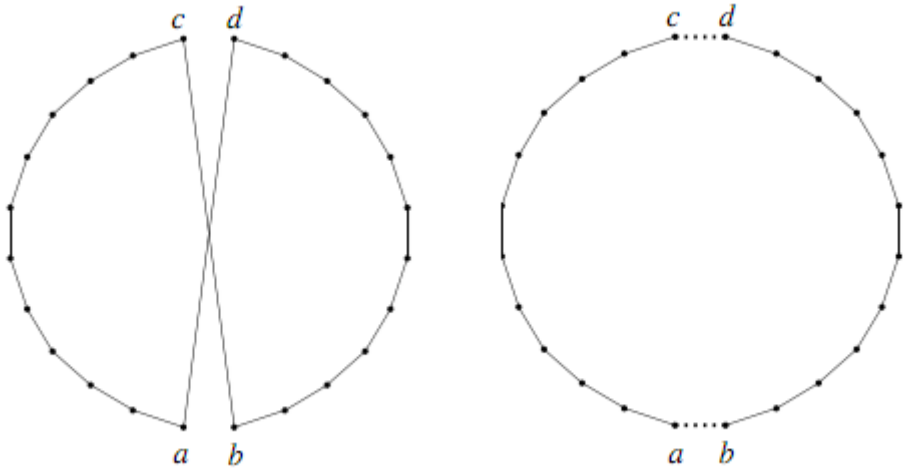
- `incrEval()`: hàm lượng giá, được tính dựa trên sự thay đổi khi áp dụng phép biến đổi hiện tại lên nghiệm. Hàm này thường được sử dụng để tính nhanh giá trị của fitness, cũng như trong các trường hợp giá trị fitness có thể tính theo dạng tăng dần (incrementally).
- `update()`: áp dụng bước chuyển hiện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

Lân cận 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.



Hình 10. Cơ chế tạo lời giải khả thi của 2-Opt.

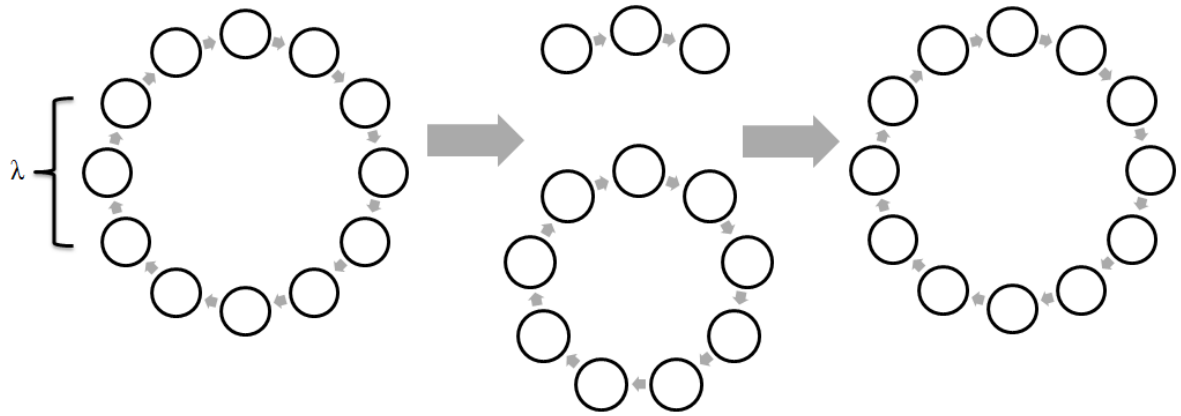
```
class tspTwoOpt: public pair<unsigned int, unsigned int>,
                 public edaMove
{
public:
  tspTwoOpt();
  tspTwoOpt(const tspTwoOpt &move);
  tspTwoOpt(const Graph &g);
  ~tspTwoOpt();
  edaMove* clone() const;
  void init(const edaSolution &sol);
  double incrEval(const edaSolution &sol) 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 fitness của bài toán.

Lân cận 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.



Hình 11. Cơ chế tạo lời giải khả thi của Or-Opt.

```
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.

3.5. 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 tìm kiếm tabu).

```
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`.

3.6. 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.

Đặc tả gen

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.

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 fitness 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());
}
```

3.7. Lựa chọn các giải thuật tìm kiếm và xây dựng Workflow:

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 ví dụ đưa 3 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);
```

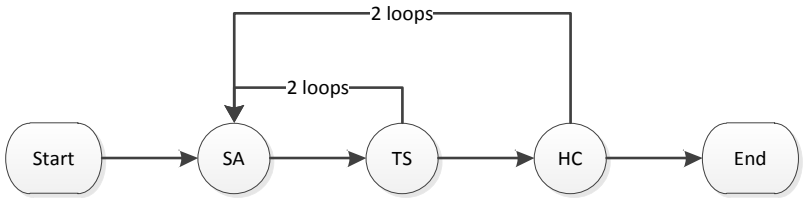
Sau đó, phương thức `insertEdge()` được sử dụng để mô tả sự phụ thuộc dữ liệu giữa các tác vụ:

```
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);
```

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



Hình 12. Workflow tính toán với ngữ cảnh lặp lồng nhau.

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 12).

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.

3.8. Thực thi tìm kiếm

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.

Để đánh giá chất lượng nghiệm đối với từng giải thuật, cũng như hiệu quả của việc kết hợp các giải thuật tối ưu khác nhau, nhiều sơ đồ tính toán được đưa ra để thử nghiệm. Các kết quả tính toán của các sơ đồ tốt trên sẽ được trình bày ở mục kết quả tính toán tại phần sau. Mục đích việc đưa hàng loạt sơ đồ tính toán nhằm minh họa sức mạnh và khả năng tùy biến của thư viện.

CHƯƠNG 4: CÁC TÍNH TOÁN THỬ NGHIỆM VÀ TIỆN ÍCH HỖ TRỢ

Như đã trình bày trong các phần trước, thư viện meta-heuristic llà một công cụ hỗ trợ tính toán tối ưu, và một trong các thế mạnh là có thể triển khai dễ dàng trên nhiều môi trường tính toán phổ biến hiện nay. Trong phần này, tài liệu sẽ giới thiệu cho người dùng 2 đề mục chính: tính toán thử nghiệm và các công cụ hỗ trợ để đánh giá kết quả tính toán.

4.1. Tính toán thử nghiệm

Để minh chứng cho việc tính toán thử nghiệm, bài toán được sử dụng là TSP đã được mô tả chỉ tiết trong chương 3. Ngoài ra, một môi trường tính toán song song cũng được thiết lập 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, một tính toán lên đến 32 quá trình MPI đồng thời (nếu không có vấn đề gì về bộ nhớ) có thể thực hiện được.

Các dữ liệu (benchmark) thử nghiệm được lấy từ nguồn Đại học Waterloo[1], Canada. Bộ dữ liệu này được sử dụng trong nhiều nghiên cứu kinh điển về nhóm bài toán *NP* và đặc biệt là bài toán TSP. Ở đây, chúng ta chỉ xem xét một số bài toán nhỏ bộ dữ liệu của Đại học Waterloo cụ thể gồm: berlin52, eil101 và ali535.

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

Qua việc thực thi và dò tìm theo phương pháp thử sai, nhóm nghiên cứu đã đưa ra các thông số cho từng loại giải thuật meta-heuristic đơn lẽ như sau:

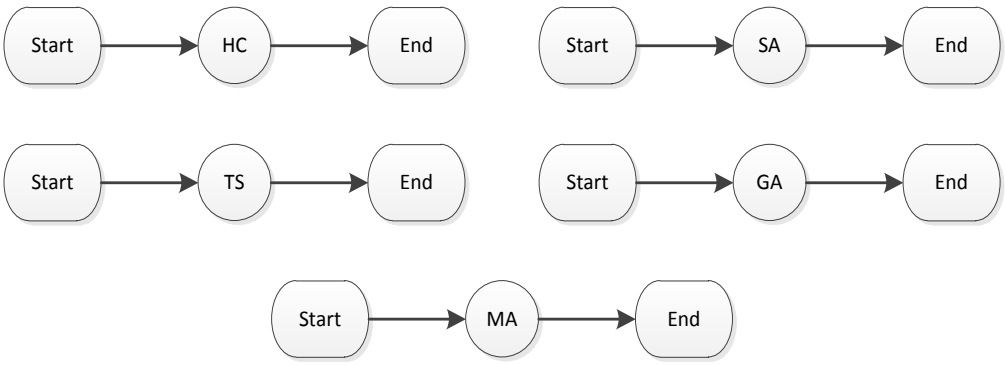
Bảng 1. Các thông số chi từng loại giải thuật meta-heuristic đơn lẽ.

	Số lời giải	**Số lần lặp**	**Chiến lược**
HC	3		Global Best + 2-Opt hoặc Or-Opt
SA	3	1500	ExpCooling(0.01, 0.98) + 2-Opt + T(50°)
TS	3	1500	Global Best + 2-Opt
GA	80	50	Roulette Wheel (0.5) + PMX + Swap(0.05)
MA	80	50	Roulette Wheel (0.5) + PMX + Swap(0.05) + HC(2-Opt)

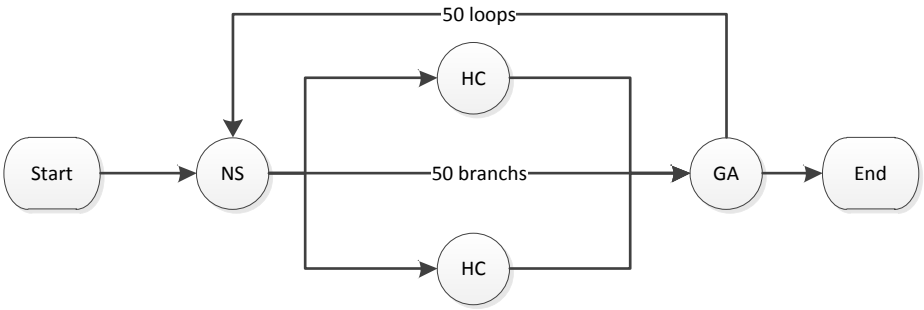
Như đã đề cập trong các phần trước, sau khi xây dựng đầy đủ các cấu trúc lân cận (dành cho nhóm thuật toán dựa trên dịch chuyển), và các cấu trúc biểu diễn, mã hoá,…(dành cho nhóm thuật toán quần thể) thì người dùng có thể triển khai nhiều dạng workflow khác nhau. Cụ thể trong ví dụ là chia thành 2 nhóm:

- Nhóm 1: các workflow chỉ sử dụng đơn lẻ một meta-heuristic như hình 13.
- Nhóm 2: các workflow dựa trên sự kết hợp các meta-heuristic lại với nhau như các hình từ 14 đến 18.

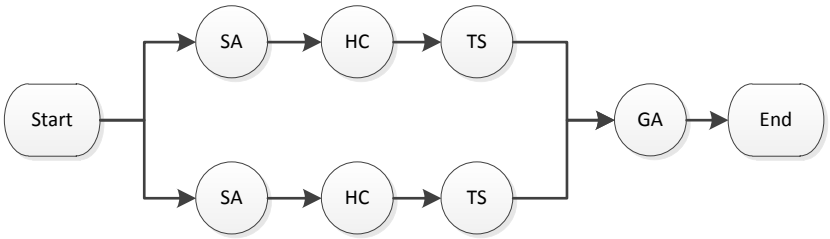
Rõ ràng là chỉ với một bộ 5 meta-heuristic được thiết lặp sẵn với các tham số tuỳ chọn bởi người dùng, rất nhiều khả năng xây dựng các workflow với các định hướng khác nhau. Báo cáo này không đi sâu vào phân tích sự lựa chọn, mà chỉ muốn nhấn mạnh tính tiện dụng của thư viện.



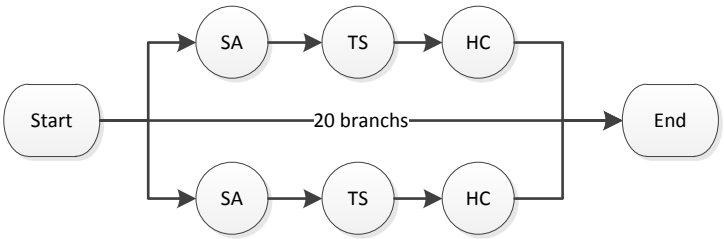
Hình 13. Các sơ đồ tối ưu đối với từng loại giải thuật meta-heuristic đơn lẽ.



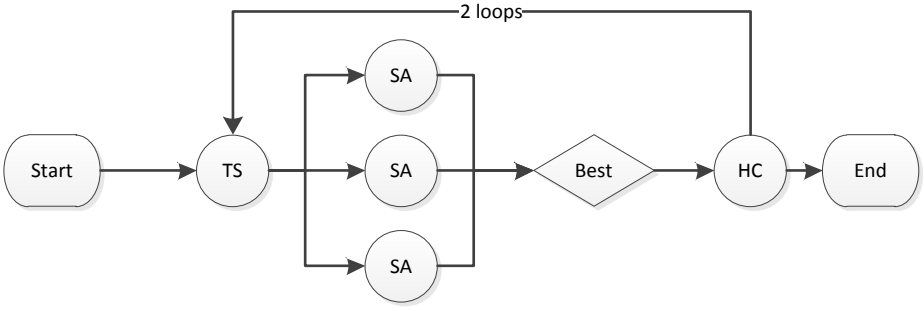
Hình 14. Hiện thực giải thuật MA song song (MPIMA) bằng workflow trên nền tảng kết hợp giải thuật HC và GA.



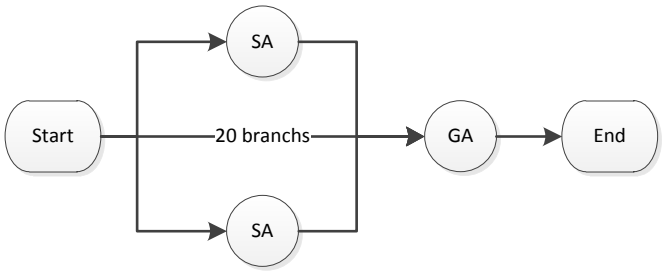
Hình 15. Workflow của SEQ01 & MPI01.



Hình 16. Workflow của SEQ02 & MPI02.



Hình 17. Workflow của SEQ03 & MPI03.



Hình 18. Workflow của SEQ04 & MPI04.

Bảng dưới đây cung cấp các kết quả tính toán khi sử dụng các workflow khác nhau.

Bảng 2. Kết quả tối ưu với các workflow tính toán trên benchmark berlin52.

Workflow	**Thời gian tính (giây)**	**Giá trị hàm mục tiêu**			
		Tốt nhất	**Xấu nhất**	**Trung bình**	**Concorde**
HC 2-Opt	0.022	-8228	-8408	-8326	-7542 (nghiệm tối ưu)
HC Or-Opt	0.041	-10752	-11328	-10947	
SA	0.327	-7760	-7986	-7871	
TS	0.318	-8043	-8476	-8282	
GA	0.170	-20368	-22029	-20659	
MA	0.456	-7598	-7598	-7598	
MPIMA	3.678	-7544	-9781	-7851	
VNS	0.047	-8678	-9347	-8978	
SEQ01	0.930	-7549	-11713	-7646	
MPI01	2.483	-7782	-9833	-7906	
SEQ02	0.216	-7865	-9353	-8415	
MPI02	1.250	-8328			
SEQ03	0.076	-9074			
MPI03	1.248	-8616			
SEQ04	0.209	-7865	-9271	-8021	
MPI04	1.250	-8959	-12143	-9165	

Bảng 3. Kết quả tối ưu với các workflow tính toán trên benchmark eil101.

Workflow	**Thời gian tính (giây)**	**Giá trị hàm mục tiêu**			
		Tốt nhất	**Xấu nhất**	**Trung bình**	**Concorde**
HC 2-Opt	0.077	-713	-747	-729	-629

HC Or-Opt	0.100	-879	-959	-908	(nghiệm tối ưu)
SA	0.831	-1267	-1280	-1272	
TS	1.000	-724	-732	-729	
GA	0.583	-2383	-2577	-2457	
MA	3.293	-649	-649	-649	
MPIMA	15.890	-643	-774	-646	
VNS	0.185	-729	-792	-758	
SEQ01	3.581	-675	-846	-689	
MPI01	4.408	-681	-833	-687	
SEQ02	0.713	-694	-746	-718	
MPI02	1.382	-694			
SEQ03	0.227	-723			
MPI03	1.284	-709			
SEQ04	0.352	-1249	-1344	-1255	
MPI04	1.345	-1347	-1498	-1358	

Bảng 4. Kết quả tối ưu với các workflow tính toán trên benchmark ali535.

Workflow	**Thời gian tính (giây)**	**Giá trị hàm mục tiêu**			
		Tốt nhất	**Xấu nhất**	**Trung bình**	**Concorde**
HC 2-Opt	28.504	-2180	-2341	-2238	-1982 (nghiệm tối ưu)
HC Or-Opt	34.830	-3805	-4054	-3942	
SA	6.551	-8043	-8196	-8118	
TS	43.510	-2190	-2240	-2212	
GA	25.304	-32544	-33904	-32958	
MA	747.110	-2033	-2033	-2033	
MPIMA	1296.560	-2048	-2463	-2067	
VNS	37.733	-2401	-2779	-2598	
SEQ01	222.820	-4554	-5451	-4576	
MPI01	156.347	-4530	-5000	-4554	
SEQ02	53.750	-4704	-5335	-5028	
MPI02	10.044	-5019			
SEQ03	5.035	-3755			
MPI03	4.959	-3908			
SEQ04	25.016	-9506	-10104	-9554	
MPI04	8.878	-10009	-10009	-10009	

4.2. 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 tập tin 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 văn bản phổ biến trong các nền tảng hệ điều hành gồm cả Linux, Unix, Window và MacOS. Người dùng có thể quan sát các kết quả từ tập tin để có được cái nhìn tổng quan từ quá trình tối ưu. Các thông tin được ghi lại trong tập tin bao gồm: loại thuật toán tối ưu đang thực thi, ID (số thứ tự) của đỉnh (đại diện cho thuật toán meta-heuristic), 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 fitness 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.

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

Hình 19. Tập tin nhật ký được ghi lại trong quá trình tối ưu của giải thuật GA.

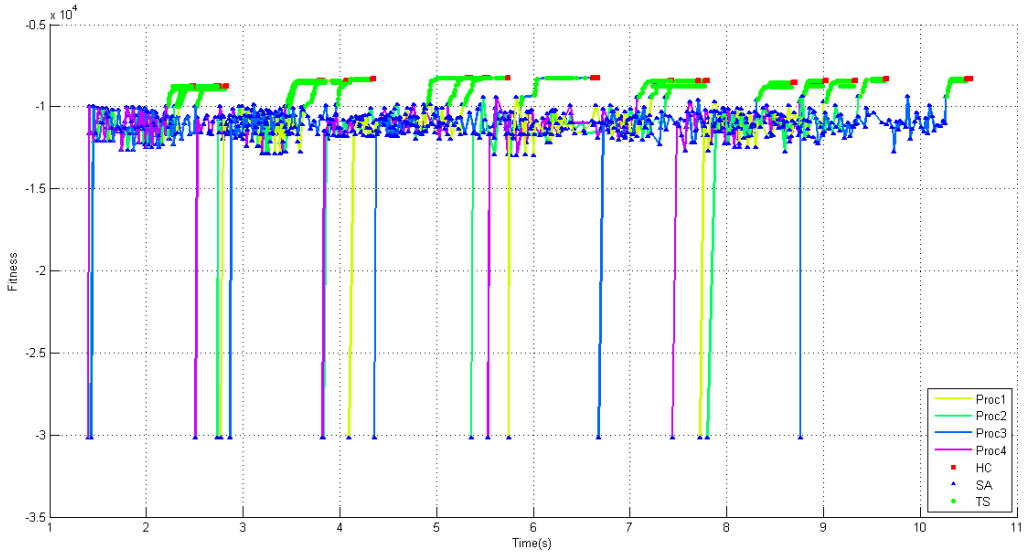
Để trực quan hóa thông tin của tập tin nhật ký, nhóm phát triển thư viện cũng đã hiện thực một script file trong môi trường MATLAB (người dùng linux có thể sử dụng với Octave để thay thế). Cú pháp để hiện thị thông tin của log file là:

```
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 tập tin). Hình 20 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 fitness, 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, tài liệu hướng dẫn sẽ không đi sâu vào chi tiết. Người dùng, cần có những kiến thức nhất định về thư viện và meta-heuristic thì mới hiệu chỉnh các thông số của quá trình tối ưu hiệu quả.



Hình 20. Biểu đồ hiển thị log file của quá trình tối ưu với các tác vụ tối HC, SA, TS với 4 đơn vị tính toán song song.

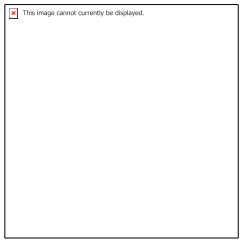
Phụ lục A. Thông tin về cấu trúc cơ sở dữ liệu

edaAdaption Class Tham chiếu

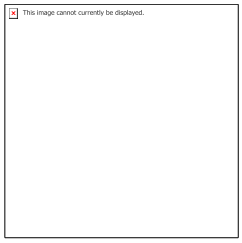
Lớp **edaAdaption** thực hiện việc cải tiến trên danh sách lời giải đầu vào dựa trên các giải thuật tìm kiếm cục bộ theo một tỷ lệ cho trước.

```
#include <edaAdaption.h>
```

Sơ đồ kế thừa cho edaAdaption:



Sơ đồ liên kết cho edaAdaption:



Các hàm thành viên Public

- **edaAdaption** ()
  Khởi tạo đối tượng.
- **edaAdaption** (**edaSearch** *lSearch, double rate=1.0)
- **edaAdaption** (const **edaAdaption** &adapt)
- **~edaAdaption** ()
  Hủy đối tượng.
- **edaAdaption** * **clone** () const
  Nhân bản đối tượng.
- void **setRate** (const double &rate)
- void **printOn** (ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.
- void **update** (**edaSolutionList** &list) const
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAADAPTION_)
  *Gán ID nhận dạng thuộc lớp **edaAdaption** cho đối tượng.*

Mô tả chi tiết

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.

Thông tin về Constructor và Destructor

*edaAdaption::edaAdaption (edaSearch * lSearch, double rate = `1.0`)*

Khởi tạo đối tượng

Các tham số:

lSearch	Toán tử tìm kiếm cục bộ trong đối tượng
rate	Tỷ lệ cải tiến trong danh sách lời giải (0 < rate < 1)

edaAdaption::edaAdaption (const edaAdaption & adapt)

Khởi tạo đối tượng

Các tham số:

adapt	Đối tượng cần sao chép

Thông tin về hàm thành viên

void edaAdaption::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

void edaAdaption::setRate (const double & rate)

Thiết lập tỷ lệ cải tiến trong danh sách lời giải

Các tham số:

rate	Tỷ lệ cải tiến trong danh sách lời giải (0 < rate < 1)

void edaAdaption::update (edaSolutionList & list) const

Thực hiện việc cải tiến lên danh sách lời giải

Các tham số:

list	Danh sách các lời giải cần cải tiến

Thông tin cho class được biên soạn từ các file sau đây:

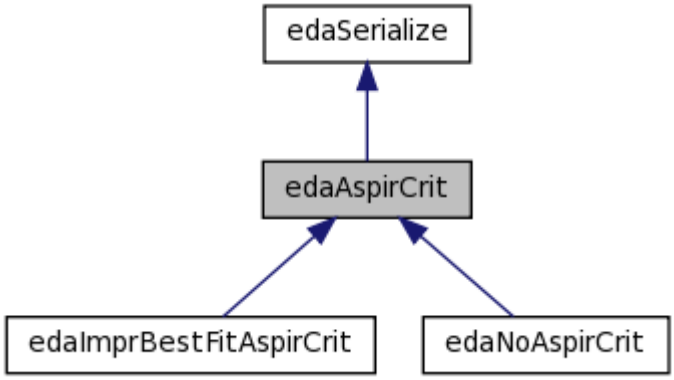
- lib/edaAdaption.h
- lib/edaAdaption.cpp

edaAspirCrit Class Tham chiếu

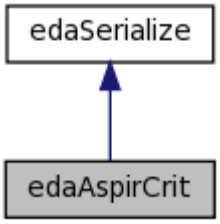
Lớp trừu tượng **edaAspirCrit** thực hiện việc kiểm tra sự hợp lệ của các bước chuyển ứng viên.

```
#include <edaAspirCrit.h>
```

Sơ đồ kế thừa cho edaAspirCrit:



Sơ đồ liên kết cho edaAspirCrit:



Các hàm thành viên Public

- **edaAspirCrit** ()
  Khởi tạo đối tượng.
- **~edaAspirCrit** ()
  Hủy đối tượng.
- virtual **edaAspirCrit** * **clone** () const =0
- virtual void **init** ()=0
  Khởi động đối tượng.
- virtual bool **check** (const **edaMove** *move, double fitness)=0
- **setClassID** (_SYSCLASSID_+_CLSID_EDAASPIRCRIT_)
  *Gán ID nhận dạng thuộc lớp **edaAdaption** cho đối tượng.*

Mô tả chi tiết

Lớp trừu tượng **edaAspirCrit** thực hiện việc kiểm tra sự hợp lệ của các bước chuyển ứng viên.

Thông tin về hàm thành viên

*virtual bool edaAspirCrit::check (const edaMove * move, double fitness)* `[pure virtual]`

Kiểm tra bước chuyển move trong giải thuật Tabu có được chấp nhận không

Các tham số:

move	Bước trong cần kiểm tra
fitness	Lượng giá của bước chuyển

Được thực hiện trong **edaImprBestFitAspirCrit** (*tr.98*), và **edaNoAspirCrit** (*tr.120*).

virtual edaAspirCrit edaAspirCrit::clone () const* `[pure virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Được thực hiện trong **edaImprBestFitAspirCrit** (*tr.98*), và **edaNoAspirCrit** (*tr.120*).

Thông tin cho class được biên soạn từ các file sau đây:

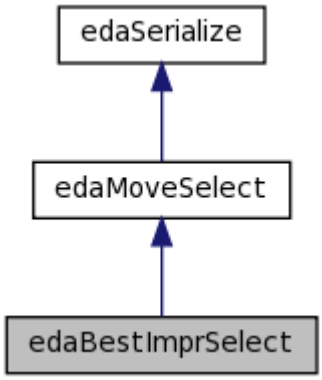
- lib/edaAspirCrit.h

edaBestImprSelect Class Tham chiếu

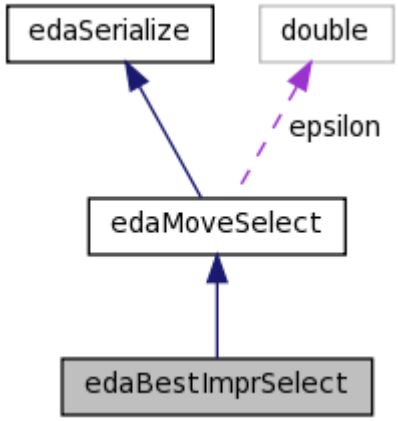
Lớp **edaBestImprSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược Global Best.

`#include <edaBestImprSelect.h>`

Sơ đồ kế thừa cho edaBestImprSelect:



Sơ đồ liên kết cho edaBestImprSelect:



Các hàm thành viên Public

- **edaBestImprSelect** (const double epsilon=1e-4)
  Khởi tạo đối tượng.
- **~edaBestImprSelect** ()
  Hủy đối tượng.
- **edaMoveSelect** * **clone** () const
  Nhân bản đối tượng.
- void **init** (double fitness)
  Khởi động đối tượng.
- bool **update** (const **edaMove** *move, double fitness)
- bool **save** (**edaMove** *move, double &fitness) const
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDABESTIMPRSELECT_)
  *Gán ID nhận dạng thuộc lớp **edaAdaption** cho đối tượng.*

Additional Inherited Members

Mô tả chi tiết

Lớp **edaBestImprSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược Global Best.

Thông tin về hàm thành viên

*bool edaBestImprSelect::save (edaMove * move, double & fit) const* `[virtual]`

Lấy thông tin bước chuyển ứng với chiến lược tương ứng

Các tham số:

move	Bước chuyển ứng với chiến lược đề ra
fitness	Lượng giá của bước chuyển

Thực hiện **edaMoveSelect** (*tr.112*).

void edaBestImprSelect::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Thực hiện **edaMoveSelect** (*tr.112*).

*bool edaBestImprSelect::update (const edaMove * move, double fit)* `[virtual]`

Thực hiện việc chọn lựa với bước chuyển theo chiến lược tương ứng

Các tham số:

move	Bước chuyển ứng viên
fitness	Lượng giá của bước chuyển

Thực hiện **edaMoveSelect** (*tr.113*).

Thông tin cho class được biên soạn từ các file sau đây:

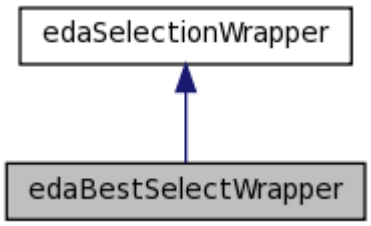
- lib/edaBestImprSelect.h
- lib/edaBestImprSelect.cpp

edaBestSelectWrapper Class Tham chiếu

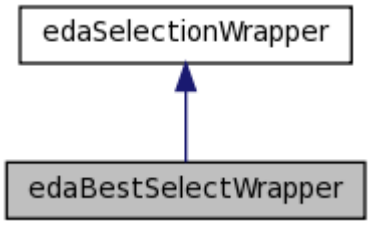
Lớp **edaBestSelectWrapper** thực hiện việc chọn lời giải có hàm lượng giá tốt nhất trong tập lời giải được đưa vào tại mỗi nút tìm kiếm.

```
#include <edaBestSelectWrapper.h>
```

Sơ đồ kế thừa cho edaBestSelectWrapper:



Sơ đồ liên kết cho edaBestSelectWrapper:



Các hàm thành viên Public

- **edaBestSelectWrapper** ()
  Khởi tạo đối tượng.
- **edaBestSelectWrapper** * **clone** () const
- bool **select** (**edaSolutionList** &list) const

Mô tả chi tiết

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.

Thông tin về hàm thành viên

*edaBestSelectWrapper * edaBestSelectWrapper::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSelectionWrapper** (*tr.151*).

bool edaBestSelectWrapper::select (edaSolutionList & list) const `[virtual]`

Chọn lựa các lời giải với chiến lược chọn tương ứng trong tập lời giải

Giá trị trả về:

Nhận giá trị TRUE nếu chọn lựa thành công, ngược lại giá trị đầu ra là FALSE

Thực hiện **edaSelectionWrapper** (*tr.151*).

Thông tin cho class được biên soạn từ các file sau đây:

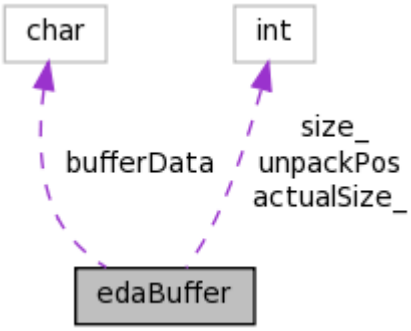
- lib/edaBestSelectWrapper.h
- lib/edaBestSelectWrapper.cpp

edaBuffer Class Tham chiếu

Lớp **edaBuffer** thực việc đóng gói (pack) và mở gói (unpack) các loại dữ liệu cơ bản.

`#include <edaBuffer.h>`

Sơ đồ liên kết cho edaBuffer:



Các hàm thành viên Public

- void **reset** ()
  Xóa tất cả các thông tin đang lưu giữ trong đối tượng.
- void **Pack** (const char *data, int count=1)
- void **UnPack** (char *data, int count=1)
- void **Pack** (const unsigned char *data, int count=1)
- void **UnPack** (unsigned char *data, int count=1)

- void **Pack** (const bool *data, int count=1)
- void **UnPack** (bool *data, int count=1)
- void **Pack** (const int *data, int count=1)
- void **UnPack** (int *data, int count=1)
- void **Pack** (const unsigned *data, int count=1)
- void **UnPack** (unsigned *data, int count=1)
- void **Pack** (const long *data, int count=1)
- void **UnPack** (long *data, int count=1)
- void **Pack** (const unsigned long *data, int count=1)
- void **UnPack** (unsigned long *data, int count=1)
- void **Pack** (const short *data, int count=1)
- void **UnPack** (short *data, int count=1)
- void **Pack** (const unsigned short *data, int count=1)
- void **UnPack** (unsigned short *data, int count=1)
- void **Pack** (const float *data, int count=1)
- void **UnPack** (float *data, int count=1)
- void **Pack** (const double *data, int count=1)
- void **UnPack** (double *data, int count=1)
- char * **getBuffer** ()
- int **getActualSize** () const
- void **setBuffer** (int as, char *buf)

Các hàm thành viên Protected

- bool **checkUnPack** (int sz)

các thuộc tính Protected

- char * **bufferData**
- int **size_**
- int **actualSize_**
- int **unpackPos**

Mô tả chi tiết

Lớp **edaBuffer** thực việc đóng gói (pack) và mở gói (unpack) các loại dữ liệu cơ bản.

Thông tin về hàm thành viên

int edaBuffer::getActualSize () const `[inline]`

Lấy kích thước dữ liệu hiện lưu trữ trong đối tượng

Giá trị trả về:

Kích thước dữ liệu được lưu giữ trong đối tượng

char edaBuffer::getBuffer ()* `[inline]`

Lấy dữ liệu trong đối tượng

Giá trị trả về:

Dữ liệu nhị phân được lưu giữ trong đối tượng

*void edaBuffer::setBuffer (int as, char * buf)* `[inline]`

Gán thông tin dữ liệu cho đối tượng

Các tham số:

as	Kích thước của dữ liệu (bội số của giá trị byte)
buf	Chuỗi chứa thông tin dữ liệu dưới dạng nhị phân

Thông tin cho class được biên soạn từ các file sau đây:

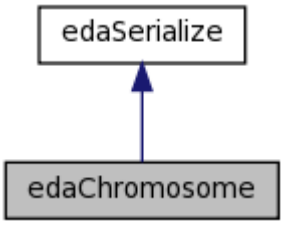
- lib/edaBuffer.h
- lib/edaBuffer.cpp

edaChromosome Class Tham chiếu

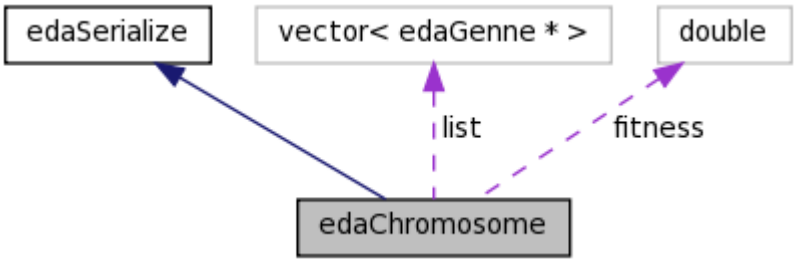
Lớp **edaChromosome** chứa thông tin về nhiễm sắc thể và các phương thức tương ứng.

`#include <edaChromosome.h>`

Sơ đồ kế thừa cho edaChromosome:



Sơ đồ liên kết cho edaChromosome:



Các hàm thành viên Public

- **edaChromosome** ()
  Khởi tạo đối tượng.
- **edaChromosome** (const **edaChromosome** &chro)
- virtual **~edaChromosome** ()
  Hủy đối tượng.
- virtual **edaChromosome** * **clone** () const
  Nhân bản đối tượng.
- virtual void **addGenne** (const **edaGenne** &genne)
- virtual **edaGenne** * **getGenne** (unsigned int index) const
- virtual double **getFitness** () const
- virtual void **setFitness** (double finess)
- virtual unsigned int **getLength** () const
- virtual **edaChromosome** & **operator=** (const **edaChromosome** &chro)
- virtual bool **operator==** (const **edaChromosome** &chro) const

- virtual bool **operator!=** (const **edaChromosome** &chro) const
- virtual void **printOn** (ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDACHROMOSOME_)
  *Gán ID nhận dạng thuộc lớp **edaAdaption** cho đối tượng.*

Các hàm thành viên Protected

- virtual void **easer** ()

các thuộc tính Protected

- vector< **edaGenne** * > **list**
- double **fitness**

Mô tả chi tiết

Lớp **edaChromosome** chứa thông tin về nhiễm sắc thể và các phương thức tương ứng.

Thông tin về Constructor và Destructor

edaChromosome::edaChromosome (const edaChromosome & chro)

Khởi tạo đối tượng

Các tham số:

chro	Đối tượng cần sao chép

Thông tin về hàm thành viên

void edaChromosome::addGenne (const edaGenne & genne) `[virtual]`

Thêm đoạn gen và cuối nhiễm sắc thể

Các tham số:

genne	Đoạn gen cần thêm

double edaChromosome::getFitness () const `[virtual]`

Lấy giá trị lượng giá của nhiễm sắc thể

Giá trị trả về:

Giá trị lượng giá của nhiễm sắc thể

*edaGenne * edaChromosome::getGenne (unsigned int index) const* `[virtual]`

Lấy đoạn gen từ nhiễm sắc thể

Các tham số:

index	Chỉ số của đoạn cần lấy

Giá trị trả về:

genne Đoạn gen cần lấy

unsigned int edaChromosome::getLength () const `[virtual]`

Lấy chiều dài (số lượng gen) của nhiễm sắc thể

Giá trị trả về:

Chiều dài của nhiễm sắc thể

void edaChromosome::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

void edaChromosome::setFitness (double finess) `[virtual]`

Gán giá trị lượng giá của nhiễm sắc thể

Các tham số:

finess	Giá trị lượng giá của nhiễm sắc thể

Thông tin cho class được biên soạn từ các file sau đây:

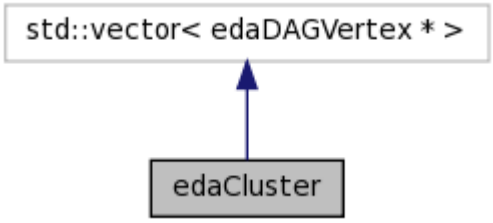
- lib/edaChromosome.h
- lib/edaChromosome.cpp

edaCluster Class Tham chiếu

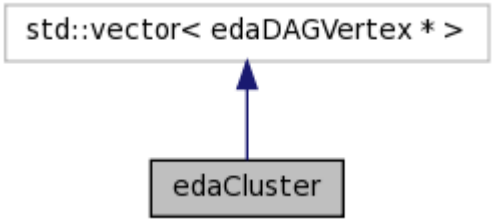
Lớp **edaCluster** chứa thông tin vì các cụm node tìm kiếm và các thao tác trên chúng.

`#include <edaCluster.h>`

Sơ đồ kế thừa cho edaCluster:



Sơ đồ liên kết cho edaCluster:



Các hàm thành viên Public

- **edaCluster** ()
  Khởi tạo đối tượng.
- virtual **~edaCluster** ()
  Hủy đối tượng.
- bool **isOverlap** (const **edaCluster** &_clus)
  Kiểm tra xem 2 cụm node tìm kiếm có phủ nhau hay không.
- bool **isCover** (const **edaCluster** &_clus)
  Kiểm tra xem 2 cụm node tìm kiếm có chứa nhau hay không.
- void **printOn** (ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.

Mô tả chi tiết

Lớp **edaCluster** chứa thông tin vì các cụm node tìm kiếm và các thao tác trên chúng.

Thông tin cho class được biên soạn từ các file sau đây:

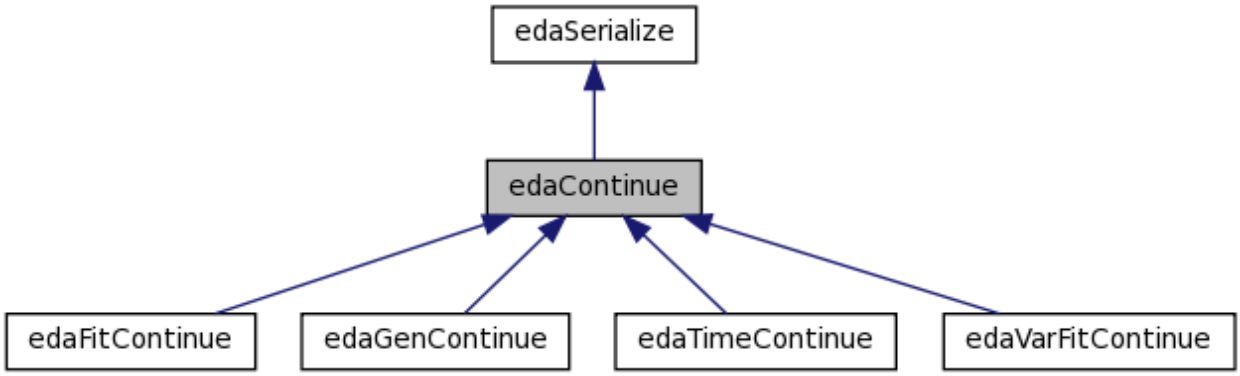
- lib/edaCluster.h
- lib/edaCluster.cpp

edaContinue Class Tham chiếu

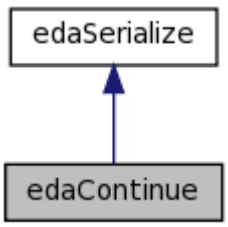
Lớp trừu tượng **edaContinue** mô tả các thông tin về điều kiện dừng trong thư viện.

```
#include <edaContinue.h>
```

Sơ đồ kế thừa cho edaContinue:



Sơ đồ liên kết cho edaContinue:



Các hàm thành viên Public

- virtual **~edaContinue** ()
  Hủy đối tượng.
- virtual **edaContinue** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **init** ()=0
  Khởi động đối tượng.
- virtual bool **check** (const **edaSolutionList** &list)=0
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- virtual void **printOn** (ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.

Mô tả chi tiết

Lớp trừu tượng **edaContinue** mô tả các thông tin về điều kiện dừng trong thư viện.

Thông tin về hàm thành viên

bool edaContinue::check (const edaSolutionList & list) `[pure virtual]`

Kiểm tra điều kiện dừng

Các tham số:

list	Danh sách các lời giải cần kiểm tra với điều kiện dừng

Được thực hiện trong **edaTimeContinue** (*tr.171*), **edaGenContinue** (*tr.88*), **edaFitContinue** (*tr.83*), và **edaVarFitContinue** (*tr.179*).

void edaContinue::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Được thực thi lại trong **edaTimeContinue** (*tr.171*), **edaGenContinue** (*tr.89*), **edaFitContinue** (*tr.83*), và **edaVarFitContinue** (*tr.179*).

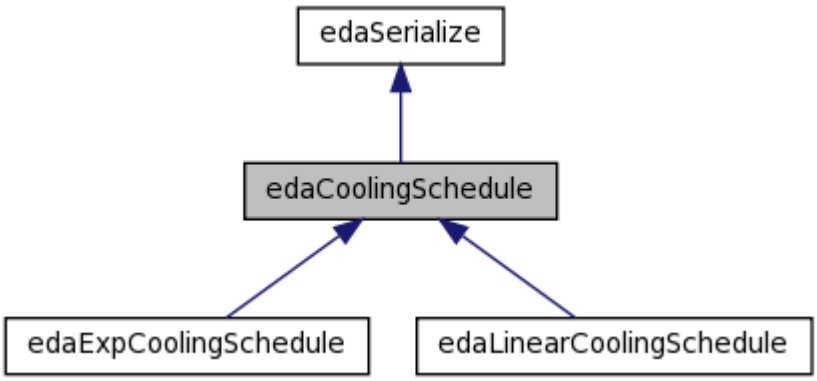
Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaContinue.h
- lib/edaContinue.cpp

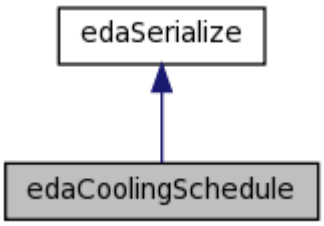
edaCoolingSchedule Class Tham chiếu

Lớp trừu tượng **edaCoolingSchedule** mô tả các thông tin về chiến lược làm

lạnh nhiệt độ.
`#include <edaCoolingSchedule.h>`
Sơ đồ kế thừa cho edaCoolingSchedule:



Sơ đồ liên kết cho edaCoolingSchedule:



Các hàm thành viên Public

- virtual **edaCoolingSchedule** * **clone** () const =0
  Nhân bản đối tượng.
- virtual bool **check** (double &temperature)=0
- virtual void **Serialize** (**edaBuffer** &buf, bool &pack)
- virtual void **printOn** (std::ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.

Mô tả chi tiết

Lớp trừu tượng **edaCoolingSchedule** mô tả các thông tin về chiến lược làm lạnh nhiệt độ.

Thông tin về hàm thành viên

virtual bool edaCoolingSchedule::check (double & temperature) `[pure virtual]`

Kiểm tra nhiệt độ hiện tại với ngưỡng nhiệt độ

Các tham số:

temperature	Nhiệt độ hiện tại

Được thực hiện trong **edaLinearCoolingSchedule** (*tr.104*), và **edaExpCoolingSchedule** (*tr.80*).

Thông tin cho class được biên soạn từ các file sau đây:

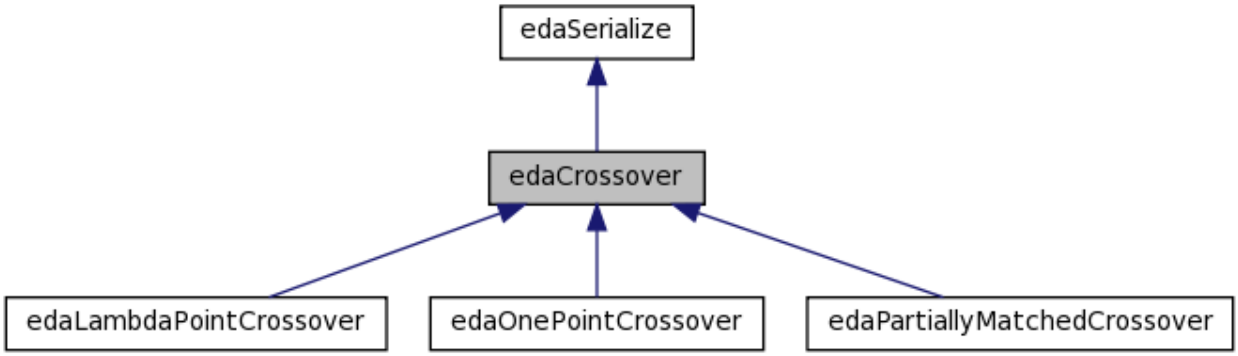
- lib/edaCoolingSchedule.h
- lib/edaCoolingSchedule.cpp

edaCrossover Class Tham chiếu

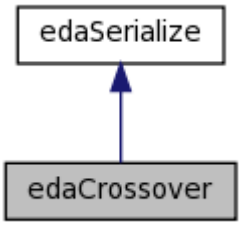
Lớp trừu tượng **edaCrossover** mô tả các thông tin về chiến lược lai chéo giữa các nhiệm sắc thể.

`#include <edaCrossover.h>`

Sơ đồ kế thừa cho edaCrossover:



Sơ đồ liên kết cho edaCrossover:



Các hàm thành viên Public

- **edaCrossover** ()
  Khởi tạo đối tượng.
- **edaCrossover** (const **edaCrossover** &cross)
  Khởi tạo đối tượng với đối tượng cần sao chép.
- virtual **~edaCrossover** ()
  Hủy đối tượng.
- virtual **edaCrossover** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **update** (**edaPopulation** &pop)=0
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)=0
- virtual void **printOn** (ostream &os) const

Mô tả chi tiết

Lớp trừu tượng **edaCrossover** mô tả các thông tin về chiến lược lai chéo giữa các nhiệm sắc thể.

Thông tin về hàm thành viên

virtual void edaCrossover::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Được thực hiện trong **edaOnePointCrossover** (*tr.126*), **edaPartiallyMatchedCrossover** (*tr.128*), và **edaLambdaPointCrossover** (*tr.102*).

virtual void edaCrossover::update (edaPopulation & pop) `[pure virtual]`

Thực hiện việc cập nhật tập dân cư với chiến lược tương ứng với từng hiện thực cụ thể

Các tham số:

pop	Tập dân cư cần lai chéo

Được thực hiện trong **edaOnePointCrossover** (*tr.126*), **edaPartiallyMatchedCrossover** (*tr.128*), và **edaLambdaPointCrossover** (*tr.102*).

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaCrossover.h
- lib/edaCrossover.cpp

edaDAG< DataType > Class Template Tham chiếu

Lớp **edaDAG** hiện thực đồ thị có hướng không vòng với kiểu dữ liệu tùy biến.
`#include <edaDAG.h>`

Các hàm thành viên Public

- **edaDAG** ()
  Khởi tạo đồ thị
- **~edaDAG** ()
  Hủy đồ thị
- int **count** () const
  Lấy chiều dài dữ liệu.
- void **insertVertex** (int key, DataType &_data)
- void **insertEdge** (int key, int fromVertex, int toVertex)
- bool **insertLoop** (int key, int &fromVertex, int &toVertex)
- bool **clusterCheck** (const int &fromVertex, const int &toVertex)
- **edaCluster** * **getCluster** (const int &fromVertex, const int &toVertex)
- **edaCluster** * **getCluster** (const **edaDAGEdge** *loop)

- **edaDAGEdge** * **findLoop** (int taskID, map< int, int > &loopStatus)
- vector< **edaDAGEdge** * > * **detectSubLoop** (const **edaDAGEdge** &loop)
- **edaDAGEdge** * **detectLoop** (int taskID, map< int, int > &loopStatus)
- **edaDAGEdge** * **getLoop** (int loopID)
  Lấy vòng lặp từ thông tin khóa của chính nó
- bool **cycleCheck** (int &cycleVertex)
  Kiểm tra sự tồn tại vòng trong đồ thị
- **edaDAGVertex** * **getVertex** (int key)
  Lấy đỉnh từ thông tin khóa của chính nó
- **edaDAGEdge** * **getEdge** (int key)
  Lấy cạnh từ thông tin khóa của chính nó
- DataType & **operator[]** (int key)
- vector< int > **getParentNodes** (int key) const
  Lấy các đỉnh cha từ thông tin khóa của đỉnh con.
- vector< int > **traverse** () const
  Duyệt qua đồ thị

Mô tả chi tiết

template<class DataType>class edaDAG< DataType >

Lớp **edaDAG** hiện thực đồ thị có hướng không vòng với kiểu dữ liệu tùy biến.

Thông tin về hàm thành viên

template<class DataType> bool edaDAG< DataType >::clusterCheck (const int & fromVertex, const int & toVertex) `[inline]`

Kiểm tra vòng kết nối thỏa điều kiện cụm node hay không

Các tham số:

fromVertex	Khóa của đỉnh bắt đầu
toVertex	Khóa của đỉnh kết thúc

template<class DataType> edaDAGEdge edaDAG< DataType >::detectLoop (int taskID, map< int, int > & loopStatus)* `[inline]`

Dò tìm các vòng lặp liên kết với một node có khóa cho trước chưa xử lý torng đồ thị

Các tham số:

taskID	Khóa của node
loopStatus	Trạng thái xử lý của vòng lặp

Giá trị trả về:

Tập các cạnh **edaDAGEdge** là vòng lặp với chưa xử lý trong đồ thị

template<class DataType> vector<edaDAGEdge>* edaDAG< DataType >::detectSubLoop (const edaDAGEdge & loop)* `[inline]`

Dò tìm các vòng lặp con thuộc vòng lặp cho trước

Các tham số:

loop	Vòng lặp cần tìm các vòng lặp con

Giá trị trả về:

Tập các cạnh **edaDAGEdge** là vòng lặp con

template<class DataType> edaDAGEdge edaDAG< DataType >::findLoop (int taskID, map< int, int > & loopStatus)* `[inline]`

Tìm vòng lặp liên kết với một node có khóa cho trước ứng với trạng thái sẵn sàng

Các tham số:

taskID	Khóa của node
loopStatus	Trạng thái xử lý của vòng lặp

Giá trị trả về:

Tập các cạnh **edaDAGEdge** là vòng lặp với trạng thái chờ sẵn sàng

template<class DataType> edaCluster edaDAG< DataType >::getCluster (const int & fromVertex, const int & toVertex)* `[inline]`

Lấy cụm node từ vòng kết nối

Các tham số:

fromVertex	Khóa của đỉnh bắt đầu
toVertex	Khóa của đỉnh kết thúc

Giá trị trả về:

Đối tượng **edaCluster** thuộc vòng kết nối

template<class DataType> edaCluster edaDAG< DataType >::getCluster (const edaDAGEdge * loop)* `[inline]`

Lấy cụm node từ vòng kết nối

Các tham số:

loop	Vòng lặp thuộc đồ thị

Giá trị trả về:

Đối tượng **edaCluster** thuộc vòng lặp

template<class DataType> void edaDAG< DataType >::insertEdge (int key, int fromVertex, int toVertex) `[inline]`

Tạo cạnh mới

Các tham số:

key	Khóa của cạnh
fromVertex	Khóa của đỉnh bắt đầu
toVertex	Khóa của đỉnh kết thúc

template<class DataType> bool edaDAG< DataType >::insertLoop (int key, int & fromVertex, int & toVertex) `[inline]`

Tạo vòng lặp mới

Các tham số:

key	Khóa của vòng lặp
fromVertex	Khóa của đỉnh bắt đầu
toVertex	Khóa của đỉnh kết thúc

template<class DataType> void edaDAG< DataType >::insertVertex (int key, DataType & _data) `[inline]`

Tạo đỉnh mới

Các tham số:

key	Khóa của đỉnh
data	Dữ liệu cần thêm vào đồ thị

template<class DataType> DataType& edaDAG< DataType >::operator[] (int key) `[inline]`

Get the reference to data

Các tham số:

key	The key of vertex to get data

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaDAG.h

edaDAGEdge Class Tham chiếu

Lớp **edaDAGEdge** là hiện thực của cạnh đồ thị DAG.

`#include <edaDAGEdge.h>`

Các hàm thành viên Public

- **edaDAGEdge** (int _key, **edaDAGVertex** *_fromVertex, **edaDAGVertex** *_toVertex)
  Khởi tạo đối tượng.
- **~edaDAGEdge** ()
  Hủy đối tượng.
- int **getKey** () const
  Lấy khóa của đối tượng.
- **edaDAGVertex** * **getSourceVertex** () const
  Lấy đỉnh đầu.
- **edaDAGVertex** * **getDestVertex** () const
  Lấy đỉnh cuối.

Mô tả chi tiết

Lớp **edaDAGEdge** là hiện thực của cạnh đồ thị DAG.

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaDAGEdge.h
- lib/edaDAGEdge.cpp

edaDAGVertex Class Tham chiếu

Lớp **edaDAGEdge** là hiện thực của đỉnh đồ thị DAG.

`#include <edaDAGVertex.h>`

Các hàm thành viên Public

- **edaDAGVertex** (int key)
  Khởi tạo đối tượng.
- **~edaDAGVertex** ()
  Hủy đối tượng.
- int **getKey** () const
  Lấy khóa của đối tượng.
- void **insertInEdge** (**edaDAGEdge** *inEdge)
  Thêm cạnh vào.
- void **insertOutEdge** (**edaDAGEdge** *outEdge)
  Thêm cạnh ra.
- vector< **edaDAGEdge** * > * **getInEdges** ()
  Lấy tất cả các cạnh vào của đỉnh.
- vector< **edaDAGEdge** * > * **getOutEdges** ()
  Lấy tất cả các cạnh ra của đỉnh.
- int **getInDegree** ()
  Lấy tổng số cạnh vào.
- int **getOutDegree** ()
  Lấy tổng số cạnh ra.
- bool **cycleCheck** (int &cycleVertex)
  Kiểm tra sự tồn tại chu trình tại đỉnh.
- void **setCycle** (bool flag)
  Thiết lập cờ đánh dấu sự xuất hiện của chu trình.

Mô tả chi tiết

Lớp **edaDAGEdge** là hiện thực của đỉnh đồ thị DAG.

Thông tin về hàm thành viên

int edaDAGVertex::getOutDegree ()

Lấy tổng số cạnh ra.

Get indegree

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaDAGVertex.h
- lib/edaDAGVertex.cpp

edaException Class Tham chiếu

Lớp **edaException** hiện thực việc xử lý ngoại lệ trong thư viện.
`#include <edaException.h>`

Các hàm thành viên Public

- **edaException** ()
  Khởi tạo đối tượng.
- **edaException** (std::string errorMessage)
- const std::string & **getErrorMessage** ()

Mô tả chi tiết

Lớp **edaException** hiện thực việc xử lý ngoại lệ trong thư viện.

Thông tin về Constructor và Destructor

edaException::edaException (std::string errorMessage) `[inline]`

Khởi tạo đối tượng

Các tham số:

errorMessage	chuỗi lỗi

Thông tin về hàm thành viên

const std::string& edaException::getErrorMessage () `[inline]`

Lấy chuỗi lỗi từ đối tượng

Giá trị trả về:

Chuỗi lỗi

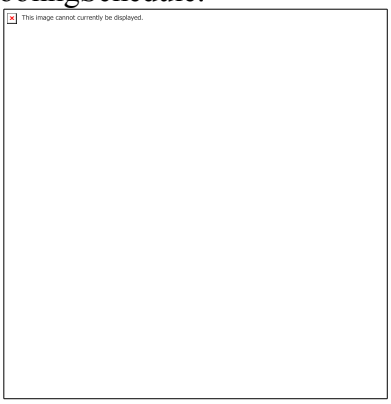
Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaException.h

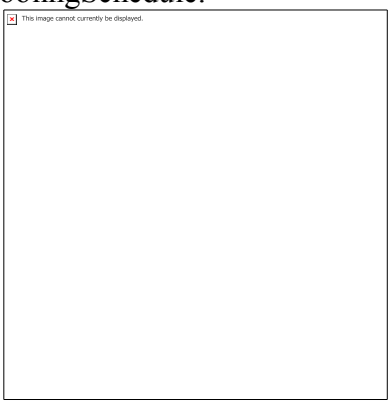
edaExpCoolingSchedule Class Tham chiếu

Lớp **edaExpCoolingSchedule** là hiện thực của chiến lược làm lạnh nhiệt độ theo hàm mũ.
`#include <edaExpCoolingSchedule.h>`

Sơ đồ kế thừa cho edaExpCoolingSchedule:



Sơ đồ liên kết cho edaExpCoolingSchedule:



Các hàm thành viên Public

- **edaExpCoolingSchedule** ()
  Khởi tạo đối tượng.
- **edaExpCoolingSchedule** (double threshold, double ratio)
- virtual **edaCoolingSchedule** * **clone** () const
  Nhân bản đối tượng.
- bool **check** (double &temperature)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EXP_COOLING_SCHEDULE_)
- void **printOn** (std::ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.

Mô tả chi tiết

Lớp **edaExpCoolingSchedule** là hiện thực của chiến lược làm lạnh nhiệt độ theo hàm mũ.

Thông tin về Constructor và Destructor

edaExpCoolingSchedule::edaExpCoolingSchedule (double threshold, double ratio)

Khỏi tạo đối tượng

Các tham số:

threshold	Ngưỡng nhiệt độ
ratio	Tỷ lệ giảm nhiệt độ (0 < ratio < 1)

Thông tin về hàm thành viên

bool edaExpCoolingSchedule::check (double & temperature) `[virtual]`

Kiểm tra nhiệt độ hiện tại với ngưỡng nhiệt độ

Các tham số:

temperature	Nhiệt độ hiện tại

Thực hiện **edaCoolingSchedule** (*tr.71*).

void edaExpCoolingSchedule::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

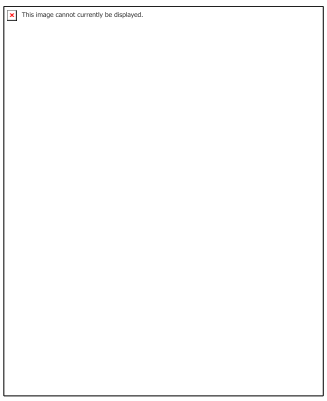
- lib/edaExpCoolingSchedule.h
- lib/edaExpCoolingSchedule.cpp

edaFirstImprSelect Class Tham chiếu

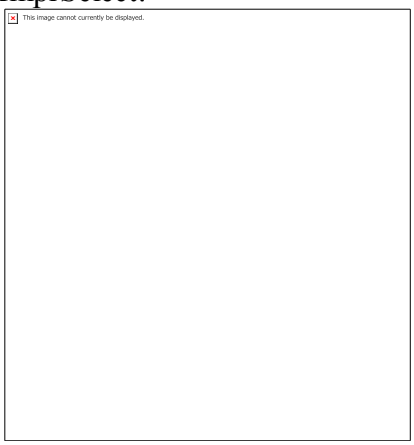
Lớp **edaBestImprSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược First Best.

```
#include <edaFirstImprSelect.h>
```

Sơ đồ kế thừa cho edaFirstImprSelect:



Sơ đồ liên kết cho edaFirstImprSelect:



Các hàm thành viên Public

- **edaFirstImprSelect** (const double epsilon=1e-4)
- **edaMoveSelect** * **clone** () const
  Nhân bản đối tượng.
- void **init** (double fitness)
  Khởi động đối tượng.
- bool **update** (const **edaMove** *move, double fitness)
- bool **save** (**edaMove** *move, double &fitness) const
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAFIRSTIMPRSELECT_)

Additional Inherited Members

Mô tả chi tiết

Lớp **edaBestImprSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược First Best.

Thông tin về hàm thành viên

*bool edaFirstImprSelect::save (edaMove * move, double & fit) const* `[virtual]`

Lấy thông tin bước chuyển ứng với chiến lược tương ứng

Các tham số:

move	Bước chuyển ứng với chiến lược đề ra
fitness	Lượng giá của bước chuyển

Thực hiện **edaMoveSelect** (*tr.112*).

void edaFirstImprSelect::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của

	đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Thực hiện **edaMoveSelect** (*tr.112*).

*bool edaFirstImprSelect::update (const edaMove * move, double fit)* `[virtual]`

Thực hiện việc chọn lựa với bước chuyển theo chiến lược tương ứng

Các tham số:

move	Bước chuyển ứng viên
fitness	Lượng giá của bước chuyển

Thực hiện **edaMoveSelect** (*tr.113*).

Thông tin cho class được biên soạn từ các file sau đây:

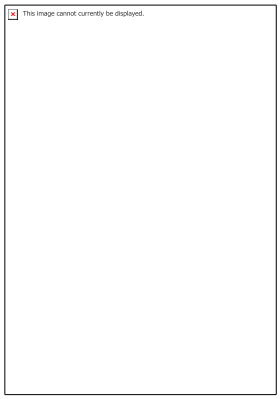
- lib/edaFirstImprSelect.h
- lib/edaFirstImprSelect.cpp

edaFitContinue Class Tham chiếu

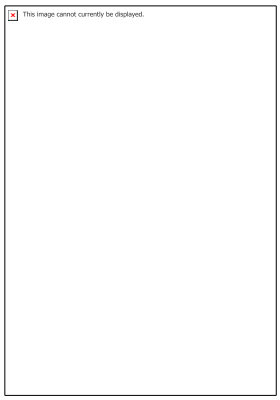
Lớp **edaFitContinue** hiện thực điều kiện dừng theo hàm lượng giá.

```
#include <edaFitContinue.h>
```

Sơ đồ kế thừa cho edaFitContinue:



Sơ đồ liên kết cho edaFitContinue:



Các hàm thành viên Public

- **edaFitContinue** ()
  Khởi tạo đối tượng.
- **edaFitContinue** (double fitness)
  Khởi tạo đối tượng với giá trị fitness.
- **~edaFitContinue** ()
  Hủy đối tượng.
- virtual **edaContinue** * **clone** () const
  Nhân bản đối tượng.
- void **init** ()
  Khởi động đối tượng.
- bool **check** (const **edaSolutionList** &list)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_FIT_CONTINUE_)

Mô tả chi tiết

Lớp **edaFitContinue** hiện thực điều kiện dừng theo hàm lượng giá.

Thông tin về hàm thành viên

bool edaFitContinue::check (const edaSolutionList & list) `[virtual]`

Kiểm tra điều kiện dừng

Các tham số:

list	Danh sách các lời giải cần kiểm tra với điều kiện dừng

Thực hiện **edaContinue** (*tr.70*).

void edaFitContinue::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaContinue** (*tr.70*).

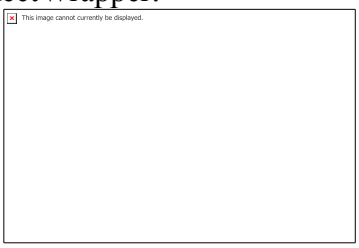
Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaFitContinue.h
- lib/edaFitContinue.cpp

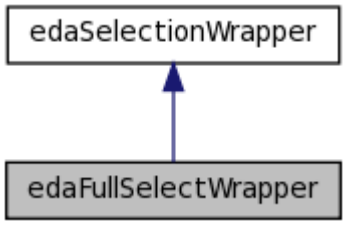
edaFullSelectWrapper Class Tham chiếu

Lớp **edaFullSelectWrapper** thực hiện việc chọn tất cả các lời giải trong danh

sách lời giải và đưa chúng vào nút tìm kiếm.
`#include <edaFullSelectWrapper.h>`
Sơ đồ kế thừa cho edaFullSelectWrapper:



Sơ đồ liên kết cho edaFullSelectWrapper:



Các hàm thành viên Public

- **edaFullSelectWrapper** ()
  Khởi tạo đối tượng.
- **~edaFullSelectWrapper** ()
  Hủy đối tượng.
- **edaFullSelectWrapper** * **clone** () const
- bool **select** (**edaSolutionList** &list) const

Mô tả chi tiết

Lớp **edaFullSelectWrapper** thực hiện việc chọn tất cả các lời giải trong danh sách lời giải và đưa chúng vào nút tìm kiếm.

Thông tin về hàm thành viên

*edaFullSelectWrapper * edaFullSelectWrapper::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSelectionWrapper** (*tr.151*).

bool edaFullSelectWrapper::select (edaSolutionList & list) const `[virtual]`

Chọn lựa các lời giải với chiến lược chọn tương ứng trong tập lời giải

Giá trị trả về:

Nhận giá trị TRUE nếu chọn lựa thành công, ngược lại giá trị đầu ra là FALSE

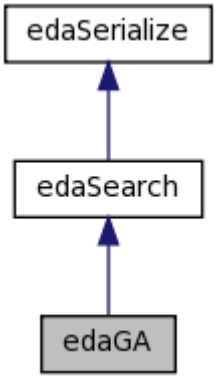
Thực hiện **edaSelectionWrapper** (*tr.151*).

Thông tin cho class được biên soạn từ các file sau đây:

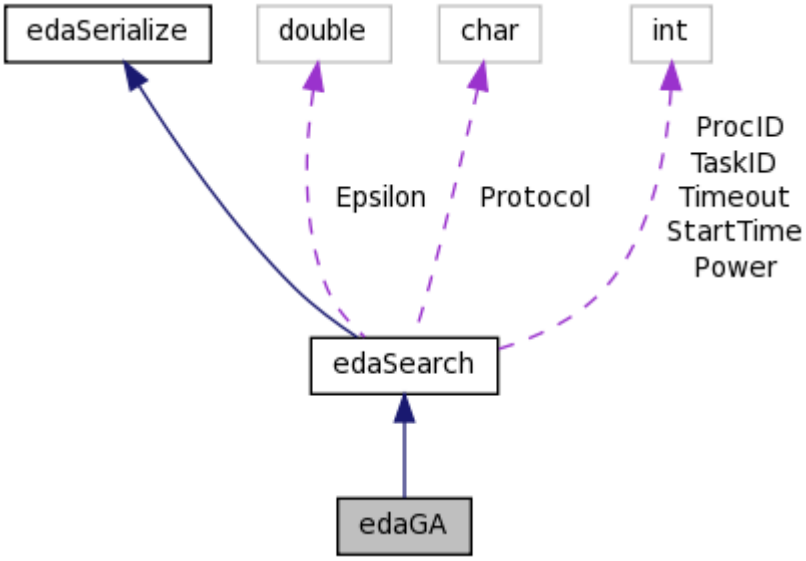
- lib/edaFullSelectWrapper.h
- lib/edaFullSelectWrapper.cpp

edaGA Class Tham chiếu

Lớp **edaGA** thực hiện việc tối ưu theo giải thuật di truyền.
`#include <edaGA.h>`
Sơ đồ kế thừa cho edaGA:



Sơ đồ liên kết cho edaGA:



Các hàm thành viên Public

- **edaGA** ()
  Khởi tạo đối tượng.
- **edaGA** (**edaContinue** *con, **edaRepresentation** *repre, **edaNaturalSelection** *slect, **edaCrossover** *cross, **edaMutation** *mute, unsigned int elite=1, int timeout=0, int power=0)

- **edaGA** (const **edaGA** &ga)
- **edaGA** * **clone** () const
- virtual **~edaGA** ()
  Hủy đối tượng.
- bool **search** (**edaSolutionList** &list)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAGA_)

Additional Inherited Members

Mô tả chi tiết

Lớp **edaGA** thực hiện việc tối ưu theo giải thuật di truyền.

Thông tin về Constructor và Destructor

*edaGA::edaGA (edaContinue * con, edaRepresentation * repre, edaNaturalSelection * slect, edaCrossover * cross, edaMutation * mute, unsigned int elite = 1, int timeout = 0, int power = 0)*

Khởi tạo đối tượng

Các tham số:

con	Điều kiện dừng
repre	Toán tử mã hóa
slect	Toán tử chọn lọc (tự nhiên)
cross	Toán tử lai chéo
mute	Toán tử đột biến
elite	Số phần tử ưu việt
timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

edaGA::edaGA (const edaGA & ga)

Khởi tạo đối tượng

Các tham số:

ga	Đối tượng search cần sao chép

Thông tin về hàm thành viên

*edaGA * edaGA::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSearch** (*tr.148*).

bool edaGA::search (edaSolutionList & list) `[virtual]`

Thực thi tối ưu

Các tham số:

list	Danh sách lời giải cần tối ưu

Giá trị trả về:

TRUE: tối ưu thành công, FALSE: tối ưu thất bại

Thực hiện **edaSearch** (*tr.148*).

void edaGA::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSearch** (*tr.148*).

Thông tin cho class được biên soạn từ các file sau đây:

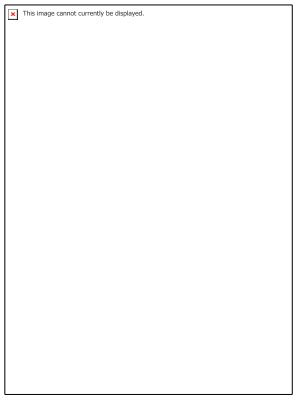
- lib/edaGA.h
- lib/edaGA.cpp

edaGenContinue Class Tham chiếu

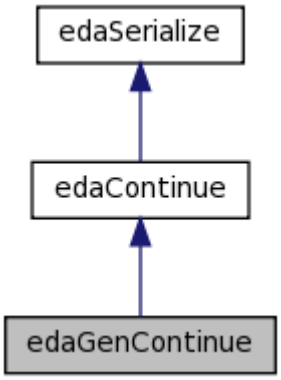
Lớp **edaGenContinue** hiện thực điều kiện dừng theo theo số lần lặp.

`#include <edaGenContinue.h>`

Sơ đồ kế thừa cho edaGenContinue:



Sơ đồ liên kết cho edaGenContinue:



Các hàm thành viên Public

- **edaGenContinue** ()
  Khởi tạo đối tượng.
- **edaGenContinue** (unsigned int maxNumGen)
- virtual **~edaGenContinue** ()
  Hủy đối tượng.
- virtual **edaContinue** * **clone** () const
  Nhân bản đối tượng.
- void **init** ()
  Khởi động đối tượng.
- bool **check** (const **edaSolutionList** &list)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_GEN_CONTINUE_)

Mô tả chi tiết

Lớp **edaGenContinue** hiện thực điều kiện dừng theo theo số lần lặp.

Thông tin về Constructor và Destructor

edaGenContinue::edaGenContinue (unsigned int maxNumGen)

Khởi tạo đối tượng

Các tham số:

maxNumGen	Số lần (thế hệ) lặp tối đa

Thông tin về hàm thành viên

bool edaGenContinue::check (const edaSolutionList & list) `[virtual]`

Kiểm tra điều kiện dừng

Các tham số:

list	Danh sách các lời giải cần kiểm tra với điều kiện dừng

Thực hiện **edaContinue** (*tr.70*).

void edaGenContinue::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaContinue** (*tr.70*).

Thông tin cho class được biên soạn từ các file sau đây:

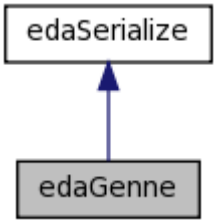
- lib/edaGenContinue.h
- lib/edaGenContinue.cpp

edaGenne Class Tham chiếu

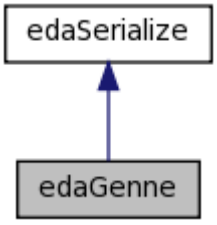
Lớp **edaGenne** là hiện thực đối tượng gen trong giải thuật tiến hóa.

`#include <edaGenne.h>`

Sơ đồ kế thừa cho edaGenne:



Sơ đồ liên kết cho edaGenne:



Các hàm thành viên Public

- **edaGenne** ()
  Khởi tạo đối tượng.
- **edaGenne** (double value)
- **edaGenne** (const **edaGenne** &genne)
- virtual **~edaGenne** ()
  Hủy đối tượng.
- virtual **edaGenne** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **swap** (**edaGenne** *genne)
- virtual void **exchange** (double rate)

- virtual void **printOn** (ostream &os) const =0
  In thông tin (dạng chuỗi) của đối tượng lên ostream.
- virtual **edaGenne** & **operator=** (const **edaGenne** &genne)=0
- virtual bool **operator==** (const **edaGenne** &genne) const =0
- bool **operator!=** (const **edaGenne** &genne) const
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)=0

Mô tả chi tiết

Lớp **edaGenne** là hiện thực đối tượng gen trong giải thuật tiến hóa.

Thông tin về Constructor và Destructor

edaGenne::edaGenne (double value) `[inline]`

Khởi tạo đối tượng

Các tham số:

value	Giá trị dược gán vào gen

edaGenne::edaGenne (const edaGenne & genne) `[inline]`

Khởi tạo đối tượng

Các tham số:

genne	Đối tượng được sao chép

Thông tin về hàm thành viên

virtual void edaGenne::exchange (double rate) `[inline], [virtual]`

Thay đổi giá trị của gen

Các tham số:

rate	Giá trị hoán đổi

virtual void edaGenne::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

*virtual void edaGenne::swap (edaGenne * genne)* `[inline], [virtual]`

Hoán đổi giá trị giữa 2 gen

Các tham số:

genne	Gen dùng hoán đổi giá trị

Thông tin cho class được biên soạn từ các file sau đây:

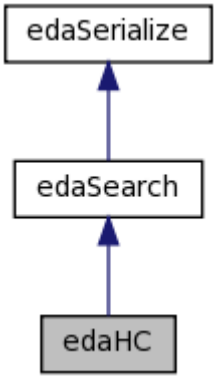
- lib/edaGenne.h
- lib/edaGenne.cpp

edaHC Class Tham chiếu

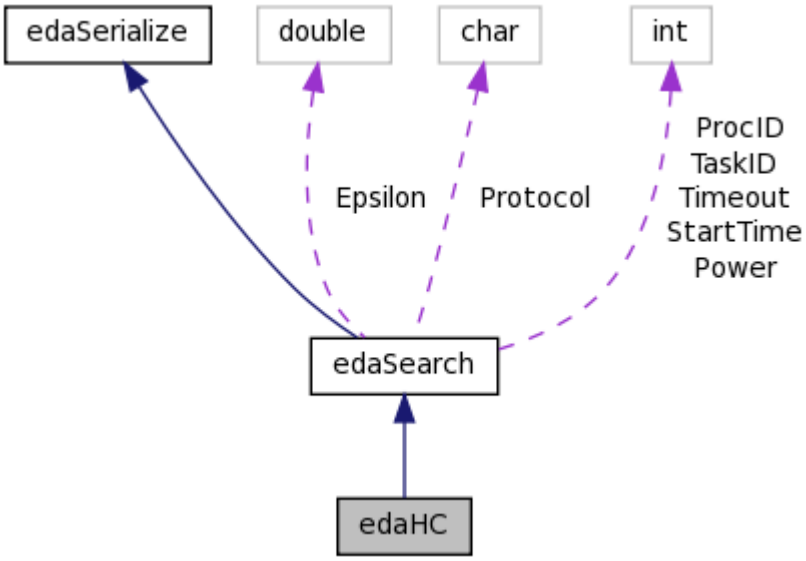
Lớp **edaHC** thực hiện việc tối ưu theo giải thuật leo đồi.

`#include <edaHC.h>`

Sơ đồ kế thừa cho edaHC:



Sơ đồ liên kết cho edaHC:



Các hàm thành viên Public

- **edaHC** ()
  Khởi tạo đối tượng.
- **edaHC** (int power)
- **edaHC** (**edaMove** *move, **edaMoveGen** *moveNext, **edaMoveSelect** *moveSelect, **edaContinue** *continueCrit, int timeout=0, int power=0)

- **edaHC** (**edaMove** *move, **edaHCMoveExpl** *moveExpl, **edaContinue** *continueCrit, int timeout=0, int power=0)
- **edaHC** (const **edaHC** &hc)
- **edaHC** * **clone** () const
- **~edaHC** ()
  Hủy đối tượng.
- bool **search** (**edaSolutionList** &list)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAHC_)

Additional Inherited Members

Mô tả chi tiết

Lớp **edaHC** thực hiện việc tối ưu theo giải thuật leo đồi.

Thông tin về Constructor và Destructor

edaHC::edaHC (int power)

Khởi tạo đối tượng

Các tham số:

power	Bậc tìm kiếm

*edaHC::edaHC (edaMove * move, edaMoveGen * moveNext, edaMoveSelect * moveSelect, edaContinue * continueCrit, int timeout = 0, int power = 0)*

Khởi tạo đối tượng

Các tham số:

move	Chiến lược bước chuyển trong không gian tìm kiếm
moveNext	Phương pháp tạo bước chuyển trong không gian tìm kiếm
moveSelect	Chiến lược chọn bước chuyển trong không gian tìm kiếm
continueCrit	Điều kiện dừng
timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

*edaHC::edaHC (edaMove * move, edaHCMoveExpl * moveExpl, edaContinue * continueCrit, int timeout = 0, int power = 0)*

Khởi tạo đối tượng

Các tham số:

move	Chiến lược bước chuyển trong không gian tìm kiếm
moveExpl	Chiến lược khai phá không gian tìm kiếm
continueCrit	Điều kiện dừng
timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

edaHC::edaHC (const edaHC & hc)

Khởi tạo đối tượng

Các tham số:

hc	Đối tượng search cần sao chép

Thông tin về hàm thành viên

*edaHC * edaHC::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSearch** (*tr.148*).

bool edaHC::search (edaSolutionList & list) `[virtual]`

Thực thi tối ưu

Các tham số:

list	Danh sách lời giải cần tối ưu

Giá trị trả về:

TRUE: tối ưu thành công, FALSE: tối ưu thất bại

Thực hiện **edaSearch** (*tr.148*).

void edaHC::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSearch** (*tr.148*).

Thông tin cho class được biên soạn từ các file sau đây:

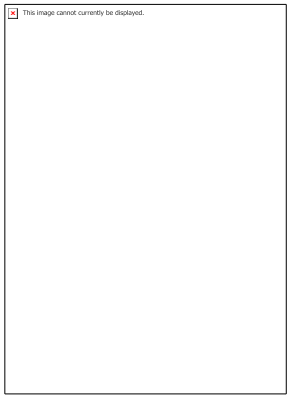
- lib/edaHC.h
- lib/edaHC.cpp

edaHCMoveExpl Class Tham chiếu

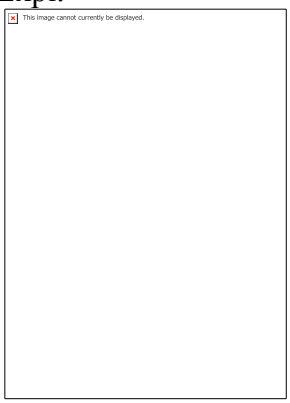
Lớp **edaHCMoveExpl** hiện thực chiến lược khai phá không gian ứng với giải thuật HC.

```
#include <edaHCMoveExpl.h>
```

Sơ đồ kế thừa cho edaHCMoveExpl:



Sơ đồ liên kết cho edaHCMoveExpl:



Các hàm thành viên Public

- **edaHCMoveExpl** ()
  Khởi tạo đối tượng.
- **edaHCMoveExpl** (const **edaMoveGen** *moveNext, const **edaMoveSelect** *moveSelect)
- **edaHCMoveExpl** (const **edaHCMoveExpl** &_moveExpl)
  Khởi tạo đối tượng.
- **~edaHCMoveExpl** ()
  Hũy đối tượng.
- **edaHCMoveExpl** * **clone** () const
  Nhân bản đối tượng.
- void **explore** (const **edaMove** *move, **edaSolution** &oldSolution, **edaSolution** &newSolution)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAHCMOVEEXPL_)

Mô tả chi tiết

Lớp **edaHCMoveExpl** hiện thực chiến lược khai phá không gian ứng với giải thuật HC.

Thông tin về Constructor và Destructor

*edaHCMoveExpl::edaHCMoveExpl (const edaMoveGen * moveNext, const edaMoveSelect * moveSelect)*

Khởi tạo đối tượng

Các tham số:

moveNext	Chiến lược tạo bước chuyển trong không gian tìm kiếm
moveSelect	Chiến lược chọn bước chuyển trong không gian tìm kiếm

---

Thông tin về hàm thành viên

*void edaHCMoveExpl::explore (const edaMove * move, edaSolution & oldSolution, edaSolution & newSolution)* `[virtual]`

Run the explorer

Các tham số:

oldSolution	The current solution
newSolution	The new solution

Thực hiện **edaMoveExpl** (*tr.110*).

void edaHCMoveExpl::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

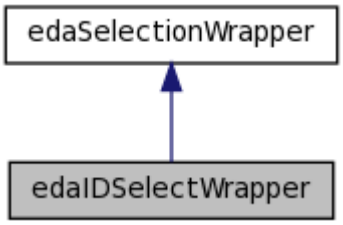
- lib/edaHCMoveExpl.h
- lib/edaHCMoveExpl.cpp

edaIDSelectWrapper Class Tham chiếu

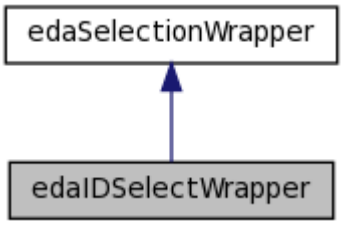
Lớp **edaIDSelectWrapper** thực hiện việc chọn lời giải theo thứ tự của chúng trong sanh sách lời giải.

```
#include <edaIDSelectWrapper.h>
```

Sơ đồ kế thừa cho edaIDSelectWrapper:



Sơ đồ liên kết cho edaIDSelectWrapper:



Các hàm thành viên Public

- **edaIDSelectWrapper** (unsigned int id=0)
- **edaIDSelectWrapper** * **clone** () const
- bool **select** (**edaSolutionList** &list) const

Mô tả chi tiết

Lớp **edaIDSelectWrapper** thực hiện việc chọn lời giải theo thứ tự của chúng trong sanh sách lời giải.

Thông tin về Constructor và Destructor

edaIDSelectWrapper::edaIDSelectWrapper (unsigned int id = `0`)

Khởi tạo đối tượng

Các tham số:

id	Thứ tự của lời giải dược chọn trong danh sách lời giải (mặc định id = 0)

Thông tin về hàm thành viên

*edaIDSelectWrapper * edaIDSelectWrapper::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSelectionWrapper** (*tr.151*).

bool edaIDSelectWrapper::select (edaSolutionList & list) const `[virtual]`

Chọn lựa các lời giải với chiến lược chọn tương ứng trong tập lời giải

Giá trị trả về:

Nhận giá trị TRUE nếu chọn lựa thành công, ngược lại giá trị đầu ra là FALSE

Thực hiện **edaSelectionWrapper** (*tr.151*).

Thông tin cho class được biên soạn từ các file sau đây:

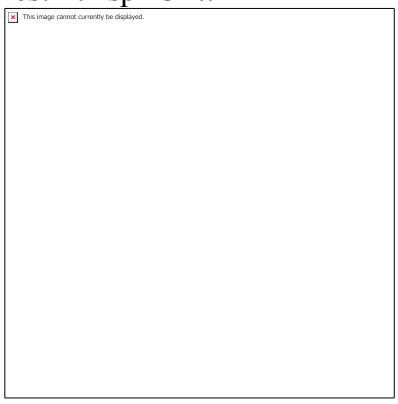
- lib/edaIDSelectWrapper.h
- lib/edaIDSelectWrapper.cpp

edaImprBestFitAspirCrit Class Tham chiếu

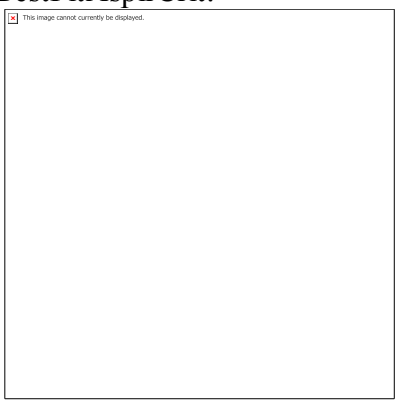
Lớp **edaImprBestFitAspirCrit** thực hiện việc lấy bước chuyển ứng viên có hàm lượng giá tốt nhất.

`#include <edaImprBestFitAspirCrit.h>`

Sơ đồ kế thừa cho edaImprBestFitAspirCrit:



Sơ đồ liên kết cho edaImprBestFitAspirCrit:



Các hàm thành viên Public

- **edaImprBestFitAspirCrit** ()
  Khởi tạo đối tượng.
- **edaImprBestFitAspirCrit** (const **edaImprBestFitAspirCrit** &ac)
- **~edaImprBestFitAspirCrit** ()
  Hủy đối tượng.
- **edaAspirCrit** * **clone** () const
- void **init** ()
  Khởi động đối tượng.
- bool **check** (const **edaMove** *_move, double fitness)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAIMPRBESTFITASPIRCRIT_)

Mô tả chi tiết

Lớp **edaImprBestFitAspirCrit** thực hiện việc lấy bước chuyển ứng viên có hàm lượng giá tốt nhất.

Thông tin về Constructor và Destructor

edaImprBestFitAspirCrit::edaImprBestFitAspirCrit (const edaImprBestFitAspirCrit & ac)

Khởi tạo đối tượng

Các tham số:

ac	Đối tượng cần sao chép

Thông tin về hàm thành viên

*bool edaImprBestFitAspirCrit::check (const edaMove * move, double fitness)* `[virtual]`

Kiểm tra bước chuyển move trong giải thuật Tabu có được chấp nhận không

Các tham số:

move	Bước trong cần kiểm tra
fitness	Lượng giá của bước chuyển

Thực hiện **edaAspirCrit** (*tr.60*).

*edaAspirCrit * edaImprBestFitAspirCrit::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaAspirCrit** (*tr.61*).

void edaImprBestFitAspirCrit::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaImprBestFitAspirCrit.h
- lib/edaImprBestFitAspirCrit.cpp

edaINIReader Class Tham chiếu

Lớp **edaINIReader** thực hiện việc xử lý file ASCII theo chuẩn INI.
`#include <edaINIReader.h>`

Các hàm thành viên Public

- **edaINIReader** ()
  Khởi tạo đối tượng.
- virtual **~edaINIReader** ()
  Hủy đối tượng.
- **edaINIReader** (std::string filename)
- std::string **GetString** (std::string section, std::string name, std::string default_value)
- long **GetInteger** (std::string section, std::string name, long default_value)
- long * **GetInteger** (std::string section, std::string name, double default_value, unsigned int number)
- double **GetDouble** (std::string section, std::string name, double default_value)
- double * **GetDouble** (std::string section, std::string name, double default_value, unsigned int number)
- bool **GetBoolean** (std::string section, std::string name, bool default_value)

Mô tả chi tiết

Lớp **edaINIReader** thực hiện việc xử lý file ASCII theo chuẩn INI.

Thông tin về Constructor và Destructor

edaINIReader::edaINIReader (std::string filename)

Đọc file INI

Các tham số:

filename	Tên file INI cần đọc

Thông tin về hàm thành viên

bool edaINIReader::GetBoolean (std::string section, std::string name, bool default_value)

Lấy thuộc tính dạng luận lý

Các tham số:

section	Mục cần đọc
name	Tên thuộc tính
default_value	Giá trị mặc định khi thuộc tính không có giá trị

double edaINIReader::GetDouble (std::string section, std::string name, double default_value)

Lấy thuộc tính dạng số thực

Các tham số:

section	Mục cần đọc
name	Tên thuộc tính
default_value	Giá trị mặc định khi thuộc tính không có giá trị

*double * edaINIReader::GetDouble (std::string section, std::string name, double default_value, unsigned int number)*

Lấy các thuộc tính dạng số thực vào mãng

Các tham số:

section	Mục cần đọc
name	Tên thuộc tính
default_value	Giá trị mặc định khi thuộc tính không có giá trị
number	Số thuộc tính

long edaINIReader::GetInteger (std::string section, std::string name, long default_value)

Lấy thuộc tính dạng số nguyên

Các tham số:

section	Mục cần đọc
name	Tên thuộc tính
default_value	Giá trị mặc định khi thuộc tính không có giá trị

*long * edaINIReader::GetInteger (std::string section, std::string name, double default_value, unsigned int number)*

Lấy các thuộc tính dạng số nguyên vào mãng

Các tham số:

section	Mục cần đọc
name	Tên thuộc tính
default_value	Giá trị mặc định khi thuộc tính không có giá trị
number	Số thuộc tính

string edaINIReader::GetString (std::string section, std::string name, std::string default_value)

Lấy thuộc tính dạng chuỗi

Các tham số:

section	Mục cần đọc
name	Tên thuộc tính
default_value	Giá trị mặc định khi thuộc tính không có giá trị

Thông tin cho class được biên soạn từ các file sau đây:

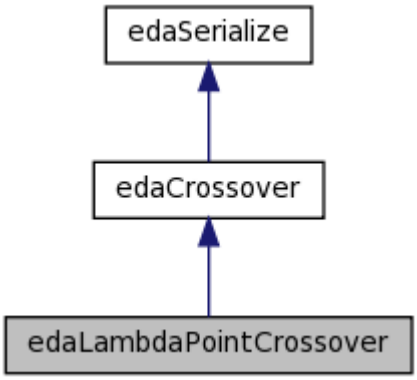
- lib/edaINIReader.h
- lib/edaINIReader.cpp

edaLambdaPointCrossover Class Tham chiếu

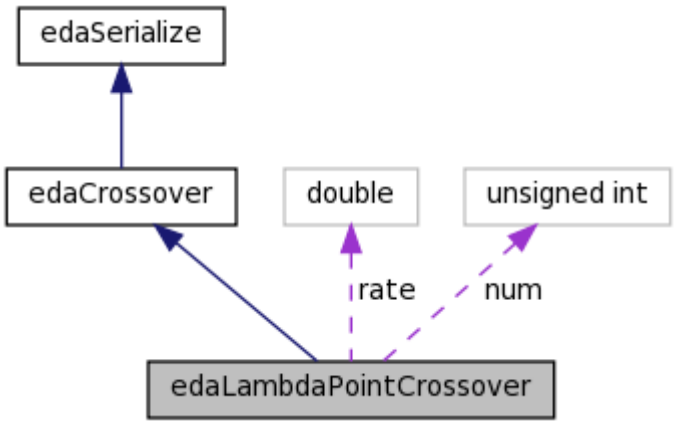
Lớp **edaLambdaPointCrossover** hiện thực chiến lược lai chéo giữa các nhiệm sắc thể với số diểm lai chéo là tùy chọn.

`#include <edaLambdaPointCrossover.h>`

Sơ đồ kế thừa cho edaLambdaPointCrossover:



Sơ đồ liên kết cho edaLambdaPointCrossover:



Các hàm thành viên Public

- **edaLambdaPointCrossover** (double rate=1.0, unsigned int num=1)
- **edaLambdaPointCrossover** (const **edaLambdaPointCrossover** &cross)
  Khởi tạo đối tượng với đối tượng cần sao chép.
- virtual **~edaLambdaPointCrossover** ()
  Hủy đối tượng.
- **edaLambdaPointCrossover** * **clone** () const
  Nhân bản đối tượng.
- void **setRate** (double value)
- void **setNumPoint** (unsigned int num)
- void **update** (**edaPopulation** &pop)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDALAMBDAPOINTCROSSOVER_)

các thuộc tính Protected

- double **rate**
- unsigned int **num**

Mô tả chi tiết

Lớp **edaLambdaPointCrossover** hiện thực chiến lược lai chéo giữa các nhiệm sắc thể với số điểm lai chéo là tùy chọn.

Thông tin về Constructor và Destructor

edaLambdaPointCrossover::edaLambdaPointCrossover (double rate = `1.0`, unsigned int num = `1`)

Khởi tạo đối tượng

Các tham số:

rate	Tỷ lệ lai chéo
num	Số điểm lai chéo (num < nữa số lượng gen trong nhiễm sắc thể)

Thông tin về hàm thành viên

void edaLambdaPointCrossover::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Thực hiện **edaCrossover** (*tr.73*).

void edaLambdaPointCrossover::setNumPoint (unsigned int num)

Thiết lập số điểm lai chéo

Các tham số:

value	Số điểm lai chéo

void edaLambdaPointCrossover::setRate (double value)

Thiết lập tỷ lệ lai chéo

Các tham số:

value	Tỷ lệ lai chéo

void edaLambdaPointCrossover::update (edaPopulation & pop) `[virtual]`

Thực hiện việc cập nhật tập dân cư với chiến lược tương ứng với từng hiện thực cụ thể

Các tham số:

pop	Tập dân cư cần lai chéo

Thực hiện **edaCrossover** (*tr.73*).

Thông tin cho class được biên soạn từ các file sau đây:

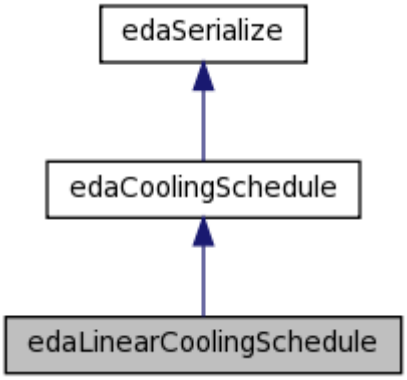
- lib/edaLambdaPointCrossover.h
- lib/edaLambdaPointCrossover.cpp

edaLinearCoolingSchedule Class Tham chiếu

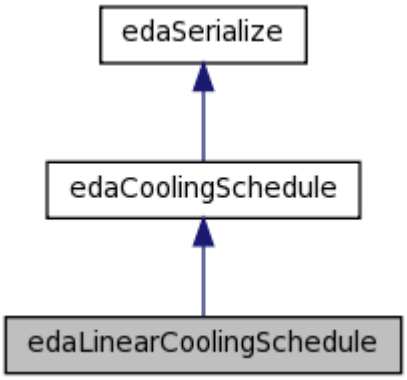
Lớp **edaExpCoolingSchedule** là hiện thực của chiến lược làm lạnh nhiệt độ theo hàm tuyến tính.

```
#include <edaLinearCoolingSchedule.h>
```

Sơ đồ kế thừa cho edaLinearCoolingSchedule:



Sơ đồ liên kết cho edaLinearCoolingSchedule:



Các hàm thành viên Public

- **edaLinearCoolingSchedule** ()
  Khởi tạo đối tượng.
- **edaLinearCoolingSchedule** (double _threshold, double _quantity)
- **edaCoolingSchedule** * **clone** () const
  Nhân bản đối tượng.

- bool **check** (double &temperature)
- void **printOn** (std::ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_LINEAR_COOLING_SCHEDULE_)

Mô tả chi tiết

Lớp **edaExpCoolingSchedule** là hiện thực của chiến lược làm lạnh nhiệt độ theo hàm tuyến tính.

Thông tin về Constructor và Destructor

edaLinearCoolingSchedule::edaLinearCoolingSchedule () `[inline]`

Khỏi tạo đối tượng.

Default constructor

edaLinearCoolingSchedule::edaLinearCoolingSchedule (double _threshold, double _quantity)

Khỏi tạo đối tượng

Các tham số:

threshold	Ngưỡng nhiệt độ
ratio	Lượng giảm nhiệt độ (0 < ratio < 1)

Thông tin về hàm thành viên

bool edaLinearCoolingSchedule::check (double & temperature) `[virtual]`

Kiểm tra nhiệt độ hiện tại với ngưỡng nhiệt độ

Các tham số:

temperature	Nhiệt độ hiện tại

Thực hiện **edaCoolingSchedule** (*tr.71*).

void edaLinearCoolingSchedule::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

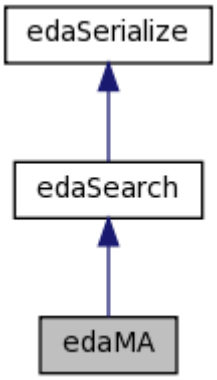
- lib/edaLinearCoolingSchedule.h
- lib/edaLinearCoolingSchedule.cpp

edaMA Class Tham chiếu

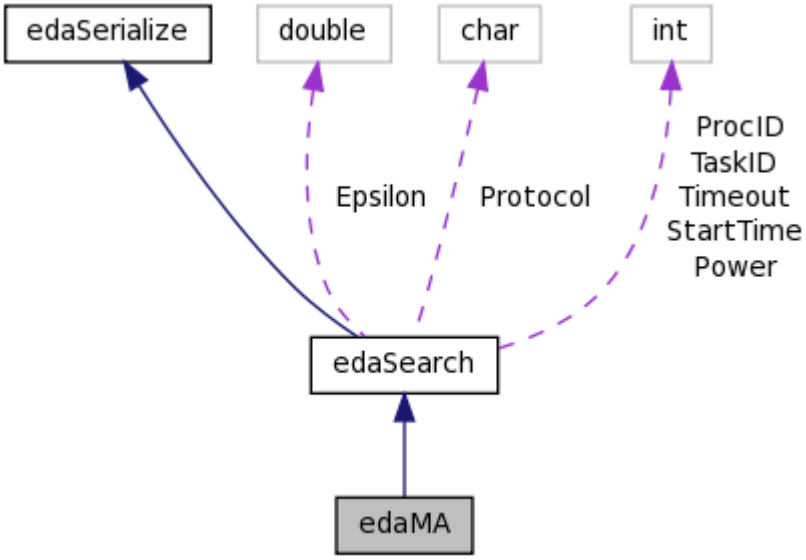
Lớp **edaMA** thực hiện việc tối ưu theo giải thuật Memetic.

```
#include <edaMA.h>
```

Sơ đồ kế thừa cho edaMA:



Sơ đồ liên kết cho edaMA:



Các hàm thành viên Public

- **edaMA** ()
  Khởi tạo đối tượng.
- **edaMA** (**edaContinue** *con, **edaRepresentation** *repre, **edaNaturalSelection** *slect, **edaCrossover** *cross, **edaMutation** *mute, **edaAdaption** *adapt, unsigned int elite=1, int timeout=0, int power=0)
- **edaMA** (const **edaMA** &ma)
- **edaMA** * **clone** () const

- virtual **~edaMA** ()
  Hủy đối tượng.
- bool **search** (**edaSolutionList** &list)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAMA_)

Additional Inherited Members

Mô tả chi tiết

Lớp **edaMA** thực hiện việc tối ưu theo giải thuật Memetic.

Thông tin về Constructor và Destructor

*edaMA::edaMA (edaContinue * con, edaRepresentation * repre, edaNaturalSelection * slect, edaCrossover * cross, edaMutation * mute, edaAdaption * adapt, unsigned int elite = `1`, int timeout = `0`, int power = `0`)*

Khởi tạo đối tượng

Các tham số:

con	Điều kiện dừng
repre	Toán tử mã hóa
slect	Toán tử chọn lọc (tự nhiên)
cross	Toán tử lai chéo
mute	Toán tử đột biến
adapt	Toán tử cải tiến
elite	Số phần tử ưu việt
timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

edaMA::edaMA (const edaMA & ma)

Khởi tạo đối tượng

Các tham số:

ma	Đối tượng search cần sao chép

Thông tin về hàm thành viên

*edaMA * edaMA::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSearch** (*tr.148*).

bool edaMA::search (edaSolutionList & list) `[virtual]`

Thực thi tối ưu

Các tham số:

list	Danh sách lời giải cần tối ưu

Giá trị trả về:

TRUE: tối ưu thành công, FALSE: tối ưu thất bại

Thực hiện **edaSearch** (*tr.148*).

void edaMA::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSearch** (*tr.148*).

Thông tin cho class được biên soạn từ các file sau đây:

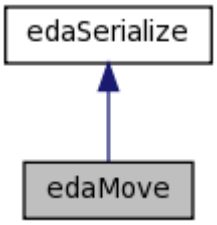
- lib/edaMA.h
- lib/edaMA.cpp

edaMove Class Tham chiếu

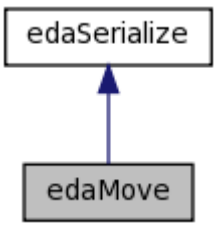
Lớp trừu tượng **edaMove** mô tả các thông tin về chiến lược duy chuyển trong không gian tìm kiếm của các giải thuật.

```
#include <edaMove.h>
```

Sơ đồ kế thừa cho edaMove:



Sơ đồ liên kết cho edaMove:



Các hàm thành viên Public

- **edaMove** ()
  Khởi tạo đối tượng.

- **edaMove** (const **edaMove** &move)
  Khởi tạo đối tượng với đối tượng cần sao chép.
- virtual **~edaMove** ()
  Hủy đối tượng.
- virtual **edaMove** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **init** (const **edaSolution** &sol)=0
  Khởi động đối tượng.
- 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

Mô tả chi tiết

Lớp trừu tượng **edaMove** mô tả các thông tin về chiến lược duy chuyển trong không gian tìm kiếm của các giải thuật.

Thông tin về hàm thành viên

virtual double edaMove::incrEval (const edaSolution & sol) const `[pure virtual]`

Đánh giá sự thay đổi của hàm lượng giá so với lời giải cũ

Các tham số:

sol	Lời giải chưa được cập nhật

virtual void edaMove::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

virtual void edaMove::update (edaSolution & sol) const `[pure virtual]`

Cập nhật lời giải với bước chuyển đang có của đối tượng

Các tham số:

sol	Lời giải hiện tại

Thông tin cho class được biên soạn từ các file sau đây:

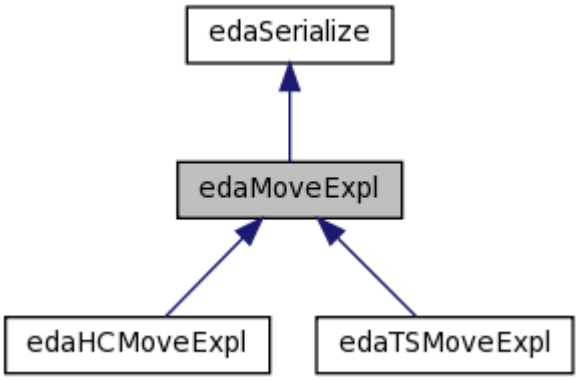
- lib/edaMove.h
- lib/edaMove.cpp

edaMoveExpl Class Tham chiếu

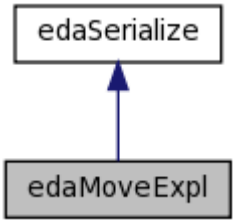
Lớp trừu tượng **edaMoveExpl** hiện thực chiến lược khai phá không gian ứng với từng giải thuật.

`#include <edaMoveExpl.h>`

Sơ đồ kế thừa cho edaMoveExpl:



Sơ đồ liên kết cho edaMoveExpl:



Các hàm thành viên Public

- virtual **~edaMoveExpl** ()
  Hũy đối tượng.
- virtual **edaMoveExpl** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **explore** (const **edaMove** *move, **edaSolution** &oldSolution, **edaSolution** &newSolution)=0

Mô tả chi tiết

Lớp trừu tượng **edaMoveExpl** hiện thực chiến lược khai phá không gian ứng với từng giải thuật.

Thông tin về hàm thành viên

*virtual void edaMoveExpl::explore (const edaMove * move, edaSolution & oldSolution, edaSolution & newSolution)* `[pure virtual]`

Khai phá không gian tìm kiếm

Các tham số:

move	Chiến lược di chuyển trong không gian
oldSolution	Lời giải trước khi áp dụng
newSolution	Lời giải sau khi áp dụng

Được thực hiện trong **edaTSMoveExpl** (*tr.177*), và **edaHCMoveExpl** (*tr.95*).

Thông tin cho class được biên soạn từ các file sau đây:

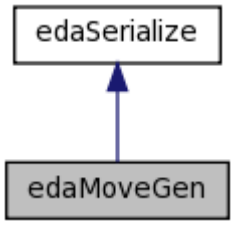
- lib/edaMoveExpl.h

edaMoveGen Class Tham chiếu

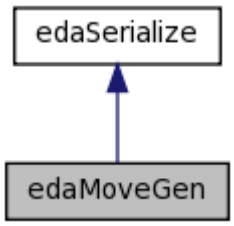
Lớp trừu tượng **edaMoveGen** mô tả các thông tin về thức tạo các bước chuyển trong không gian tìm kiếm của các giải thuật.

`#include <edaMoveGen.h>`

Sơ đồ kế thừa cho edaMoveGen:



Sơ đồ liên kết cho edaMoveGen:



Các hàm thành viên Public

- **edaMoveGen** ()
  Khởi tạo đối tượng.
- **edaMoveGen** (const **edaMoveGen** &m)
  Khởi tạo đối tượng với đối tượng cần sao chép.
- virtual **~edaMoveGen** ()
  Hủy đối tượng.

- virtual **edaMoveGen** * **clone** () const =0
  Nhân bản đối tượng.
- virtual bool **generate** (**edaMove** *move, const **edaSolution** &sol)=0
- **setClassID** (_SYSCLASSID_+_CLSID_EDAMOVEGEN_)

Mô tả chi tiết

Lớp trừu tượng **edaMoveGen** mô tả các thông tin về thức tạo các bước chuyển trong không gian tìm kiếm của các giải thuật.

Thông tin về hàm thành viên

*virtual bool edaMoveGen::generate (edaMove * move, const edaSolution & sol)* `[pure virtual]`

Cập nhật bước chuyển mới cho lời giải

Các tham số:

move	Bước chuyển cần được cập nhật
sol	Lời giải cần xác lập bước chuyển

Thông tin cho class được biên soạn từ các file sau đây:

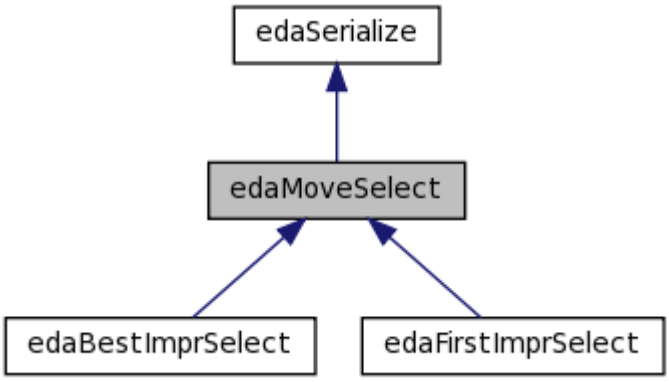
- lib/edaMoveGen.h

edaMoveSelect Class Tham chiếu

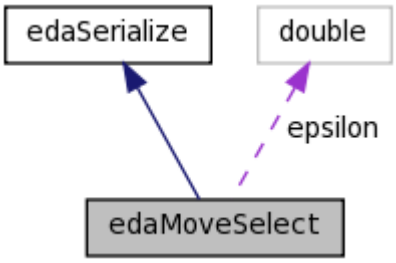
Lớp trừu tượng **edaMoveSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược tương ứng.

```
#include <edaMoveSelect.h>
```

Sơ đồ kế thừa cho edaMoveSelect:



Sơ đồ liên kết cho edaMoveSelect:



Các hàm thành viên Public

- **edaMoveSelect** (const double epsilon)
  Khởi tạo đối tượng.
- virtual **~edaMoveSelect** ()
  Hủy đối tượng.
- virtual **edaMoveSelect** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **init** (double fitness)=0
  Khởi động đối tượng.
- virtual bool **update** (const **edaMove** *move, double fit)=0
- virtual bool **save** (**edaMove** *move, double &fit) const =0
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)=0
- **setClassID** (_SYSCLASSID_+_CLSID_EDAMOVESELECT_)

các thuộc tính Protected

- double **epsilon**

Mô tả chi tiết

Lớp trừu tượng **edaMoveSelect** thực hiện việc chọn lựa các bước chuyển theo chiến lược tương ứng.

Thông tin về hàm thành viên

*virtual bool edaMoveSelect::save (edaMove * move, double & fit) const* `[pure virtual]`

Lấy thông tin bước chuyển ứng với chiến lược tương ứng

Các tham số:

move	Bước chuyển ứng với chiến lược đề ra
fitness	Lượng giá của bước chuyển

Được thực hiện trong **edaBestImprSelect** (*tr.62*), và **edaFirstImprSelect** (*tr.81*).

virtual void edaMoveSelect::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của

	đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Được thực hiện trong **edaBestImprSelect** (*tr.62*), và **edaFirstImprSelect** (*tr.81*).

*virtual bool edaMoveSelect::update (const edaMove * move, double fit)* `[pure virtual]`

Thực hiện việc chọn lựa với bước chuyển theo chiến lược tương ứng

Các tham số:

move	Bước chuyển ứng viên
fitness	Lượng giá của bước chuyển

Được thực hiện trong **edaBestImprSelect** (*tr.62*), và **edaFirstImprSelect** (*tr.82*).

Thông tin cho class được biên soạn từ các file sau đây:

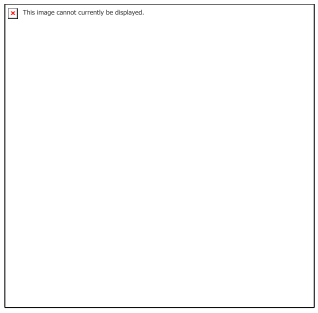
- lib/edaMoveSelect.h

edaMpiProcStatus Struct Tham chiếu

Cấu trúc **edaMpiProcStatus** chứa thông tin tra trạng thái của các node tìm kiếm.

`#include <edaMpiWrapperControl.h>`

Sơ đồ liên kết cho edaMpiProcStatus:



các trường dữ liệu

- int **stat_**
- int **taskID_**

Mô tả chi tiết

Cấu trúc **edaMpiProcStatus** chứa thông tin tra trạng thái của các node tìm kiếm.

Thông tin cho struct được biên soạn từ các file sau đây:

- lib/edaMpiWrapperControl.h

edaMpiWorker Class Tham chiếu

Lớp **edaMpiWorker** thực thi việc phân chia tác vụ giữa giữa máy chủ và các máy trạm.

`#include <edaMpiWorker.h>`

Các hàm thành viên Public

- **edaMpiWorker** (int master=0)
- **~edaMpiWorker** ()
  Hủy đối tượng.
- void **serve** ()
  Thực thi tác vụ do máy chủ phân phối.

Mô tả chi tiết

Lớp **edaMpiWorker** thực thi việc phân chia tác vụ giữa giữa máy chủ và các máy trạm.

Thông tin về Constructor và Destructor

edaMpiWorker::edaMpiWorker (int master = 0)

Khởi tạo đối tượng

Các tham số:

master	ID của node master

Thông tin cho class được biên soạn từ các file sau đây:

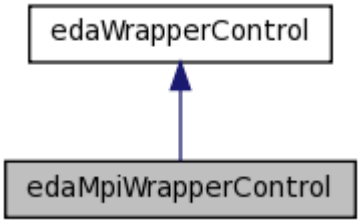
- lib/edaMpiWorker.h
- lib/edaMpiWorker.cpp

edaMpiWrapperControl Class Tham chiếu

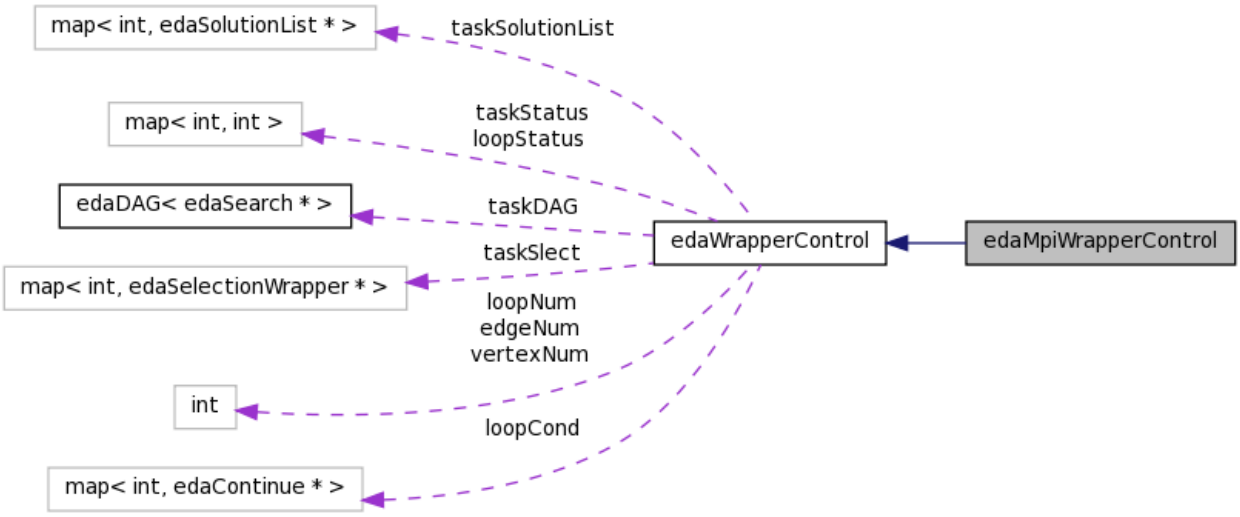
Lớp **edaMpiWrapperControl** điều khiển quá trình tối ưu bằng cơ chế MPI.

`#include <edaMpiWrapperControl.h>`

Sơ đồ kế thừa cho edaMpiWrapperControl:



Sơ đồ liên kết cho edaMpiWrapperControl:



Các hàm thành viên Public

- **edaMpiWrapperControl** ()
  Khởi tạo đối tượng.
- **~edaMpiWrapperControl** ()
  Hủy dối tượng.
- bool **search** (**edaSolutionList** &list)
- int **polling** (int &nodeID)

Additional Inherited Members

Mô tả chi tiết

Lớp **edaMpiWrapperControl** điều khiển quá trình tối ưu bằng cơ chế MPI.

Thông tin về hàm thành viên

int edaMpiWrapperControl::polling (int & nodeID)

Thăm dò các kết quả trả về của các máy trạm

Các tham số:

nodeID	ID của node cần thăm dò

bool edaMpiWrapperControl::search (edaSolutionList & list) `[virtual]`

Thực thi việc tối ưu trên trình diều kiển

Các tham số:

list	Danh sách các lời giải cần tối ưu

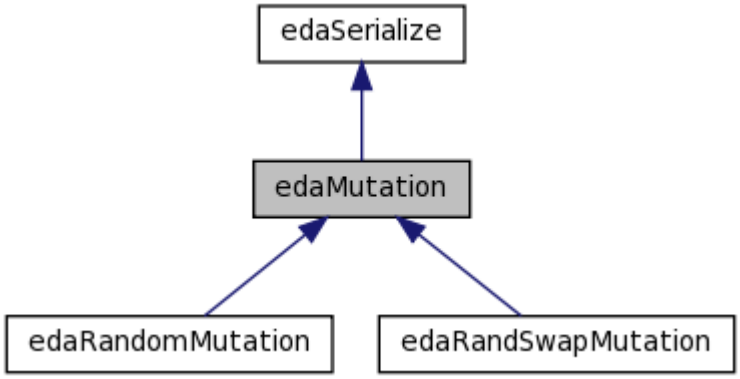
Thực hiện **edaWrapperControl** (*tr.181*).

Thông tin cho class được biên soạn từ các file sau đây:

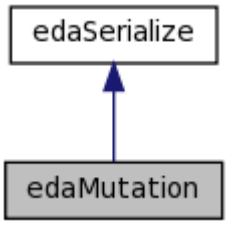
- lib/edaMpiWrapperControl.h
- lib/edaMpiWrapperControl.cpp

edaMutation Class Tham chiếu

Lớp trừ tượng **edaMutation** hiện thực các chiến lược đột biến.
`#include <edaMutation.h>`
Sơ đồ kế thừa cho edaMutation:



Sơ đồ liên kết cho edaMutation:



Các hàm thành viên Public

- **edaMutation** ()
  Khởi tạo đối tượng.
- **edaMutation** (const **edaMutation** &mute)
- virtual **~edaMutation** ()
  Hủy đối tượng.
- virtual **edaMutation** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **update** (**edaPopulation** &pop) const =0
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)=0
- virtual void **printOn** (ostream &os) const

Mô tả chi tiết

Lớp trừ tượng **edaMutation** hiện thực các chiến lược đột biến.

Thông tin về hàm thành viên

virtual void edaMutation::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Được thực hiện trong **edaRandomMutation** (*tr.134*), và **edaRandSwapMutation** (*tr.136*).

virtual void edaMutation::update (edaPopulation & pop) const `[pure virtual]`

Thực thi chiến lược đột biến lên tập dân cư

Các tham số:

pop	Tập dân cư cần đột biến

Được thực hiện trong **edaRandomMutation** (*tr.134*), và **edaRandSwapMutation** (*tr.136*).

Thông tin cho class được biên soạn từ các file sau đây:

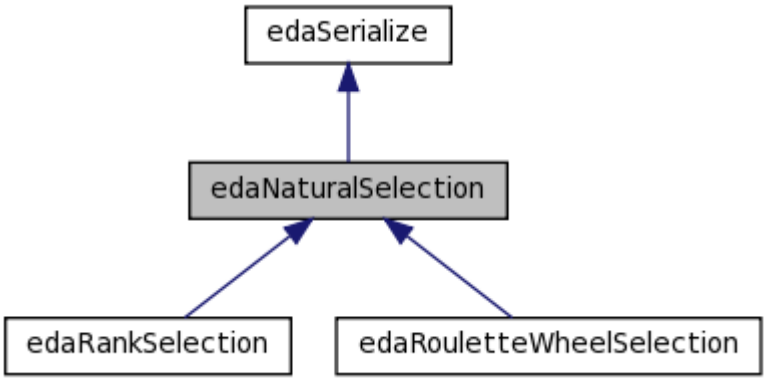
- lib/edaMutation.h
- lib/edaMutation.cpp

edaNaturalSelection Class Tham chiếu

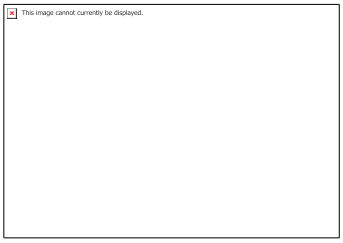
Lớp trừ tượng **edaNaturalSelection** hiện thực các chiến lược chọn lọc tự nhiên.

`#include <edaNaturalSelection.h>`

Sơ đồ kế thừa cho edaNaturalSelection:



Sơ đồ liên kết cho edaNaturalSelection:



Các hàm thành viên Public

- **edaNaturalSelection** ()
  Khởi tạo đối tượng.
- **edaNaturalSelection** (const **edaNaturalSelection** &slect)
- virtual **~edaNaturalSelection** ()
  Hủy đối tượng.
- virtual **edaNaturalSelection** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **update** (**edaPopulation** &pop)=0
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)=0
- virtual void **easer** ()=0
  óa thông tin chứa trong đối tượng
- virtual void **printOn** (ostream &os) const
  In thông tin (dạng chuỗi) của đối tượng lên ostream.

Mô tả chi tiết

Lớp trừ tượng **edaNaturalSelection** hiện thực các chiến lược chọn lọc tự nhiên.

Thông tin về Constructor và Destructor

edaNaturalSelection::edaNaturalSelection (const edaNaturalSelection & slect) `[inline]`

Khởi tạo đối tượng với dối tượng cần sao chép

Các tham số:

Đối	tượng cần sao chép

Thông tin về hàm thành viên

virtual void edaNaturalSelection::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Được thực hiện trong **edaRankSelection** (*tr.138*), và **edaRouletteWheelSelection** (*tr.143*).

virtual void edaNaturalSelection::update (edaPopulation & pop) `[pure virtual]`

Cập nhật đối tượng với chiến lược chọn lọc tự nhiên

Các tham số:

pop	Tập dân cư cần thực hiện chọn lọc

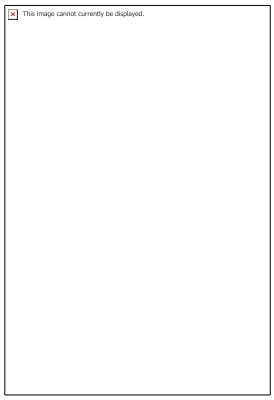
Được thực hiện trong **edaRankSelection** (*tr.138*), và **edaRouletteWheelSelection** (*tr.144*).

Thông tin cho class được biên soạn từ các file sau đây:

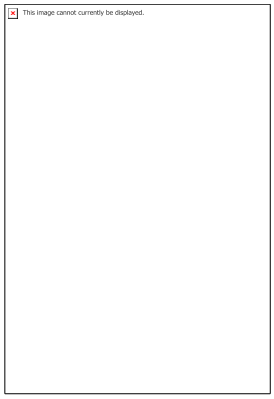
- lib/edaNaturalSelection.h
- lib/edaNaturalSelection.cpp

edaNoAspirCrit Class Tham chiếu

Lớp **edaImprBestFitAspirCrit** không thực hiện việc thay đổi danh sách Tabu.
`#include <edaNoAspirCrit.h>`
Sơ đồ kế thừa cho edaNoAspirCrit:



Sơ đồ liên kết cho edaNoAspirCrit:



Các hàm thành viên Public

- **edaNoAspirCrit** ()

Khởi tạo đối tượng.

- **edaNoAspirCrit** (const **edaNoAspirCrit** &nac)
- **~edaNoAspirCrit** ()
  Hủy đối tượng.
- **edaAspirCrit** * **clone** () const
- void **init** ()
  Khởi động đối tượng.
- bool **check** (const **edaMove** *_move, double fitness)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDANOASPIRCRIT_)

Mô tả chi tiết

Lớp **edaImprBestFitAspirCrit** không thực hiện việc thay đổi danh sách Tabu.

Thông tin về Constructor và Destructor

edaNoAspirCrit::edaNoAspirCrit (const edaNoAspirCrit & nac)

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thông tin về hàm thành viên

*bool edaNoAspirCrit::check (const edaMove * move, double fitness)* `[virtual]`

Kiểm tra bước chuyển move trong giải thuật Tabu có được chấp nhận không

Các tham số:

move	Bước trong cần kiểm tra
fitness	Lượng giá của bước chuyển

Thực hiện **edaAspirCrit** (*tr.60*).

*edaAspirCrit * edaNoAspirCrit::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaAspirCrit** (*tr.61*).

void edaNoAspirCrit::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

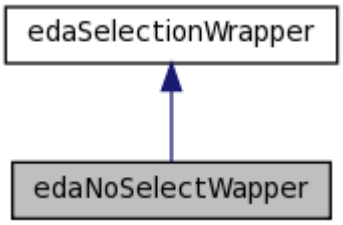
- lib/edaNoAspirCrit.h
- lib/edaNoAspirCrit.cpp

edaNoSelectWapper Class Tham chiếu

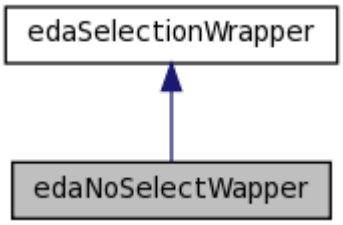
Lớp **edaNoSelectWapper** thực hiện việc loại bỏ tất cả các lời giải trong sanh sách lời giải.

`#include <edaNoSelectWapper.h>`

Sơ đồ kế thừa cho edaNoSelectWapper:



Sơ đồ liên kết cho edaNoSelectWapper:



Các hàm thành viên Public

- **edaNoSelectWapper** * **clone** () const
- bool **select** (**edaSolutionList** &list) const

Mô tả chi tiết

Lớp **edaNoSelectWapper** thực hiện việc loại bỏ tất cả các lời giải trong sanh sách lời giải.

Thông tin về hàm thành viên

*edaNoSelectWapper * edaNoSelectWapper::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSelectionWrapper** (*tr.151*).

bool edaNoSelectWapper::select (edaSolutionList & list) const `[virtual]`

Chọn lựa các lời giải với chiến lược chọn tương ứng trong tập lời giải

Giá trị trả về:

Nhận giá trị TRUE nếu chọn lựa thành công, ngược lại giá trị đầu ra là FALSE

Thực hiện **edaSelectionWrapper** (*tr.151*).

Thông tin cho class được biên soạn từ các file sau đây:

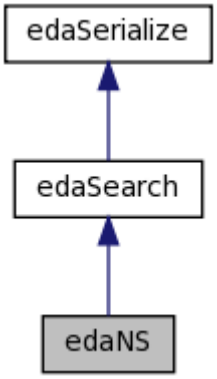
- lib/edaNoSelectWapper.h
- lib/edaNoSelectWapper.cpp

edaNS Class Tham chiếu

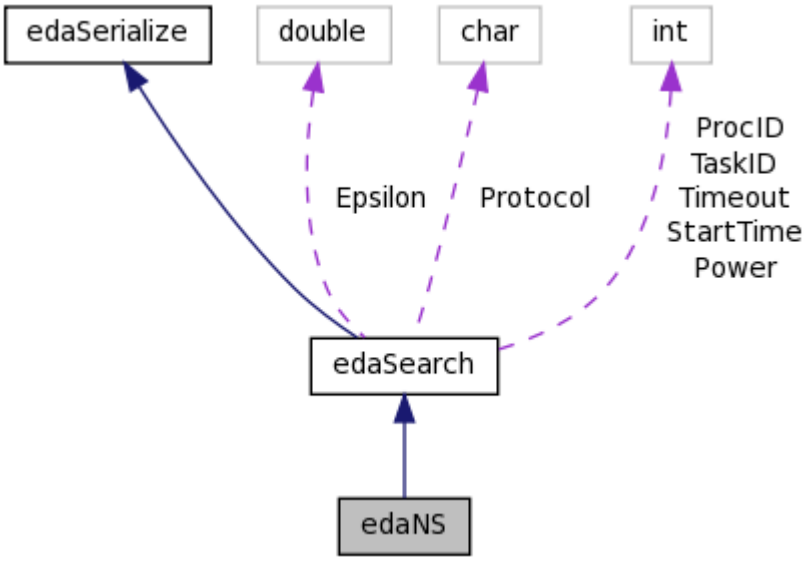
Lớp **edaNS** là một chiến lược tối ưu rỗng (không thực hiện quá trình tối ưu).

```
#include <edaNS.h>
```

Sơ đồ kế thừa cho edaNS:



Sơ đồ liên kết cho edaNS:



Các hàm thành viên Public

- **edaNS** (int _power)
  Khởi tạo đối tượng.
- **edaNS** (int timeout=0, int power=0)
- **edaNS** (const **edaNS** &ns)
- **edaNS** * **clone** () const
- virtual **~edaNS** ()
  Hủy đối tượng.
- bool **search** (**edaSolutionList** &list)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDANS_)

Additional Inherited Members

Mô tả chi tiết

Lớp **edaNS** là một chiến lược tối ưu rỗng (không thực hiện quá trình tối ưu).

Thông tin về Constructor và Destructor

edaNS::edaNS (int timeout = `0`, int power = `0`)

Khởi tạo đối tượng

Các tham số:

timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

edaNS::edaNS (const edaNS & ns)

Khởi tạo đối tượng

Các tham số:

ns	Đối tượng search cần sao chép

Thông tin về hàm thành viên

*edaNS * edaNS::clone () const `[virtual]`*

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSearch** (*tr.148*).

bool edaNS::search (edaSolutionList & list) `[virtual]`

Thực thi tối ưu

Các tham số:

list	Danh sách lời giải cần tối ưu

Giá trị trả về:

TRUE: tối ưu thành công, FALSE: tối ưu thất bại

Thực hiện **edaSearch** (*tr.148*).

void edaNS::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSearch** (*tr.148*).

Thông tin cho class được biên soạn từ các file sau đây:

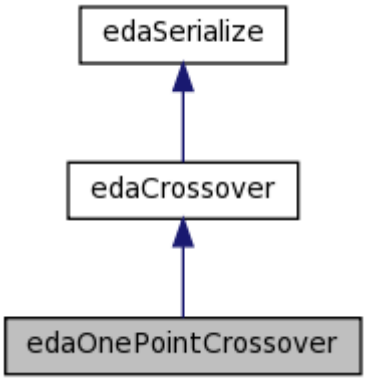
- lib/edaNS.h
- lib/edaNS.cpp

edaOnePointCrossover Class Tham chiếu

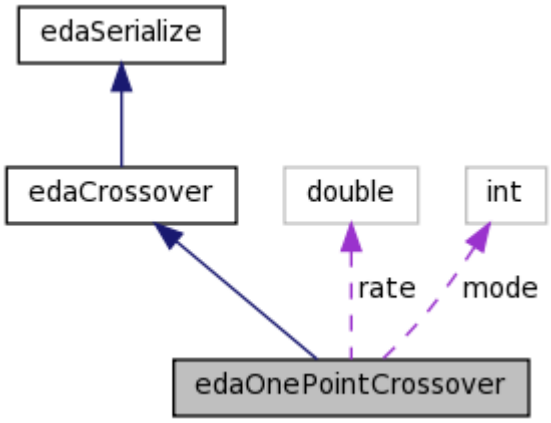
Lớp **edaOnePointCrossover** hiện thực chiến lược lai chéo giữa các nhiệm sắc thể với một diểm lai chéo với các mode lai khác nhau.

```
#include <edaOnePointCrossover.h>
```

Sơ đồ kế thừa cho edaOnePointCrossover:



Sơ đồ liên kết cho edaOnePointCrossover:



Public Types

- enum **CrossPointMode** { **MEDIAN**, **RANDOM** }

Các hàm thành viên Public

- **edaOnePointCrossover** (double rate=1.0)
- **edaOnePointCrossover** (const **edaOnePointCrossover** &cross)
  Khởi tạo đối tượng với đối tượng cần sao chép.
- virtual **~edaOnePointCrossover** ()
  Hủy đối tượng.
- **edaOnePointCrossover** * **clone** () const
  Nhân bản đối tượng.
- void **setRate** (double value)
- void **setMode** (**CrossPointMode** mode=RANDOM)
- void **update** (**edaPopulation** &pop)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAONEPOINTCROSSOVER_)

các thuộc tính Protected

- double **rate**
- unsigned int **mode**

Mô tả chi tiết

Lớp **edaOnePointCrossover** hiện thực chiến lược lai chéo giữa các nhiệm sắc thể với một điểm lai chéo với các mode lai khác nhau.

Thông tin về Member Enumeration

enum edaOnePointCrossover::CrossPointMode

Các mode lai chéo một điểm được hỗ trợ

Các tham số:

MEDIAN	Điểm lai chéo tại trung vị của chuỗi gen
RANDOM	Điểm lai chéo được chọn ngẫu nhiên

Thông tin về Constructor và Destructor

edaOnePointCrossover::edaOnePointCrossover (double rate = `1.0`)

Khởi tạo đối tượng

Các tham số:

rate	Tỷ lệ lai chéo

Thông tin về hàm thành viên

void edaOnePointCrossover::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Thực hiện **edaCrossover** (*tr.73*).

void edaOnePointCrossover::setMode (CrossPointMode mode = `RANDOM`*)*

Thiết lập mode lai chéo

Các tham số:

mode	Mode lai chéo (mặc định là ngẫu nhiên)

void edaOnePointCrossover::setRate (double value)

Thiết lập tỷ lệ lai chéo

Các tham số:

value	Tỷ lệ lai chéo

void edaOnePointCrossover::update (edaPopulation & pop) `[virtual]`

Thực hiện việc cập nhật tập dân cư với chiến lược tương ứng với từng hiện thực cụ thể

Các tham số:

pop	Tập dân cư cần lai chéo

Thực hiện **edaCrossover** (*tr.73*).

Thông tin cho class được biên soạn từ các file sau đây:

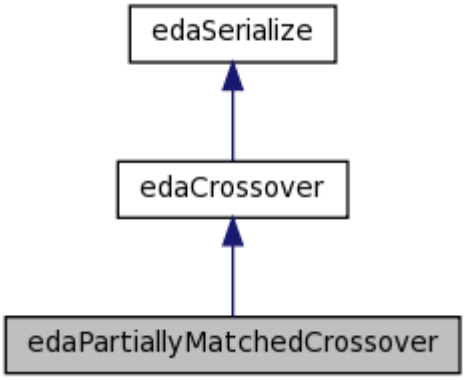
- lib/edaOnePointCrossover.h
- lib/edaOnePointCrossover.cpp

edaPartiallyMatchedCrossover Class Tham chiếu

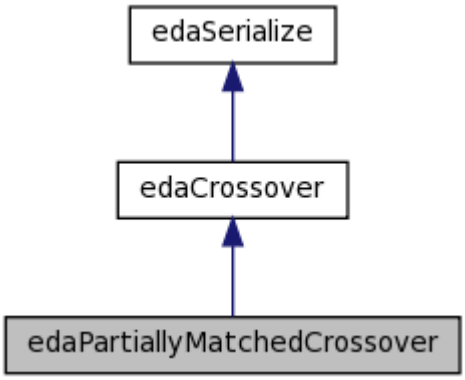
Lớp **edaPartiallyMatchedCrossover** hiện thực chiến lược lai chéo PMX giữa các nhiệm sắc thể.

`#include <edaPartiallyMatchedCrossover.h>`

Sơ đồ kế thừa cho edaPartiallyMatchedCrossover:



Sơ đồ liên kết cho edaPartiallyMatchedCrossover:



Các hàm thành viên Public

- **edaPartiallyMatchedCrossover** (double rate=1.0, unsigned int num=1)
- **edaPartiallyMatchedCrossover** (const **edaPartiallyMatchedCrossover** &cross)
  Khởi tạo đối tượng với đối tượng cần sao chép.
- virtual **~edaPartiallyMatchedCrossover** ()
  Hủy đối tượng.
- void **setRate** (double value)
- void **setNumPoint** (unsigned int num)
- **edaPartiallyMatchedCrossover** * **clone** () const
  Nhân bản đối tượng.
- void **update** (**edaPopulation** &pop)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- void **printOn** (ostream &os) const
- **setClassID** (_SYSCLASSID_+_CLSID_EDAPARTIALLYMATCHEDCROSSOVER_)

Mô tả chi tiết

Lớp **edaPartiallyMatchedCrossover** hiện thực chiến lược lai chéo PMX giữa các nhiệm sắc thể.

Thông tin về Constructor và Destructor

edaPartiallyMatchedCrossover::edaPartiallyMatchedCrossover (double rate = `1.0`, unsigned int num = `1`)

Khởi tạo đối tượng

Các tham số:

rate	Tỷ lệ lai chéo
num	Số điểm lai chéo

Thông tin về hàm thành viên

void edaPartiallyMatchedCrossover::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Thực hiện **edaCrossover** (*tr.73*).

void edaPartiallyMatchedCrossover::setNumPoint (unsigned int num)

Thiết lập số điểm lai chéo

Các tham số:

value	Số điểm lai chéo

void edaPartiallyMatchedCrossover::setRate (double value)

Thiết lập tỷ lệ lai chéo

Các tham số:

value	Tỷ lệ lai chéo

void edaPartiallyMatchedCrossover::update (edaPopulation & pop) `[virtual]`

Thực hiện việc cập nhật tập dân cư với chiến lược tương ứng với từng hiện thực cụ thể

Các tham số:

pop	Tập dân cư cần lai chéo

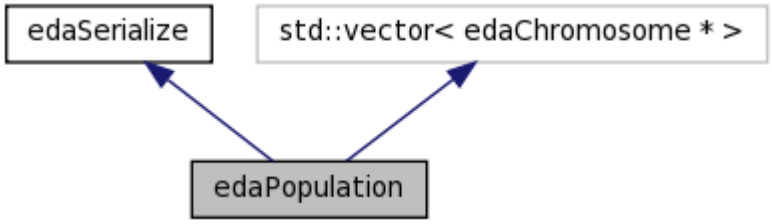
Thực hiện **edaCrossover** (*tr.73*).

Thông tin cho class được biên soạn từ các file sau đây:

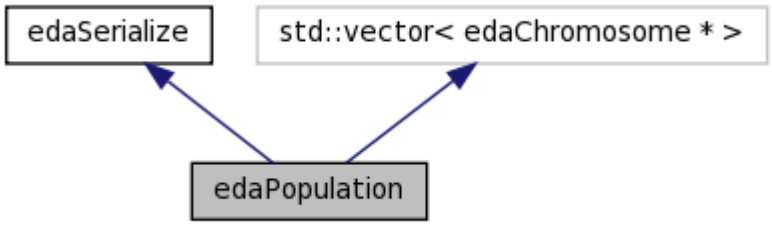
- lib/edaPartiallyMatchedCrossover.h
- lib/edaPartiallyMatchedCrossover.cpp

edaPopulation Class Tham chiếu

Lớp **edaPopulation** hiện thực đối tượng quần thể trong giải thuật tiến hóa.
`#include <edaPopulation.h>`
Sơ đồ kế thừa cho edaPopulation:



Sơ đồ liên kết cho edaPopulation:



Các hàm thành viên Public

- **edaPopulation** ()
  Khởi tạo đối tượng.
- **edaPopulation** (unsigned int num)
- **edaPopulation** (const **edaPopulation** &pop)
- **edaPopulation** * **clone** () const
  Nhân bản đối tượng.
- void **printOn** (ostream &os) const
- virtual **~edaPopulation** ()
  Hũy đối tượng.
- **edaChromosome** * **pop_back** ()
- double **mean** () const
- double **std** () const
- double **min** () const
- double **max** () const
- **edaPopulation** & **operator=** (const **edaPopulation** &pop)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDAPOPULATION_)
- void **easer** ()
  Xóa thông tin trong đối tượng.
- void **sort** ()
  Sắp xếp các cá thể trong tập dân cư theo thứ tự hàm lượng giá

Mô tả chi tiết

Lớp **edaPopulation** hiện thực đối tượng quần thể trong giải thuật tiến hóa.

Thông tin về Constructor và Destructor

edaPopulation::edaPopulation (unsigned int num)

Khởi tạo đối tượng

Các tham số:

num	Số dân cư trong quần thể

edaPopulation::edaPopulation (const edaPopulation & pop)

Các tham số:

pop	Đối tượng cần sao chép

Thông tin về hàm thành viên

double edaPopulation::max () const

Lấy giá trị lượng giá cao nhất của tập dân cư

Giá trị trả về:

Giá trị lượng giá cao nhất

double edaPopulation::mean () const

Lấy giá trị lượng giá trung bình của tập dân cư

Giá trị trả về:

Giá trị lượng giá trung bình

double edaPopulation::min () const

Lấy giá trị lượng giá thấp nhất của tập dân cư

Giá trị trả về:

Giá trị lượng giá thấp nhất

*edaChromosome * edaPopulation::pop_back ()*

Lấy cá thể nằm cuối tập dân cư

Giá trị trả về:

Cá thể **edaChromosome** nằm ở vị trí cuối tập dân cư

void edaPopulation::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

double edaPopulation::std () const

Lấy độ lệch chuẩn của lượng giá của tập dân cư

Giá trị trả về:

Giá trị độ lệch chuẩn

Thông tin cho class được biên soạn từ các file sau đây:

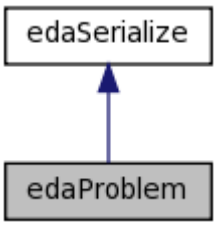
- lib/edaPopulation.h
- lib/edaPopulation.cpp

edaProblem Class Tham chiếu

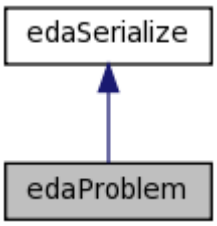
Lớp trừu tượng **edaProblem** chứa thông tin bài toán cho từng vấn đề tối ưu cụ thể.

```
#include <edaProblem.h>
```

Sơ đồ kế thừa cho edaProblem:



Sơ đồ liên kết cho edaProblem:



Các hàm thành viên Public

- **edaProblem** ()
  Khởi tạo đối tượng.
- **edaProblem** (const char *filename)
- **edaProblem** (const **edaProblem** &pro)
- virtual **~edaProblem** ()
  Hũy đối tượng.
- virtual **edaProblem** * **clone** () const =0
  Nhân bản đối tượng.
- virtual **edaProblem** & **operator=** (const **edaProblem** &pro)=0
- virtual void **printOn** (ostream &os) const
- void **Serialize** (**edaBuffer** &buf, bool pack)=0

Mô tả chi tiết

Lớp trừu tượng **edaProblem** chứa thông tin bài toán cho từng vấn đề tối ưu cụ thể.

Thông tin về Constructor và Destructor

*edaProblem::edaProblem (const char * filename)* `[inline]`

Khởi tạo đối tượng

Các tham số:

filename	Tên file chứa thông tin của bài toán tối ưu

edaProblem::edaProblem (const edaProblem & pro) `[inline]`

Khởi tạo đối tượng

Các tham số:

pro	Đối tượng cần sao chép

Thông tin về hàm thành viên

void edaProblem::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

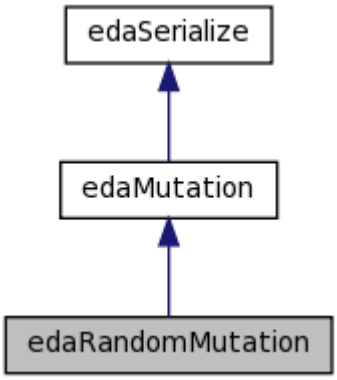
Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

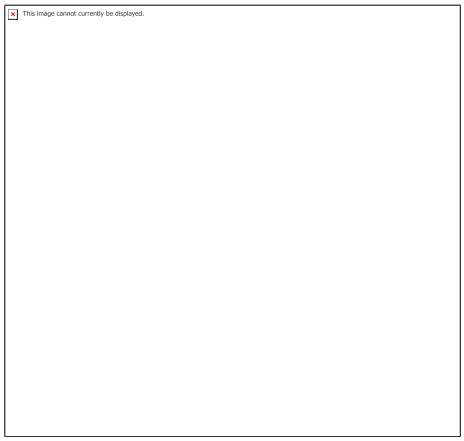
- lib/edaProblem.h
- lib/edaProblem.cpp

edaRandomMutation Class Tham chiếu

Lớp **edaRandomMutation** hiện thực chiến lược đột biến phát sinh ngẫu nhiên.
`#include <edaRandomMutation.h>`
Sơ đồ kế thừa cho edaRandomMutation:



Sơ đồ liên kết cho edaRandomMutation:



Các hàm thành viên Public

- **edaRandomMutation** (double rare=0.1)
- **edaRandomMutation** (const **edaRandomMutation** &mute)
- virtual **~edaRandomMutation** ()
  Hủy đối tượng.
- virtual **edaRandomMutation** * **clone** () const
  Nhân bản đối tượng.
- virtual void **setRate** (double value)
- virtual void **update** (**edaPopulation** &pop) const
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDARANDOMMUTATION_)

các thuộc tính Protected

- double **rate**

Mô tả chi tiết

Lớp **edaRandomMutation** hiện thực chiến lược đột biến phát sinh ngẫu nhiên.

Thông tin về Constructor và Destructor

edaRandomMutation::edaRandomMutation (double rare = `0.1`)

Khởi tạo đối tượng

Các tham số:

rate	Tỷ lệ đột biến

edaRandomMutation::edaRandomMutation (const edaRandomMutation & mute)

Khởi tạo đối tượng

Các tham số:

mute	Đối tượng cần sao chép

Thông tin về hàm thành viên

void edaRandomMutation::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Thực hiện **edaMutation** (*tr.117*).

void edaRandomMutation::setRate (double value) `[virtual]`

Thiết lập tỷ lệ đột biến

Các tham số:

value	Tỷ lệ đột biến

void edaRandomMutation::update (edaPopulation & pop) const `[virtual]`

Thực thi chiến lược đột biến lên tập dân cư

Các tham số:

pop	Tập dân cư cần đột biến

Thực hiện **edaMutation** (*tr.117*).

Thông tin cho class được biên soạn từ các file sau đây:

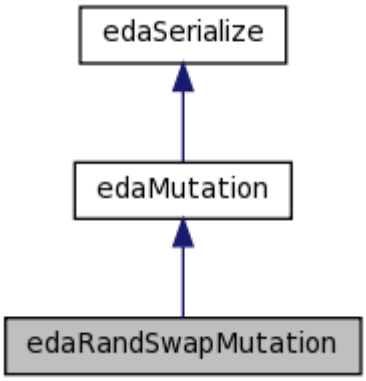
- lib/edaRandomMutation.h
- lib/edaRandomMutation.cpp

edaRandSwapMutation Class Tham chiếu

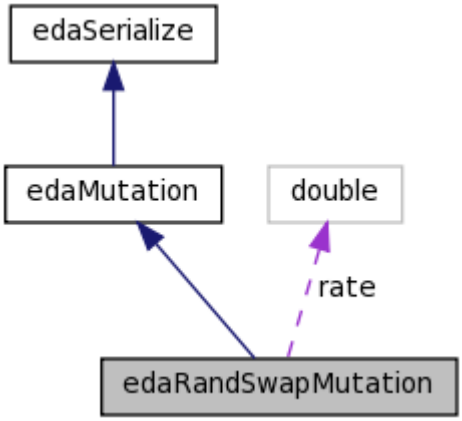
Lớp **edaRandSwapMutation** hiện thực chiến lược đột biến hoán đổi gen.

```
#include <edaRandSwapMutation.h>
```

Sơ đồ kế thừa cho edaRandSwapMutation:



Sơ đồ liên kết cho edaRandSwapMutation:



Các hàm thành viên Public

- **edaRandSwapMutation** (double rate=0.1)
- **edaRandSwapMutation** (const **edaRandSwapMutation** &mute)
- **~edaRandSwapMutation** ()
  Hủy đối tượng.
- **edaRandSwapMutation** * **clone** () const
  Nhân bản đối tượng.
- void **printOn** (ostream &os) const
- void **update** (**edaPopulation** &pop) const
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDARANDSWAPMUTATION_)

Các hàm thành viên Protected

- void **getRandPermutation** (**edaChromosome** *chro) const

các thuộc tính Protected

- double **rate**

Mô tả chi tiết

Lớp **edaRandSwapMutation** hiện thực chiến lược đột biến hoán đổi gen.

Thông tin về Constructor và Destructor

edaRandSwapMutation::edaRandSwapMutation (double rate = `0.1`)

Khởi tạo đối tượng

Các tham số:

rate	Tỷ lệ đột biến

edaRandSwapMutation::edaRandSwapMutation (const edaRandSwapMutation & mute)

Khởi tạo đối tượng

Các tham số:

mute	Đối tượng cần sao chép

Thông tin về hàm thành viên

void edaRandSwapMutation::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Thực hiện **edaMutation** (*tr.117*).

void edaRandSwapMutation::update (edaPopulation & pop) const `[virtual]`

Thực thi chiến lược đột biến lên tập dân cư

Các tham số:

pop	Tập dân cư cần đột biến

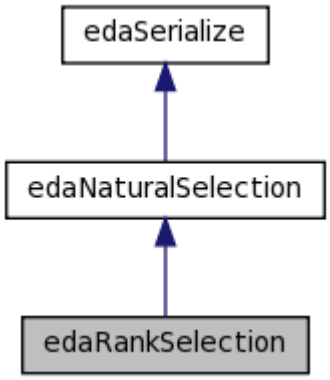
Thực hiện **edaMutation** (*tr.117*).

Thông tin cho class được biên soạn từ các file sau đây:

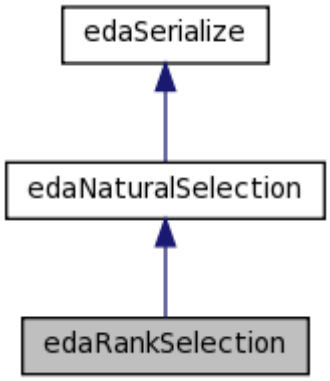
- lib/edaRandSwapMutation.h
- lib/edaRandSwapMutation.cpp

edaRankSelection Class Tham chiếu

Lớp **edaRankSelection** hiện thực chiến lược chọn lọc tự nhiên theo thứ hạng.
`#include <edaRankSelection.h>`
Sơ đồ kế thừa cho edaRankSelection:



Sơ đồ liên kết cho edaRankSelection:



Các hàm thành viên Public

- **edaRankSelection** (const double rate=1.0)
- **edaRankSelection** (const **edaRankSelection** &slect)
- virtual **~edaRankSelection** ()
  Hũy đối tượng.
- **edaRankSelection** * **clone** () const
  Nhân bản đối tượng.
- void **setRate** (double value)
- void **update** (**edaPopulation** &pop)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- void **printOn** (ostream &os) const
  In thông tin (dạng chuổi) của đối tượng lên ostream.
- **setClassID** (_SYSCLASSID_+_CLSID_EDARANKSELECTION_)

Mô tả chi tiết

Lớp **edaRankSelection** hiện thực chiến lược chọn lọc tự nhiên theo thứ hạng.

Thông tin về Constructor và Destructor

edaRankSelection::edaRankSelection (const double rate = 1.0)

Khởi tạo đối tượng

Các tham số:

rate	Tỷ lệ chọn lọc

edaRankSelection::edaRankSelection (const edaRankSelection & slect)

Khởi tạo đối tượng

Các tham số:

slect	Đối tượng cần sao chép

Thông tin về hàm thành viên

void edaRankSelection::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Thực hiện **edaNaturalSelection** (*tr.118*).

void edaRankSelection::setRate (double value)

Gán tỷ lệ chọn lọc

Các tham số:

value	Tỷ lệ chọn lọc

void edaRankSelection::update (edaPopulation & pop) `[virtual]`

Cập nhật đối tượng với chiến lược chọn lọc tự nhiên

Các tham số:

pop	Tập dân cư cần thực hiện chọn lọc

Thực hiện **edaNaturalSelection** (*tr.119*).

Thông tin cho class được biên soạn từ các file sau đây:

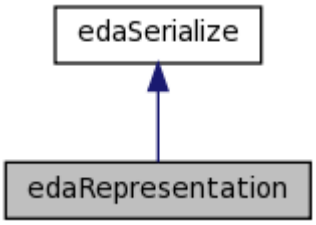
- lib/edaRankSelection.h
- lib/edaRankSelection.cpp

edaRepresentation Class Tham chiếu

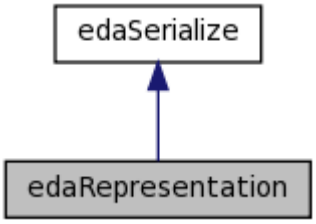
Lớp trừu tượng **edaRepresentation** hiện thực việc chứa phương pháp mã hóa từ lời giải thành nhiệm sắc thể.

```
#include <edaRepresentation.h>
```

Sơ đồ kế thừa cho edaRepresentation:



Sơ đồ liên kết cho edaRepresentation:



Các hàm thành viên Public

- **edaRepresentation** ()
  Khởi tạo đối tượng.
- **edaRepresentation** (const **edaProblem** &pro)
- **edaRepresentation** (const **edaRepresentation** &repre)
- virtual **edaRepresentation** * **clone** () const =0
  Nhân bản đối tượng.
- virtual **~edaRepresentation** ()
  Hũy đối tượng.
- virtual void **init** (const **edaSolutionList** &list, **edaPopulation** &pop) const =0
- virtual void **decode** (const **edaPopulation** &pop, **edaSolutionList** &list) const =0
- virtual void **encode** (const **edaSolutionList** &list, **edaPopulation** &pop) const =0
- virtual void **printOn** (ostream &os) const
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)=0

Mô tả chi tiết

Lớp trừu tượng **edaRepresentation** hiện thực việc chứa phương pháp mã hóa từ lời giải thành nhiệm sắc thể.

Thông tin về Constructor và Destructor

edaRepresentation::edaRepresentation (const edaProblem & pro) `[inline]`

Khởi tạo đối tượng

Các tham số:

pro	Thông tin bài toán

edaRepresentation::edaRepresentation (const edaRepresentation & repre) `[inline]`

Khởi tạo đối tượng

Các tham số:

repre	Đối tượng cần sao chép

Thông tin về hàm thành viên

virtual void edaRepresentation::decode (const edaPopulation & pop, edaSolutionList & list) const `[pure virtual]`

Mã hóa tập dân cư từ danh sách lời giải

Các tham số:

list	Danh sách lời giải
pop	Tập dân cư

virtual void edaRepresentation::encode (const edaSolutionList & list, edaPopulation & pop) const `[pure virtual]`

Giải mã danh sách lời giải từ tập dân cư

Các tham số:

list	Danh sách lời giải
pop	Tập dân cư

virtual void edaRepresentation::init (const edaSolutionList & list, edaPopulation & pop) const `[pure virtual]`

Khởi động đối tượng

Các tham số:

list	Danh sách lời giải ban đầu
pop	Tập dân cư cần được mã hóa

virtual void edaRepresentation::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaRepresentation.h
- lib/edaRepresentation.cpp

edaRNG Class Tham chiếu

Lớp **edaRNG** hiện thực cơ chế tạo các số ngẫu nhiên.
`#include <edaRNG.h>`

Các hàm thành viên Public

- **edaRNG** (ulong x_)
- **edaRNG** (ulong z_, ulong w_, ulong jsr_, ulong jcong_)
- double **RNOR** ()
- double **REXP** ()
- double **nfix** (slong h, ulong i)
- double **efix** (ulong j, ulong i)
- void **zigset** ()
- void **init** ()
- void **init** (ulong rank)
- void **init** (ulong z_, ulong w_, ulong jsr_, ulong jcong_)
- long **rand_int31** ()
- double **rand_closed01** ()
- double **rand_open01** ()
- double **rand_halfclosed01** ()
- double **rand_halfopen01** ()
- double **uniform** (double x=0.0, double y=1.0)
- double **normal** (double mu=0.0, double sd=1.0)
- double **exponential** (double lambda=1)
- double **gamma** (double shape=1, double scale=1)
- double **chi_square** (double df)
- double **beta** (double a1, double a2)
- void **uniform** (vector< double > &res, double x=0.0, double y=1.0)
- void **normal** (vector< double > &res, double mu=0.0, double sd=1.0)
- void **exponential** (vector< double > &res, double lambda=1)
- void **gamma** (vector< double > &res, double shape=1, double scale=1)
- void **chi_square** (vector< double > &res, double df)
- void **beta** (vector< double > &res, double a1, double a2)
- unsigned int **random** (unsigned int n)
- unsigned int **random** (unsigned int min, unsigned max)
- int **poisson** (double mu)
- int **binomial** (double p, int n)
- void **multinom** (unsigned int n, const vector< double > &probs, vector< uint > &samp)
- void **multinom** (unsigned int n, const double *prob, uint K, uint *samp)
- void **poisson** (vector< int > &res, double lambda)
- void **binomial** (vector< int > &res, double p, int n)
- double **round** (double value, unsigned int digit)
- double * **linspace** (double a, double b, unsigned int n)
- unsigned int **sum** (vector< unsigned int > v)
- double **sum** (vector< double > v)
- void **swap** (unsigned int &a, unsigned int &b)

Mô tả chi tiết

Lớp **edaRNG** hiện thực cơ chế tạo các số ngẫu nhiên.

Thông tin cho class được biên soạn từ các file sau đây:

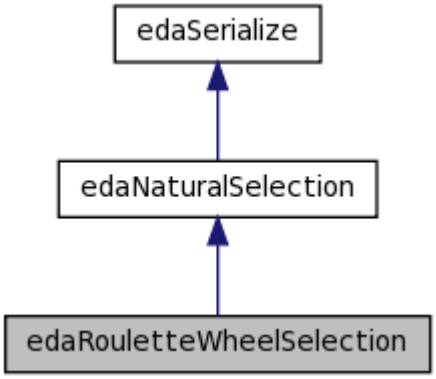
- lib/edaRNG.h
- lib/edaRNG.cpp

edaRouletteWheelSelection Class Tham chiếu

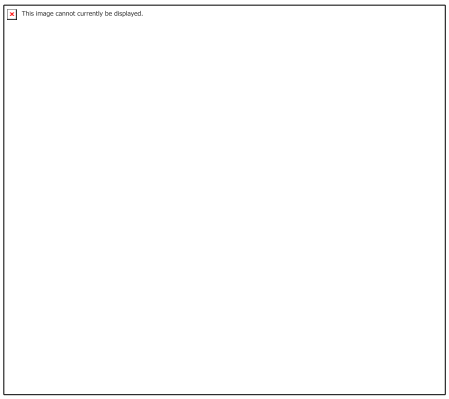
Lớp **edaRouletteWheelSelection** hiện thực chiến lược chọn lọc tự nhiên theo trọng số

`#include <edaRouletteWheelSelection.h>`

Sơ đồ kế thừa cho edaRouletteWheelSelection:



Sơ đồ liên kết cho edaRouletteWheelSelection:



Các hàm thành viên Public

- **edaRouletteWheelSelection** (const double rate=1.0, const double alpha=2)
- **edaRouletteWheelSelection** (const **edaRouletteWheelSelection** &slect)
- virtual **~edaRouletteWheelSelection** ()
  Hũy đối tượng.
- **edaRouletteWheelSelection** * **clone** () const
  Nhân bản đối tượng.
- void **printOn** (ostream &os) const
  In thông tin (dạng chuổi) của đối tượng lên ostream.
- void **computeCostWeight** (const **edaPopulation** &pop)
- void **setAlpha** (double value)
- void **update** (**edaPopulation** &pop)
- void **Serialize** (**edaBuffer** &buf, bool pack)

- **setClassID** (_SYSCLASSID_+_CLSID_EDAROULETTEWHEELSELECTION_)

Mô tả chi tiết

Lớp **edaRouletteWheelSelection** hiện thực chiến lược chọn lọc tự nhiên theo trọng số

Thông tin về Constructor và Destructor

edaRouletteWheelSelection::edaRouletteWheelSelection (const double rate = `1.0`, const double alpha = `2`)

Khởi tạo đối tượng

Các tham số:

rate	Tỷ lệ chọn lọc
alpha	Hệ số nền dùng để tính trọng số cho cá thể được chọn lọc

edaRouletteWheelSelection::edaRouletteWheelSelection (const edaRouletteWheelSelection & slect)

Khởi tạo đối tượng

Các tham số:

slect	Đối tượng cần sao chép

Thông tin về hàm thành viên

void edaRouletteWheelSelection::computeCostWeight (const edaPopulation & pop)

Tính toán trọng số cho các cá thể trong tập dân cư

Các tham số:

pop	Tập dân cư cần tính toán

void edaRouletteWheelSelection::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Thực hiện **edaNaturalSelection** (*tr.118*).

void edaRouletteWheelSelection::setAlpha (double value)

Gán hệ số nền dùng để tính trọng số cho cá thể được chọn lọc

Các tham số:

value	Hệ số nền

void edaRouletteWheelSelection::update (edaPopulation & pop) `[virtual]`

Cập nhật đối tượng với chiến lược chọn lọc tự nhiên

Các tham số:

pop	Tập dân cư cần thực hiện chọn lọc

Thực hiện **edaNaturalSelection** (*tr.119*).

Thông tin cho class được biên soạn từ các file sau đây:

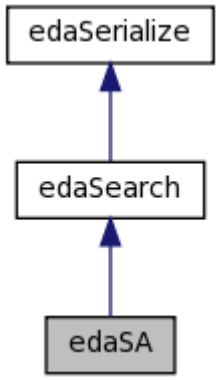
- lib/edaRouletteWheelSelection.h
- lib/edaRouletteWheelSelection.cpp

edaSA Class Tham chiếu

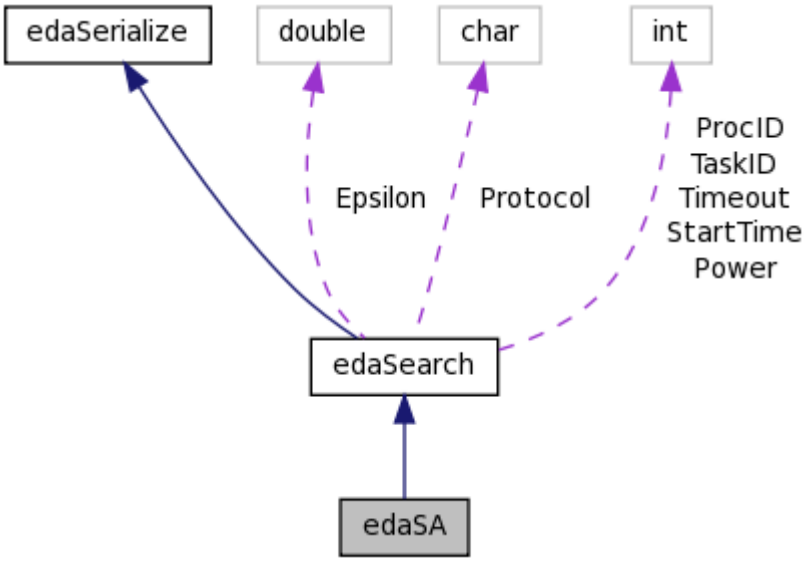
Lớp **edaSA** thực hiện việc tối ưu theo giải thuật mô phỏng luyện kim.

`#include <edaSA.h>`

Sơ đồ kế thừa cho edaSA:



Sơ đồ liên kết cho edaSA:



Các hàm thành viên Public

- **edaSA** ()
  Khởi tạo đối tượng.
- **edaSA** (int _power)
- **edaSA** (**edaMove** *move, **edaMoveGen** *moveRandom, **edaContinue** *continueCriteria, double initTemperature, **edaCoolingSchedule** *coolingSchedule, int timeout=0, int power=0)
- **edaSA** (const **edaSA** &sa)
- **edaSA** * **clone** () const
- bool **search** (**edaSolutionList** &list)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDASA_)

Additional Inherited Members

Mô tả chi tiết

Lớp **edaSA** thực hiện việc tối ưu theo giải thuật mô phỏng luyện kim.

Thông tin về Constructor và Destructor

edaSA::edaSA (int _power)

Khởi tạo đối tượng

Các tham số:

power	Bậc tìm kiếm

*edaSA::edaSA (edaMove * move, edaMoveGen * moveRandom, edaContinue * continueCriteria, double initTemperature, edaCoolingSchedule * coolingSchedule, int timeout = 0, int power = 0)*

Khởi tạo đối tượng

Các tham số:

move	Chiến lược bước chuyển trong không gian tìm kiếm
moveRandom	Phương pháp tạo bước chuyển ngẫu nhiên trong không gian tìm kiếm
continueCriteria	Điều kiện dừng
coolingSchedule	Chiến luộc làm lạnh nhiệt độ
initTemperature	Nhiệt độ khởi động ban đầu
timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

edaSA::edaSA (const edaSA & sa)

Khởi tạo đối tượng

Các tham số:

sa	Đối tượng search cần sao chép

Thông tin về hàm thành viên

*edaSA * edaSA::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSearch** (*tr.148*).

bool edaSA::search (edaSolutionList & list) `[virtual]`

Thực thi tối ưu

Các tham số:

list	Danh sách lời giải cần tối ưu

Giá trị trả về:

TRUE: tối ưu thành công, FALSE: tối ưu thất bại

Thực hiện **edaSearch** (*tr.148*).

void edaSA::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSearch** (*tr.148*).

Thông tin cho class được biên soạn từ các file sau đây:

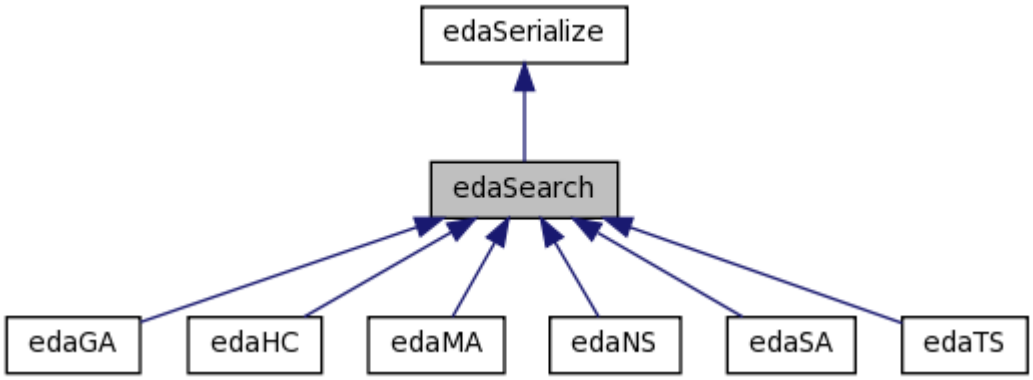
- lib/edaSA.h
- lib/edaSA.cpp

edaSearch Class Tham chiếu

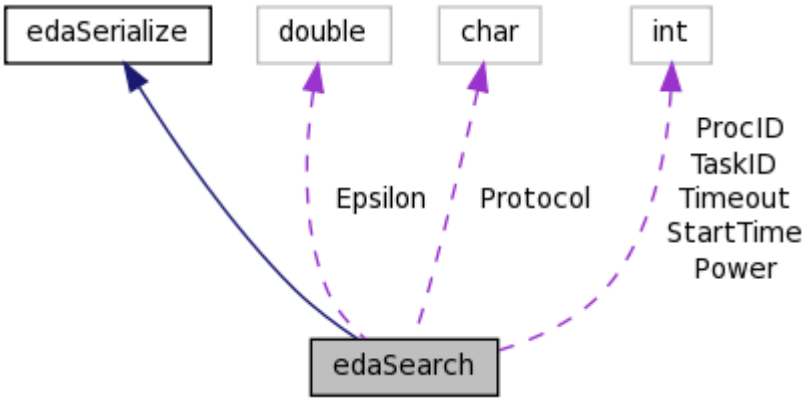
Lớp trừu tượng **edaSearch** thực hiện việc tối ưu với các chiến lược hỗ trợ trong thư viện hoặc do người dùng xây dựng.

```
#include <edaSearch.h>
```

Sơ đồ kế thừa cho edaSearch:



Sơ đồ liên kết cho edaSearch:



Các hàm thành viên Public

- **edaSearch** ()
  Khởi tạo đối tượng.
- **edaSearch** (int power, const char *protocol="http")
- **edaSearch** (int timeout, int power)
- **edaSearch** (const **edaSearch** &**search**)
- virtual **edaSearch** * **clone** () const =0
- virtual **~edaSearch** ()
  Hủy đối tượng.
- virtual bool **search** (**edaSolutionList** &list)=0
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)

các trường dữ liệu

- int **Timeout**
- int **StartTime**
- double **Epsilon**
- int **Power**
- char **Protocol** [1024]
- int **TaskID**
- int **ProcID**

Mô tả chi tiết

Lớp trừu tượng **edaSearch** thực hiện việc tối ưu với các chiến lược hỗ trợ trong thư viện hoặc do người dùng xây dựng.

Thông tin về Constructor và Destructor

*edaSearch::edaSearch (int power, const char * protocol = "http")*

Khởi tạo đối tượng

Các tham số:

power	Bậc tìm kiếm
protocol	Giao thức

edaSearch::edaSearch (int timeout, int power)

Khởi tạo đối tượng

Các tham số:

timeout	Thời gian thoát
power	Bậc tìm kiếm

edaSearch::edaSearch (const edaSearch & search)

Khởi tạo đối tượng

Các tham số:

search	Đối tượng search cần sao chép

Thông tin về hàm thành viên

virtual edaSearch edaSearch::clone () const* `[pure virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Được thực hiện trong **edaTS** (*tr.175*), **edaHC** (*tr.93*), **edaMA** (*tr.106*), **edaGA** (*tr.86*), **edaSA** (*tr.146*), và **edaNS** (*tr.123*).

virtual bool edaSearch::search (edaSolutionList & list) `[pure virtual]`

Thực thi tối ưu

Các tham số:

list	Danh sách lời giải cần tối ưu

Giá trị trả về:

TRUE: tối ưu thành công, FALSE: tối ưu thất bại

Được thực hiện trong **edaTS** (*tr.175*), **edaHC** (*tr.93*), **edaGA** (*tr.86*), **edaMA** (*tr.106*), **edaSA** (*tr.146*), và **edaNS** (*tr.123*).

void edaSearch::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Được thực thi lại trong **edaTS** (*tr.175*), **edaGA** (*tr.87*), **edaHC** (*tr.93*), **edaMA** (*tr.107*), **edaSA** (*tr.146*), và **edaNS** (*tr.124*).

Thông tin cho class được biên soạn từ các file sau đây:

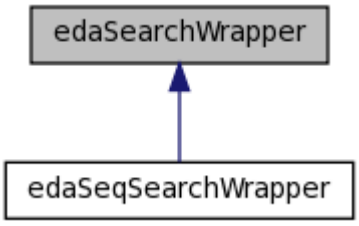
- lib/edaSearch.h
- lib/edaSearch.cpp

edaSearchWrapper Class Tham chiếu

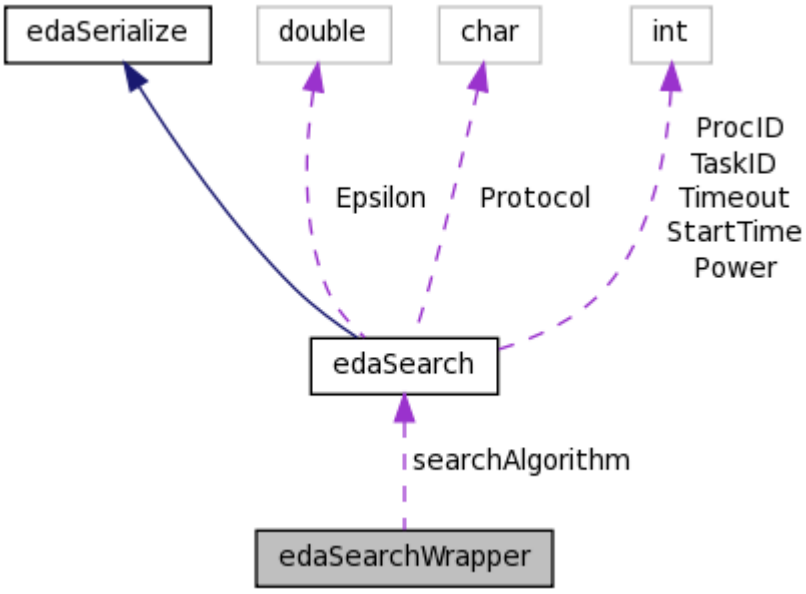
Lớp **edaSearchWrapper** hỗ trợ cơ chế tối ưu với thông tin tìm kiếm đã được đóng gói.

```
#include <edaSearchWrapper.h>
```

Sơ đồ kế thừa cho edaSearchWrapper:



Sơ đồ liên kết cho edaSearchWrapper:



Các hàm thành viên Public

- **edaSearchWrapper** ()
  Khởi tạo đối tượng.
- **edaSearchWrapper** (**edaBuffer** &buf)
- virtual **~edaSearchWrapper** ()
  Hũy đối tượng.
- virtual void **search** (**edaBuffer** &buf_in, **edaBuffer** &buf_out)=0

Các hàm thành viên Protected

- void **setAlgorithm** (**edaBuffer** &_buf)

các thuộc tính Protected

- **edaSearch** * **searchAlgorithm**

Mô tả chi tiết

Lớp **edaSearchWrapper** hổ trợ cơ chế tối ưu với thông tin tìm kiếm đã được đóng gói.

Thông tin về Constructor và Destructor

edaSearchWrapper::edaSearchWrapper (edaBuffer & buf)

Khởi tạo đối tượng với bộ đệm

Các tham số:

buf	Bộ đệm

Thông tin về hàm thành viên

virtual void edaSearchWrapper::search (edaBuffer & buf_in, edaBuffer & buf_out) `[pure virtual]`

Thực thi tìm kiếm

Các tham số:

buf_in	Bộ đệm đầu vào
buf_out	Bộ đệm đầu ra

Được thực hiện trong **edaSeqSearchWrapper** (*tr.153*).

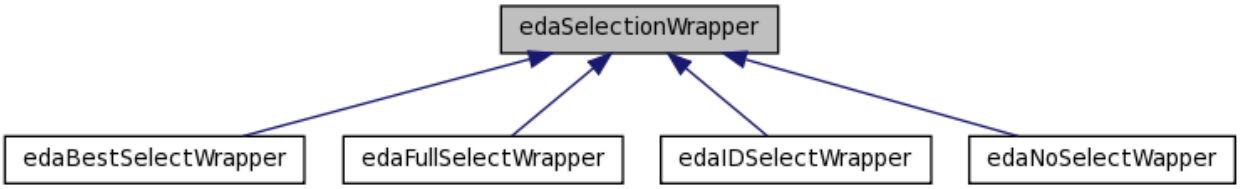
Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaSearchWrapper.h
- lib/edaSearchWrapper.cpp

edaSelectionWrapper Class Tham chiếu

Lớp trừu tượng **edaSelectionWrapper** thực hiện việc chọn lời giải với các chiến lược hổ trợ trong thư viện hoặc do người dùng xây dựng.

```
#include <edaSelectionWrapper.h>
```
Sơ đồ kế thừa cho edaSelectionWrapper:



Các hàm thành viên Public

- **edaSelectionWrapper** ()
  Khởi tạo đối tượng.
- **edaSelectionWrapper** (const **edaSelectionWrapper** &slect)
- virtual **~edaSelectionWrapper** ()
  Hủy đối tượng.
- virtual **edaSelectionWrapper** * **clone** () const =0
- virtual bool **select** (**edaSolutionList** &list) const =0

Mô tả chi tiết

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.

Thông tin về Constructor và Destructor

edaSelectionWrapper::edaSelectionWrapper (const edaSelectionWrapper & slect) `[inline]`

Khởi tạo đối tượng

Các tham số:

slect	Đối tượng cần sao chép

Thông tin về hàm thành viên

virtual edaSelectionWrapper edaSelectionWrapper::clone () const* `[pure virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Được thực hiện trong **edaFullSelectWrapper** (*tr.84*), **edaBestSelectWrapper** (*tr.64*), **edaIDSelectWrapper** (*tr.96*), và **edaNoSelectWapper** (*tr.121*).

virtual bool edaSelectionWrapper::select (edaSolutionList & list) const `[pure virtual]`

Chọn lựa các lời giải với chiến lược chọn tương ứng trong tập lời giải

Giá trị trả về:

Nhận giá trị TRUE nếu chọn lựa thành công, ngược lại giá trị đầu ra là FALSE

Được thực hiện trong **edaBestSelectWrapper** (*tr.64*), **edaFullSelectWrapper** (*tr.84*), **edaIDSelectWrapper** (*tr.96*), và **edaNoSelectWapper** (*tr.122*).

Thông tin cho class được biên soạn từ các file sau đây:

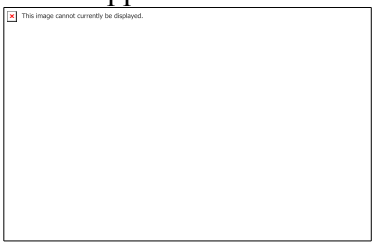
- lib/edaSelectionWrapper.h

edaSeqSearchWrapper Class Tham chiếu

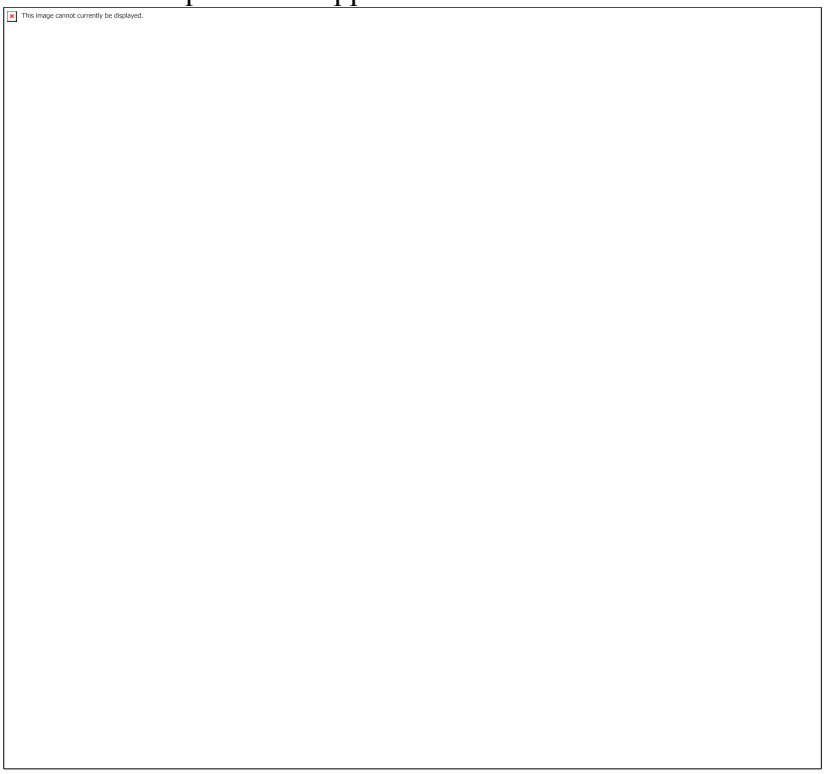
Lớp trừu tượng **edaSeqSearchWrapper** điều khiển quá trình tối ưu bằng cơ chế tuần tự

```
#include <edaSeqSearchWrapper.h>
```

Sơ đồ kế thừa cho edaSeqSearchWrapper:



Sơ đồ liên kết cho edaSeqSearchWrapper:



Các hàm thành viên Public

- **edaSeqSearchWrapper** ()
  Khởi tạo đối tượng.
- **edaSeqSearchWrapper** (**edaBuffer** &buf)
- virtual **~edaSeqSearchWrapper** ()
  Hủy đối tượng.
- void **search** (**edaBuffer** &_buf_in, **edaBuffer** &_buf_out)

Additional Inherited Members

Mô tả chi tiết

Lớp trừu tượng **edaSeqSearchWrapper** điều khiển quá trình tối ưu bằng cơ chế tuần tự

Thông tin về Constructor và Destructor

edaSeqSearchWrapper::edaSeqSearchWrapper (edaBuffer & buf)

Khởi tạo đối tượng

Các tham số:

buf	Bộ đệm

Thông tin về hàm thành viên

void edaSeqSearchWrapper::search (edaBuffer & buf_in, edaBuffer & buf_out) `[virtual]`

Thực thi tìm kiếm

Các tham số:

buf_in	Bộ đệm đầu vào
buf_out	Bộ đệm đầu ra

Thực hiện **edaSearchWrapper** (*tr.150*).

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaSeqSearchWrapper.h
- lib/edaSeqSearchWrapper.cpp

edaSequentialControl Class Tham chiếu

Lớp trừu tượng **edaSequentialControl** điều khiển quá trình thiết lập workflow tuần tự

```
#include <edaSequentialControl.h>
```

Các hàm thành viên Public

- **edaSequentialControl** ()

Khởi tạo đối tượng.

- int **insertVertex** (**edaSearch** *sa)
- int **insertEdge** (const int from, const int to)
- bool **search** (**edaSolutionList** &list)

Mô tả chi tiết

Lớp trừu tượng **edaSequentialControl** điều khiển quá trình thiết lập workflow tuần tự

Thông tin về hàm thành viên

int edaSequentialControl::insertEdge (const int from, const int to)

Đưa cạnh tìm kiếm vào workflow

Các tham số:

from	Đỉnh tìm kiếm bắt đầu của cạnh
to	Đỉnh tìm kiếm kết thúc của cạnh

Giá trị trả về:

ID của cạnh tìm kiếm

*int edaSequentialControl::insertVertex (edaSearch * sa)*

Đưa đỉnh tìm kiếm vào workflow

Các tham số:

sa	Đỉnh tìm kiếm

Giá trị trả về:

ID của đỉnh tìm kiếm

bool edaSequentialControl::search (edaSolutionList & list)

Thực thi việc tối ưu trên trình diều kiển

Các tham số:

list	Danh sách các lời giải cần tối ưu

Thông tin cho class được biên soạn từ các file sau đây:

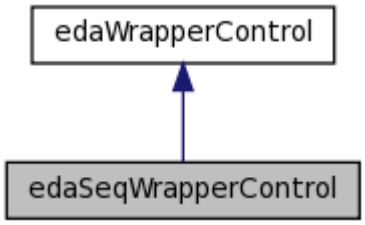
- lib/edaSequentialControl.h
- lib/edaSequentialControl.cpp

edaSeqWrapperControl Class Tham chiếu

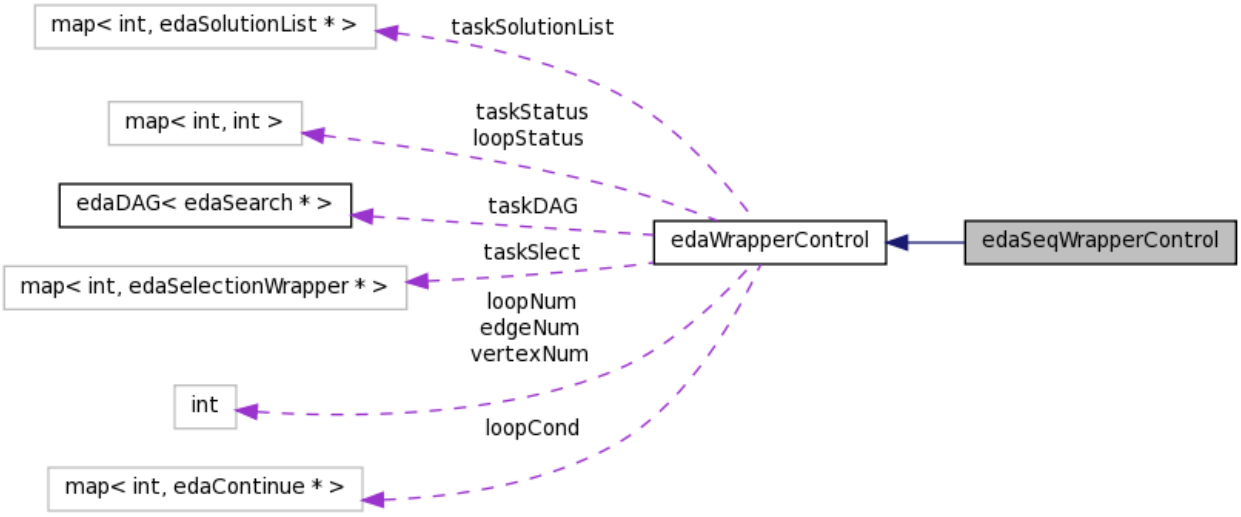
Lớp trừu tượng **edaWrapperControl** điều khiển quá trình tối ưu tuần tự

```
#include <edaSeqWrapperControl.h>
```

Sơ đồ kế thừa cho edaSeqWrapperControl:



Sơ đồ liên kết cho edaSeqWrapperControl:



Các hàm thành viên Public

- **edaSeqWrapperControl** ()
  Khởi tạo đối tượng.
- bool **search** (**edaSolutionList** &list)

Additional Inherited Members

Mô tả chi tiết

Lớp trừu tượng **edaWrapperControl** điều khiển quá trình tối ưu tuần tự

Thông tin về hàm thành viên

bool edaSeqWrapperControl::search (edaSolutionList & list) `[virtual]`

Thực thi việc tối ưu trên trình diều kiển

Các tham số:

list	Danh sách các lời giải cần tối ưu

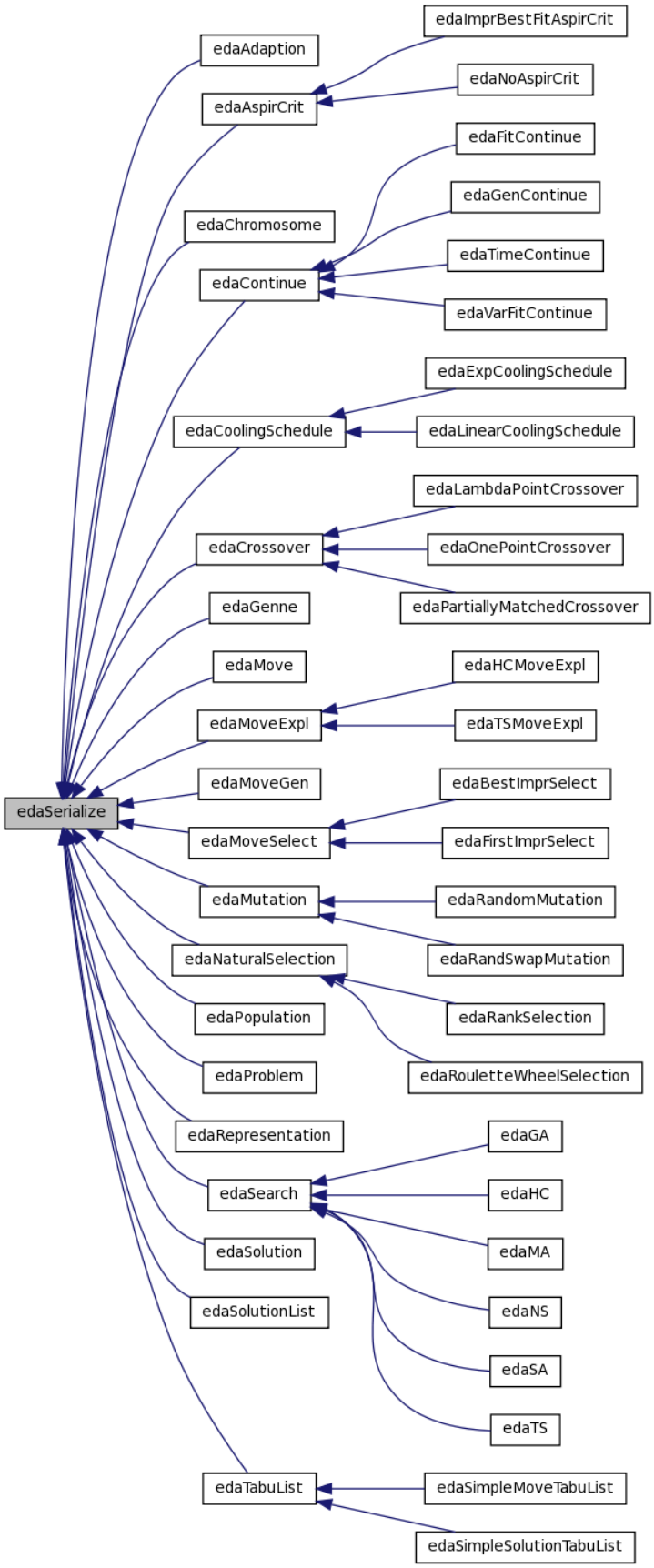
Thực hiện **edaWrapperControl** (*tr.181*).

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaSeqWrapperControl.h
- lib/edaSeqWrapperControl.cpp

edaSerialize Class Tham chiếu

Lớp **edaSerialize** hiện thực cơ chế gói và giải gói cho các đối tượng phức tạp.
`#include <edaSerialize.h>`
Sơ đồ kế thừa cho edaSerialize:


edaImprBestFitAspirCrit
edaAdaption
edaNoAspirCrit
edaAspirCrit
edaFitContinue
edaGenContinue
edaChromosome
edaTimeContinue
edaContinue
edaVarFitContinue
edaExpCoolingSchedule
edaCoolingSchedule
edaLinearCoolingSchedule
edaLambdaPointCrossover
edaCrossover
edaOnePointCrossover
edaPartiallyMatchedCrossover
edaGenne
edaMove
edaHCMoveExpl
edaMoveExpl
edaTSMoveExpl
edaMoveGen
edaBestImprSelect
edaSerialize
edaMoveSelect
edaFirstImprSelect
edaMutation
edaRandomMutation
edaNaturalSelection
edaRandSwapMutation
edaPopulation
edaRankSelection
edaProblem
edaRouletteWheelSelection
edaRepresentation
edaGA
edaSearch
edaHC
edaSolution
edaMA
edaSolutionList
edaNS
edaSA
edaTS
edaTabuList
edaSimpleMoveTabuList
edaSimpleSolutionTabuList

Các hàm thành viên Public

- void **doSerialize** (**edaBuffer** &buf, bool pack=true)
- void * **createObject** (**edaBuffer** &buf)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_INVALID_)

Mô tả chi tiết

Lớp **edaSerialize** hiện thực cơ chế gói và giải gói cho các đối tượng phức tạp.

Thông tin về hàm thành viên

void edaSerialize::createObject (edaBuffer & buf)*

Phục hồi đối đã được đóng gói từ bộ đệm

Các tham số:

buf	Bộ đệm

void edaSerialize::doSerialize (edaBuffer & buf, bool pack = `true`)

Hiện thực việc đóng gói cho đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

void edaSerialize::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

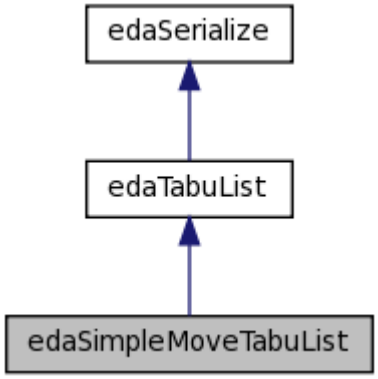
Được thực thi lại trong **edaSolutionList** (*tr.167*), **edaChromosome** (*tr.68*), **edaTS** (*tr.175*), **edaGA** (*tr.87*), **edaHC** (*tr.93*), **edaMA** (*tr.107*), **edaAdaption** (*tr.59*), **edaPopulation** (*tr.130*), **edaSA** (*tr.146*), **edaRepresentation** (*tr.140*), **edaMoveSelect** (*tr.112*), **edaGenne** (*tr.90*), **edaBestImprSelect** (*tr.62*), **edaSearch** (*tr.148*), **edaSolution** (*tr.164*), **edaTSMoveExpl** (*tr.177*), **edaOnePointCrossover** (*tr.126*), **edaPartiallyMatchedCrossover** (*tr.128*), **edaRankSelection** (*tr.138*), **edaRouletteWheelSelection** (*tr.143*), **edaLambdaPointCrossover** (*tr.102*), **edaProblem** (*tr.132*), **edaContinue** (*tr.70*), **edaCrossover** (*tr.73*), **edaLinearCoolingSchedule** (*tr.104*), **edaHCMoveExpl** (*tr.95*), **edaMove** (*tr.108*), **edaNaturalSelection** (*tr.118*), **edaRandomMutation** (*tr.134*), **edaMutation** (*tr.117*), **edaRandSwapMutation** (*tr.136*), **edaSimpleMoveTabuList** (*tr.160*), **edaSimpleSolutionTabuList** (*tr.163*), **edaFirstImprSelect** (*tr.81*), **edaExpCoolingSchedule** (*tr.80*), **edaImprBestFitAspirCrit** (*tr.98*), **edaNoAspirCrit** (*tr.120*), **edaNS** (*tr.124*), **edaTimeContinue** (*tr.171*), **edaGenContinue** (*tr.89*), **edaFitContinue** (*tr.83*), và **edaVarFitContinue** (*tr.179*).

Thông tin cho class được biên soạn từ các file sau đây:

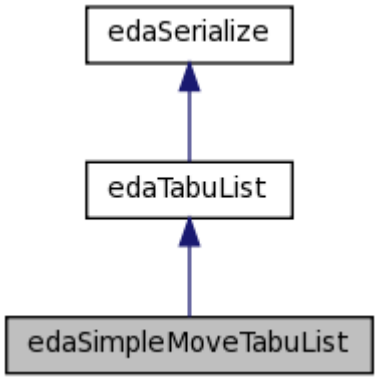
- lib/edaSerialize.h
- lib/edaSerialize.cpp

edaSimpleMoveTabuList Class Tham chiếu

Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo lớp **edaSolution**.
`#include <edaSimpleMoveTabuList.h>`
Sơ đồ kế thừa cho edaSimpleMoveTabuList:



Sơ đồ liên kết cho edaSimpleMoveTabuList:



Các hàm thành viên Public

- **edaSimpleMoveTabuList** ()
  Khởi tạo danh sách Tabu.
- **edaSimpleMoveTabuList** (unsigned int maxSize)
- **edaSimpleMoveTabuList** (const **edaSimpleMoveTabuList** &tl)
- **~edaSimpleMoveTabuList** ()
  Hũy đối tượng.
- **edaTabuList** * **clone** () const
  Nhân bản đối tượng.
- void **init** ()
  Khởi động danh sách Tabu.
- void **add** (const **edaMove** *_move, const **edaSolution** *_sol)
- void **update** ()
  Cập nhật danh sách Tabu.
- bool **check** (const **edaMove** *_move, const **edaSolution** *_sol)
- void **Serialize** (**edaBuffer** &buf, bool pack)

- **setClassID** (_SYSCLASSID_+_CLSID_EDASIMPLEMOVETABULIST_)

Mô tả chi tiết

Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo lớp **edaSolution**.

Thông tin về Constructor và Destructor

edaSimpleMoveTabuList::edaSimpleMoveTabuList (unsigned int maxSize)

Khởi tạo danh sách Tabu

Các tham số:

maxSize	Số lượng phần tử tối đa trong danh sách

edaSimpleMoveTabuList::edaSimpleMoveTabuList (const edaSimpleMoveTabuList & tl)

Khởi tạo danh sách Tabu

Các tham số:

tl	Đối tượng cần sao chép

Thông tin về hàm thành viên

*void edaSimpleMoveTabuList::add (const edaMove * move, const edaSolution * sol)* `[virtual]`

Đưa bước chuyển vào danh sách Tabu

Các tham số:

move	Bước chuyển
sol	lời giải hiện tại

Thực hiện **edaTabuList** (*tr.169*).

*bool edaSimpleMoveTabuList::check (const edaMove * move, const edaSolution * sol)* `[virtual]`

Kiểm tra bước chuyển có thuộc danh sách Tabu hay không

Các tham số:

move	Bước chuyển
Lời	giải

Giá trị trả về:

TRUE: nếu thuộc, FALSE: nếu không thuộc

Thực hiện **edaTabuList** (*tr.169*).

void edaSimpleMoveTabuList::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

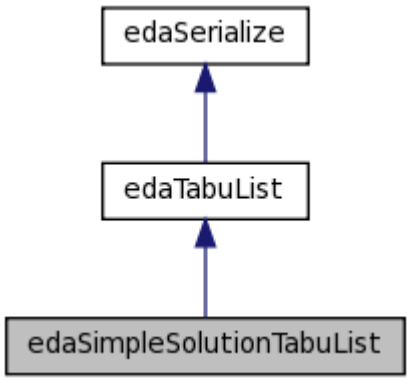
- lib/edaSimpleMoveTabuList.h
- lib/edaSimpleMoveTabuList.cpp

edaSimpleSolutionTabuList Class Tham chiếu

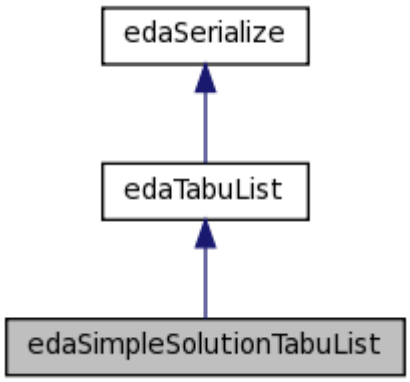
Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo lớp **edaMove**.

```
#include <edaSimpleSolutionTabuList.h>
```

Sơ đồ kế thừa cho edaSimpleSolutionTabuList:



Sơ đồ liên kết cho edaSimpleSolutionTabuList:



Các hàm thành viên Public

- **edaSimpleSolutionTabuList** ()
  Khởi tạo danh sách Tabu.
- **edaSimpleSolutionTabuList** (unsigned int maxSize)
- **edaSimpleSolutionTabuList** (const **edaSimpleSolutionTabuList** &tl)
- virtual **~edaSimpleSolutionTabuList** ()

Hũy đối tượng.

- **edaTabuList** * **clone** () const
  Nhân bản đối tượng.
- void **init** ()
  Khởi động danh sách Tabu.
- void **add** (const **edaMove** *_move, const **edaSolution** *_sol)
- void **update** ()
  Cập nhật danh sách Tabu.
- bool **check** (const **edaMove** *_move, const **edaSolution** *_sol)
- void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDASIMPLESOLUTIONTABULIST_)

Mô tả chi tiết

Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo lớp **edaMove**.

Thông tin về Constructor và Destructor

edaSimpleSolutionTabuList::edaSimpleSolutionTabuList (unsigned int maxSize)

Khởi tạo danh sách Tabu

Các tham số:

maxSize	Số lượng phần tử tối đa trong danh sách

edaSimpleSolutionTabuList::edaSimpleSolutionTabuList (const edaSimpleSolutionTabuList & tl)

Khởi tạo danh sách Tabu

Các tham số:

tl	Đối tượng cần sao chép

Thông tin về hàm thành viên

*void edaSimpleSolutionTabuList::add (const edaMove * move, const edaSolution * sol)* `[virtual]`

Đưa bước chuyển vào danh sách Tabu

Các tham số:

move	Bước chuyển
sol	lời giải hiện tại

Thực hiện **edaTabuList** (*tr.169*).

*bool edaSimpleSolutionTabuList::check (const edaMove * move, const edaSolution * sol)* `[virtual]`

Kiểm tra bước chuyển có thuộc danh sách Tabu hay không

Các tham số:

move	Bước chuyển
Lời	giải

Giá trị trả về:

TRUE: nếu thuộc, FALSE: nếu không thuộc

Thực hiện **edaTabuList** (*tr.169*).

void edaSimpleSolutionTabuList::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

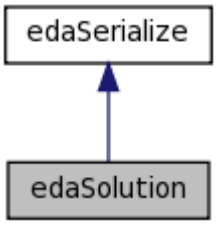
Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaSimpleSolutionTabuList.h
- lib/edaSimpleSolutionTabuList.cppedaSolution Class Tham chiếu

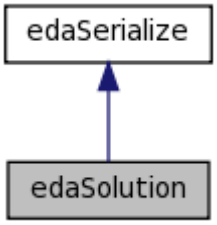
Lớp trừu tượng **edaSolution** hiện thực lời giải của bài toán.

```
#include <edaSolution.h>
```

Sơ đồ kế thừa cho edaSolution:



Sơ đồ liên kết cho edaSolution:



Các hàm thành viên Public

- **edaSolution** ()
  Khởi tạo đối tượng.
- **edaSolution** (const **edaSolution** &sol)
  Khởi tạo đối tượng.
- **edaSolution** (const **edaProblem** &pro)

Khởi tạo đối tượng.

- virtual **~edaSolution** ()
  Hủy đối tượng.
- virtual **edaSolution** * **clone** () const =0
- virtual void **init** ()
  Khởi động đối tượng.
- 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

Mô tả chi tiết

Lớp trừu tượng **edaSolution** hiện thực lời giải của bài toán.

Thông tin về hàm thành viên

virtual edaSolution edaSolution::clone () const* `[pure virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng cần nhân bản

virtual double edaSolution::evaluate () `[pure virtual]`

Tính hàm lượng giá của lời giải

Các tham số:

\return	Giá trị hàm lượng giá của lời giải

virtual double edaSolution::getCost () const `[pure virtual]`

Lấy giá trị lượng giá (đã tính) của lời giải

Các tham số:

\return	Giá trị hàm lượng giá của lời giải

virtual void edaSolution::Serialize (edaBuffer & buf, bool pack) `[pure virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

virtual void edaSolution::setCost (double value) `[inline], [virtual]`

Thiết lập giá trị lượng giá cho lời giải

Các tham số:

value	Giá trị lượng giá

Thông tin cho class được biên soạn từ các file sau đây:

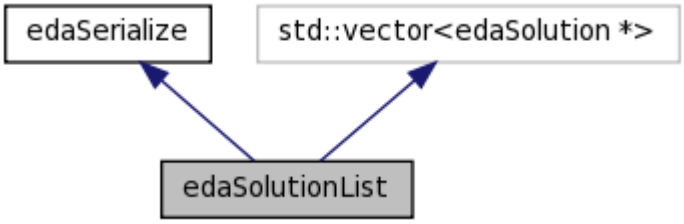
- lib/edaSolution.h
- lib/edaSolution.cpp

edaSolutionList Class Tham chiếu

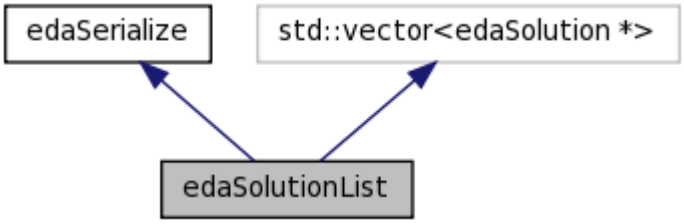
Lớp trừu tượng **edaSolutionList** hiện thực danh sách lời giải của bài toán.

```
#include <edaSolutionList.h>
```

Sơ đồ kế thừa cho edaSolutionList:



Sơ đồ liên kết cho edaSolutionList:



Các hàm thành viên Public

- **edaSolutionList** ()
  Khởi tạo đối tượng.
- **edaSolutionList** (const **edaSolutionList** &orig)
  Khởi tạo đối tượng.
- virtual **~edaSolutionList** ()
  Hũy đối tượng.
- **edaSolutionList** * **clone** () const
- double **evaluate** () const
- double **mean** () const
- double **std** () const
- double **min** () const
- double **max** () const
- void **sort** ()
  Sắp xếp các cá thể trong tập lời giải theo thứ tự hàm lượng giá
- bool **replace** (**edaSolutionList** &list)
- **edaSolutionList getList** (unsigned int num)
- void **printOn** (ostream &os) const
- **edaSolution** * **getBest** () const
- unsigned int **getBestID** () const

- virtual **edaSolutionList** & **operator=** (const **edaSolutionList** &sol)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDASOLUTIONLIST_)
- void **easer** ()
  Xóa thông tin chứa trong đối tượngt.

Mô tả chi tiết

Lớp trừu tượng **edaSolutionList** hiện thực danh sách lời giải của bài toán.

Thông tin về hàm thành viên

*edaSolutionList * edaSolutionList::clone () const*

Nhân bản đối tượng

Giá trị trả về:

Đối tượng cần nhân bản

double edaSolutionList::evaluate () const

Tính hàm lượng giá của lời giải

Các tham số:

\return	Giá trị hàm lượng giá của lời giải

*edaSolution * edaSolutionList::getBest () const*

Lấy thông tin lời giải tốt nhất

Giá trị trả về:

edaSolution lời giải với giá trị lượng giá tốt nhất

unsigned int edaSolutionList::getBestID () const

Lấy thông tin ID lời giải tốt nhất

Giá trị trả về:

ID của lời giải với giá trị lượng giá tốt nhất

edaSolutionList edaSolutionList::getList (unsigned int num)

Lấy danh sách lời giải con với số lượng cho trước

Các tham số:

num	Số lời giải cần lấy

Giá trị trả về:

edaSolutionList Danh sách (tập) lời giải con

double edaSolutionList::max () const

Lấy giá trị lượng giá cao nhất của tập lời giải

Giá trị trả về:

Giá trị lượng giá cao nhất

double edaSolutionList::mean () const

Lấy giá trị lượng giá trung bình của tập lời giải

Giá trị trả về:

Giá trị lượng giá trung bình

double edaSolutionList::min () const

Lấy giá trị lượng giá thấp nhất của tập lời giải

Giá trị trả về:

Giá trị lượng giá thấp nhất

bool edaSolutionList::replace (edaSolutionList & list)

Thay thế (gán giá trị) cho tập lời giải

Các tham số:

list	Tập lời giả cần thay thế

void edaSolutionList::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

double edaSolutionList::std () const

Lấy độ lệch chuẩn của lượng giá của tập lời giải

Giá trị trả về:

Giá trị độ lệch chuẩn

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaSolutionList.h
- lib/edaSolutionList.cpp

edaString Class Tham chiếu

Lớp **edaString** hiện thực việc hổ trợ xử lý chuỗi trong thư viện.

```
#include <edaString.h>
```

Các hàm thành viên Public

- **edaString** ()
  Khởi tạo đối tượng.
- **edaString** (const **edaString** &orig)
- virtual **~edaString** ()
  Hủy đối tượng.

Mô tả chi tiết

Lớp **edaString** hiện thực việc hỗ trợ xử lý chuỗi trong thư viện.

Thông tin về Constructor và Destructor

edaString::edaString (const edaString & orig)

Khởi tạo đối tượng

Các tham số:

orig	Đối tượng gốc cần sao chép

Thông tin cho class được biên soạn từ các file sau đây:

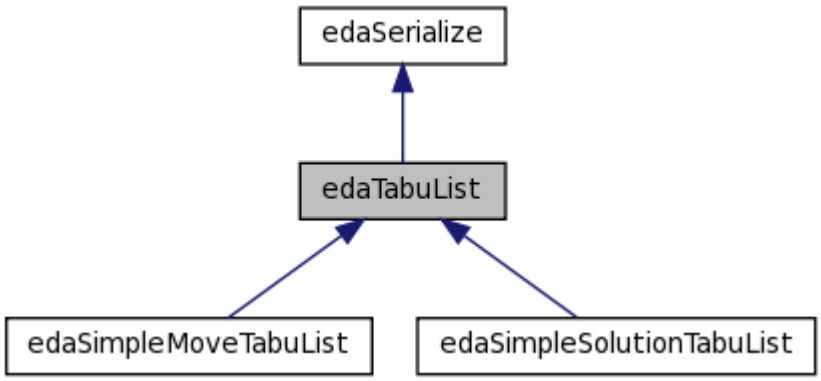
- lib/edaString.h
- lib/edaString.cpp

edaTabuList Class Tham chiếu

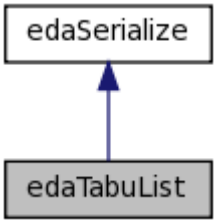
Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo các đối tượng khác nhau.

```
#include <edaTabuList.h>
```

Sơ đồ kế thừa cho edaTabuList:



Sơ đồ liên kết cho edaTabuList:



Các hàm thành viên Public

- **edaTabuList** ()

Khởi tạo danh sách Tabu.

- virtual **~edaTabuList** ()
  Hũy danh sách Tabu.
- virtual **edaTabuList** * **clone** () const =0
  Nhân bản đối tượng.
- virtual void **add** (const **edaMove** *move, const **edaSolution** *sol)=0
- virtual void **update** ()=0
  Cập nhật danh sách Tabu.
- virtual void **init** ()=0
  Khởi động danh sách Tabu.
- virtual bool **check** (const **edaMove** *move, const **edaSolution** *sol)=0
- **setClassID** (_SYSCLASSID_+_CLSID_EDATABULIST_)

Mô tả chi tiết

Lớp trừu tượng **edaTabuList** hiện thực danh sách Tabu theo các đối tượng khác nhau.

Thông tin về hàm thành viên

*virtual void edaTabuList::add (const edaMove * move, const edaSolution * sol)* `[pure virtual]`

Đưa bước chuyển vào danh sách Tabu

Các tham số:

move	Bước chuyển
sol	lời giải hiện tại

Được thực hiện trong **edaSimpleMoveTabuList** (*tr.160*), và **edaSimpleSolutionTabuList** (*tr.162*).

*virtual bool edaTabuList::check (const edaMove * move, const edaSolution * sol)* `[pure virtual]`

Kiểm tra bước chuyển có thuộc danh sách Tabu hay không

Các tham số:

move	Bước chuyển
Lời	giải

Giá trị trả về:

TRUE: nếu thuộc, FALSE: nếu không thuộc

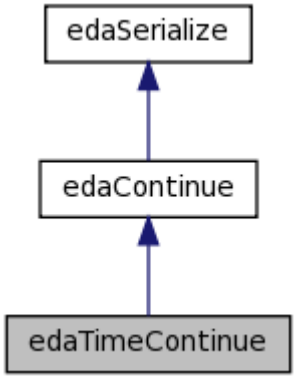
Được thực hiện trong **edaSimpleMoveTabuList** (*tr.160*), và **edaSimpleSolutionTabuList** (*tr.162*).

Thông tin cho class được biên soạn từ các file sau đây:

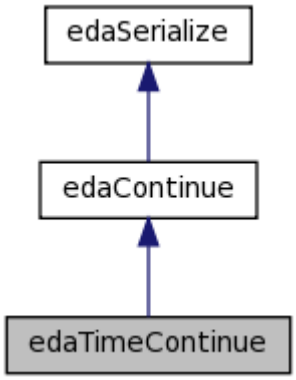
- lib/edaTabuList.h

edaTimeContinue Class Tham chiếu

Lớp **edaTimeContinue** hiện thực điều kiện dừng theo theo thời gian chạy.
`#include <edaTimeContinue.h>`
Sơ đồ kế thừa cho edaTimeContinue:



Sơ đồ liên kết cho edaTimeContinue:



Các hàm thành viên Public

- **edaTimeContinue** ()
  Khởi tạo đối tượng.
- **edaTimeContinue** (unsigned int maxTime, unsigned int startTime=0)
- virtual **~edaTimeContinue** ()
  Hủy đối tượng.
- virtual **edaContinue** * **clone** () const
  Nhân bản đối tượng.
- void **init** ()
  Khởi động đối tượng.
- bool **check** (const **edaSolutionList** &list)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_TIME_CONTINUE_)

Mô tả chi tiết

Lớp **edaTimeContinue** hiện thực điều kiện dừng theo theo thời gian chạy.

Thông tin về Constructor và Destructor

edaTimeContinue::edaTimeContinue (unsigned int maxTime, unsigned int startTime = 0)

Khởi tạo đối tượng

Các tham số:

maxTime	Thời gian giới hạn cho việc lặp
startTime	Thời gian bắt đầu (mặc định startTime = 0)

Thông tin về hàm thành viên

bool edaTimeContinue::check (const edaSolutionList & list) `[virtual]`

Kiểm tra điều kiện dừng

Các tham số:

list	Danh sách các lời giải cần kiểm tra với điều kiện dừng

Thực hiện **edaContinue** (*tr.70*).

void edaTimeContinue::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hỗ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaContinue** (*tr.70*).

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaTimeContinue.h
- lib/edaTimeContinue.cpp

edaTimer Class Tham chiếu

Lớp **edaTimer** hiện thực việc hỗ trợ tính toán thời gian trong thư viện.

```
#include <edaTimer.h>
```

Các hàm thành viên Public

- **edaTimer** ()
  Khởi tạo đối tượng.
- void **start** (bool reset=false)

- double **stop** ()
- double **restart** ()
- double **elapse** () const
- double **duration** () const

Mô tả chi tiết

Lớp **edaTimer** hiện thực việc hổ trợ tính toán thời gian trong thư viện.

Thông tin về hàm thành viên

double edaTimer::duration () const

Lấy thời khoảng hiện tại

Giá trị trả về:

Thời khoảng hiện tại

double edaTimer::elapse () const

Lấy thời điểm hiện tại

Giá trị trả về:

Thời gian hiện tại

double edaTimer::restart ()

Start lại quá trình tính toán thời gian

Giá trị trả về:

Trả lại thời điểm cuối cùng khi bắt đầu restart lại

void edaTimer::start (bool reset = `false`)

Bắt đầu do thời gian chạy

Các tham số:

reset	Có start lại từ đầu thời điểm

double edaTimer::stop ()

Dừng tính toán thời gian

Giá trị trả về:

Trả lại thời điểm cuối cùng khi bắt đầu dừng

Thông tin cho class được biên soạn từ các file sau đây:

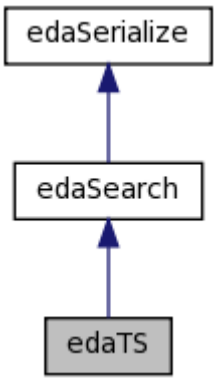
- lib/edaTimer.h
- lib/edaTimer.cpp

edaTS Class Tham chiếu

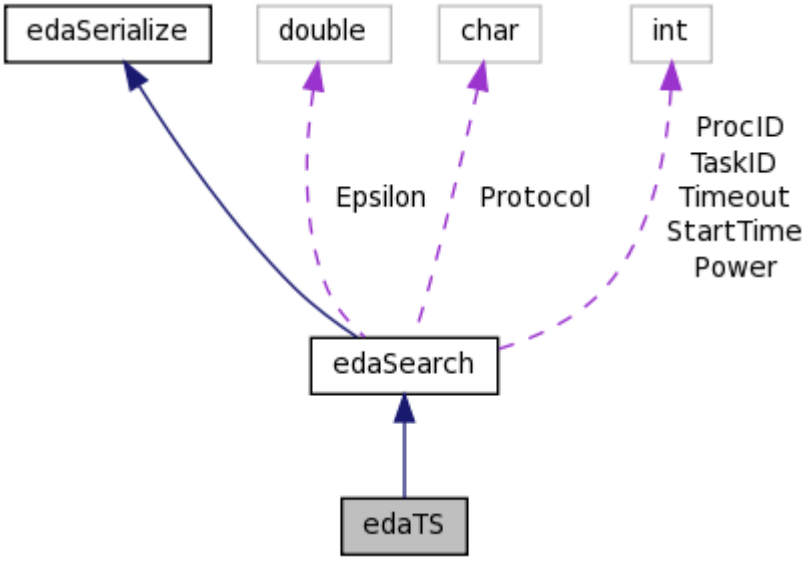
Lớp trừu tượng **edaTS** thực hiện việc tối ưu theo giải thuật tìm kiếm Tabu.

```
#include <edaTS.h>
```

Sơ đồ kế thừa cho edaTS:



Sơ đồ liên kết cho edaTS:



Các hàm thành viên Public

- **edaTS** ()
  Khởi tạo đối tượng.
- **edaTS** (int power)
- **edaTS** (**edaMove** *move, **edaMoveGen** *moveNext, **edaTabuList** *tabuList, **edaAspirCrit** *aspirCrit, **edaContinue** *continueCriteria, int timeout=0, int power=0)
- **edaTS** (**edaMove** *move, **edaTSMoveExpl** *moveExpl, **edaContinue** *continueCriteria, int timeout=0, int power=0)
- **edaTS** (const **edaTS** &ts)
- **edaTS** * **clone** () const
- **~edaTS** ()
  Hủy đối tượng.
- bool **search** (**edaSolutionList** &list)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDATS_)

Additional Inherited Members

Mô tả chi tiết

Lớp trừu tượng **edaTS** thực hiện việc tối ưu theo giải thuật tìm kiếm Tabu.

Thông tin về Constructor và Destructor

edaTS::edaTS (int power)

Khởi tạo đối tượng

Các tham số:

power	Bậc tìm kiếm

*edaTS::edaTS (edaMove * move, edaMoveGen * moveNext, edaTabuList * tabuList, edaAspirCrit * aspirCrit, edaContinue * continueCriteria, int timeout = 0, int power = 0)*

Khởi tạo đối tượng

Các tham số:

move	Chiến lược bước chuyển trong không gian tìm kiếm
moveNext	Phương pháp tạo bước chuyển trong không gian tìm kiếm
aspirCrit	Chiến lược kiểm tra sự hợp lệ của các bước chuyển ứng viên
continueCriteria	Điều kiện dừng
timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

*edaTS::edaTS (edaMove * move, edaTSMoveExpl * moveExpl, edaContinue * continueCriteria, int timeout = 0, int power = 0)*

Khởi tạo đối tượng

Các tham số:

move	Chiến lược bước chuyển trong không gian tìm kiếm
edaTSMoveExpl	Chiến lược khai phá không gian tìm kiếm cho giải thuật TS
continueCriteria	Điều kiện dừng
timeout	Thời gian thực thi tìm kiếm
power	Bậc tìm kiếm

edaTS::edaTS (const edaTS & ts)

Khởi tạo đối tượng

Các tham số:

ma	Đối tượng search cần sao chép

Thông tin về hàm thành viên

*edaTS * edaTS::clone () const* `[virtual]`

Nhân bản đối tượng

Giá trị trả về:

Đối tượng được nhân bản

Thực hiện **edaSearch** (*tr.148*).

bool edaTS::search (edaSolutionList & list) `[virtual]`

Thực thi tối ưu

Các tham số:

list	Danh sách lời giải cần tối ưu

Giá trị trả về:

TRUE: tối ưu thành công, FALSE: tối ưu thất bại

Thực hiện **edaSearch** (*tr.148*).

void edaTS::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gói (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSearch** (*tr.148*).

Thông tin cho class được biên soạn từ các file sau đây:

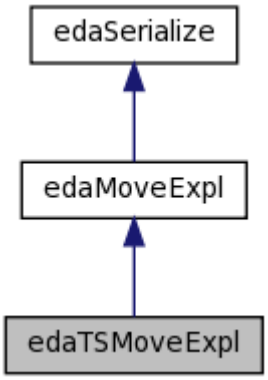
- lib/edaTS.h
- lib/edaTS.cpp

edaTSMoveExpl Class Tham chiếu

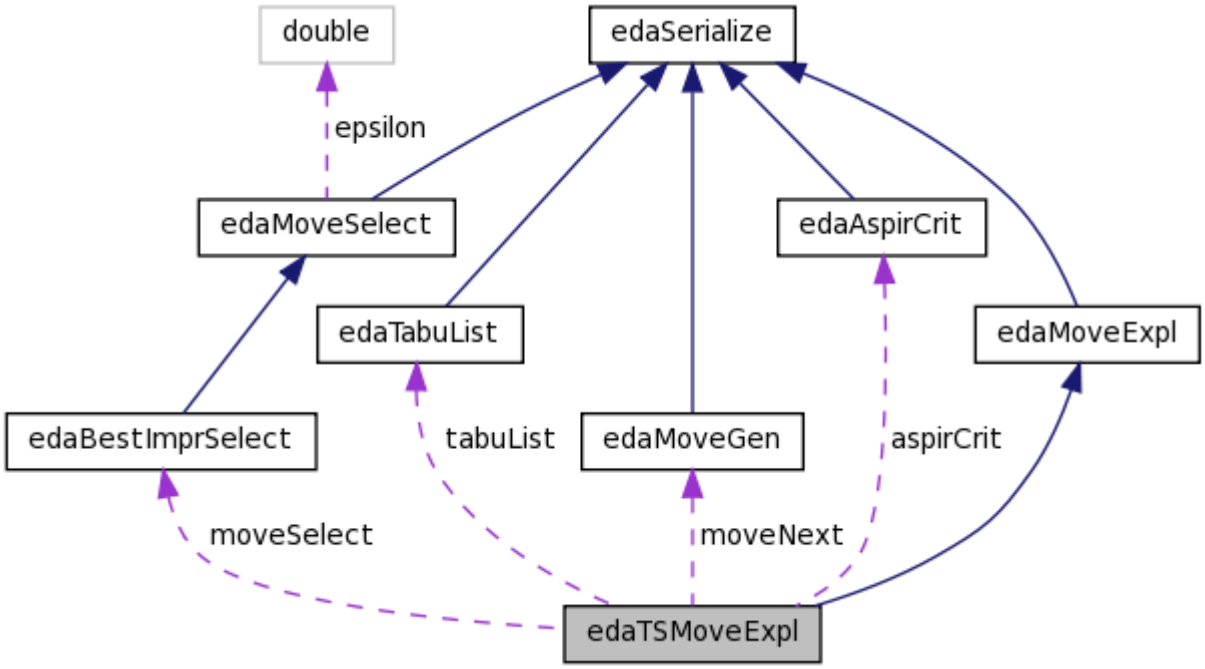
Lớp **edaTSMoveExpl** hiện thực chiến lược khai phá không gian ứng với giải thuật TS.

`#include <edaTSMoveExpl.h>`

Sơ đồ kế thừa cho edaTSMoveExpl:



Sơ đồ liên kết cho edaTSMoveExpl:



Các hàm thành viên Public

- **edaTSMoveExpl** ()
  Khởi tạo đối tượng.
- **edaTSMoveExpl** (const **edaMoveGen** *moveNext, const **edaTabuList** *tabuList, const **edaAspirCrit** *aspirCrit)
- **edaTSMoveExpl** (const **edaTSMoveExpl** &moveExpl)
- virtual **~edaTSMoveExpl** ()
  Hủy đối tượng.
- virtual **edaTSMoveExpl** * **clone** () const
  Nhân bản đối tượng.
- virtual void **explore** (const **edaMove** *move, **edaSolution** &oldSolution, **edaSolution** &newSolution)
- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_EDATSMOVEEXPL_)

Các hàm thành viên Protected

- void **clean** ()

các thuộc tính Protected

- **edaMoveGen * moveNext**
- **edaTabuList * tabuList**
- **edaAspirCrit * aspirCrit**
- **edaBestImprSelect * moveSelect**

Mô tả chi tiết

Lớp **edaTSMoveExpl** hiện thực chiến lược khai phá không gian ứng với giải thuật TS.

Thông tin về Constructor và Destructor

*edaTSMoveExpl::edaTSMoveExpl (const edaMoveGen * moveNext, const edaTabuList * tabuList, const edaAspirCrit * aspirCrit)*

Khởi tạo đối tượng

Các tham số:

moveNext	Chiến lược tạo bước chuyển trong không gian tìm kiếm
tabuList	Danh sách Tabu
aspirCrit	Chiến lược chọn bước chuyển trong không gian tìm kiếm

edaTSMoveExpl::edaTSMoveExpl (const edaTSMoveExpl & moveExpl)

Khởi tạo đối tượng

Các tham số:

moveExpl	Đối tượng cần sao chép

Thông tin về hàm thành viên

*void edaTSMoveExpl::explore (const edaMove * move, edaSolution & oldSolution, edaSolution & newSolution)* `[virtual]`

Khai phá không gian tìm kiếm

Các tham số:

move	Chiến lược di chuyển trong không gian
oldSolution	Lời giải trước khi áp dụng
newSolution	Lời giải sau khi áp dụng

Thực hiện **edaMoveExpl** (*tr.110*).

void edaTSMoveExpl::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaSerialize** (*tr.158*).

Thông tin cho class được biên soạn từ các file sau đây:

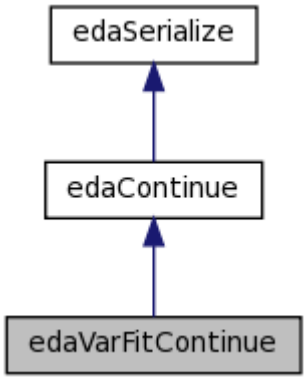
- lib/edaTSMoveExpl.h
- lib/edaTSMoveExpl.cpp

edaVarFitContinue Class Tham chiếu

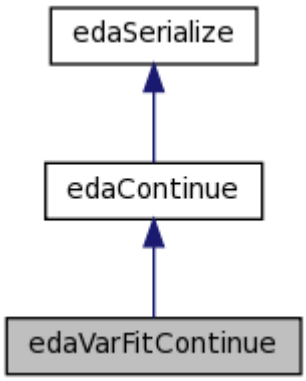
Lớp **edaVarFitContinue** hiện thực điều kiện dừng theo theo hàm lượng giá có khả năng thay đổi.

`#include <edaVarFitContinue.h>`

Sơ đồ kế thừa cho edaVarFitContinue:



Sơ đồ liên kết cho edaVarFitContinue:



Các hàm thành viên Public

- **edaVarFitContinue** (unsigned int num_loop=1)
  Khởi tạo đối tượng.
- **~edaVarFitContinue** ()
  Hủy đối tượng.
- virtual **edaContinue** * **clone** () const
  Nhân bản đối tượng.
- void **init** ()
  Khởi động đối tượng.
- bool **check** (const **edaSolutionList** &list)

- virtual void **Serialize** (**edaBuffer** &buf, bool pack)
- **setClassID** (_SYSCLASSID_+_CLSID_VAR_FIT_CONTINUE_)

Mô tả chi tiết

Lớp **edaVarFitContinue** hiện thực điều kiện dừng theo theo hàm lượng giá có khả năng thay đổi.

Thông tin về hàm thành viên

bool edaVarFitContinue::check (const edaSolutionList & list) `[virtual]`

Kiểm tra điều kiện dừng

Các tham số:

list	Danh sách các lời giải cần kiểm tra với điều kiện dừng

Thực hiện **edaContinue** (*tr.70*).

void edaVarFitContinue::Serialize (edaBuffer & buf, bool pack) `[virtual]`

Hiện thực việc đóng gói và mở gói các thông tin của đối tượng

Các tham số:

buf	Bộ đệm có hổ trợ việc đóng gói và nhận các thông tin của đối tượng
pack	Cờ hiệu: đóng gới (pack = 1), và giải gói (pack = 0))

Được thực thi lại từ **edaContinue** (*tr.70*).

Thông tin cho class được biên soạn từ các file sau đây:

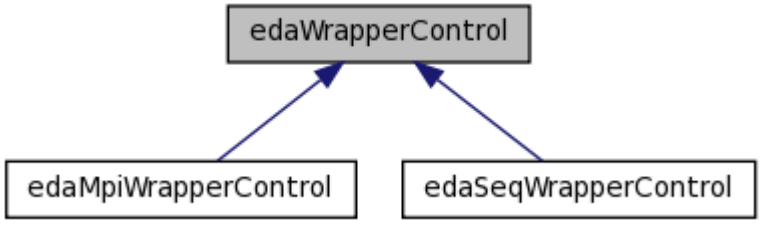
- lib/edaVarFitContinue.h
- lib/edaVarFitContinue.cpp

edaWrapperControl Class Tham chiếu

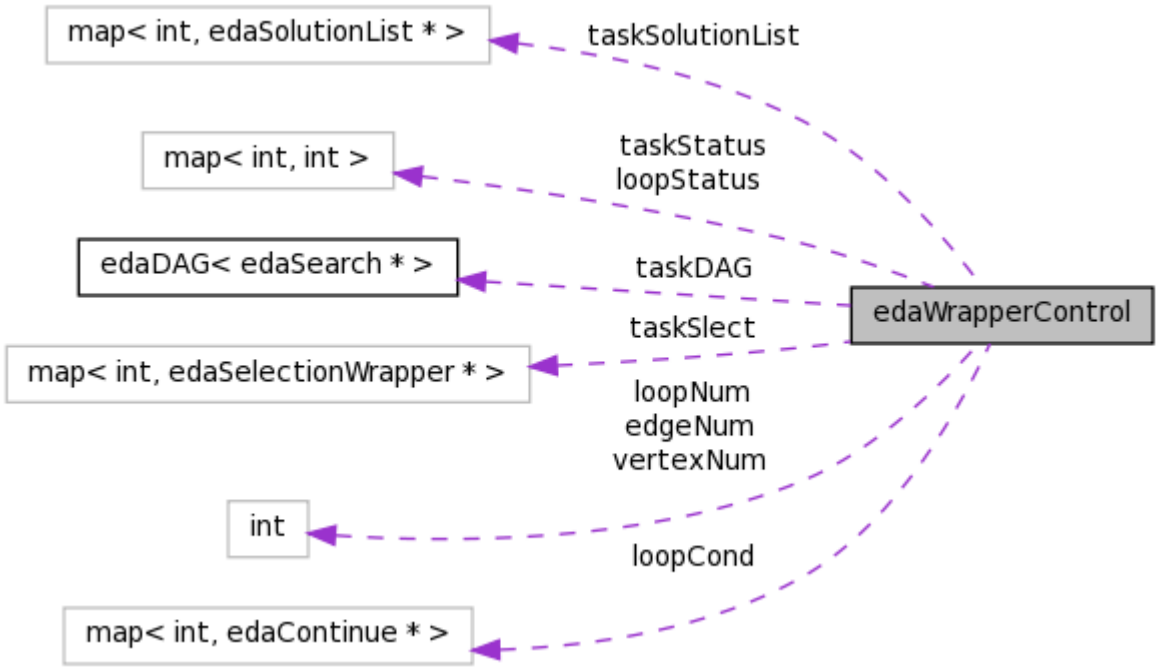
Lớp trừu tượng **edaWrapperControl** điều khiển quá trình tối ưu.

```
#include <edaWrapperControl.h>
```

Sơ đồ kế thừa cho edaWrapperControl:



Sơ đồ liên kết cho edaWrapperControl:



Các hàm thành viên Public

- **edaWrapperControl** ()
  Khởi tạo đối tượng.
- virtual **~edaWrapperControl** ()
  Hủy dối tượng.
- 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

Các hàm thành viên 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** ()

các thuộc tính Protected

- 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**

Mô tả chi tiết

Lớp trừu tượng **edaWrapperControl** điều khiển quá trình tối ưu.

Thông tin về hàm thành viên

int edaWrapperControl::insertEdge (const int from, const int to) `[virtual]`

Thêm cạnh mới vào trình diều khiển

Các tham số:

from	Đỉnh đầu
to	Đỉnh cuối

int edaWrapperControl::insertLoop (const int from, const int to, const edaContinue & con) `[virtual]`

Thêm vòng lặp mới vào trình diều khiển

Các tham số:

from	Đỉnh bắt đầu lặp
to	Đỉnh kết thúc vòng lặp
con	Điều kiện lặp

*int edaWrapperControl::insertVertex (edaSearch * sa)* `[virtual]`

Thêm node tìm kiếm mới vào trình điều khiển

Các tham số:

sa	Node tìm kiếm mới

*int edaWrapperControl::insertVertex (edaSearch * sa, const edaSelectionWrapper & slect)* `[virtual]`

Thêm node tìm kiếm mới vào trình diều khiển với điều kiên chọn

Các tham số:

sa	Node tìm kiếm mới
slect	Chiến lược chọn lựa lời giải cho quá trình tối ưu

virtual bool edaWrapperControl::search (edaSolutionList & list) `[pure virtual]`

Thực thi việc tối ưu trên trình diều kiển

Các tham số:

list	Danh sách các lời giải cần tối ưu

Được thực hiện trong **edaMpiWrapperControl** (*tr.115*), và **edaSeqWrapperControl** (*tr.155*).

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaWrapperControl.h

- lib/edaWrapperControl.cpp

edaWriter Class Tham chiếu

Lớp **edaWriter** hiện thực việc hổ trợ đọc và ghi file.

```
#include <edaWriter.h>
```

Các hàm thành viên Public

- **edaWriter** ()
  Khởi tạo đối tượng.
- **edaWriter** (const **edaWriter** &orig)
- virtual **~edaWriter** ()
  Hủy đối tượng.

Mô tả chi tiết

Lớp **edaWriter** hiện thực việc hổ trợ đọc và ghi file.

Thông tin về Constructor và Destructor

edaWriter::edaWriter (const edaWriter & orig)

Khởi tạo đối tượng

Các tham số:

orig	Đối tượng gốc cần sao chép

Thông tin cho class được biên soạn từ các file sau đây:

- lib/edaWriter.h
- lib/edaWriter.cpp

TÀI LIỆU THAM KHẢO

[David06] David L. Applegate, Robert E. Bixby, Vasek Chvátal & William J. Cook, "*The Traveling Salesman Problem: A Computational Study*", Princeton University Press, 2006, ISBN: 978-1-400-84110-3.

[Donald10] Donald Davendra, "*Traveling Salesman Problem, Theory and Applications*", Published by InTech, 2010, ISBN 978-953-307-426-9.

[El09] El-Ghazali Talbi, "*Metaheuristics From Design To Implementation*", John Wiley & Sons, 2009, ISBN 978-0-470-27858-1.

[Fred03] Fred Glover, Gary A. Kochenberger, "*Handbook of Metaheuristics"*, Kluwer Academic Publishers, 2003, ISBN 1-4020-7263-5.

[Hoai14] Trần Văn Hoài, "*Báo Cáo Tổng Kết Xây Dựng Thư Viện Lập Trình Hỗ Trợ Tối Ưu Tổ Hợp trên Môi Trường Tính Toán Song Song và Phân Bố*", Đại Học Bách Khoa Tp.HCM, 2014.

[Michel10] Michel Gendreau, Jean-Yves Potvin, "*International Series in Operations Research & Management Science Volume 146: Handbook of Metaheuristics*", Springer, 2010, ISBN 978-1-4419-1663-1.

[POP12] The POP-C++ Team, Grid & Cloud Computing Group, "*Parallel Object Programming in C++ User and Installation Manual*", Version : 2.5-a, 2009.

[Sean13] Sean Luke, "Essentials of Metaheuristics A Set of Undergraduate Lecture Notes", Online Version 2.0, 2013, ISBN 978-1-300-54962-8.

[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.