TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

ĐỒ ÁN
TỐT NGHIỆP ĐẠI HỌC

NGÀNH CÔNG NGHỆ THÔNG TIN

Giải thuật tìm kiếm cục bộ dựa trên ràng buộc giải bài toán đóng thùng 2D

Sinh viên thực hiện: **Cao Văn Phúc**
Lớp CNTT4 - K55

Giảng viên hướng dẫn: **TS. Phạm Quang Dũng**

HÀ NỘI 12 - 2015

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên: Cao Văn Phúc Email: phuccaotp@gmail.com

Lớp: CNTT4 Hệ đào tạo: Chính quy

Đồ án tốt nghiệp được thực hiện tại: Phòng 902 - B1

Thời gian làm ĐATN: Từ ngày 20/02/2015 đến 20/08/2015

2. Mục đích nội dung của ĐATN

Tìm hiểu giải thuật tìm kiếm cục bộ dựa trên ràng buộc và cài đặt bài toán đóng thùng 2D (Bin Packing 2D) sử dụng thư viện JOpenCBLS.

3. Các nhiệm vụ cụ thể của ĐATN

- Tìm hiểu giải thuật tìm kiếm cục bộ giải các bài toán có ràng buộc,
- Tìm hiểu thư viện JOpenCBLS,
- Phát triển giải thuật tìm kiếm cục bộ để giải bài toán đóng thùng 2D,
- Cài đặt ứng dụng tương tác với người dùng,
- Thử nghiệm và đánh giá thư viện JOpenCBLS trên bài toán BP2D.

4. Lời cam đoan của sinh viên:

Tôi - *Cao Văn Phúc* - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của học *TS. Phạm Quang Dũng*. Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

Hà Nội, ngày 14 tháng 12 năm 2015

Tác giả ĐATN

Cao Văn Phúc

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

Sinh viên đã hoàn thành tốt các nhiệm vụ của ĐATN đại học. Tôi đề nghị cho sinh viên được bảo vệ tốt nghiệp.

Hà Nội, ngày 28 tháng 12 năm 2015

Giáo viên hướng dẫn

TS. Phạm Quang Dũng

Tóm Tắt Nội Dung Đồ Án Tốt Nghiệp

Bài toán tối ưu tổ hợp là bài toán rất phổ biến trong đời sống xã hội, đặc biệt trong các lĩnh vực như phân công, quản lý, lập kế hoạch. Mục tiêu của các bài toán này là tìm một lời giải thỏa mãn một tập các ràng buộc được đặt ra và đồng thời có thể tối ưu một số mục tiêu nào đó. Thông thường các bài toán này đều thuộc lớp các bài toán NP-Khó, các phương pháp giải bài toán tối ưu tổ hợp được chia làm hai hướng: hướng tiếp cận đúng và hướng tiếp cận gần đúng.

Trong các hướng tiếp cận thì hướng tiếp cận gần đúng sẽ là khả thi hơn khi khối lượng tính toán trong các bài toán tối ưu này vượt xa khả năng tính toán của các máy tính hiện tại (2015) nếu theo hướng tiếp cận đúng. Phương pháp tìm kiếm cục bộ dựa trên ràng buộc là một phương pháp theo hướng tiếp cận gần đúng có thể sử dụng trong các bài toán tối ưu này.

Lớp các bài toán đóng thùng (Bin Packing) [3, 4, 5, 8] là các bài toán liên quan đến việc xếp các vật phẩm với kích thước cho trước vào một số các thùng (Bin) thỏa mãn các ràng buộc không chồng lấn về mặt không gian và thực hiện tối ưu một số mục tiêu nào đó.

Đồ án này nghiên cứu tìm kiếm lời giải cho bài toán đóng thùng 2D với phương pháp tìm kiếm cục bộ dựa trên ràng buộc [2, 6] sử dụng bộ thư viện JOpenCBLS [7].

Nội dung đồ án bao gồm:

Chương 1 Giới thiệu về bài toán tối ưu hóa tổ hợp trong khoa học máy tính, các phương pháp tiếp cận, ưu nhược điểm và các kết quả thực tiễn. Phương pháp tìm kiếm cục bộ dựa trên ràng buộc, các kết quả và công việc còn phải thực hiện và giới thiệu về thư viện JOpenCBLS.

Chương 2 giới thiệu về bài toán Bin Packing 2D: mô tả, mô hình toán học, ứng dụng của trong thực tiễn.

Chương 3 phân tích, cài đặt cách thức thực hiện tìm kiếm cục bộ trên bài toán đóng thùng và áp dụng thư viện JOpenCBLS vào để giải bài toán đóng thùng 2D.

Chương 4 minh họa chương trình đã được cài đặt với thư viện JOpenCBLS với

giao diện tương tác người dùng sử dụng công nghệ J2EE, Javascript, Bootstrap. Mục này bao gồm các kết quả thống kê trên các bộ dữ liệu kiểm tra có sẵn, kết quả trên các bộ này cho thấy chương trình cho khả năng giải tốt các bài này.

Mục lục

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP **1**

Mở đầu **2**

Mục lục **4**

Danh sách hình ảnh **6**

Danh sách bảng **7**

Danh mục từ viết tắt **8**

1 Bài Toán Tối Ưu Tổ Hợp **9**
1.1 Tổ hợp và không gian tìm kiếm 9
1.1.1 Khái niệm 9
1.1.2 Ví dụ 10
1.1.3 Một số kỹ thuật giải bài toán tối ưu tổ hợp 12
1.2 Tìm kiếm cục bộ dựa trên ràng buộc 13
1.3 Thư viện JOpenCBLS 14
1.3.1 Tổng quan 14
1.3.2 Các thành phần của JOpenCBLS 15
1.3.2.1 Mô hình 15
1.3.2.2 Tìm kiếm (Search) 16
1.3.2.3 Ví dụ minh họa 17

2 Bài Toán Đóng Thùng 2D - Bin Packing 2D **20**
2.1 Phát biểu bài toán 20
2.2 Mô hình toán học 20
2.3 Ứng dụng 23

3 Áp dụng thư viện JOpenCBLS vào bài toán BP2D **24**
3.1 Các chức năng cài đặt thêm 24
3.1.1 Phép chia nguyên (Function) 24
3.1.2 Phép lấy phần dư (Function) 25
3.1.3 Các lớp có chức năng tìm kiếm 26
3.1.3.1 Tabu Search 26
3.1.3.2 Random Search 27
3.2 Dữ liệu và mô hình hóa 27
3.2.1 Dữ liệu 27
3.2.2 Mô hình 28
3.2.2.1 Khai báo và cấp phát bộ nhớ 28
3.2.2.2 Tải các ràng buộc 29
3.2.2.3 Thực hiện tìm kiếm 31

4 Chương Trình Minh Họa **32**
4.1 Mô hình chương trình 32
4.1.1 Tổng Quan 32
4.1.2 Server 32
4.1.2.1 Core 32
4.1.2.2 Servlet 34
4.1.3 Client 36
4.1.4 Giao tiếp 36
4.2 Giao diện người dùng 38
4.3 Kết quả thực nghiệm 40
4.3.1 Thực nghiệm 40
4.3.2 Đánh giá chương trình 40

Kết luận và hướng phát triển **42**

Phụ lục A Dữ liệu kiểm tra **44**

Phụ lục **44**

Tài liệu tham khảo **45**

Danh sách hình ảnh

1 Bài Toán Tối Ưu Tổ Hợp

1.1 Lời giải của bài toán 8 con hậu [1]. 11

1.2 Lời giải của bài toán tô màu đồ thị 11

2 Bài Toán Đóng Thùng 2D - Bin Packing 2D

2.1 Kích thước và tọa độ các vật phẩm 21

4 Chương Trình Minh Họa

4.1 Kiến trúc tổng quan ứng dụng 33

4.2 Giao diện khởi tạo chương trình 38

4.3 Giao diện thực hiện tìm kiếm 39

4.4 Giao diện kết quả chương trình 39

Danh sách bảng

4.1 Kết quả thực thi trên các bộ dữ liệu mẫu của chương trình với thuật toán tìm kiếm Tabu cài đặt trong lớp *SearchTabuAssign* 40

A.1 Kích thước và số lượng vật phẩm các bộ dữ liệu 44

Danh mục từ viết tắt

CBLS Constraint Base Local Search: thuật toán tìm kiếm cục bộ dựa trên ràng buộc.

JOpenCBLS Thư viện tìm kiếm cục bộ dựa trên ràng buộc được thiết kế và phát triển bởi TS. Phạm Quang Dũng và các thành viên lab Thuật toán và tối ưu - ĐH Bách Khoa Hà Nội

BP Bin Packing

BP2D Bin Packing 2D

MVP Kiến trúc Model-View-Presenter

CP Constraint Programming: Quy hoạch ràng buộc.

Chương 1

Bài Toán Tối Ưu Tổ Hợp

1.1 Tổ hợp và không gian tìm kiếm

1.1.1 Khái niệm

Bài Toán Tối ưu tổ hợp

Bài toán tối ưu tổ hợp [6] là một dạng của bài toán tối ưu, nó có dạng tổng quát như sau:

- Cho hàm $f(X)$ xác định trên một tập hữu hạn các phần tử D, hàm $f(x)$ được gọi là hàm mục tiêu.
- Mỗi phần tử X thuộc D có dạng $X = (x_1, \ldots, x_n)$ được gọi là một phương án.
- Cần tìm một phương án X thuộc D sao cho hàm $f(X)$ đạt cực tiểu, phương án X như vậy được gọi là phương án tối ưu.

Bài toán thỏa mãn ràng buộc

Bài toán thỏa mãn ràng buộc [6] là bài toán tối ưu tổ hợp bổ sung thêm các điều kiện ràng buộc được xác định bởi bộ

$$\{X, D, C, F\}$$

với:

$X = \{X_1, \ldots, X_n\}$ là tập các biến.

$D = \{D_1, \ldots, D_n\}$ là tập miền xác định tương ứng của các biến.

$C = \{C_1, \ldots, C_m\}$ là tập các ràng buộc.

$F = \{F_1, \ldots, F_k\}$ là tập các hàm mục tiêu cần tối ưu.

Tập $S = \{t_1, \ldots, t_n\}$ với phần tử t_i là giá trị tương ứng của mỗi biến X_i và S thỏa mãn tất cả các ràng buộc định nghĩa trong C thì S được gọi là tập lời giải của bài toán thỏa mãn ràng buộc trên bộ $\{X, D, C\}$.

Trong nhiều bài toán chúng ta còn cần phải tối ưu thêm các mục tiêu:

$$F_i = f(X_1, \ldots, X_n) \quad 1 \leq i \leq k. \tag{1.1}$$

Lời giải S^* trong tập các lời giải S của bài toán thỏa mãn ràng buộc trên đồng thời cho giá trị cực tiểu của các hàm mục tiêu F_i thì S^* được gọi là lời giải tối ưu của bài toán.

Không gian tìm kiếm lời giải

Tất cả các bài toán tối ưu tổ hợp đều có độ phức tạp tính toán hàm mũ, chúng thuộc lớp các bài toán NP-khó và chưa có các thuật toán hiệu quả để tìm kiếm lời giải.

1.1.2 Ví dụ

N-QUEEN

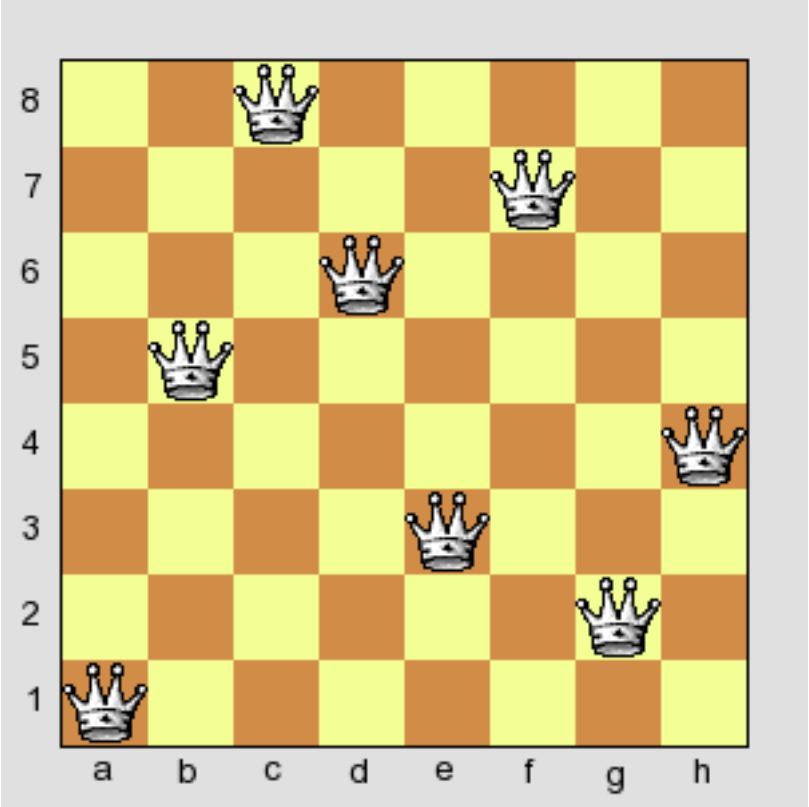
Bài toán sắp xếp $N(N > 4)$ quân hậu yêu cầu sắp xếp N quân hậu trên bàn cờ vua sao cho không có quân hậu nào có thể ăn được các quân hậu còn lại theo quy tắc cờ vua. Hay có thể phát biểu rằng không có hai con hậu nào cùng nằm tren một hàng, một cột hoặc một đường chéo.

Mô hình toán học:

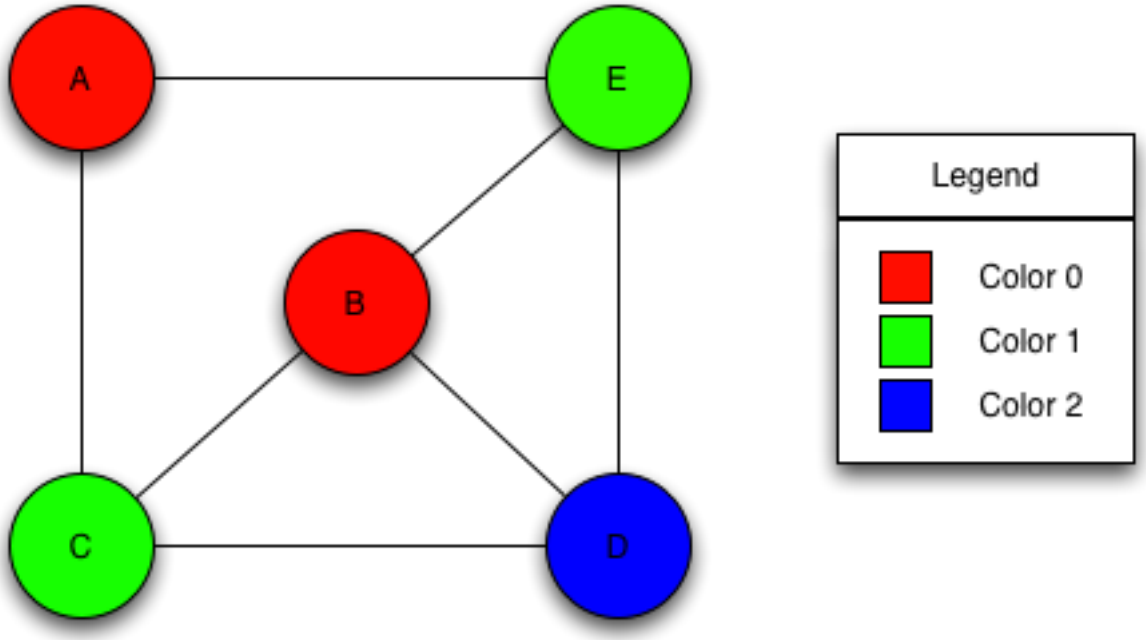
- $X = \{x_i | 1 \leq i \leq N\}$ với x_i là chỉ số cột của con hậu hàng thứ i.
- $D = \{d_i | 1 \leq i \leq N \, | d_i \in \{1, 2, \ldots, N\}\}$ là các miền giá trị của x_i.
- Tập các ràng buộc C:

$$C = \begin{cases} x_i \neq x_j, 1 \leq i < j \leq N & \text{Không có hai con hậu nào trên cùng một cột} \\ x_i - i \neq x_j - j, 1 \leq i < j \leq N & \text{Ràng buộc đường chéo thuận} \\ x_i + i \neq x_j + j, 1 \leq i < j \leq N & \text{Ràng buộc đường chéo ngược} \end{cases}$$

Hình 1.1 là một phương án thỏa mãn ràng buộc của bài toán 10 con hậu:



Hình 1.1: Lời giải của bài toán 8 con hậu [1].



Hình 1.2: Lời giải của bài toán tô màu đồ thị

Graph-Coloring

Bài toán tô màu đồ thị yêu cầu tô màu cho các đỉnh của đồ thị sao cho không có hai đỉnh liền kề nhau có cùng màu.

Mô hình toán học:

Đồ thị $G(V, E), K$ màu cho trước .

- $X = \{X_i | 1 \leq i \leq |V|\}$ là các màu tô trên các đỉnh của đồ thị.
- $D = \{D_i | 1 \leq i \leq |V|\}$ là miền giá trị cho các X_i, $D_i = \{1, \ldots, K\}$
- Tập ràng buộc C: $X_i \neq X_j$ nếu $\exists e \in E, e = (V_i, V_j)$
- Hàm mục tiêu: Cực tiểu số lượng màu cần phải tô tren đồ thị $G(V, E)$

Hình 1.2 là một lời giải cho bài toán tô màu đồ thị.

1.1.3 Một số kỹ thuật giải bài toán tối ưu tổ hợp

Quy hoạch ràng buộc [2, 6]

Là kỹ thuật cơ bản để giải quyết bài toán thỏa mãn ràng buộc, mô hình của kỹ thuật quy hoạch ràng buộc như sau:

$$CP = Model + Search$$

Trong đó:

- Model là tập các biến, miền giá trị và ràng buộc của bài toán.
- Search là thủ tục tìm kiếm lời giải dựa trên các ràng buộc, thủ tục này có thể là các thuật toán tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu…và có sử dụng các kỹ thuật cắt tỉa phù hợp làm giảm không gian tìm kiếm.

Cụ thể các bước thực hiện như sau:

- Khởi tạo mô hình tìm kiếm, không cần thiết phải khởi tạo giá trị cho các biến bộ phận.
- Trong mỗi bước lặp, các biến bộ phận được lần lượt gán giá trị trong các thủ tục đệ quy, thủ tục này sẽ đệ quy cho đến khi tất cả các biến bộ phận được gán đầy đủ giá trị phù hợp với ràng buộc của bài toán và sẽ quay lui khi tìm được một phương án hoặc là phát hiện ra hướng tiếp theo không có phương án nào phù hợp.

Kỹ thuật này có ưu điểm dễ cài đặt, ý tưởng trong sáng tuy nhiên khi gặp bài toán có không gian lời giải lớn và rất lớn thì nó tỏ ra không hiệu quả khi không thể đưa ra được một phương án dù đó là phương án tồi vì nó yêu cầu duyệt gần như toàn bộ không gian lời giải.

Tìm kiếm cục bộ [7]

Là kỹ thuật tìm kiếm lời giải dựa trên việc lặp đi lặp lại thao tác lựa chọn lời giải có chất lượng tốt hơn trong không gian lời giải lân cận, phương pháp này có nhiều ưu điểm hơn trong những bài toán có không gian tìm kiếm lớn, cụ thể về phương pháp này tôi xin trình bày ở mục tiếp theo.

1.2 Tìm kiếm cục bộ dựa trên ràng buộc

Với các bài toán tối ưu tổ hợp chúng ta luôn phải làm việc với các bộ giá trị gán cho các biến để tìm ra bộ giá trị đảm bảo thỏa mãn các ràng buộc và tối ưu hóa hàm mục tiêu. Số lượng các tập có thể của các bộ này rất lớn nên chúng ta khó có thể đưa ra một giải thuật hiệu quả để duyệt hết phương án cũng như khó có thể đưa ra một giải thuật cắt tỉa phù hợp để loại bỏ các nhánh không cần thiết.

Tiếp cận theo hướng tìm kiếm cục bộ giảm được các khó khăn nêu trên, nó không yêu cầu nhiều bộ nhớ, không yêu cầu quá nhiều thời gian tính toán. Trong tìm kiếm cục bộ ta sử dụng một số khái niệm:

- **Láng giềng** của một phương án S là các phương án $\{S_1, S_2, \ldots, S_m\}$ được sinh ra từ S bằng cách thay đổi một vài giá trị bộ phận trong phương án của S.
- **Cực tiểu địa phương** là phương án thỏa mãn các ràng buộc và có giá trị hàm mục tiêu nhỏ hơn so với các láng giềng quanh nó.

Với phương pháp tìm kiếm cục bộ, chúng ta không phải duyệt hết tất cả các bộ tổ hợp của bài toán mà thay vào đó sẽ tìm kiếm các phương án địa phương thỏa mãn các ràng buộc và cực tiểu hóa hàm mục tiêu. Từ các phương án địa phương này ta sẽ lần lượt đi đến các phương án địa phương khác tốt hơn và tiếp tục cho đến khi ta có một phương án mới là phương án tối ưu toàn cục. Sau một số bước nhất định việc tìm kiếm cục bộ sẽ đưa chúng ta đến phương án cục bộ mới tốt hơn và có thể đến được phương án tối ưu hoặc là phương án gần tối ưu mà chúng ta coi nó là chấp nhận được với yêu cầu bài toán.

Các bước của một giải thuật tìm kiếm cục bộ như sau:

- Khởi tạo: Sinh lời giải xuất phát s, tính toán các vi phạm ràng buộc, hàm mục tiêu.
- Tìm kiếm lân cận: từ lời giải hiện tại, giải thuật sinh ra tập lời giải láng giềng bằng

  $$S = N(s)$$

  cách thay đổi một vài thành phần của phương án hiện tại. Từ tập lời giải mới này ta cần tìm ra phương án s' có chất lượng tốt nhất.
- Di chuyển: sau khi tìm được phương án mới có chất lượng tốt, ta thực hiện di chuyển lời giải sang phương án mới này và cập nhật lại giá trị vi phạm và hàm mục tiêu.
- Kiểm tra: kiểm tra các điều kiện dừng, nếu thỏa mãn thì kết thúc tìm kiếm, nếu không sẽ thực hiện quay lại bước Tìm kiếm lân cận.

Các giải thuật tìm kiếm cục bộ được cải thiện đáng kể nhờ sử dụng các Heuristic để tăng cường chất lượng các lời giải láng giềng.

Giải thuật tìm kiếm cục bộ sử dụng một số phương pháp tìm kiếm sau:

- Kỹ thuật leo đồi [2] lời giải được bắt đầu từ một vị trí ngẫu nhiên và quan sát xung quanh để xem phương án nào tốt hơn quanh nó để đi đến, sau một số lần lặp, nó sẽ đưa đến một phương án tối ưu địa phương.
- Kỹ thuật tìm kiếm tabu [2] trong bước dịch chuyển của thuật toán tìm kiếm cục bộ, một phương án được phép dịch chuyển nếu như nó hoàn toàn không bị cấm. Một phương án bị cấm là phương án đã được dịch chuyển đến trước đó. Nhờ sử dụng kỹ thuật này chúng ta giảm được sự tìm kiếm xoay vòng quanh các phương án trước đó.

1.3 Thư viện JOpenCBLS

1.3.1 Tổng quan

JOpenCBLS là bộ thư viện được viết bằng ngôn ngữ JAVA do TS. Phạm Quang Dũng và các thành viên Lab thuật toán và tối ưu - Viện Công nghệ thông tin truyền thông, đại học Bách Khoa Hà Nội thiết kế và cài đặt.

Thư viện góp phần giảm công sức trong việc giải các bài toán tối ưu thỏa mãn ràng buộc bằng phương pháp tìm kiếm cục bộ. Để sử dụng hiệu quả thư viện người

dùng cần biểu diễn miền giá trị, biểu diễn lời giải và các ràng buộc thích hợp. Dựa vào các thông tin được cung cấp sẵn thư viện sẽ tự động thực hiện công việc tìm kiếm kết quả theo phương pháp tìm kiếm cục bộ. Kiến trúc của thư viện JOpenCBLS dựa trên kiến trúc của Constraint-based Local Search - CBLS - Van Hentenryck & Michel 2005. Thư viện được thiết kế và cài đặt trên ngôn ngữ JAVA do đó thư viện có được tính linh hoạt, tùy biến mở rộng và tái sử dụng của nền tảng JAVA. Điều đó giúp cho JOpenCBLS đã sẽ và đang góp phần rất tích cực trong quá trình giảng dạy, học tập và nghiên cứu của các cán bộ, sinh viên viện công nghệ thông tin trong Viện công nghệ thông tin - đại học Bách Khoa.

JOpenCBLS hướng đến việc đơn giản hóa các công việc xây dựng chương trình, cấu trúc dữ liệu và thiết kế thuật toán với bộ các thuật toán được cài đặt sẵn trong thư viện. Nó giúp cho người sử dụng có thêm thời gian tập trung nghiên cứu biểu diễn dữ liệu, mô hình hóa các ràng buộc của bài toán, cài đặt thử nghiệm các chiến lược tìm kiếm heuristic và meta heuristic để giải quyết bài toán.

Cấu trúc thư viện JOpenCBLS:

- Mô hình (Model) nhằm mô hình hóa các bài toán, biểu diễn chúng để sau đó có thể thực hiện quá trình tìm kiếm:
  - Các biến.
  - Các đại lượng bất biến.
  - Các hàm.
  - Các ràng buộc.
- Tìm kiếm (Search) chứa các thủ tục tìm kiếm được cài đặt sẵn với các giải thuật:
  - Tabu Search.
  - SA Search.

1.3.2 Các thành phần của JOpenCBLS

1.3.2.1 Mô hình

Biến (Variable)

- Biến trong JOpenCBLS là một đối tượng JAVA dùng để biểu diễn một bộ phận của phương án trong quá trình tìm kiếm, tập các biến cho ta một phương án.

- JOpenCBLS có 2 lớp cơ bản biểu diễn biến là VarInt và VarIntLS, trong đó VarInt là lớp biểu diễn cơ bản cho một biến, lớp VarIntLS kế thừa VarInt và cài đặt thêm các phương thức để thao tác trong quá trình tìm kiếm.

Bất Biến (Invariants)

- Đại lượng bất biến là các đối tượng JAVA lưu giữ mối quan hệ giữa các biến, miền giá trị, mối quan hệ giữa biến và ràng buộc. Đại lượng này không thay đổi trong quá trình thực hiện tìm kiếm lời giải.
- Trong JOpenCBLS, Invariant là một *interface*, nó cần được cài đặt lại khi cần thể hiện ràng buộc giữa các biến.

Hàm (Functions)

- Là các đối tượng biểu diễn các hàm thay đổi giá trị được phụ thuộc vào giá trị các biến trong mô hình.

Ràng buộc (Constraints)

- Là các đối tượng JAVA biểu diễn ràng buộc của bài toán, cũng giống như các hàm, nó thay đổi được giá trị và phụ thuộc vào giá trị các biến.

1.3.2.2 Tìm kiếm (Search)

JOpenCBLS có sẵn các phương thức để thao tác với biến trong quá trình tìm kiếm trong lớp MinMaxSelector:

- *MinMaxSelector(IConstraint s)*
  là hàm khởi tạo đối tượng lựa chọn trong mô hình gắn với hệ thống ràng buộc s.
- *selectMostViolatedVariable()*
  là hàm trả về đối tượng biến trong mô hình có số lượng ràng buộc bị vi phạm liên quan đến nó nhiều nhất.
- *selectMostPromissingValue(VarIntLS x)*
  trả về giá trị làm giảm số lượng vi phạm nhiều nhất trong với biến x.

JOpenCBLS cũng đã cài đặt các kỹ thuật tìm kiếm trong lớp TabuSearch để giải quyết các bài toán với các thủ tục tìm kiếm tabu như:

- *search(s, tabuLength, maxTime, maxIter, maxStable)*
  Tìm kiếm với tabu heuristic nhận tham số đầu vào:

  - *s* Hệ thống ràng buộc.
  - *tabuLength* số bước lặp bị cấm với cùng một thao tác dịch chuyển láng giềng.
  - *maxTime* thời gian chay tối đa (tính theo giây) của thủ tục tìm kiếm tabu này.
  - *maxIter* số vòng lặp (láng giềng) tối đa trong quá trình tìm kiếm.
  - *maxStable* số lượng tối đa các láng giềng đi qua nếu không tìm thấy sự cải thiện hàm mục tiêu.

- *searchAssignSwap(s, tabuLength, maxIter, maxTime)*
  Tìm kiếm tabu bằng cách xem xét cả việc gán giá trị cho các biến và tráo đổi giá trị từng cặp biến với nhau, nhận tham số đầu vào:
  - *s* hệ thống ràng buộc.
  - *tabuLength* số bước bị cấm dịch chuyển sau khi thực hiện dịch chuyển đó.
  - *maxIter* số bước lặp tối đa.
  - *maxTime* thời gian tối đa thực hiện tìm kiếm.

- *searchMaintainConstraint(f, s, maxTime, maxIter, maxStable)*
  Tìm kiếm tabu thực hiện thỏa mãn tất cả ràng buộc trong s và cực tiểu hóa hàm mục tiêu f, nhận tham số đầu vào:
  - *f* hàm mục tiêu cần cực tiểu hóa.
  - *s* hệ thống ràng buộc.
  - *maxTime* thời gian chạy tối đa của thủ tục tìm kiếm.
  - *maxIter* số lượng bước lặp tối đa.
  - *maxStable* số lần lặp tối đa để khởi động lại lời giải.

1.3.2.3 Ví dụ minh họa

Dưới đây là một ví dụ minh họa việc cài đặt bài toán N-QUEENS sử dụng JOpenCBLS:

- **Mô hình bài toán**

```
//NQueen.java

...
```

```
int n = 10000;

LocalSearchManager ls = new LocalSearchManager();
VarIntLS[] x = new VarIntLS[n];

for (int i = 0; i < n; i++){
    x[i] = new VarIntLS(ls, 0, n - 1);

ConstraintSystem s = new ConstraintSystem(ls);
s.post(new AllDifferent(x));

IFunction[] f1 = new IFunction[n];
for (int i = 0; i < n; i++)
    f1[i] = new FuncPlus(x[i], i);
s.post(new AllDifferent(f1));

IFunction[] f2 = new IFunction[n];
for (int i = 0; i < n; i++)
    f2[i] = new FuncPlus(x[i], -i);
s.post(new AllDifferent(f2));

ls.close();
...
```

Để thực hiện khởi tạo chúng ta cần khai báo một LocalSearchManager quả lý tất cả các biến, sau đó khai báo các biến trong mô hình và khởi tạo nó. Tiếp theo chúng ta cần có một hệ thống ràng buộc và thực hiện lưu giữ các ràng buộc lại trong hệ thống ràng buộc s. Ở đây, với bài toán n-queens chúng ta có 3 ràng buộc cơ bản:

- các quân hậu không cùng nằm trên cùng một cột,
- các quân hậu không cùng nằm trên cùng một đường chéo thuận,
- các quân hậu không cùng nằm trên cùng một đường chéo ngược.

Sau khi khởi tạo các biến, mô hình và tải các ràng buộc, chúng ta cần đóng mô hình để thư viện thực hiện các thủ tục khởi tạo, cập nhật hệ thống ràng buộc cần thiết.

- **Thực hiện tìm kiếm**

```
//NQueen.java
MinMaxSelector mms = new MinMaxSelector(s);
int it = 0;
while(it < 10000 && s.violations() > 0){
    VarIntLS variable = mms.selectMostViolatedVariable();
    int selectValue = mms.selectMostPromissingValue(variable);
    variable.setValuePropagate(selectValue);
    it++;
}
System.out.println("Current violations: " + s.violation());
```

Với đoạn mã thực hiện việc tìm kiếm này chúng ta có một đối tượng lựa chọn để lấy các thông tin, các đối tượng mà chúng ta yêu cầu sau đó tiếp tục thực hiện gán giá trị cho các biến chúng ta lấy được. Cụ thể ở đây ta luôn chọn đối tượng biến có số lượng vi phạm có liên quan đến nó là nhiều nhất sau đó gán cho nó giá trị mới sao cho số lượng vi phảm được giảm đi cũng là nhiều nhất, tiếp tục lặp thao tác vừa rồi cho tới khi không còn vi phạm hoặc số vòng lặp vượt quá tối đa. Kết thúc vòng lặp chúng ta có kết quả của thủ tục tìm kiếm này.

Chương 2

Bài Toán Đóng Thùng 2D - Bin Packing 2D

2.1 Phát biểu bài toán

Trong thực tế cuộc sống chúng ta rât hay gặp phải vấn đề cần giải quyết: liệu có thể xếp tất cả những vật này vào cái túi này, cái hộp này hay cái thùng kia không, lựa chọn cái nào phù hợp để vừa chứa đủ những thứ này? Các yêu cầu này nằm trong lớp bài toán đóng thùng [3].

Để mô hình hóa tốt hơn tôi sẽ xét bài toán này trong không gian 2 chiều, khi đó các vật phẩm chỉ còn lại kích thước rộng (w_i) và cao (h_i), và thùng có kích thước $W \times H$. Như vậy chúng ta có bài toán đóng thùng 2 chiều (Bin Packing 2D) với phát biểu như sau:
Cho trước một cái thùng với kích thước $W \times H$ và N vật phẩm $\{X_1, \ldots, X_n\}$, các vật phẩm X_i có kích thước tương ứng là $w_i \times h_i$ ($1 \leq i \leq N$). Ta cần đặt các vật phẩm (có thể xoay theo chiều ngang hoặc dọc) này vào các vị trí trong thùng sao cho các vật không chồng lấn lên nhau, không vượt khỏi biên của thùng đã cho. Câu hỏi đặt ra là liệu có một phương án sắp xếp nào đó thỏa mãn yêu cầu trên không và nếu có hãy đưa ra phương án sắp xếp đó.

2.2 Mô hình toán học

Mô hình cơ bản

Cho trước một cái thùng với kích thước $(W \times H)$ và N vật với kích thước tương ứng được cho bởi: $(w_1, h_1), \ldots, (w_n, h_n)$.

Với các biến:

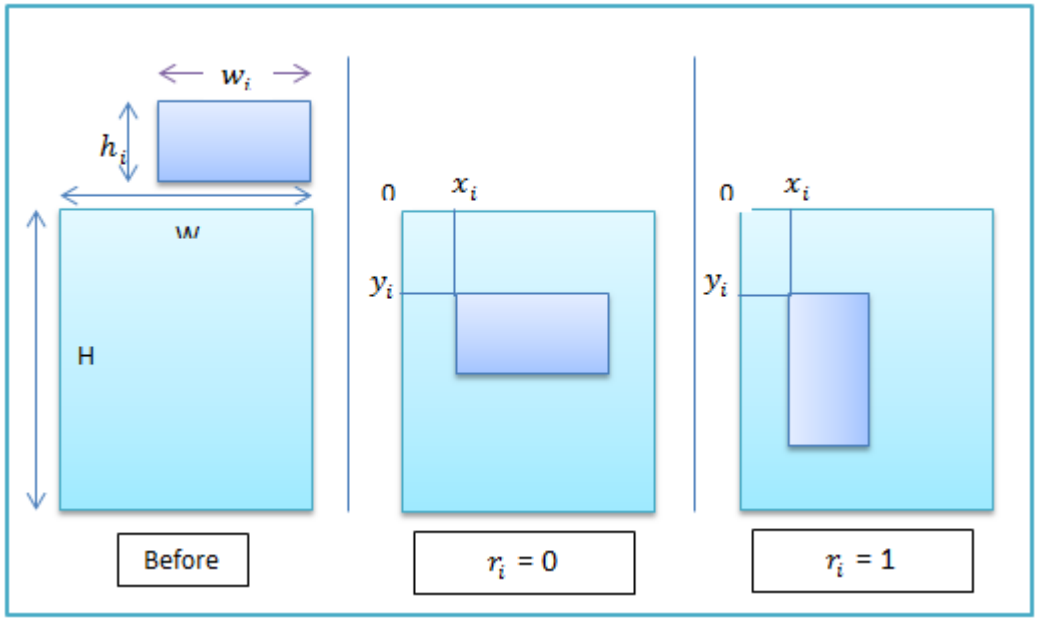
$$X_i \in \{0, \ldots, W-1\} \qquad 1 \leq i \leq N$$
$$Y_i \in \{0, \ldots, H-1\} \qquad 1 \leq i \leq N$$
$$R_i \in \{0, 1\} \qquad 1 \leq i \leq N$$

Các kích thước và tọa độ vật phẩm như hình 2.1 trong đó:

X_i là tọa độ theo phương ngang đặt vật thứ i trong thùng.

Y_i là tọa độ theo phương dọc đặt vật thứ i trong thùng.

R_i xác định vật phẩm được xoay hoặc không.



Hình 2.1: Kích thước và tọa độ các vật phẩm

Thỏa mãn các ràng buộc:

$$C_1 : \begin{cases} X_i + w_i \leq W & \text{nếu } R_i = 0 \\ X_i + h_i \leq W & \text{nếu } R_i = 1 \end{cases} \tag{2.1}$$

$$C_2 : \begin{cases} Y_i + h_i \leq H & \text{nếu } R_i = 0 \\ Y_i + w_i \leq H & \text{nếu } R_i = 1 \end{cases} \tag{2.2}$$

$$C_3 : \begin{cases} X_i + w_i \leq X_j || X_j + w_j \leq X_i || Y_i + h_i \leq Y_j || Y_j + h_j \leq Y_i & \text{nếu } R_i = 0, R_j = 0 \\ X_i + w_i \leq X_j || X_j + h_j \leq X_i || Y_i + h_i \leq Y_j || Y_j + w_j \leq Y_i & \text{nếu } R_i = 0, R_j = 1 \\ X_i + h_i \leq X_j || X_j + w_j \leq X_i || Y_i + w_i \leq Y_j || Y_j + h_j \leq Y_i & \text{nếu } R_i = 1, R_j = 0 \\ X_i + h_i \leq X_j || X_j + h_j \leq X_i || Y_i + w_i \leq Y_j || Y_j + w_j \leq Y_i & \text{nếu } R_i = 1, R_j = 1 \end{cases} \tag{2.3}$$

Với:

C_1 là ràng buộc các vật phẩm không được vượt quá biên rộng của thùng.

C_2 là ràng buộc các vật phẩm không được vượt quá biên cao của thùng.

C_3 là ràng buộc mọi vật phẩm không chồng lấn lên nhau.

Mô hình mở rộng

Ở mô hình trước chúng ta có ba mảng chứa các biến là X, Y và R, để thực hiện tìm kiếm trong mô hình tốt hơn chúng tôi đề xuất thực hiện việc gắn các biến tọa độ x và y với nhau thông qua một biến pos. Biến pos này sẽ bao gồm thông tin của hai biến x và y như sau:

$$pos = y \times W + x \tag{2.4}$$

với

pos là biến chứa thông tin tọa độ của vật trong thùng.

x là biến tọa độ theo phương ngang trong thùng như ở mô hình trước.

y là biến tọa độ theo phương dọc trong thùng như mô hình trước.

W là chiều rộng của thùng.

Từ tọa độ pos chúng ta có thể tính toán được tọa độ của các vật như sau:

$$\begin{cases} x = pos_{modeW} \\ y = pos/W \end{cases} \tag{2.5}$$

2.3 Ứng dụng

Bài toán đóng thùng có ứng dụng rộng rãi trong đóng gói, vận chuyển và bố trí linh kiện.

- Các hãng vận chuyển thường phải xếp các vật phẩm vào các container, kích thước container là cố định nên yêu cầu cần phải xếp sao cho tiết kiệm diện tích nhất số lượng các vật phẩm cần vận chuyển.
- Các công ty chuyển phát nhanh cần đóng gói các gói hàng trước khi đưa lên máy bay, kích thước cho phép trên máy bay là giới hạn nên yêu cầu cần phải sắp xếp một cách tiết kiệm nhất.
- Phân chia các khu vực bố trí linh kiện điện tử cũng rất quan trọng bên cạnh việc nghiên cứu làm giảm kích thước linh kiện khi các thiết bị ngày nay yêu cầu nhỏ gọn mà vẫn đảm bảo hiệu năng cao, an toàn cho người sử dụng.
- . . .

Chương 3

Áp dụng thư viện JOpenCBLS vào bài toán BP2D

Trong chương này tôi sẽ trình bày cách cài đặt thuật toán tìm kiếm cục bộ dựa trên ràng buộc vào bài toán đóng thùng 2d đã được trình bày ở chương 2.

3.1 Các chức năng cài đặt thêm

Để cài đặt với mô hình mở rộng của bài toán đóng thùng 2D đã trình bày ở mục trước, chúng tôi đề xuất cài đặt thêm một số lớp có các chức năng sau:

3.1.1 Phép chia nguyên (Function)

Phép chia nguyên là phép toán được định nghĩa trên vành số nguyên, nó bao gồm hai toán tử là toán tử chia và toán tử bị chia. Trong bài toán BP2D tôi chỉ thao tác chính trên số nguyên nên việc sử dụng thêm phép chia nguyên là cần thiết. Cụ thể trong BP2D để gắn liền tọa độ theo phương x và theo phương y của vật phẩm tôi sử dụng một biến trong mô hình lưu giữ một đại lượng

$$posCombine_i = x_i \times W + y_i \quad 1 \leq i \leq N$$

Nó cho phép chúng ta có thêm nhiều láng giềng hơn trong quá trình duyệt tìm kiếm. Việc lấy giá trị tọa độ y_i được thực hiện đơn giản bằng một phép chia nguyên.

Phép chia nguyên được cài đặt bởi lớp Devide là một định nghĩa hàm, nó nhận khởi tạo là một biến và một số nguyên cố định. Giá trị của hàm này chỉ phụ thuộc vào giá trị của biến tham số truyền vào hàm khởi tạo.

```
// Divide.java
```

```
...
public Divide(VarIntLS var, int mod) {
    this.variable = var;
    this.mod = mod;

    this.manager = var.getLocalSearchManager();

    manager.post(this);
}
...
```

trong đó:

- *var* là tham chiếu đến đối tượng VarIntLS chứa giá trị bị chia.
- *mod* là số nguyên chứa giá trị số chia.

3.1.2 Phép lấy phần dư (Function)

Bên cạnh phép chia nguyên là phép lấy phần dư (module) cũng được định nghĩa trên vành số nguyên gồm hai toán tử là toán tử lấy mô-đun và toán tử mô-đun. Trong BP2D tôi sử dụng toán tử này để lấy lại giá trị tọa độ x_i của vật phẩm trong túi từ đại lượng $posCombine_i$.

Phép lấy phần dư được cài đặt bởi lớp Module là một định nghĩa hàm, nhận khởi tạo là một biến và một số nguyên cố định, giá trị của hàm này là kết quả phép chia lấy phần dư của giá trị biến truyền vào và số nguyên đó.

```
// Module.java
...
public Module(VarIntLS var, int mod) {
    this.variable = var;
    this.mod = mod;

    this.manager = var.getLocalSearchManager();
    manager.post(this);
}

...
```

trong đó:

- *var* là tham chiếu đến đối tượng VarIntLS chứa giá trị lấy phần dư.

- *mod* là số nguyên chứa giá trị số chia.

3.1.3 Các lớp có chức năng tìm kiếm

Ngoài các lớp có sẵn trong thư viện JOpenCBLS tôi có cài đặt thử nghiệm thêm các lớp thực hiện các phương thức tìm kiếm khác nhau nhằm phù hợp hơn với mô hình và kiến trúc của ứng dụng.

3.1.3.1 Tabu Search

Tabu search là một kỹ thuật sử dụng phổ biến và rất hiệu quả trong các bài toán tìm kiếm cục bộ, ở đây tôi cài đặt các lớp thực hiện các công việc tìm kiếm *tabu* trong các mô hình của JOpenCBLS. Các lớp tìm kiếm này sẽ cài đặt một *interface* là SearchMethod chứa các phương thức điều khiển:

- *run()* sẽ bắt đầu quá trình tìm kiếm.
- *stop()* sẽ kết thúc tiến trình tìm kiếm hiện tại.

Khi bắt đầu một tiến trình tìm kiếm thông thường các lớp sẽ gọi phương thức tìm kiếm tương ứng của mình đã được cài đặt:

- SearchTabuAssign là lớp cài đặt phương thức tìm kiếm tabu bằng cách xem xét láng giềng là các bộ tổ hợp mới có được từ bộ tổ hợp cũ bằng cách thay đổi duy nhất một thành phần trong nó. Đối tượng của lớp này sẽ thực hiện tìm kiếm trong phương thức *searchAssign()*.
- SearchTabuSwap là lớp cài đặt phương thức tìm kiếm tabu bằng cách xem xét láng giềng là các bộ tổ hợp mới có được bằng cách tráo đổi giá trị của đúng 2 thành phần trong bộ tổ hợp cũ. Đối tượng của lớp này sẽ thực hiện tìm kiếm trong phương thức *searchSwap()*.
- SearchTabuMixAssignSwap là lớp cài đặt phương thức tìm kiếm tabu bằng cách xem xét láng giềng là hợp của các láng giềng trong tìm kiếm với SearchTabuAssign và SearchTabuSwap. Đối tượng của lớp này sẽ thực hiện tìm kiếm trong phương thức *searchMix()*.

Mỗi lớp tìm kiếm này có thể chạy độc lập trong môi trường đa luồng, chúng ta có thể khởi tạo nó bằng một luồng mới và thực thi công việc tìm kiếm độc lập, việc này giúp cho phần ứng dụng trở nên thông suốt hơn. Các tiến trình mới này có thể bị hủy trước khi hoàn thành công việc tìm kiếm nếu như chúng ta gọi thủ tục kết thúc (*stop()*) từ một tiến trình khác.

3.1.3.2 Random Search

Lớp RandomSearch là một lớp có chức năng tìm kiếm với phương thức tìm kiếm ngẫu nhiên, lớp này cài đặt với mục đích thử nghiệm.

3.2 Dữ liệu và mô hình hóa

Mô hình của bài toán được đặt hoàn toàn trong lớp BpModelCombine, nó kế thừa một *interface* là SearchModel, cài đặt *interface* này cho phép các phương thức tìm kiếm ở trên có thể sử dụng model này trong quá trình tìm kiếm, tạo sự độc lập giữa cài đặt phương thức tìm kiếm và cài đặt mô hình.

3.2.1 Dữ liệu

Trong lớp BpModelCombine bao gồm:

- **Các thuộc tính**

```
//BpModelCombine.java
private LocalSearchManager manager;
private ConstraintSystem cs;

private SearchEventPool pool;

private BpData data;
private int itemCount;
private int binWidth;
private int binHeight;
private int[] itemWidths;
private int[] itemHeights;
```

Có tác dụng như sau:

  - *manager* là đối tượng quản lý toàn bộ mô hình.
  - *cs* là đối tượng quản lý hệ thống ràng buộc.
  - *pool* là đối tượng giữ lại các mốc trong quá trình tìm kiếm.
  - *data* là đối tượng chịu trách nhiệm đọc dữ liệu từ tệp tin và trích xuất ra các thông tin cần thiết.
  - *itemCount* là số lượng vật phẩm (N).

- *binWidth* là chiều rộng của thùng (W).
- *binHeight* là chiều cao của thùng (H).
- *itemWidths* là mảng lưu trữ chiều rộng của các vật phẩm: $[w_1, \ldots, w_N]$.
- *itemHeights* là mảng lưu trữ chiều cao của các vật phẩm: $[h_1, \ldots, h_N]$.

- **Biến mô hình**

```
//BpModelCombine.java

private VarIntLS[] combinedPos;
private VarIntLS[] rotated;
```

có tác dụng:

- *combinedPos* là đối tượng lưu giữ giá trị thể hiện vị trí của vật phẩm trong thùng như đã trình bày ở mục trước:

$$combinedPos[i] = Y_i \times W + X_i \quad \text{với } 1 \leq i \leq N$$

- *rotated* là đối tượng lưu giữ trạng thái hiện thời của vật phẩm: được xoay hay không được xoay.

$$\begin{cases} rotated[i] = 1 & \text{nếu vật thứ } i \text{ đã được xoay} \\ rotated[i] = 0 & \text{nếu vật thứ } i \text{ không được xoay} \end{cases}$$

3.2.2 Mô hình

3.2.2.1 Khai báo và cấp phát bộ nhớ

Các thao tác khai báo và cấp phát bộ nhớ được thực hiện trong phạm vi của thủ tục:

```
//BpModelCombine.java
...
private void allocateVariable(){
    final int COMBINE_MAX = (binWidth - 1) * binHeight;
    combinedPos = new VarIntLS[itemCount];
    rotated = new VarIntLS[itemCount];
    for(int i = 0; i< itemCount; i++){
        combinedPos[i] = new VarIntLS(manager, 0, COMBINE_MAX - 1);
```

```
            rotated[i] = new VarIntLS(manager, 0, 1);
        }
    }
    ...
```

bao gồm:

- khai báo và khởi tạo giá trị cho các phần tử của mảng $combinedPos$,
- khai báo và khởi tạo giá trị cho các phần tử của mảng $rotated$.

3.2.2.2 Tải các ràng buộc

Thao tác khai báo các ràng buộc được thực hiện trong thủ tục *loadConstraints()*:

- Tính toán vị trí tọa độ x, y:

```
// Calculate x position and y position of all items
IFunction[] left = new IFunction[itemCount];
IFunction[] top = new IFunction[itemCount];
for(int i = 0; i< itemCount; i++){
    left[i] = new Module(combinedPos[i], binWidth);
    top[i] = new Divide(combinedPos[i], binWidth);
}
```

- Tính toán tọa độ biên của các hộp trong thùng:

```
// Calculate bound of all item
IFunction[] right = new IFunction[itemCount];
IFunction[] bottom = new IFunction[itemCount];
IFunction[] rightRot = new IFunction[itemCount];
IFunction[] bottomRot = new IFunction[itemCount];
for(int i = 0; i< itemCount; i++){
    right[i] = new FuncPlus(left[i], itemWidths[i]);
    bottom[i] = new FuncPlus(top[i], itemHeights[i]);
    rightRot[i] = new FuncPlus(left[i], itemHeights[i]);
    bottomRot[i] = new FuncPlus(top[i], itemWidths[i]);
}
```

- Tải ràng buộc không chồng lấn giữa các vật:

```
// All items must not overlap
for(int i = 0; i< itemCount; i++){
    for(int j = i + 1; j< itemCount; j++){
        IConstraint[] allCase = new IConstraint[4];
        // Case 0 item 1 NOT rotated, item 2 NOT rotated
        IConstraint[] tmp = new IConstraint[4];
        tmp[0] = new LessOrEqual(right[i], left[j]);
        tmp[1] = new LessOrEqual(right[j], left[i]);
        tmp[2] = new LessOrEqual(bottom[i], top[j]);
        tmp[3] = new LessOrEqual(bottom[j], top[i]);
        IConstraint[] tmpCase = new IConstraint[3];
        tmpCase[0] = new OR(tmp);
        tmpCase[1] = new IsEqual(rotated[i], 0);
        tmpCase[2] = new IsEqual(rotated[j], 0);
        allCase[0] = new AND(tmpCase);
        // Case 1 item 1 NOT rotated, item 2 rotated
        ...
        allCase[1] = new AND(tmpCase);
        // Case 2 item 1 rotated, item 2 NOT rotated
        ...
        allCase[2] = new AND(tmpCase);
        // Case 3 item 1 rotated, item 2 rotated
        tmp = new IConstraint[4];
        ...
        allCase[3] = new AND(tmpCase);

        // Combine all case
        cs.post(new OR(allCase));
    }
}
```

- Tải ràng buộc vật phẩm không vượt quá phạm vi của thùng:

```
// All items must be not overlap with bin-bound
for(int i = 0; i< itemCount; i++){
    IConstraint[] allCase = new IConstraint[2];
    // Case 0 : item NOT rotated
    IConstraint[] tmp = new IConstraint[3];
```

```
            tmp[0] = new LessOrEqual(right[i], binWidth);
            tmp[1] = new LessOrEqual(bottom[i], binHeight);
            tmp[2] = new IsEqual(rotated[i], 0);
            allCase[0] = new AND(tmp);
            // Case 1 : item rotated
            tmp = new IConstraint[3];
            tmp[0] = new LessOrEqual(rightRot[i], binWidth);
            tmp[1] = new LessOrEqual(bottomRot[i], binHeight);
            tmp[2] = new IsEqual(rotated[i], 1);
            allCase[1] = new AND(tmp);

            // Combine all case
            cs.post(new OR(allCase));
        }
```

3.2.2.3 Thực hiện tìm kiếm

Chúng ta sẽ thực hiện tìm kiếm cục bộ bằng cách: sử dụng một đối tượng tìm kiếm (SearchTabuSwap, RandomSearch,...) khởi tạo với mô hình vừa được khai báo gọi thủ tục *run()*.

Chương 4

Chương Trình Minh Họa

4.1 Mô hình chương trình

Chương trình minh họa được cài đặt trên môi trường ứng dụng WEB sử dụng các công nghệ **Servlet, Bootstrap, Javascript AngularJS**.

4.1.1 Tổng Quan

Hình 4.1 mô tả kiến trúc tổng quan của ứng dụng web cho bài toán đóng thùng 2D, chương trình gồm 3 phần:

- **Core** chứa các lớp xử lý nghiệp vụ của bài toán đóng thùng, bao gồm đọc ghi dữ liệu, mô hình hóa và thực hiện tìm kiếm.
- **Servlet** chứa các lớp xử lý yêu cầu từ phía trình duyệt web của người dùng, nó xử lý các thao tác điều khiển của người dùng và cũng là nơi trả lại các thông tin cần thiết về cho client.
- **Client** là ứng dụng ở phía người dùng xử lý các thông tin trả về từ servlet và hiển trị lên màn hình trình duyệt web.

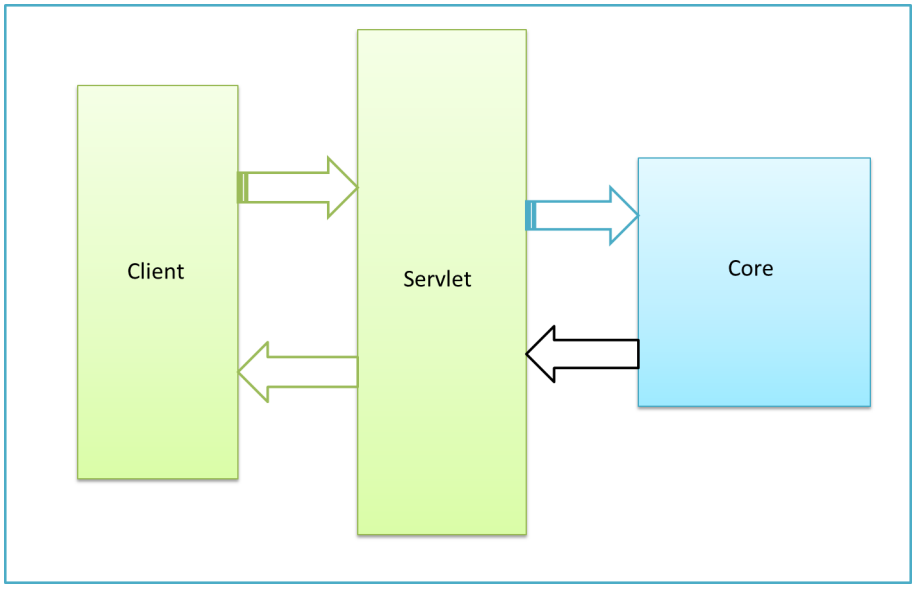
Cụ thể các phần của chương trình như sau:

4.1.2 Server

4.1.2.1 Core

Phần **Core** bao gồm:

- Các lớp xử lý việc đọc dữ liệu từ hệ thống tệp tin của Server, các lớp này nằm trong gói *binpacking.io*



Hình 4.1: Kiến trúc tổng quan ứng dụng

– BpData chứa các thông tin của tệp tin dữ liệu, mỗi đối tượng của lớp này chứa đầy đủ thông tin của một tệp tin dữ liệu của BP2D: tệp tin (*file*), chiều rộng thùng (*binWidth*), chiều cao thùng (*binHeight*), số lượng vật phẩm (*itemCount*), chiều rộng các vật phẩm (*itemHeights*), chiều cao các vậy phẩm (*itemHeights*). Ngoài ra các đối tượng của lớp này chỉ thực sự đọc dữ liệu khi cần thiết, điều này giúp cải thiện hiệu năng hệ thống, tiết kiệm tài nguyên máy chủ.

```
//BpData.java

private File file;
private boolean isReady;
private int binWidth;
private int binHeight;
private int itemCount;

private int[] itemWidths;
private int[] itemHeights;
```

– BpDataManager quản lý toàn bộ các đối tượng BpData, bảo đảm mỗi đối tượng BpData luôn chỉ cần khởi tạo một lần và có thể sử dụng cho nhiều mô hình khác nhau. Đối tượng của lớp này sẽ là nơi cung cấp dữ liệu cho các lớp yêu cầu xử lý mô hình, tìm kiếm….

- Lớp mô hình hóa cho bài toán BpModelCombine nằm trong gói *binpacking2d.model* có tác dụng mô hình bài toán đóng thùng vào thư viện JOpenCBLS.
- Các lớp khởi tạo phương án bắt đầu cho bài toán, mỗi lớp khởi tạo sẽ có một cách thức khởi tạo riêng cho các mô hình tương ứng. Với bài toán BP2D có các cách khởi tạo tương ứng với các lớp:
  - BpAllZeroInitMethod khởi tạo phương án bằng cách xếp tất cả các vật phẩm vào vị trí góc của thùng và không có vật phẩm nào được xoay.
  - BpRandomInitMethod khởi tạo một cách ngẫu nhiên vị trí các vật phẩm, chúng cũng có thể được xoay hoặc không và hoàn toàn ngẫu nhiên.

  Ngoài ra các đối tượng của các lớp khởi tạo này cũng được quản lý bởi đối tượng của lớp BpInitMethodManager, đối tượng này sẽ chỉ cho hệ thống những lớp khởi tạo nào được phép sử dụng và khởi tạo sẵn các lớp khởi tạo.
- Chương trình còn cài đặt thêm một số lớp có chức năng tương tự như trong thư viện JOpenCBLS có tác dụng thử nghiệm và phù hợp hơn với chương trình đa luồng.

4.1.2.2 Servlet

Các Servlet là các lớp nằm trong gói *binpacking2d.servlet* có vai trò giao tiếp với client, bao gồm:

- **control - ControlServlet** là Servlet chịu trách nhiệm nhận các thông điệp điều khiển từ phía người dùng sau đó nó tương tác với phần Core để thực hiện các tác vụ này. ControlServlet xử lý các thông điệp từ các phương thức GET của http:
  - *control?action=init&initMode=<modeId>&fileId=<fileId>* khởi tạo mô hình bằng chỉ số của thủ tục tìm kiếm và chỉ số của tệp tin trên hệ thống, các chỉ số định danh này được cố định và được cung cấp bởi các quản lý tương ứng: SearchMethodManager, BpDataManager.
  - *control?action=start* bắt đầu tiến trình tìm kiếm cục bộ đã được khởi tạo trước đó, nếu chưa được khởi tạo trước thì thao tác này sẽ không thực hiện tìm kiếm.
  - *action=stop* sẽ dừng tiến trình tìm kiếm đang thực thi trên máy chủ.
- **events - EventsServlet** sẽ liệt kê tất cả các sự kiện diễn ra trên máy chủ trong quá trình tìm kiếm, các sự kiện được trả dưới dạng JSON cấu trúc như sau:

```
{
    "state": {
        "running": RUNNING_VALUE,
        "ready": READY_VALUE
    }
    "events":[
        {
            "violations": VIOLATION,
            "name": EVENT_NAME,
            "items": LIST_ITEMS
        },
        ...
    ]
}
```

- **fetch - ResourceServlet** là Servlet chịu trách nhiệm xử lý các yêu cầu lấy thông tin tài nguyên từ người dùng, nó cung cấp các loại tài nguyên:
  - *fetch?res=files* khi nhận được yêu cầu này nó sẽ trả về dưới dạng JSON các tệp tin dữ liệu của bài toán BP2D đang có trên thư mục dữ liệu của máy chủ. Thông điệp trả về sẽ có dạng JSON:

```
[
    {
        "name": FILE_NAME,
        "id" : FILE_ID
    },
    ...
]
```

  - *fetch?res=search* cung cấp các thuật toán tìm kiếm đã được cài đặt trong ứng dụng của máy chủ, thông tin trả về dưới dạng JSON:

```
[
    SEARCH_METHOD,
    ...
]
```

  - *fetch?res=init* cung cấp các thuật toán khởi tạo dưới dạng JSON:

```
[
    INIT_METHOD,
    ...
]
```

4.1.3 Client

Chương trình phía client được cài đặt sử dụng công nghệ AngularJS và Bootstrap:

- AngularJS là một framework javascript được phát triển bởi các kỹ sư của Google, nó thay đổi cách các lập trình viên làm việc với javascript, cung cấp các cách thức lập trình nhanh, logic, hiệu quả trên các ứng dụng web.
- Bootstrap3 là một thư viện css3 nhằm làm cho việc thiết kế các trang web trở nên đơn giản, chuẩn mực.

Trong ứng dụng phía client của bài toán BP2D sử dụng các thành phần:

- *bin.js* định nghĩa chỉ thị của AngularJS, khi nó được khai báo trên html canvas, nó sẽ vẽ ra hình ảnh để mô tả vị trí của các vật phẩm trong thùng, nếu các vật chồng lấn lên nhau nó sẽ thông báo bằng cách vẽ viền của nó màu đỏ.
- *poller.js* khai báo một dịch vụ mới trên AngularJS, dịch vụ này được AngularJS thực thi khi khởi tạo và nó được thực thi trong suốt quá trình ứng dụng phía client được mở. Khi được hoạt động, nó sẽ định kỳ lấy các sự kiện từ trên máy chủ về và dùng các hàm xử lý được khai báo trước để xử lý các dữ liệu sự kiện này. Thông thường các hàm xử lý sẽ cập nhật hiển thị các vật nằm trong thùng.
- *index.js* định nghĩa ứng dụng AngularJS, thiết lập các hàm xử lý sự kiện cho *poller* và thực hiện các thao tác khởi tạo cần thiết.
- *index.html* là trang html chính của ứng dụng BP2D, nó GET từ Server và được cập nhật liên tục khi các sự kiện xảy ra.

4.1.4 Giao tiếp

Khi máy chủ tiếp nhận một yêu cầu từ phía client, nó sẽ gửi về tệp tin *index.html* cho client để thực thi trên trình duyệt web. Khi bắt đầu ứng dụng web phía client sẽ thực hiện lấy các thông tin tài nguyên từ máy chủ như các tệp tin, các thủ tục khởi tạo,

các thuật toán tìm kiếm và đồng thời nó cũng yêu cầu máy chủ khởi tạo sẵn với một số tham số mặc định. Với ứng dụng web phía client này, người dùng có thể thực hiện các thao tác:

- Lựa chọn tệp tin dữ liệu: ứng dụng cho phép người dùng lựa chọn tệp tin dữ liệu thông qua bảng chọn. Khi người dùng lựa chọn xong tệp tin dữ liệu, client sẽ yêu cầu server thực hiện luôn thao tác khởi tạo tương ứng.
- Lựa chọn thủ tục khởi tạo: khi người dùng thay đổi thủ tục khởi tạo chương trình cũng sẽ tự động gửi yêu cầu khởi tạo lên server và tự động cập nhật trạng thái lên màn hình.
- Lựa chọn thuật toán tìm kiếm: chương trình cung cấp nhiều lựa chọn tìm kiếm cho bài toán, các thuật toán được cài đặt một cách tổng quát và có thể dùng cho nhiều mô hình.
- Bắt đầu tìm kiếm: Người sử dụng được cung cấp một nút bấm để thực hiện tìm kiếm sau khi các thao tác khởi tạo được hoàn thành. Khi người sử dụng thực hiện thao tác này, client sẽ gửi một thông điệp điều khiển lên server yêu cầu bắt đầu thuật toán tìm kiếm.
- Kết thúc: Trong quá trình thực thi tìm kiếm, người sử dụng có thể dừng tiến trình này bằng cách bấm nút kết thúc, client sẽ gửi một thông điệp điều khiển kết thúc lên server và ngay lập tức server sẽ dừng công việc tìm kiếm.

Ngoài ra trong quá trình thực hiện tìm kiếm, client luôn luôn gửi các yêu cầu lấy danh sách sự kiện diễn ra trên server để cập nhật lên màn hình trình duyệt, các yêu cầu này được gửi một cách định kỳ.

4.2 Giao diện người dùng

Khởi tạo dữ liệu

Hình 4.2: Giao diện khởi tạo chương trình

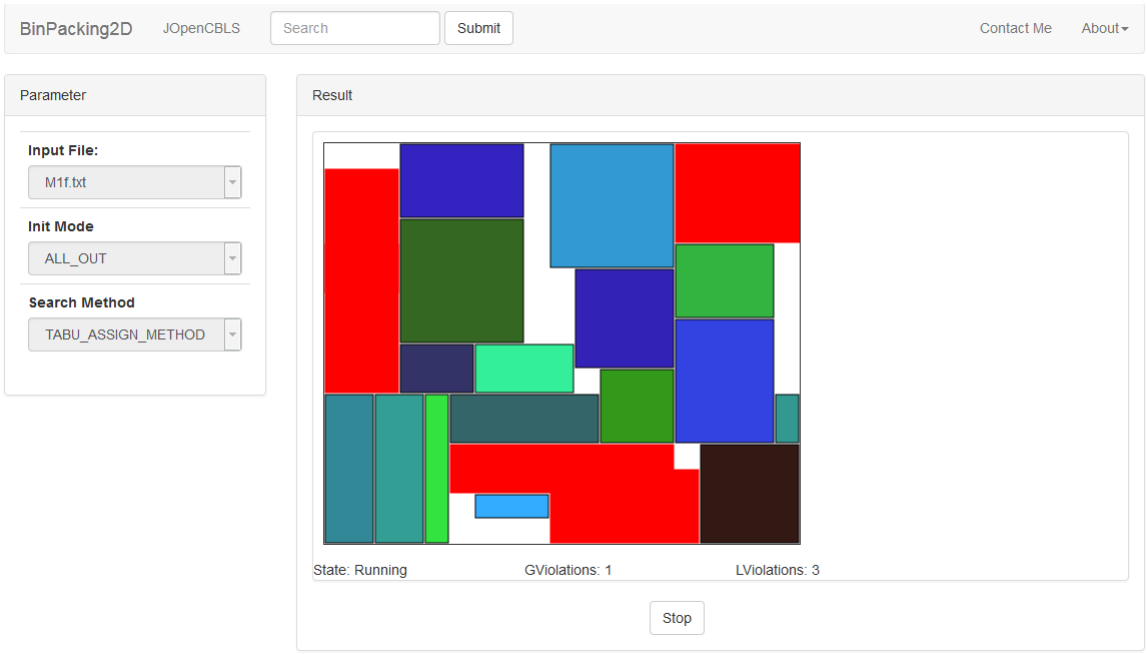
Khi ứng dụng được khởi chạy, khi người dùng lựa chọn thay đổi bộ dữ liệu hoặc khi thay đổi thủ tục khởi tạo thì chương trình sẽ thực hiện khởi tạo mặc định các tham số và hiển thị lên màn hình như hình 4.2. Chương trình sẽ hiển thị các vật phẩm được xếp vào thùng không thỏa mãn các ràng buộc bởi màu đỏ, các vật thỏa mãn các ràng buộc sẽ được hiển thị bởi một màu nhẹ ngẫu nhiên.

Thực hiện tìm kiếm

Trong khi thực hiện tiến trình tìm kiếm, màn hình sẽ có dạng như hình 4.3, nó sẽ thống kê số vi phạm hiện tại và số vi phạm tốt nhất tìm thấy trên hệ thống. Giao diện sẽ thay đổi theo trạng thái tìm kiếm hiện tại trên máy chủ.

Hiển thị kết quả

Hình 4.4 là giao diện sau khi thực hiện chương trình.



Hình 4.3: Giao diện thực hiện tìm kiếm

Hình 4.4: Giao diện kết quả chương trình

4.3 Kết quả thực nghiệm

4.3.1 Thực nghiệm

Kết quả thực tế chạy trên các bộ dữ liệu mẫu được thống kê trong bảng 4.1

Bộ DL[1]	RANDOM[2]		ALLZERO[2]	
	Violations	Time(m)	Violations	Time (m)
M0a.txt	0	1	0	<1
M0b.txt	0	10	0	7
M1a.txt	0	133	0	19
M1b.txt	0	243	0	89
M1c.txt	0	102	0	267
M1d.txt	0	156	0	78
M1e.txt	0[3]	378	0[3]	326
M1f.txt	1	?	1	?

[1] Các bộ dữ liệu được cho trong A.1

[2] Các số liệu được lấy từ giá trị trung bình 10 lần chạy trên mõi bộ dữ liệu bao gồm số bước lặp và thời gian thực thi.

[3] Mục này chứa một vài lần thực thi cho kết quả vi phạm khác 0.

[*] Kết quả thống kê được thực hiện trên máy tính intel với tốc độ xử lý 3.1GH với nền tảng Java 8.

Bảng 4.1: Kết quả thực thi trên các bộ dữ liệu mẫu của chương trình với thuật toán tìm kiếm Tabu cài đặt trong lớp *SearchTabuAssign*

Các kết quả thống kê cho thấy bên cạnh thuật toán sẵn có trong thư viện JOpenCBLS thì các thuật toán cài đặt thêm đã cải thiện khá tốt khả năng tìm kiếm trong bài toán BP2D.

4.3.2 Đánh giá chương trình

- **Ưu điểm**

  - Giải quyết bài toán đóng thùng 2 chiều một cách có hiệu quả.
  - Giao diện phầm mềm trực quan, dễ thao tác.

- Áp dụng nhiều công nghệ mới như AngularJS, Bootstrap, Java servlet, kiến trúc MVP để xây dựng chương trình.
- Sử dụng được tính mềm dẻo, tái sử dụng của thư viện JOpenCBLS.

- **Nhược điểm**
  - Chưa thực hiện và giải quyết trên bộ dữ liệu lớn.

Kết luận và hướng phát triển

Kết luận

Đồ án tốt nghiệp với đề tài Nghiên cứu và áp dụng thuật toán tìm kiếm cục bộ dựa trên ràng buộc để giải bài toán đóng thùng đã hoàn thành các mục tiêu yêu cầu:

- Nghiên cứu thuật toán tìm kiếm cục bộ để giải các bài toán thỏa mãn ràng buộc.
- Tìm hiểu và sử dụng thành thạo các thành phần của thư viện JOpenCBLS.
- Mô hình hóa và giải bài toán đóng thùng ở mức 2 chiều.
- Xây dựng ứng dụng Web sử dụng công nghệ JAVA và các công nghệ phía browser khác.
- Cài đặt các thuật toán tìm kiếm khác ngoài các thuật toán cài đặt sẵn để chạy thử, so sánh hiệu năng.
- Mở rộng thư viện JOpenCBLS để sử dụng tốt hơn trong các ứng dụng Web.

Hướng phát triển

Đồ án đã nghiên cứu thành công phương pháp giải bài toán đóng thùng với phương pháp tìm kiếm cục bộ dựa trên ràng buộc sử dụng thư viện JOpenCBLS của TS. Phạm Quang Dũng. Nhưng trong phạm vi đồ án tôi không phân tích sử dụng được hết mọi kỹ năng của thuật toán tìm kiếm cục bộ dựa trên ràng buộc cũng như không thể nghiên cứu rộng hơn về lớp các bài toán đóng thùng. Sau đây là những hướng mở rộng nghiên cứu của đồ án này:

- Nghiên cứu thêm các Heuristic tìm kiếm mới hiệu quả hơn, góp phần cải thiện hiệu năng tìm kiếm trong các bài toán tìm kiếm cục bộ dựa trên ràng buộc.
- Phát triển đầy đủ bộ công cụ mô hình cho thư viện JOpenCBLS.

- Cải thiện hiệu quả của thuật toán đã cài đặt để có thể sử dụng tốt hơn cho các bộ dữ liệu lớn hơn.
- Mở rộng bài toán đóng thùng 2 chiều sang bài toán n chiều và thỏa mãn thêm nhiều ràng buộc hơn:
  - Ràng buộc thứ tự đặt các vật vào thùng.
  - Ràng buộc yêu cầu xếp cạnh nhau các vật.
- Phát triển bài toán đóng thùng với yêu cầu mới: lựa chọn số lượng các vật cho vào thùng để tối ưu một hay nhiều mục tiêu nào đó.

Tiếp tục nghiên cứu các kỹ thuật nhằm tăng hiệu quả của thuật toán, giải quyết tốt hơn các bộ dữ liệu lớn.

Phụ lục A

Dữ liệu kiểm tra

Bộ Dữ liệu	Chiều rộng BIN	Chiều cao BIN	Số vật phẩm
M0a.txt	9	7	6
M0b.txt	16	10	10
M1a.txt	18	19	15
M1b.txt	23	23	18
M1c.txt	23	24	18
M1d.txt	24	25	19
M1e.txt	20	21	19
M1f.txt	19	16	21

Bảng A.1: Kích thước và số lượng vật phẩm các bộ dữ liệu

Bảng A.1 liệt kê các thông tin đầu vào của các bộ dữ liệu sử dụng để đánh giá chương trình ứng dụng.

Tài liệu tham khảo

[1] Paul Butler. N-queens in a tweet. 2009.

[2] P. Van Beek F. Rossi and T. Walsh. Handbook of Constraint Programming, 2006.

[3] E. Hopper. Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. 2000.

[4] Vigo D. Lodi A., Martello S. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, 1999.

[5] van Vuuren J. H. Ortmann F. G., Ntene N. New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems, 2010.

[6] Lauren Michel Pascal van Hentenryck. *Constraint-Based Local Search*. PhD thesis, 2006.

[7] Dung Pham-Quang. A java library for constraint-based local search. 2015.

[8] Sigurd M. Pisinger D. The two-dimensional bin packing problem with variable bin sizes and costs, 2005.