

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN MÔN PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

**Single- and multi-objective evolutionary  
algorithms for the knapsack problem with  
dynamically changing constraints**

*Người hướng dẫn:* TS NGUYỄN CHÍ THIỆN

*Người thực hiện:* LIÊU THANH LÂM – C1900015

LÊ MINH QUÂN – 52100094

VÕ NGUYỄN ANH KHOA – 52100049

Lớp : 190C0101 - 21050201

Khoá : 23 - 25

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM  
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG  
KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN MÔN PHÂN TÍCH THIẾT KẾ  
GIẢI THUẬT**

**Single- and multi-objective evolutionary  
algorithms for the knapsack problem with  
dynamically changing constraints**

Người hướng dẫn: **TS NGUYỄN CHÍ THIỆN**

Người thực hiện: **LIỄU THANH LÂM**

**LÊ MINH QUÂN**

**VÕ NGUYỄN ANH KHOA**

Lớp : **190C0101 - 21050201**

Khoá : **23 - 25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

## LỜI CẢM ƠN

Đây là phần tác giả **tự viết** ngắn gọn, thể hiện sự biết ơn của mình đối với những người đã giúp mình hoàn thành Luận văn/Luận án. Tuyệt đối không sao chép theo mẫu những “lời cảm ơn” đã có.

## **ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của TS Nguyễn Chí Thiện;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 14 tháng 10 năm 2023*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Liều Thanh Lâm*

*Lê Minh Quân*

*Võ Nguyễn Anh Khoa*

## **PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN**

### **Phần xác nhận của GV hướng dẫn**

---

---

---

---

---

---

---

---

Tp. Hồ Chí Minh, ngày      tháng      năm  
(kí và ghi họ tên)

### **Phần đánh giá của GV chấm bài**

---

---

---

---

---

---

---

---

Tp. Hồ Chí Minh, ngày      tháng      năm  
(kí và ghi họ tên)

## **TÓM TẮT**

Trình bày tóm tắt vấn đề nghiên cứu, các hướng tiếp cận, cách giải quyết vấn đề và một số kết quả đạt được, những phát hiện cơ bản trong vòng 1 -2 trang.

## MỤC LỤC

LỜI CẢM ƠN .....	i
PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN .....	iii
TÓM TẮT .....	iv
MỤC LỤC .....	1
DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ .....	5
CHƯƠNG 1. MỞ ĐẦU .....	6
PHẦN 1.1. LÝ DO CHỌN ĐỀ TÀI .....	6
PHẦN 1.2. MỤC TIÊU ĐỀ TÀI.....	6
PHẦN 1.3. ĐỐI TƯỢNG VÀ PHẠM VI NGUYÊN CỨU .....	6
PHẦN 1.3.1. ĐỐI TƯỢNG NGHIÊN CỨU .....	6
PHẦN 1.3.2. PHẠM VI NGHIÊN CỨU .....	6
PHẦN 1.4. Ý NGHĨA KHOA HỌC VÀ Ý NGHĨA THỰC TIỄN .....	6
PHẦN 1.4.1. Ý NGHĨA KHOA HỌC.....	6
PHẦN 1.4.2. Ý NGHĨA THỰC TIỄN.....	7
PHẦN 1.5. CƠ SỞ KHOA HỌC .....	7
CHƯƠNG 2. EVOLUTION ALGORITHM .....	7
PHẦN 2.1. LỊCH SỬ HÌNH THÀNH .....	7
PHẦN 2.2. GIẢI THUẬT TIẾN HÓA LÀ GÌ?.....	8
PHẦN 2.3. SINGLE-OBJECTIVE ALGORITHM.....	10
PHẦN 2.3.1. ĐỊNH NGHĨA.....	10
Single-objective algorithm là thuật toán giải quyết các bài toán tối ưu hóa có duy nhất một mục tiêu (objective). ....	10
PHẦN 2.3.2. CÁC THUẬT TOÁN.....	10
Các thuật toán tiêu biểu như greedy (tham lam), recursion (đệ quy), lặp ( ...iterative), định hướng tìm kiếm (Directed search algorithm), dẫn hướng (Hill climbing algorithm).....	10

PHẦN 2.4. MULTIPLE – OBJECTIVES ALGORITHM .....	10
PHẦN 2.4.1. ĐỊNH NGHĨA.....	10
PHẦN 2.4.2. CÁC THUẬT TOÁN.....	10
CHƯƠNG 3. KNAPSACK.....	11
PHẦN 3.1. ĐỊNH NGHĨA BÀI TOÁN KNAPSACK .....	11
PHẦN 3.2. 1+1 EA .....	12
PHẦN 3.2.1. BÀI TOÁN (1+1)EA LÀ GÌ.....	12
PHẦN 3.2.2. GIẢI THUẬT CỦA BÀI TOÁN (1+1)EA.....	12
PHẦN 3.3. MOEA và MOEAD.....	13
PHẦN 3.3.1. BÀI TOÁN MOEA LÀ GÌ? .....	13
PHẦN 3.3.2. GIẢI THUẬT CỦA BÀI TOÁN MOEA VÀ MOEAD....	14
Bảng 4: Mã giả repair.....	16
PHẦN 3.3.3. PHÂN TÍCH LÝ THUYẾT VÀ TIẾP CẬN CƠ SỞ.....	16
PHẦN 3.4. NSGA-II.....	18
PHẦN 3.4.1. BÀI TOÁN NSGA-II LÀ GÌ? .....	18
PHẦN 3.3.2. GIẢI THUẬT CỦA BÀI TOÁN NSGA-II .....	18
PHẦN 3.4. SPEA2 .....	20
PHẦN 3.4.1. BÀI TOÁN SPEA2 LÀ GÌ? .....	20
PHẦN 3.4.2. GIẢI THUẬT SPEA2.....	20
PHẦN 3.4.3. CÔNG THỨC MỚI CHO BÀI TOÁN KNAPSACK .....	21
CHƯƠNG 4. THỰC NGHIỆM .....	23
PHẦN 4.1. GIẢI THUẬT (1+1) EA, MOEA VÀ <b>MOEAD</b> ,(1+1)EA.....	23
MOEA và MOEAD .....	23
PHẦN 4.2. GIẢI THUẬT NSGA- II VÀ SPEA2 .....	29
PHẦN 4.3. ĐÁNH GIÁ BENCHMARK VÀ HIỆU SUẤT.....	33
TÀI LIỆU THAM KHẢO.....	34
PHỤ LỤC .....	35



PHẦN 6.1. EA .....	35
PHẦN 6.2. (1+1) EA.....	37
PHẦN 6.3. MOEA .....	37
Định nghĩa 2: Bắt đầu từ 1 dân số tùy ý, thời gian dự kiến cho MOEA và MOEAD để sản sinh ra 1 giải pháp tối ưu cho bất kỳ dung lượng $C$ * trong $\{0, \dots, 2n - 1\}$ cho bài toán T là $O(n^2 \log n)$ nếu $\delta \geq n$ .....	
<b>Error!</b>	
<b>Bookmark not defined.</b>	
PHẦN 6.4. <b>MOEAD</b> .....	40
PHẦN 6.5. NSGA II .....	41
PHẦN 6.6. SPEA2 .....	<b>Error! Bookmark not defined.</b>

## DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

### CÁC KÝ HIỆU

$f$  Tần số của dòng điện và điện áp (Hz)

$p$  Mật độ điện tích khối (C/m<sup>3</sup>)

### CÁC CHỮ VIẾT TẮT

EA Giải thuật tiến hóa

## DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

### DANH MỤC HÌNH

Hình 1. Các sự kiện lịch sử phát triển giải thuật EC.....8

Hình 2.....**Error! Bookmark not defined.**

### DANH MỤC BẢNG

Bảng 1. Mã giả thuật toán tiến hóa .....10

## CHƯƠNG 1. MỞ ĐẦU

### PHẦN 1.1. LÝ DO CHỌN ĐỀ TÀI

Phân tích thiết kế yêu cầu là học phần nghiên cứu các khái niệm và phân tích kỹ thuật phân tích độ phức tạp của thuật toán.

Knapsack (bài toán xếp balo) là là một bài toán tối ưu hóa tổ hợp. Bài toán được đặt tên từ vấn đề chọn những gì quan trọng có thể nhét vừa vào trong một cái túi (với giới hạn khối lượng) để mang theo trong một chuyến đi. [1]

### PHẦN 1.2. MỤC TIÊU ĐỀ TÀI

Nâng cao kiến thức về thuật toán và các kỹ thuật phân tích độ phức tạp của thuật toán. Cụ thể ở đề tài này, nghiên cứu về bài toán ba lô(knapsack): Khái niệm, phân loại, các hướng phát triển,...

### PHẦN 1.3. ĐỐI TƯỢNG VÀ PHẠM VI NGUYÊN CỨU

#### PHẦN 1.3.1. ĐỐI TƯỢNG NGHIÊN CỨU

Giải thuật tiến hóa, bài toán Knapsack và các vấn đề liên quan

#### PHẦN 1.3.2. PHẠM VI NGHIÊN CỨU

Đề tài giới hạn sử dụng các thuật toán tiến hóa với tham số điều chỉnh để nghiên cứu bài toán knapsack.

### PHẦN 1.4. Ý NGHĨA KHOA HỌC VÀ Ý NGHĨA THỰC TIỄN

#### PHẦN 1.4.1. Ý NGHĨA KHOA HỌC

Bài toán xếp ba lô là một trong những bài toán tổ hợp phổ biến được nghiên cứu trong lĩnh vực tối ưu hóa. Một số điểm quan trọng về bài toán này:

- Đầu vào là tập hợp các vật phẩm cần vận chuyển, mỗi vật phẩm có khối lượng và giá trị nhất định.
- Mục tiêu là chọn ra tập hợp các vật phẩm để đảm bảo tổng khối lượng không vượt quá dung tích ba lô, và tổng giá trị là lớn nhất.
- Bài toán thuộc dạng tổ hợp 0-1 với điều kiện ràng buộc. Đòi hỏi phải sử dụng các thuật toán tối ưu như tham lam, đệ quy, Lựa chọn gene,...

- Là một trong những bài toán cơ bản để kiểm tra hiệu quả của các thuật toán tối ưu.

### ***PHẦN 1.4.2. Ý NGHĨA THỰC TIỄN***

Bài toán xếp ba lô có nhiều ý nghĩa thực tiễn quan trọng trong lĩnh vực vận tải và logistics:

- Xếp hàng hóa trên tàu biển, máy bay: Giúp tận dụng tối đa dung tích, tăng doanh thu hoặc giảm chi phí vận chuyển.
- Sắp xếp hàng hóa trên xe tải, container: Bố trí hợp lý để vận chuyển nhiều item khác nhau.
- Lập kế hoạch lô hàng giao cho người giao hàng: Tối ưu hóa tải trọng xe và lộ trình giao hàng.
- Quản lý kho hàng: Bố trí khoa học để tiết kiệm diện tích, tăng năng suất.
- Hoạch định sản xuất: Phân bổ nguyên vật liệu hợp lý để sản xuất đa dạng item.
- Lập kế hoạch giao dịch: Tối ưu danh mục đầu tư, phân bổ tài sản, nguồn lực.

### **PHẦN 1.5. CƠ SỞ KHOA HỌC**

Cơ sở khoa học của bài toán knapsack là lý thuyết về lựa chọn tối ưu của từng phần tử trong tập hợp, sao cho tổng hợp lợi ích/giá trị lớn nhất mà không vượt quá giới hạn tương ứng.

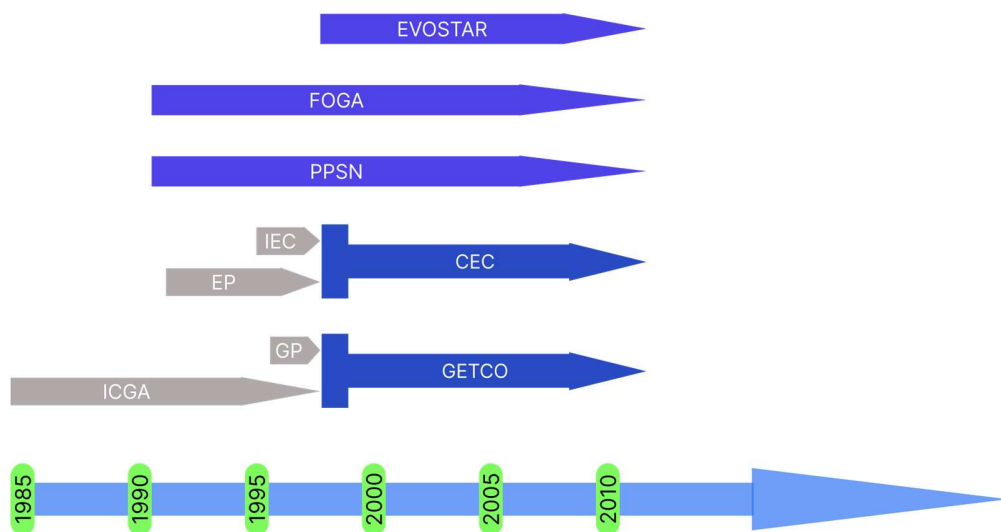
Cụ thể:

- Lý thuyết về lựa chọn tối đa: Lựa chọn từng phần tử trong tập hợp sao cho tổng hợp lợi ích lớn nhất.
- Lý thuyết hạn chế: Có sự hạn chế về khối lượng/trọng lượng tổng thể không vượt quá giới hạn cho phép.
- Lý thuyết phân bổ tài nguyên hữu hạn: Tài nguyên/không gian có hạn không thể chứa hết tất cả phần tử.
- Toán học ứng dụng: Sử dụng các phương pháp như lập kế hoạch tuyến tính, thuật toán greedy, thuật toán động lực học,... để giải quyết bài toán.

## **CHƯƠNG 2. EVOLUTION ALGORITHM**

### **PHẦN 2.1. LỊCH SỬ HÌNH THÀNH**

Phỏng theo học thuyết di truyền học Darwin, thuật toán tiến hóa [2] ra đời và có nhiều ứng dụng trong nghiên cứu khoa học. Ý tưởng được phát triển trước khi máy tính ra đời vào những năm 40 của thế kỷ 20. Năm 1948, Turing đề xuất “genetical or evolutionary search”. Năm 1962, Bremermann tính được độ phức tạp của giải thuật trong “optimization through evolution and recombination”. Trong thời gian đó, ba phân ban đầu của ý tưởng cơ bản ban đầu được phát triển ở 3 nơi khác nhau: Ở Mỹ, Fogel, Owens và Walsh giới thiệu giải thuật tiến hóa Evolutionary Programming, trong khi Holland gọi nó là Generical Algorithm; ở Đức, Rechenberg và Schwefel phát minh evolution strategies. Đến 1990s, kỹ thuật đó được gọi chung là Evolution Computing(gọi tắt là EC – tạm dịch: Tính toán tiến hóa).



Hình 1. Các sự kiện lịch sử phát triển giải thuật EC

## PHẦN 2.2. GIẢI THUẬT TIẾN HÓA LÀ GÌ?

Giải thuật tiến hóa (Evolution Algorithm – EA) [2] là một thuật toán tối ưu với ý tưởng chính: “Trong một quần thể trong tự nhiên với nguồn lực hạn chế, việc chọn lọc tự nhiên sẽ xảy ra.” Với việc đó, các cá thể con được sinh ra và sàng lọc từ các cá thể bố mẹ cho tới khi có một bộ gen tốt nhất. Áp dụng nguyên lý đó, các thuật toán sau này được sinh ra với đầu vào là 2 hay nhiều biến số tạo ra 1 hay nhiều biến đầu ra mới. Qua thời gian, nhiều nhà khoa học phát triển ra nhiều biến thể của giải thuật này.

Thuật toán tiến hóa thường được áp dụng trong các bài toán tối ưu hóa, nơi mục tiêu là tìm ra giải pháp tốt nhất trong một không gian tìm kiếm. Giải pháp được biểu diễn bởi một cá thể trong quần thể, và mức độ tốt/xấu của giải pháp được đánh giá bằng cách áp dụng một hàm mục tiêu. Mục tiêu của thuật toán tiến hóa là tìm ra giải pháp tối ưu, tức là giải pháp có giá trị của hàm mục tiêu là nhỏ nhất (hoặc lớn nhất) có thể.

Thuật toán tiến hóa gồm các thành phần chính như quần thể (population), hàm mục tiêu (objective function), các toán tử di truyền (genetic operators) và quy tắc chọn lọc (selection rules). Quần thể bao gồm một tập hợp các cá thể, mỗi cá thể biểu diễn một giải pháp tiềm năng. Hàm mục tiêu đánh giá mức độ tốt/xấu của mỗi cá thể dựa trên giá trị của hàm mục tiêu cần tối ưu. Các toán tử di truyền bao gồm lai ghép (crossover) và đột biến (mutation), được áp dụng lên quần thể để tạo ra các thế hệ cá thể mới. Quy tắc chọn lọc xác định cách chọn lọc cá thể tốt nhất để tiếp tục tồn tại trong quần thể và tham gia vào quá trình tái sản xuất.

Các thuật toán tiến hóa có những ưu điểm và hạn chế. Ưu điểm bao gồm khả năng tìm kiếm trên không gian tìm kiếm lớn và không liên tục, khả năng xử lý các bài toán tối ưu hóa không khả vi hoặc không có đạo hàm, và tính toán song song cao. Tuy nhiên, các thuật toán này cũng có nhược điểm như thời gian tính toán tốn kém, không đảm bảo tìm ra giải pháp tối ưu chính xác, và cần cân nhắc vấn đề về sự đa dạng và hội tụ của quần thể để tránh suy giảm hiệu suất tìm kiếm.

```

BEGIN
    INITIALISE population with random candidate solutions;
    EVALUATE each candidate;
    REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
        1 SELECT parents;
        2 RECOMBINE pairs of parents;
        3 MUTATE the resulting offspring;
        4 EVALUATE new candidates;
        5 SELECT individuals for the next generation;
    OD
END

```

Bảng 1. Mã giả thuật toán tiến hóa

## **PHẦN 2.3. SINGLE-OBJECTIVE ALGORITHM**

### ***PHẦN 2.3.1. ĐỊNH NGHĨA***

Single-objective algorithm là thuật toán giải quyết các bài toán tối ưu hóa có duy nhất một mục tiêu (objective).

### ***PHẦN 2.3.2. CÁC THUẬT TOÁN***

Các thuật toán tiêu biểu như greedy (tham lam), recursion (đệ quy), lặp ( iterative), định hướng tìm kiếm (Directed search algorithm), dẫn hướng (Hill climbing algorithm)

## **PHẦN 2.4. MULTIPLE – OBJECTIVES ALGORITHM**

### ***PHẦN 2.4.1. ĐỊNH NGHĨA***

Đây là những thuật toán tối ưu hóa cho bài toán có nhiều hơn một hàm mục tiêu cần tối ưu cùng lúc.

### ***PHẦN 2.4.2. CÁC THUẬT TOÁN***

Các thuật toán phổ biến: NSGA-II, MOEA/D, SPEA2,  $\epsilon$ -MOEA...



## CHƯƠNG 3. KNAPSACK

### PHẦN 3.1. ĐỊNH NGHĨA BÀI TOÁN KNAPSACK

Bài toán Knapsack (còn được gọi là bài toán cái túi) là một bài toán tối ưu hóa trong lĩnh vực quy hoạch động. Bài toán yêu cầu lựa chọn một tập hợp các đối tượng có giá trị và trọng lượng khác nhau để đặt vào một cái túi có khả năng chứa trọng lượng tối đa.

Định dạng chung của bài toán Knapsack như sau:

- Cho trước một tập hợp các đối tượng, mỗi đối tượng có một giá trị và một trọng lượng.
- Cho trước một khối lượng tối đa mà cái túi có thể chứa.
- Mục tiêu là chọn một tập hợp các đối tượng sao cho tổng giá trị của chúng là lớn nhất có thể, đồng thời đảm bảo tổng trọng lượng của chúng không vượt quá khối lượng tối đa của túi.

Ví dụ:

- Input:
  - Tập hợp các đối tượng:  $[(10, 5), (8, 3), (15, 7), (6, 2)]$   
(Mỗi đối tượng có giá trị  $w$  và trọng lượng  $p$  tương ứng)
  - Khối lượng tối đa của túi: 20
- Output:
  - Tập hợp các đối tượng được chọn:  $[(10, 5), (15, 7)]$
  - Tổng giá trị của các đối tượng: 25

Trong ví dụ này, tập hợp các đối tượng gồm 4 phần tử, mỗi phần tử là một cặp (giá trị, trọng lượng). Bài toán yêu cầu chọn các đối tượng sao cho tổng giá trị là lớn nhất có thể, nhưng tổng trọng lượng không vượt quá 20. Trong trường hợp này, chọn các đối tượng có giá trị 10 và trọng lượng 5, cùng với đối tượng có giá trị 15 và trọng lượng 7, tổng giá trị của các đối tượng này là 25, và tổng trọng lượng là 12, vẫn nằm trong khối lượng tối đa của túi.

## PHẦN 3.2. 1+1 EA

### PHẦN 3.2.1. BÀI TOÁN (1+1)EA LÀ GÌ

Bài toán (1+1) EA (1+1 Evolutionary Algorithm) là một trong những bài toán đơn giản nhất trong lĩnh vực thuật toán di truyền. Nó bao gồm các yếu tố sau:

- Chỉ có 1 cá thể (parent) trong quần thể hiện tại.
- Tạo ra 1 cá thể con (offspring) bằng cách thực hiện đột biến đối với parent.
- Đánh giá khả năng thích nghi của cả hai cá thể (fitness).
- Chọn cá thể có khả năng thích nghi cao hơn làm parent cho thế hệ tiếp theo.
- Cá thể còn lại bị loại khỏi quần thể.
- Lặp lại quá trình cho đến khi đạt tiêu chí dừng.

Mục đích là tìm ra cá thể (giải pháp) phù hợp nhất qua quá trình tối ưu hóa đơn giản này. Đây là mô hình cơ bản để nghiên cứu các thuật toán di truyền.

### PHẦN 3.2.2. GIẢI THUẬT CỦA BÀI TOÁN (1+1)EA

Bài toán (1+1) EA nhằm mục tiêu tối đa hóa hàm độ thích nghi  $f_{1+1}$ , bao gồm hai phần:

- Phần thứ nhất là tổng profit của các mặt hàng được chọn.
- Phần thứ hai là phần trừng phạt áp dụng cho các giải pháp không khả thi.

Số tiền phạt được đảm bảo sao cho giải pháp khả thi luôn thống trị giải pháp không khả thi.

Hơn nữa, đối với hai giải pháp không khả thi, giải pháp có trọng lượng gần hơn với C sẽ thống trị giải pháp còn lại.

Điều này khuyến khích thuật toán tìm kiếm giải pháp khả thi và gần nhất với ràng buộc trọng lượng C.

```

1  $x \leftarrow \text{previous best solution};$ 
2 while stopping criterion not met do
3    $y \leftarrow \text{flip each bit of } x \text{ independently with probability of } 1/n;$ 
4   if  $f_{1+1}(y) \geq f_{1+1}(x)$  then
5      $x \leftarrow y;$ 
```

Bảng 1: Mã giả (1+1)EA

Quá trình hoạt động của thuật toán (1+1) EA như sau:

- Bước 1: Chọn ngẫu nhiên một giải pháp ban đầu  $x$
- Bước 2: Thực hiện đột biến cho  $x$  bằng cách lật trạng thái của mỗi bit với xác suất  $1/n$ , tạo ra  $x'$
- Bước 3: Đánh giá hàm độ thích nghi của  $x$  và  $x'$  theo công thức:

$$f_{1+1}(x) = p(x) - (n * p_{max} + 1) * v(x) \text{ với:}$$

- $p(x)$ : tổng profit các mặt hàng trong x
  - $p_{max}$ : profit max
  - $v(x) = \max(0, w(x) - C)$ : vi phạm ràng buộc
  - Bước 4: Chọn giữa x và x' theo giá trị  $f_{1+1}$  cao hơn làm giải pháp mới x
  - Bước 5: Lặp lại bước 2 đến bước 4 cho đến khi đạt tiêu chí dừng
- Mục đích tìm ra giải pháp x có hàm độ thích nghi cao nhất.

Độ phức tạp của thuật toán (1+1) EA được tính như sau:

- Độ phức tạp trong mỗi lần lặp:
- Bước 1: Chọn ngẫu nhiên giải pháp ban đầu:  $O(1)$
- Bước 2: Thực hiện đột biến cho mỗi bit với xác suất  $1/n$ :  $O(n)$
- Bước 3: Đánh giá hàm độ thích nghi:  $O(n)$
- Bước 4: So sánh và chọn giải pháp:  $O(1)$

Nên độ phức tạp trong mỗi lần lặp là  $O(n)$

Không xác định trước số lần cần thiết để tìm được giải pháp tối ưu.

Tuy nhiên thông thường sẽ kết thúc trong một số lần lặp hữu hạn.

Nên độ phức tạp tổng thể của thuật toán là  $O(n*T)$

với T là số lần lặp cần thiết.

Do đó, thuật toán (1+1) EA thuộc dạng độ phức tạp bậc tuyến tính  $O(n)$  đối với mỗi lần lặp.

### PHẦN 3.3. MOEA và MOEAD

#### PHẦN 3.3.1. BÀI TOÁN MOEA LÀ GÌ?

MOEA là viết tắt của Multi-Objective Evolutionary Algorithms (Thuật toán di truyền nhiều mục tiêu), là kỹ thuật mở rộng thuật toán di truyền cho những bài toán tối ưu hóa nhiều mục tiêu.

Đây là lĩnh vực mở rộng của thuật toán di truyền để giải quyết các bài toán tối ưu hóa có nhiều hơn một hàm mục tiêu cần được tối ưu hóa cùng lúc.

Một số đặc điểm chính của MOEA:

- Bài toán có 2 mục tiêu trở lên cần tối ưu hóa.
- Kết quả là tập hợp các giải pháp không thể cải thiện về một mục tiêu mà làm giảm một mục tiêu khác.
- Sử dụng các phép toán trên không gian mục tiêu để so sánh và chọn giữa các giải pháp.
- Áp dụng cho nhiều lĩnh vực như thiết kế, lập lịch, tối ưu hóa tài nguyên...

Có một số điểm khác biệt chính giữa EA (Evolutionary Algorithm) và MOEA (Multi-Objective Evolutionary Algorithm):

- Số mục tiêu: EA giải quyết bài toán với 1 mục tiêu duy nhất cần tối ưu hóa trong khi MOEA giải quyết bài toán với 2 mục tiêu trở lên cần tối ưu hóa cùng lúc.
- Kết quả: EA trả về 1 giải pháp tối ưu duy nhất cho mục tiêu đơn. MOEA trả về tập hợp các giải pháp không thể cải thiện về một mục tiêu mà làm giảm một mục tiêu khác, gọi là Pareto.
- Cách so sánh giải pháp: EA dựa vào giá trị hàm mục tiêu duy nhất. MOEA dựa vào không gian mục tiêu và phép so sánh Pareto để xác định các giải pháp tối ưu hơn.
- Ứng dụng: EA thường được áp dụng cho bài toán tối ưu hóa đơn mục tiêu. MOEA phù hợp hơn với bài toán có nhiều mâu thuẫn/xung đột giữa các mục tiêu.

**Pareto** là khái niệm liên quan đến các vấn đề đa mục tiêu. Cụ thể:

- Pareto là trường hợp một giải pháp được coi là "không thống trị" so với các giải pháp khác.
- Để là Pareto, một giải pháp phải đáp ứng điều kiện: không có giải pháp nào tốt hơn về một mục tiêu mà xấu hơn về các mục tiêu còn lại.
- Tập hợp các giải pháp Pareto là bộ giải pháp tối ưu hiệu quả nhất, gọi là bộ giải pháp Pareto.
- Các thuật toán như NSGA-II, SPEA2 sử dụng xếp hạng Pareto để xác định các cá thể không thống trị trong quần thể.
- Mục tiêu của các thuật toán này là tìm ra được bộ giải pháp gần với bộ giải pháp Pareto nhất.

### ***PHẦN 3.3.2. GIẢI THUẬT CỦA BÀI TOÁN MOEA VÀ MOEAD***

MOEA là một thuật toán đa mục tiêu được thiết kế dựa trên nghiên cứu lý thuyết về hiệu suất của phương pháp tiến hóa trong bài toán tối ưu hóa lại hàm tuyến tính dưới tác động của ràng buộc biến đổi theo thời gian.

Trong trường hợp ràng buộc đồng dạng, trọng lượng  $w(x)$  của giải pháp  $x$  được xác định bằng số lượng phần tử được chọn, tức là  $w(x) = |x|_1$ .

Mỗi độ thích nghi của giải pháp  $x$  trong không gian mục tiêu hai mục tiêu là một điểm hai chiều  $f_{MOEA}(x) = (w(x), p(x))$ .

Chúng ta nói rằng giải pháp  $y$  thống trị giải pháp  $x$  theo  $f_{MOEA}$ , ký hiệu là:

$$y \succ_{MOEA} x, \text{ nếu } w(y) = w(x) \wedge f_{1+1}(y) \geq f_{1+1}(x).$$

Theo định nghĩa của  $\succcurlyeq$  MOEA, hai giải pháp chỉ so sánh được nếu chúng có cùng trọng lượng.

Lưu ý rằng nếu  $x$  và  $y$  không khả thi và có thể so sánh, thì giải pháp có profit cao hơn thống trị.

MOEA sử dụng một tham số được ký hiệu bằng  $\delta$ , xác định số lượng cá thể tối đa mà thuật toán được phép lưu xung quanh  $C$  hiện tại.

Đối với bất kỳ trọng lượng nào trong  $[C - \delta, C + \delta]$ , MOEA giữ lại một giải pháp.

Thuật toán chuẩn bị cho sự thay đổi động bằng cách lưu trữ các giải pháp gần đó, ngay cả khi chúng không khả thi vì chúng có thể trở nên khả thi sau khi thay đổi tiếp theo.

Tuy nhiên, một  $\delta$  lớn sẽ dẫn đến việc lưu giữ quá nhiều giải pháp, làm giảm xác suất lựa chọn một giải pháp cụ thể. Vì thuật toán chỉ chọn một giải pháp để biến đổi trong mỗi lần lặp, điều này ảnh hưởng đến hiệu suất của MOEA trong việc tìm ra giải pháp tối ưu.

Sau mỗi thay đổi động, MOEA cập nhật các tập hợp giải pháp. Nếu xảy ra thay đổi khiến tất cả các giải pháp hiện tại được lưu trữ nằm ngoài phạm vi lưu trữ, tức là  $[C - \delta, C + \delta]$ , thuật toán xem xét giải pháp tối ưu trước đó là giải pháp khởi tạo và sử dụng hàm repair hoạt động tương tự như (1+1) EA, cho đến khi tìm thấy một giải pháp có khoảng cách trọng lượng tối đa  $\delta$  từ  $C$ .

Lưu ý rằng nếu tất cả các giải pháp trong khoảng trước đó không khả thi, giải pháp tối ưu trước đó là giải pháp có trọng lượng thấp nhất.

Để khắc phục tốc độ cải thiện chậm của MOEA do  $\delta$  lớn, chúng tôi sử dụng định nghĩa chuẩn về sự thống trị trong tối ưu hóa đa mục tiêu - nghĩa là giải pháp  $y$  thống trị giải pháp  $x$ , ký hiệu là  $y \prec_{\text{MOEAD}} x$ , nếu  $w(y) \leq w(x) \wedge p(y) \geq p(x)$ .

Thuật toán mới này được gọi là MOEAD, được thu được bằng cách thay thế  $\succcurlyeq$ MOEA bằng  $\succcurlyeq$ MOEAD trong các dòng 16-19 của Thuật toán MOEA.

Cần lưu ý rằng nếu  $y$  là giải pháp không khả thi thì chỉ được so sánh với các giải pháp không khả thi khác, và nếu  $y$  là giải pháp khả thi thì chỉ được so sánh với các giải pháp khả thi khác.

MOEAD lưu giữ ít giải pháp hơn MOEA và chất lượng tổng thể của các giải pháp được lưu giữ cao hơn, bởi chúng không bị thống trị bởi bất kỳ giải pháp nào khác trong quần thể.

```

1 Update C;
2  $S^+ \leftarrow \{z \in S^+ \cup S^- \mid C < w(z) \leq C + \delta\}$ ;
3  $S^- \leftarrow \{z \in S^+ \cup S^- \mid C - \delta \leq w(z) \leq C\}$ ;
4 if  $S^+ \cup S^- = \emptyset$  then
5      $q \leftarrow$  best found solution before the dynamic change;
6 if  $C < w(q) \leq C + \delta$  then
7      $S^+ \leftarrow \{q\} \cup S^+$ ;
8 else if  $C - \delta \leq w(q) \leq C$  then
9      $S^- \leftarrow \{q\} \cup S^-$ ;

```

```

10 while a change happens do
11   if  $S^+ \cup S^- = \emptyset$  then
12     Initialize  $S^+$  and  $S^-$  by Repair( $q, \delta, C$ );
13   else
14     choose  $x \in S^+ \cup S^-$  uniformly at random;
15      $y \leftarrow$  flip each bit of  $x$  independently with probability  $1/n$ ;
16     if  $(C < w(y) \leq C + \delta) \wedge (\nexists p \in S^{++}: p \succcurlyeq MOEA y)$  then
17        $S^+ \leftarrow (S^+ \cup \{y\}) \setminus \{z \in S^+ \mid y \succcurlyeq MOEA z\}$ 
18     if  $(C - \delta \leq w(y) \leq C) \wedge (\nexists p \in S^-: p$ 
19        $S^- \leftarrow (S^- \cup \{y\}) \setminus \{z \in S^- \mid y > MOEA z\};$ 

```

Bảng 3: Mã giả MOEA

```

input : Initial solution  $q, \delta, C$ 
output :  $S^+$  and  $S^-$  such that  $|S^+ \cup S^-| = 1$ 
1 while  $S^+ \cup S^- = \emptyset$  do
2    $y \leftarrow$  flip each bit of  $q$  independently with probability of  $1/n$ ;
3   if  $f_{1+1}(y) \geq f_{1+1}(q)$  then
4      $q \leftarrow y$ ;
5     if  $C < w(q) \leq C + \delta$  then
6        $S^+ \leftarrow \{q\} \cup S^+$ ;
7     else if  $C - \delta \leq w(q) \leq C$  then
8        $S^- \leftarrow \{q\} \cup S^-$ ;

```

Bảng 4: Mã giả repair

Đối với MOEA:

- Độ phức tạp không gian:  $O(N)$  với  $N$  là số giải pháp được lưu trong quá trình thực thi thuật toán.  $N$  phụ thuộc vào tham số  $\delta$  quy định phạm vi lưu giữ các giải pháp xung quanh  $C$ .
- Độ phức tạp thời gian của mỗi lần lặp:  $O(N \log N)$  do cần so sánh và sắp xếp các giải pháp để chọn giải pháp biến đổi.

Đối với MOEAD:

- Độ phức tạp không gian:  $O(N)$  với  $N$  nhỏ hơn so với MOEA do lưu ít giải pháp hơn.
- Độ phức tạp thời gian của mỗi lần lặp: Tương tự MOEA, là  $O(N \log N)$ .

Do MOEAD lưu ít giải pháp hơn MOEA nên cả về độ phức tạp không gian và thời gian mỗi lần lặp đều thấp hơn so với MOEA.

### ***PHẦN 3.3.3. PHÂN TÍCH LÝ THUYẾT VÀ TIẾP CẬN CƠ SỞ***

Định lý 1: Thời gian cho đến khi  $(1+1)$  EA tạo ra từ giải pháp tối ưu  $x^*$  cho thể tích  $C = n - 1$  cho bài toán T, một giải pháp tối ưu cho thể tích  $C^* = C + 1 = n$  sau khi xảy ra thay đổi động là ít nhất  $n^{n/2}$  với xác suất ít nhất là  $1 - n^{-n/2}$ .

Chứng minh: Giải pháp tối ưu  $x^*$  cho thể tích  $C = n - 1$  bao gồm  $n - 1$  item đầu tiên trong khi giải pháp tối ưu  $y^*$  cho thể tích  $C^* = n$  chỉ bao gồm item cuối cùng. Sau khi đạt được giải pháp  $x^*$ ,  $(1+1)$  EA chỉ chấp nhận  $y^*$  vì đây là giải pháp duy nhất  $y$  với  $p(y) \geq p(x^*)$  và  $w(y) \leq C$ . Xác suất tạo ra  $y^*$  từ  $x^*$  là  $n^{-n}$  vì tất cả các bit của  $x^*$  phải đảo ngược cùng một lúc. Do đó, thời gian tối ưu hóa trung bình là  $nn$  và thời gian tối ưu hóa ít nhất là  $n^{\frac{n}{2}}$  với xác suất ít nhất  $1 - n^{-\frac{n}{2}}$  bằng giới hạn hợp.

Định lý 2: Khởi đầu với quần thể bất kỳ, thời gian trung bình cho đến khi MOEA và MOEAD tạo ra một giải pháp tối ưu cho bất kỳ thể tích  $C^* \in \{0, \dots, 2n - 1\}$  của bài toán T là  $O(n^2 \log n)$  nếu  $\delta \geq n$ .

Chứng minh:

- Tất cả các điểm tìm kiếm có giá trị  $w(x) = p(x)$  thuộc  $\{0, \dots, 2n - 1\}$
- Điều này ngụ ý rằng số lượng các giải pháp thỏa hiệp tối đa là  $2n$
- Kích thước quần thể luôn được giới hạn bởi  $2n$

Vì:

- Bài toán T chỉ có  $2n$  giải pháp hợp lệ khác nhau
- Mỗi giải pháp đại diện cho 1 điểm trong không gian giải pháp
- Do đó, tối đa chỉ có  $2n$  điểm khác nhau trong không gian giải pháp
- Kích thước quần thể bị giới hạn bởi số lượng điểm tối đa là  $2n$

Định lý 3: Khởi đầu từ quần thể tối ưu cho bài toán T,  $C = 0.75n$  và  $\delta = 0.25n - 1$ , thời gian cho đến khi MOEA và MOEAD tạo ra một giải pháp được chấp nhận sau khi thay đổi thể tích thành  $C^* = n$  là ít nhất  $e^{\frac{n}{4}}$  với xác suất  $1 - e^{-\frac{n}{4}}$ .

Chứng minh:

- Thay đổi giới hạn thành  $C^* = n$  có nghĩa là quần thể được cập nhật gồm các giải pháp tối ưu cho  $\{0.75n + 1, \dots, n - 1\}$
- Sau thay đổi, chỉ chấp nhận các giải pháp có trọng lượng trong  $\{0.75n + 1, \dots, 1.25n - 1\}$
- Để tạo ra từ bất kỳ giải pháp có trọng lượng trong tập  $\{0.75n + 1, \dots, n - 1\}$  một giải pháp có trọng lượng trong tập  $\{n, \dots, 1.25n - 1\}$ , phân tử  $n$  phải được đưa vào.
- Ngoài ra, còn phải loại bỏ ít nhất  $(0.75n + 1) + n - (1.25n - 1) = 0.75n + 1 - (0.25n - 1) > 0.5n$  phân tử trong  $n - 1$  phân tử đầu tiên
- Xác suất cùng lúc biến đổi tối thiểu  $0.5n$  bit là ít hơn  $e^{-\frac{n}{2}}$  vì cần đảo ngược tối thiểu  $\frac{n}{2}$  bit

- Do đó, thời gian cho sự kiện trên xảy ra là ít nhất  $e^{\frac{n}{4}}$  với xác suất  $1 - e^{-\frac{n}{4}}$  theo giới hạn hợp.

## PHẦN 3.4. NSGA-II

### PHẦN 3.4.1. BÀI TOÁN NSGA-II LÀ GÌ?

NSGA-II (Non-dominated Sorting Genetic Algorithm II) là một thuật toán tiến hóa đa mục tiêu nổi tiếng. Nó là phiên bản cải tiến của thuật toán NSGA đầu tiên.

Một số điểm chính của NSGA-II:

- Là thuật toán tiến hóa đa mục tiêu, tìm ra một tập hợp các giải pháp không thống trị (non-dominated solutions) sau mỗi lần tiến hóa thay vì một giải pháp duy nhất.
- Sử dụng phép sắp xếp theo không thống trị (non-dominated sorting) để sắp xếp các giải pháp.
- Cân bằng quần thể bằng cách giữ lại kích thước của mỗi phân tử trong mỗi tập hợp không thống trị (non-dominated fronts).
- Áp dụng hàm đánh giá đa mục tiêu và khoảng cách từ một giải pháp đến các giải pháp khác để lựa chọn giải pháp trong quá trình giao phối và đột biến.

### PHẦN 3.3.2. GIẢI THUẬT CỦA BÀI TOÁN NSGA-II

Những điểm chính trong cách hoạt động của NSGA-II:

Bước khởi tạo:

- Tạo ngẫu nhiên quần thể ban đầu  $P_0$ .
- Sắp xếp các cá thể dựa trên bậc thống trị không.
- Tạo dân số con  $Q_0$  bằng phép lai ghép và biến đổi.

Sau mỗi thay đổi:

- Ghép P và Q thành tập hợp kết hợp.
- Sắp xếp các cá thể thành các mặt phẳng không thống trị  $F_1, F_2, \dots$
- Chọn cá thể từ  $F_1, F_2, \dots$  cho đến khi vượt quá kích thước quần thể N để tạo  $P_{t+1}$ .
- Tính khoảng cách đám đông để phân biệt các cá thể trong cùng mặt phẳng.
- Gán khoảng cách vô hạn cho các cá thể ở mức cực trị.
- Sử dụng toán tử tiến hóa để tạo  $Q_{t+1}$  từ  $P_{t+1}$  dựa trên bậc và khoảng cách hạn chế.

Khoảng cách đám đông (crowding distance) được NSGA-II tính toán cho mỗi cá thể để phân biệt các cá thể trong cùng một mặt phẳng không thống trị.

- Để tính, NSGA-II lấy 2 láng giềng gần nhất của mỗi cá thể theo mỗi mục tiêu, so sánh giá trị mục tiêu và tính độ rộng của khoảng trống.



- Crowding distance càng lớn có nghĩa cá thể đó nằm ở vùng không gian rộng hơn, ít bị chi phối bởi các cá thể khác trong cùng mặt phẳng.
- NSGA-II sẽ ưu tiên lựa chọn các cá thể có khoảng cách đám đông lớn nhằm duy trì sự phân bố đa dạng của quần thể

```

1 Update C;
2
3 while stopping criterion not met do
4    $R_t \leftarrow P_t \cup Q_t$  .
5    $F \leftarrow \text{non-dominated-sort}(R_t)$ 
6    $F = (F_1, F_2, \dots)$ , all non-dominated fronts of  $R_t$ 
7    $P_{t+1} \leftarrow \emptyset$  and  $i \leftarrow 1$ ;
8   while  $|P_{t+1}| + |F_i| \leq N$  do
9      $P_{t+1} \leftarrow P_{t+1} \cup F_i$ ;
10     $i \leftarrow i + 1$ ;
11    crowding-distance-assignment( $F_i$ );
12    Sort  $F_i$  based on the crowding distance in descending order;
13     $P_{t+1} \leftarrow P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$ ;
14     $Q_{t+1} \leftarrow \text{make-new-pop}(P_{t+1})$ ;
15     $t \leftarrow t + 1$ ;

```

Bảng 5: Mã giả NSGA-II

Độ phức tạp của thuật toán NSGA-II (Non-dominated Sorting Genetic Algorithm II) như sau:

- Phép so sánh thống trị để xếp hạng các cá thể:  $O(MN^2)$

Ở đây:

- M là số mục tiêu cần tối ưu hóa
- N là quy mô dân số
- Xếp hạng các cá thể theo trình tự ưu tiên:  $O(MN^2)$
- Tính toán khoảng cách đám đông cho mỗi cá thể:  $O(N^2)$
- Lựa chọn để tạo thế hệ tiếp theo:  $O(N \log N)$
- Chạy thuật toán trong tổng cộng T thế hệ:

Độ phức tạp tổng cộng của NSGA-II là  $O(T * (MN^2 + N^2 \log N))$

Trong đó:

- N là quy mô dân số
- M là số mục tiêu
- T là tổng số thế hệ

- Do vậy, độ phức tạp của NSGA-II phụ thuộc vào quy mô dân số  $N$ , số mục tiêu  $M$  và tổng số thế hệ  $T$ . Thuật toán có độ phức tạp bậc  $O(N^2)$ .

## PHẦN 3.4. SPEA2

### PHẦN 3.4.1. BÀI TOÁN SPEA2 LÀ GÌ?

SPEA2 là một thuật toán tiến hóa đa mục tiêu nhằm tìm ra các giải pháp không thống trị và phân bố đều đặn trong không gian giải pháp.

Một số điểm chính của SPEA2:

- SPEA2 sử dụng khái niệm "sức mạnh Pareto" (Pareto strength) để đánh giá độ phù hợp của một cá thể. Sức mạnh Pareto đếm số lượng các cá thể thống trị cá thể đó.
- SPEA2 áp dụng khái niệm "ước lượng mật độ" (density estimation) để duy trì sự phân bố đều đặn của các cá thể.
- SPEA2 lưu trữ các cá thể không thống trị trong bộ nhớ kết quả và sử dụng nó để tạo ra thế hệ kế tiếp.
- Độ phù hợp cuối cùng của cá thể là tổng của sức mạnh Pareto và ước lượng mật độ.

### PHẦN 3.4.2. GIẢI THUẬT SPEA2

SPEA2 sẽ lưu giữ các giải pháp không thống trị tốt nhất ở mỗi thế hệ, sau đó sử dụng chúng như là "bộ lưu trữ" để sinh ra thế hệ kế tiếp thông qua lai ghép và biến đổi. Đây là cách tiếp cận khác với NSGA-II để duy trì sự phân bố của các giải pháp.

- Thuật toán này lưu giữ các giải pháp không thống trị tốt nhất của mỗi thế hệ trong bộ lưu trữ  $\bar{P}_t$  có kích thước  $\bar{N}$ .
- Nó tạo ra tập giải pháp dân số kế tiếp  $P_{t+1}$ , cũng kích thước  $N$ , bằng cách áp dụng các toán tử tiến hóa lên bộ lưu trữ trước đó  $P_t$ . Các toán tử tiến hóa thường được sử dụng trong tiến hóa gồm: lai ghép (crossover), đột biến (mutation), chọn lọc (selection).

Độ phù hợp của giải pháp  $x$  trong SPEA2 được tính toán dựa trên hai yếu tố:

- Độ phù hợp thô là số nguyên  $0 \leq R(x)$ , biểu diễn sức thống trị của các giải pháp thống trị  $x$ .
- Ước lượng mật độ  $0 < D(x) \leq 1/2$ , được tính toán dựa trên nghịch đảo khoảng cách đến láng giềng thứ  $k$  của  $x$  với cùng giá trị  $R(x)$ . Giá trị mặc định của  $k$  là  $k = \sqrt{N + \bar{N}}$ .

```

1 Update C;
2 Update objective values of solutions in populations set  $P_t$  and archive set  $\bar{P}_t$ ;
3 while stopping criterion not met do
4   Mating selection: Generate a mating pool by tournament selection from  $\bar{P}_t$ ;
5   Variation: Apply crossover and mutation operators on the mating pool to produce
      $P_{t+1}$ ;
6   Fitness assignment: Calculate fitness values of solutions in  $P_{t+1} \cup \bar{P}_t$ ;
7   Environmental selection: Generate  $\bar{P}_{t+1}$  by choosing  $\bar{N}$  non-dominated solutions
     from  $P_{t+1} \cup \bar{P}_t$ ;

```

Bảng 6: Mã giả SPEA2

Cả hai thuật toán NSGA-II và SPEA2, không giống như MOEAD, sử dụng các kỹ thuật cụ thể để đảm bảo tập hợp các giải pháp được phân bố một cách đều đặn. Điều này rất quan trọng để tìm ra bộ giải pháp gần với bộ giải pháp Pareto nhất. Bên cạnh đó, các thuật toán cần xem xét giới hạn về công suất để đáp ứng yêu cầu của bài toán, đồng thời phải có khả năng áp dụng cho các thay đổi động trong tương lai.

Độ phức tạp của thuật toán SPEA (Strength Pareto Evolutionary Algorithm) như sau:

- Tính toán độ mạnh của mỗi cá thể:  $O(N^2)$  : Ở đây N là quy mô dân số. Để tính độ mạnh của một cá thể cần so sánh với tất cả các cá thể còn lại, do đó độ phức tạp là  $O(N^2)$ .
- Sắp xếp các cá thể dựa trên độ mạnh:  $O(N \log N)$
- Lựa chọn bộ thể cho thế hệ tiếp theo bằng cách Sử dụng phương pháp chọn cặp sau đó đưa vào bộ lọc elitism.:  $O(N \log N)$
- Thực hiện trong T thế hệ

Độ phức tạp tổng cộng của SPEA là  $O(T * (N^2 + N \log N))$

Tóm lại, độ phức tạp của thuật toán SPEA là  $O(N^2)$ . Phức tạp chủ yếu đến từ phép tính độ mạnh của các cá thể có độ phức tạp  $O(N^2)$

### ***PHẦN 3.4.3. CÔNG THỨC MỚI CHO BÀI TOÁN KNAPSACK***

Khác với MOEAD sử dụng hai tập giải pháp riêng biệt để lưu trữ các giải pháp không khả thi nhằm chuẩn bị cho các thay đổi động tiếp theo, chúng tôi khai thác khả năng của SPEA2 và NSGA-II trong việc tìm ra một tập hợp các giải pháp không thống trị được phân bố đều đặn.

Ta buộc các thuật toán phải tìm kiếm các giải pháp không thống trị có trọng lượng nằm trong khoảng  $[C - \delta, C + \delta]$ . Ta có:

$$w_{MO} = \begin{cases} w(x) & \text{nếu } w(x) \in [C - \delta, C + \delta] \\ w(x) + (n \cdot w_{max} + 1) \cdot \alpha(x) & \text{mặt khác,} \end{cases}$$

Đối với giải pháp  $x$  mà

$w(x) \notin [C - \delta, C + \delta]$ ,  $\alpha(x) = \{|w(x) - C - \delta|, |w(x) - C + \delta|\}$ . Tương tự như khối lượng  $w$ , ta có:

$$p_{MO} = \begin{cases} p(x) & \text{nếu } w(x) \in [C - \delta, C + \delta] \\ p(x) + (n \cdot p_{max} + 1) \cdot \alpha(x) & \text{mặt khác,} \end{cases}$$

Penalty đảm bảo rằng bất kỳ giải pháp nào trong khoảng ưu tiên sẽ thống trị các giải pháp bên ngoài, và các giải pháp gần khoảng hơn sẽ thống trị các giải pháp xa hơn. Bằng cách này, nếu các thuật toán tạo ra một tập hợp các giải pháp không thống trị được phân bố đều đặn, dự kiến vẫn có các giải pháp khả thi chất lượng cao ngay cả sau khi xảy ra các thay đổi động.

## CHƯƠNG 4. THỰC NGHIỆM

### PHẦN 4.1. GIẢI THUẬT (1+1) EA, MOEA VÀ $MOEA_D$ , (1+1)EA, MOEA và MOEAD

- (1+1) EA

Độ phức tạp của thuật toán:  $BigTheta(n^2)$

Input: listchoices quần thể của các items trong knapsack có được chọn hay không, tỉ lệ đột biến `mutate_rate`

Output: các quần thể của items sau khi đột biến



```
1 static bool[] Mutate(bool[] listchoices, double mutate_ra
2 {
3     Random rd = new Random();
4     for (int i = 0; i < listchoices.Length; i++)
5     {
6         if (rd.NextDouble() < mutate_rate)
7         {
8             if (listchoices[i])
9             {
10                 listchoices[i] = false;
11             }
12             else
13             {
14                 listchoices[i] = true;
15             }
16         }
17     }
18     return listchoices;
19 }
```

Độ phức tạp của thuật toán:  $\text{BigTheta}(n \cdot a^2) \Leftrightarrow \text{BigTheta}(n^3)$

Input: Một Knapsack cùng với số lần lặp

Output: một knapsack sau khi đã qua xử lý thuật toán

```

1  internal static Knapsack OnePlusOne(Knapsack solution, int iter)
2  {
3      Random rd = new Random();
4      for (int i = 0; i < iter; i++)
5      {
6          Knapsack y = solution;
7          bool[] oldlist = solution.listChoice;
8          bool[] M = Mutate(oldlist, rd.NextDouble());
9          y.setListChoices(M);
10         if (y.Fitness >= solution.Fitness)
11         {
12             solution = y;
13         }
14     }
15     return solution;
16 }

```

- **MOEA, *MOEA<sub>D</sub>***

Độ phức tạp của thuật toán: Repair BigTheta( $n^2 \log n$ )

Input: đầu vào là một knapsack với chỉ số chênh lệch capacity cùng với số lần lặp tau

Output: list knapsack  $s^+$ ,  $s^-$  sau khi tạo sinh từ knapsack ban đầu



```

1 public List<List<Knapsack>> Repair(Knapsack q, int delta, int tau)
2 {
3     List<Knapsack> Splus = new List<Knapsack>();
4     List<Knapsack> Sminus = new List<Knapsack>();
5     int C = q.Capacity;
6     Random rd = new Random();
7
8     while (Splus.Count + Sminus.Count == 0)
9     {
10         Knapsack y = q;
11         bool[] oldlist = q.listChoice;
12         bool[] M = Mutate(oldlist, rd.NextDouble());
13         y.setListChoices(M);
14         if (y.Fitness >= q.Fitness)
15         {
16             q = y;
17             if (q.TotalWeight > C && q.TotalWeight <= C + delta)
18             {
19                 Splus.Add(q);
20             }
21             else if (q.TotalWeight >= C - delta && q.TotalWeight <
22             {
23                 Sminus.Add(q);
24             }
25         }
26     }
27     return new List<List<Knapsack>> { Splus, Sminus };
28 }

```

Độ phức tạp của thuật toán: Moea BigTheta( $n^3$ )

Input: đầu vào là một knapsack với chỉ số chênh lệch capacity cùng với số lần lặp tau



Output: một knapsack sau khi đã qua xử lý thuật toán



```

1  public Knapsack GetAlgorithm(Knapsack solution, int delta, int tau)
2  {
3      List<Knapsack> Splus = new List<Knapsack>();
4      List<Knapsack> Sminus = new List<Knapsack>();
5      Random rd = new Random();
6
7      for (int i = 0; i < tau; i++)
8      {
9          List<Knapsack> intersaction = new List<Knapsack>();
10         List<List<Knapsack>> temp = new List<List<Knapsack>>();
11
12         if (Splus.Count + Sminus.Count == 0)
13         {
14             temp = Repair(solution, delta, tau);
15             Splus = temp[0];
16             Sminus = temp[1];
17         }
18         else
19         {
20             foreach (Knapsack p in Splus)
21             {
22                 intersaction.Add(p);
23             }
24             foreach (Knapsack p in Sminus)
25             {
26                 intersaction.Add(p);
27             }
28             solution = intersaction[rd.Next(intersaction.Count)];
29
30             Knapsack y = solution;
31             bool[] oldlist = solution.listChoice;
32             bool[] M = Mutate(oldlist, rd.NextDouble());
33             y.setListChoices(M);
34             int C = solution.Capacity;
35             int totalYWeight = y.TotalWeight;
36             if (totalYWeight > C && totalYWeight <= C + delta && CheckGre
37             {
38                 List<Knapsack> Splus1 = Splus;
39                 Splus.Add(y);
40                 foreach (Knapsack z in Splus1)
41                 {
42                     if (GreaterThan(z, y))
43                     {
44                         Splus.Add(z);
45                     }
46                 }
47             }

```

## PHẦN 4.2. GIẢI THUẬT NSGA- II

Độ phức tạp của thuật toán: Getoffstring  $O(n^2)$

Input: quần thể knapsack cha mẹ

Output: quần thể con



```

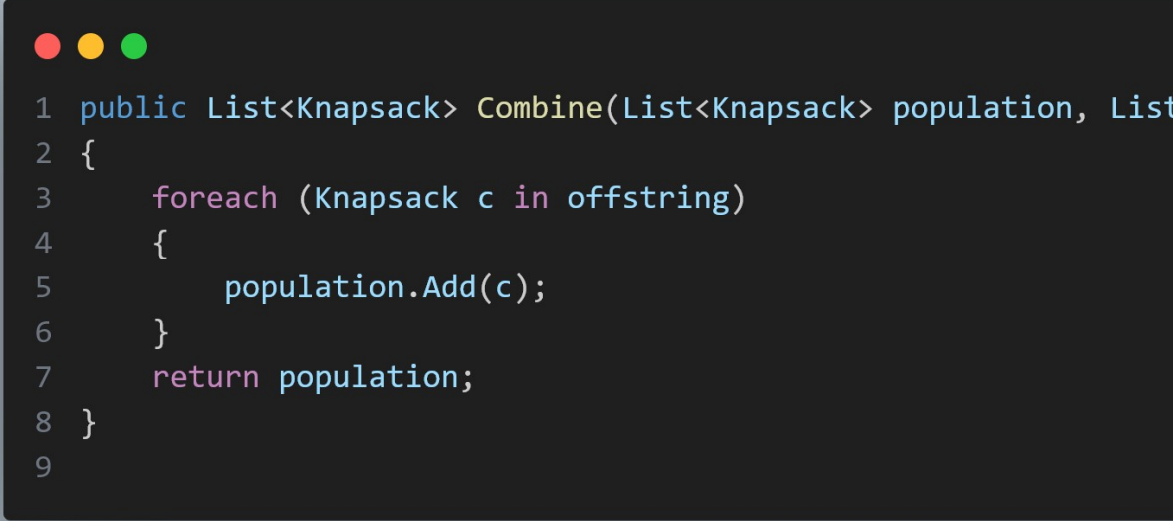
1  public List<Knapsack> GetOffstring(List<Knapsack>
2  {
3      List<Knapsack> offstring = new List<Knapsack>()
4      Random rd = new Random();
5      for (int i = 0; i < parents.Count - 1; i++)
6      {
7          for (int j = i + 1; j < parents.Count; j++)
8          {
9              Knapsack parent = parents[i], parent2 = parents[j];
10             parent.setCapacity(parent.Capacity + parent2.Capacity);
11             offstring.Add(parent);
12         }
13     }
14     return offstring;
15 }

```

Độ phức tạp của thuật toán: Combine  $O(n^3)$

Input: list knapsack cha mẹ và con cái

Output: một list knapsack sau khi gộp

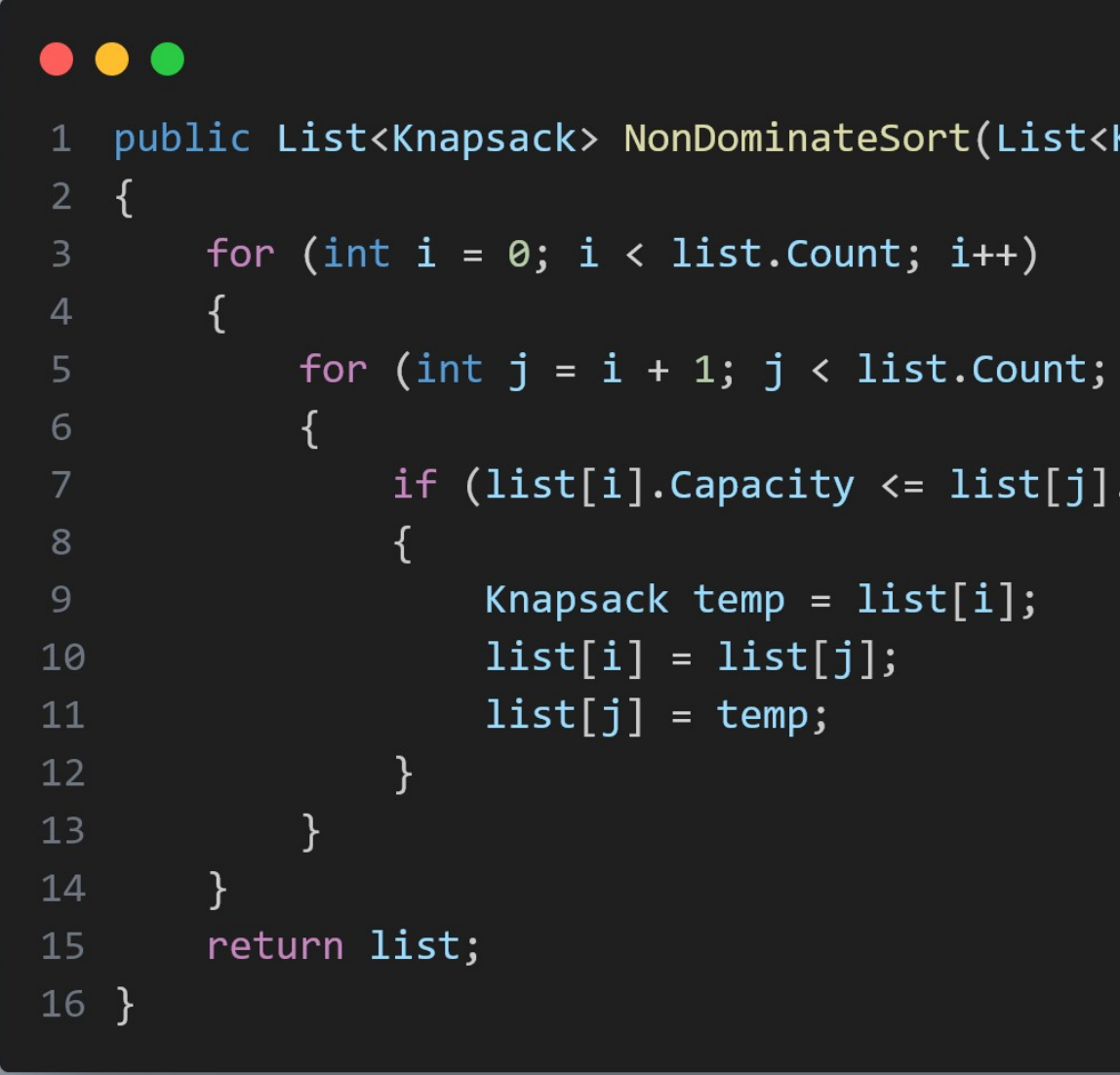


```
1 public List<Knapsack> Combine(List<Knapsack> population, List<Knapsack> offstring)
2 {
3     foreach (Knapsack c in offstring)
4     {
5         population.Add(c);
6     }
7     return population;
8 }
9
```

Độ phức tạp của thuật toán: Nondominatesort sắp xếp chọn  $O(n^2)$

Input: list knapsack

Output: list knapsack sau khi được sắp xếp bởi capacity (hạng của knapsack)



```

1  public List<Knapsack> NonDominateSort(List<Knapsack> list)
2  {
3      for (int i = 0; i < list.Count; i++)
4      {
5          for (int j = i + 1; j < list.Count; j++)
6          {
7              if (list[i].Capacity <= list[j].Capacity)
8              {
9                  Knapsack temp = list[i];
10                 list[i] = list[j];
11                 list[j] = temp;
12             }
13         }
14     }
15     return list;
16 }

```

Độ phức tạp của thuật toán:  $O(n(a^2+b^3+c^2+n))$  tương đương  $O(n^4)$

Input: đầu vào là một knapsack với chỉ số chênh lệch capacity cùng với số lần lặp lại

Output: list knapsack sau khi được chọn lọc



```

1  public List<Knapsack> getAlgorithm(Knapsack
2  {
3      int population_size = delta;
4      Random rd = new Random();
5      MOEAD mOEAD = new MOEAD();
6      List<Knapsack> population = new List<Knapsack>();
7      for (int i = 0; i < population_size; i++)
8      {
9          Knapsack s = solution;
10         bool[] oldlist = solution.listChoices;
11         bool[] M = Mutate(oldlist, rd.NextDouble());
12         s.setListChoices(M);
13         population.Add(s);
14     }
15
16
17
18     for (int i = 0; i < tau; i++)
19     {
20         List<Knapsack> offstring = GetOffspring(population);
21         List<Knapsack> R = Combine(population, offstring);
22         List<Knapsack> front = NonDominatedSort(R);
23         int j = 0;
24         List<Knapsack> new_population = new List<Knapsack>();
25         for (j = 0; j < population_size; j++)
26         {
27             new_population.Add(front[rd.NextDouble()]);
28         }
29         population = new_population;

```

### PHẦN 4.3. ĐÁNH GIÁ BENCHMARK VÀ HIỆU SUẤT

Nên sử dụng:

- 1+1EA khi bài toán đơn giản, yêu cầu hiệu suất thấp.
  - MOEA khi cần kết quả đa dạng.
  - MOEAD khi cần hiệu suất cao, giải quyết các bài toán phức tạp.
  - NSGA-II khi yêu cầu tốc độ. SPEA2 nếu không quan tâm tới tốc độ.
- 
- ✓ Độ đa dạng: NSGA-II và SPEA2 có độ đa dạng cao hơn MOEA/D do chúng duy trì toàn bộ bộ phận không thống trị.
  - ✓ Chất lượng: Hiệu suất của các thuật toán tương đương nhau đối với hầu hết các bài toán. Nhưng MOEA/D thường cho chất lượng cao hơn khi số mục tiêu tăng lên.
  - ✓ Tính ổn định: Kết quả của NSGA-II và SPEA2 ít ổn định hơn do dựa trên khoảng cách đám đông. MOEA/D ổn định hơn.
  - ✓ Tốc độ: NSGA-II nhanh nhất, tiếp theo là SPEA2. MOEA/D chậm hơn do phân tách hàm mục tiêu.
  - ✓ Độ phức tạp: NSGA-II đơn giản nhất, độ phức tạp  $O(MN^2)$ . SPEA2 phức tạp hơn với  $O(N^2)$ . MOEA/D phức tạp nhất,  $O(TN^3)$ .
  - ✓ Nên sử dụng NSGA-II nếu yêu cầu tốc độ, MOEA/D khi cần chất lượng cao, SPEA2 cân bằng giữa các yếu tố.

## TÀI LIỆU THAM KHẢO

- [1] S. Martello and P. Toth, Knapsack problems - Algorithms and Computer Implementations, England: Great Britain by Biddies Ltd, Guildford, 1990.
- [2] A. Eiben and J. Smith, Introduction to Envolutary Computing - Second Edition, Springer, 2015.
- [3] V. Roostapour, A. Neumann và F. Neumann, "Single- and multi-objective evolutionary algorithms for the knapsack problem with dynamically changing constraints," *ScienceDirect*, pp. 1-21, 2022.
- [4] S. Droste, T. Jansen and I. Wegener, "On the analysis of the  $(1 + 1)$  evolutionary algorithm," *Elsevier*, pp. 51-81, 2001.



## **PHỤ LỤC**

### **PHẦN 6.1. KNAPSACK**



```
1  class Knapsack
2  {
3      private Individual[] individuals;
4      private int capacity;
5
6      public Knapsack(Individual[] individuals, int capacity)
7      {
8          this.individuals = individuals;
9          this.capacity = capacity;
10     }
11
12     public int Capacity
13     {
14         get
15         {
16             return capacity;
17         }
18     }
19     public void setCapacity(int capacity)
20     {
21         this.capacity = capacity;
22     }
23     public int TotalWeight
24     {
25         get
26         {
27             int total = 0;
28             for(int i=0;i<individuals.Length; i++)
29             {
30                 if (individuals[i].isChoice)
31                 {
32                     total += individuals[i].Weight;
33                 }
34             }
35         }
36     }
37 }
```

## PHẦN 6.2. (1+1) EA

```

1  internal static Knapsack OnePlusOne(Knapsack solution, int iter)
2  {
3      Random rd = new Random();
4      for (int i = 0; i < iter; i++)
5      {
6          Knapsack y = solution;
7          bool[] oldlist = solution.listChoice;
8          bool[] M = Mutate(oldlist, rd.NextDouble());
9          y.setListChoices(M);
10         if (y.Fitness >= solution.Fitness)
11         {
12             solution = y;
13         }
14     }
15     return solution;
16 }

```

## PHẦN 6.3. MOEA

Định nghĩa 2: Bắt đầu từ 1 dân số tùy ý, thời gian dự kiến cho MOEA và MOEAD để sản sinh ra 1 giải pháp tối ưu cho bất kỳ dung lượng  $C^*$  trong  $\{0, \dots, 2n - 1\}$  cho bài toán T là  $O(n^2 \log n)$  nếu  $\delta \geq n$ .

Chứng minh: Tất cả các điểm tìm kiếm có giá trị  $w(x) = p(x)$  trong  $\{0, \dots, 2n - 1\}$  ngụ ý rằng số lượng các giải pháp đánh đổi là tối đa  $2n$  và kích thước dân số luôn nhỏ hơn hoặc bằng  $2n$ . Chúng ta xem xét 2 trường hợp. Trước hết giả sử rằng  $0 \leq C \leq n-1$ . Khi đó giải pháp tối ưu chọn chính xác  $C$  mục từ tập hợp  $n-1$  mục đầu tiên.

Vì  $\delta \geq n$  nên các điểm tìm kiếm với giá trị  $p(x)$  khác nhau trong  $\{0, \dots, n-1\}$  được chấp nhận. Cụ thể, mọi giải pháp hợp lệ  $x$  có lợi ích  $p(x)$  trong  $\{C - \delta, \dots, C + \delta\}$ . Nếu MOEA chưa sản sinh được 1 giải  $x$  với  $p(x)$  trong  $\{C - \delta, C + \delta\}$ , kích thước dân số là 1, giải hiện tại  $x$  không hợp lệ và  $(1+1)$  EA được triển khai để tạo ra 1 giải hợp lệ...

Sử dụng luận án mức độ tương thích với giá trị  $p(x)$  khác nhau, thời gian dự kiến để thu được giải  $x^*$  với  $p(x^*) = C$  là  $O(n^2 \log n)$ . Tương tự đối với trường hợp  $n \leq C \leq 2n - 1$ . Định lý trước cho thấy việc sử dụng dân số với giá trị  $\delta$  phù hợp trong MOEA và MOEAD có thể dẫn đến tốc độ nhanh hơn nhiều so với  $(1+1)$  EA. Sau đó chúng minh khi biến đổi hạn chế chỉ lớn hơn  $\delta$  là 1, thời gian tái tối ưu của MOEA và MOEAD sẽ theo hàm mũ...

```


1  internal abstract class MOEA
2  {
3      public abstract bool GreaterEqualThan(Knapsack x, Knapsack y);
4      public abstract bool GreaterThan(Knapsack x, Knapsack y);
5      public bool CheckGreaterEqualThan(List<Knapsack> S, Knapsack y)
6      {
7          foreach (Knapsack p in S)
8          {
9              if (GreaterEqualThan(p, y))
10             {
11                 return false;
12             }
13         }
14         return true;
15     }
16
17     public List<List<Knapsack>> Repair(Knapsack q, int delta, int tau)
18     {
19         List<Knapsack> Splus = new List<Knapsack>();
20         List<Knapsack> Sminus = new List<Knapsack>();
21         int C = q.Capacity;
22         Random rd = new Random();
23
24         while (Splus.Count + Sminus.Count == 0)
25         {
26             Knapsack y = q;
27             bool[] oldlist = q.listChoice;
28             bool[] M = Mutate(oldlist, rd.NextDouble());
29             y.setListChoices(M);
30             if (y.Fitness >= q.Fitness)
31             {
32                 q = y;
33                 if (q.TotalWeight > C && q.TotalWeight <= C + delta)
34                 {
35                     Splus.Add(q);
36                 }
37                 else if (q.TotalWeight >= C - delta && q.TotalWeight <=
38                 {
39                     Sminus.Add(q);
40                 }
41             }
42         }
43         return new List<List<Knapsack>> { Splus, Sminus };

```

### 6.3.1 MOEA Cơ Bản

```
1  internal class BaseMOEA : MOEA
2  {
3      public override bool GreaterEqualThan(Knapsack x, Knapsack y)
4      {
5          if (y.TotalWeight == x.TotalWeight && x.Fitness >= y.Fitness)
6              return false;
7      }
8
9      public override bool GreaterThan(Knapsack x, Knapsack y)
10     {
11         if (y.TotalWeight == x.TotalWeight && x.Fitness > y.Fitness)
12             return false;
13     }
14 }
```

### 6.3.2 MOEA<sub>D</sub>



```
1  internal class MOEAD : MOEA
2  {
3      public override bool GreaterEqualThan(Knapsack x, Knapsack y)
4      {
5          if (y.TotalWeight <= x.TotalWeight && x.TotalValue >= y.TotalValue)
6              return true;
7          return false;
8      }
9
10     public override bool GreaterThan(Knapsack x, Knapsack y)
11     {
12         if (y.TotalWeight < x.TotalWeight && x.TotalValue > y.TotalValue)
13             return true;
14         return false;
15     }
16 }
```

## PHẦN 6.5. NSGA II

```

1  internal class NSGA2
2  {
3      public List<Knapsack> GetOffstring(List<Knapsack> parents)
4      {
5          List<Knapsack> offstring = new List<Knapsack>();
6          Random rd = new Random();
7          for (int i = 0; i < parents.Count - 1; i++)
8          {
9              for (int j = i + 1; j < parents.Count; j++)
10             {
11                 Knapsack parent = parents[i], parent2 = parents[j];
12                 parent.setCapacity(parent.Capacity + parent2.Capacity);
13                 offstring.Add(parent);
14             }
15         }
16         return offstring;
17     }
18     public List<Knapsack> Combine(List<Knapsack> population, List<Knapsack> offstring)
19     {
20         foreach (Knapsack c in offstring)
21         {
22             population.Add(c);
23         }
24         return population;
25     }
26 }
27
28 private List<List<Knapsack>> Repair(Knapsack q, int delta,
29 {
30     List<Knapsack> Splus = new List<Knapsack>();
31     List<Knapsack> Sminus = new List<Knapsack>();
32     int C = q.Capacity;
33     Random rd = new Random();
34
35     while (Splus.Count + Sminus.Count == 0)
36     {
37         Knapsack y = q;
38         bool[] oldlist = q.listChoice;

```