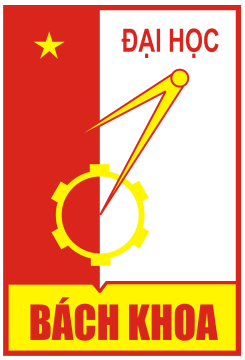
ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA TOÁN - TIN



ỨNG DỤNG DEEP REINFORCEMENT LEARNING GIẢI BÀI TOÁN NGƯỜI DU LỊCH ĐA MỤC TIÊU

ĐỒ ÁN I

Chuyên ngành: Toán - Tin

Chuyên sâu: Tin học

Giảng viên hướng dẫn: TS. Đoàn Duy Trung

Sinh viên thực hiện: Doãn Chí Thường

Mã số sinh viên: 20210831

HÀ NỘI – 2024

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

1. Mục tiêu và nội dung của đồ án

(a) Mục tiêu:

(b) Nội dung:

2. Kết quả đạt được

(a)

(b)

(c)

3. Ý thức làm việc của sinh viên:

(a)

(b)

(c)

Hà Nội, ngày ... tháng ... năm 2024

Giảng viên hướng dẫn

TS. Đoàn Duy Trung

Tóm tắt nội dung Đồ án

1. **Chương 1: Cơ sở lý thuyết** sẽ trình bày các kiến thức cơ bản liên quan đến tối ưu đa mục tiêu, kiến trúc *Neural Network*, lý thuyết về *Reinforcement Learning* cũng như trình bày về các bài toán TSP, MOTSP và một phương pháp đã được áp dụng để giải 2 bài toán trên là các giải thuật tiến hóa đa mục tiêu.

2. **Chương 2: Ứng dụng Deep Reinforcement Learning giải bài toán MOTSP** sẽ trình bày về kiến trúc mạng *Pointer Network* cũng như quy tắc huấn luyện *Actor-Critic* trong *Reinforcement Learning* được sử dụng để tối ưu mạng.

3. **Chương 3: Cài đặt chương trình và trình bày kết quả thử nghiệm** sẽ tiến hành cài đặt chương trình kiểm thử và so sánh 2 cách tiếp cận để giải bài toán MOTSP là DRL-MOA và thuật toán tiến hóa đa mục tiêu NGSA-II.

Dù rất cố gắng tìm hiểu và hoàn thiện. Tuy nhiên, những hạn chế là không thể tránh khỏi. Vì vậy, em rất mong thầy có thể đưa ra những ý kiến góp ý bổ ích để đồ án này tiếp tục được phát triển và có những kết quả mới tốt hơn.

Hà Nội, ngày 1 tháng 6 năm 2024

Tác giả đồ án

Doãn Chí Thường

Mục lục

Bảng ký hiệu và chữ viết tắt

Bảng thuật ngữ tên viết tắt:

DRL	Deep Reinforcement Learning
DRL-MOA	Deep Reinforcement Learning Multi-objective Algorithm
MOP	Multi-objective Optimization Problems
MOTSP	Multi-objective Travelling Salesman Problem
TSP	Travelling Salesman Problem

Bảng ký hiệu:

$\lambda \in \mathbb{R}^p$	Vecto trọng số
$\mathbb{R}^P_>$	$\{y \in \mathbb{R}^p : y > 0\}$
$\mathbb{R}^P_\geq$	$\{y \in \mathbb{R}^p : y \geq 0\}$
$\triangleq$	gán cho đối tượng toán học là một biểu thức
i.i.d	biến ngẫu nhiên độc lập có phân phối đồng nhất
$\mathbb{R}^p_+$	tập các vecto không âm của không gian $\mathbb{R}^p$

Danh sách hình ảnh

Mở đầu

Tối ưu tổ hợp là một lớp bài toán trong lĩnh vực khoa học máy tính với nhiều ứng dụng cụ thể trong đời sống như những bài toán tìm đường đi ngắn nhất, bài toán *Knapsack*,... Một ví dụ điển hình cho các bài toán tối ưu tổ hợp đó là bài toán *Traveling Saleman (TSP)*. Việc tìm kiếm lời giải tối ưu cho bài toán TSP là bài toán thuộc lớp *NP-hard [3]* , kể cả trong trường hợp trọng số giữa các đỉnh trong đồ thị được tính bằng khoảng cách *Euclide*.

Hiện nay có rất nhiều giải thuật từ giải thuật chính xác đến các giải thuật *heuristic* để giải bài toán TSP, trong báo cáo này tác giả xin phép trình bày một hướng tiếp cận giải bài toán TSP và TSP đa mục tiêu thông qua phương pháp *Học tăng cường Reinforcement Learning* và kết hợp với các *mạng neural* của *Học sâu Deep Learning*. Bằng việc xây dựng mô hình dựa trên Học tăng cường và Học sâu, mô hình tối ưu sau qua trình huấn luyện sẽ có thể đưa ra lời giải tối ưu cho các bộ dữ liệu khác mà không cần tốn chi phí và thời gian huấn luyện lại. Các kết quả thực nghiệm đã cho thấy kết quả vô cùng tốt của hướng tiếp cận này.

Báo cáo này sẽ trình bày mô hình giải bài toán TSP đa mục tiêu (MOTSP) bằng phương pháp *Deep Reinfocement Learning* thông qua việc chia nhỏ bài toán ban đầu thành N bài toán vô hướng con và kết hợp nghiệm tối ưu của N bài toán con đó sẽ tìm được *Pareto Front* của bài toán ban đầu. Báo cáo sẽ trình bày cụ thể kiến trúc mạng được sử dụng để giải bài toán MOTSP đó là *Pointer Network* và đồng thời sẽ tiến hành cài đặt mô hình và tiến hành so sánh giữa *Pointer Network* với giải thuật tiến hóa đa mục tiêu giải bài toán MOTSP.

Lời cảm ơn

Báo cáo này được thực hiện và hoàn thành tại Đại học Bách Khoa Hà Nội, nằm trong nội dung học phần *Đồ án I* của kì học 2023-2.

Em xin được dành lời cảm ơn chân thành đến **TS. Đoàn Duy Trung** là giảng viên đã và đang trực tiếp hướng dẫn cho em trong *Đồ án* này. Đồng thời thầy cũng đã giúp đỡ tận tình và có những đóng góp bổ ích để em có thể hoàn thành báo cáo này một cách tốt nhất. Em xin chúc thầy thật nhiều sức khỏe, vẫn nhiệt huyết với giáo dục. Hy vọng có thể được tiếp tục làm việc với thầy trong những *Đồ án* tiếp theo.

Hà Nội, ngày 1 tháng 6 năm 2024
Tác giả đồ án

Doãn Chí Thường

Chương 1

Cơ sở lý thuyết

1.1 Tối ưu đa mục tiêu

Trong phần này, em sẽ trình bày về lý thuyết của bài toán tối ưu đa mục tiêu và phương pháp vô hướng hóa cho bài toán này. Nội dung phần này được tham khảo từ [1].

1.1.1 Giới thiệu bài toán tối ưu đa mục tiêu

Trong những năm 50 của thế kỷ 20, *Quy hoạch đa mục tiêu*, hay còn được gọi là *Tối ưu đa mục tiêu* hoặc *Tối ưu véc tơ*, đã trở thành một chuyên ngành toán học, thu hút sự quan tâm của nhiều tác giả và được phát triển mạnh mẽ suốt gần 70 năm qua. Các thành tựu của Quy hoạch đa mục tiêu được ứng dụng rộng rãi trong thực tế, đặc biệt là trong lý thuyết ra quyết định, kinh tế, tài chính, kỹ thuật, viễn thông,...

Bài toán tối ưu đa mục tiêu được phát biểu như sau [1]:

$$\begin{aligned} &\text{Min } f(x) = f_k(x) \qquad \text{k} = 1,..., \text{p} \qquad \text{(MOP)} \\ &\text{v.đ.k } x \in X \subset \mathbb{R}^n_+ \end{aligned}$$

trong đó, n là số chiều của biến $f_k(x) : \mathbb{R}^n_+ \to \mathbb{R}$ dùng để biểu diễn hàm mục tiêu thứ k. Để xây dựng bài toán, ta có một số định nghĩa như sau:

Định nghĩa (1.1.0) [1]: (Điểm hữu hiệu) *Cho $\emptyset \neq Q \subset \mathbb{R}^p$. q^* là điểm hữu hiệu (efficient point) - điểm không trội (nondominated point) - điểm Pareto của Q nếu không tồn tại $q \in Q$ sao cho $q \leq q^*$ và $q \neq q^*$ tức là:* $(q^* - \mathbb{R}^p_+) \cap Q = \{q^*\}$.

Định nghĩa (1.1.1) [1]: (Điểm hữu hiệu yếu) *Cho* $\emptyset \neq Q \subset \mathbb{R}^p$. q^* *là điểm hữu hiệu yếu (weakly efficient point) - điểm không trội yếu - điểm Pareto yếu của* Q *nếu không tồn tại* $q \in Q$ *sao cho* $q \ll q^*$ *tức là:* $(q^* - int\mathbb{R}^p_+) \cap Q = \emptyset$.

Định nghĩa (1.1.2) [1]: (Điểm lý tưởng) *Cho* $\emptyset \neq Q \subset \mathbb{R}^p$. *Ta nói:* q^* *là điểm lý tưởng (ideal point) của* Q *nếu:* $q \geq q^*$ *với* $\forall q \in Q$, *tức là* $Q \subset q^* + \mathbb{R}^p_+$.

Định lý (1.1.3) [1]: (Tìm điểm hữu hiệu - hữu hiệu yếu) *Cho* $\emptyset \neq Q \subset \mathbb{R}^p$ *và* $\lambda \in \mathbb{R}^p \setminus \{0\}$. *Giả sử,* $q^* \in Argmin\{\langle \lambda, q\rangle | q \in Q\}$. *Khi đó:*

$$\text{Nếu } \lambda > 0 \text{ thì } q^* \in WMinQ$$
$$\text{Nếu } \lambda \gg 0 \text{ thì } q^* \in MinQ$$

Đặc biệt, nếu Q là *compact* thì $MinQ \neq \emptyset$.

Định nghĩa (1.1.4) [1]: (Nghiệm hữu hiệu) *Tập các nghiệm hữu hiệu của bài toán (MOP) là:*

$$X_E = \{x^0 \in X | \nexists x \in X\} \text{ sao cho } f(x^0) \geq f(x) \text{ và } f(x^0) \neq f(x)$$

Định nghĩa (1.1.5) [1]: (Nghiệm hữu hiệu yếu) *Tập các nghiệm hữu hiệu yếu của bài toán (MOP) là:*

$$X_{WE} = \{x^0 \in X | \nexists x \in X\} \text{ sao cho } f(x^0) \gg f(x)$$

Đặt $Y_E = f(x_E)$ và $Y_{WE} = f(x_{WE})$. Tập Y_E là tập ảnh hữu hiệu và Y_{WE} là tập ảnh hữu hiệu yếu của bài toán (MOP). Khi đó:

$$Y_E = MinY \text{ và } Y_{WE} = WMinY$$

1.1.2 Phương pháp vô hướng hóa

Bài toán (MOP) đã trình bày ở trên có thể giải được thông qua phương pháp vô hướng hóa tức ta đi giải bài toán tối ưu sau:

$$\min \sum_{k=1}^{p} \lambda_k f_k(x) \qquad \text{v.đ.k } x \in X$$

với $\lambda \in \mathbb{R}^p_+$ là vector trọng số với các thành phần $\lambda_k, k = 1, ..., p$ dương. Để giải các bài toán (MOP) ta thường tiến hành giải trên không gian ảnh với tập Y trước do số chiều của không gian ảnh thường nhỏ hơn số chiều của không gian quyết định. Trong phần này, em sẽ trình bày mối liên hệ giữa các điểm không trội với giá trị $\sum_{k=1}^{p} \lambda_k y_k$ cũng như chứng minh lời giải tối ưu cho bài toán vô hướng hóa cũng là nghiệm của bài toán (MOP) với các điều kiện nhất định.

Cho $Y \subset \mathbb{R}^p$ với mọi $\lambda \in \mathbb{R}^p_{\geq}$ cho trước, ta ký hiệu như sau:

$$S(\lambda, Y) := \left\{ \widehat{y} \in Y : \langle \lambda, \widehat{y} \rangle = \min_{y \in Y} \langle \lambda, y \rangle \right\}$$

là tập nghiệm tối ưu của Y ứng với λ.

$$S(Y) := \bigcup_{\lambda \in \mathbb{R}^p_>} S(\lambda, Y) = \bigcup_{\{\lambda > 0 : \sum_{k=1}^{p} \lambda_k = 1\}} S(\lambda, Y)$$
$$\text{và } S_0(Y) := \bigcup_{\lambda \in \mathbb{R}^p_{\geq}} S(\lambda, Y) = \bigcup_{\{\lambda \geq 0 : \sum_{k=1}^{p} \lambda_k = 1\}} S(\lambda, Y)$$

Định nghĩa (1.2.0)][1] : *Mọi tập $Y \in \mathbb{R}^p$ được gọi là $\mathbb{R}^p_{\geq}$-lồi nếu $Y + \mathbb{R}^p_{\geq}$. Mọi tập lồi Y hiển nhiên cũng là tập $\mathbb{R}^p_{\geq}$-lồi.*

Một số định lý chứng minh tính đúng đắn của phương pháp vô hướng hóa.

Định lý (1.2.1) [1]: *Với mọi tập $Y \subset \mathbb{R}^p$ ta có $S_0(Y) \subset Y_{wN}$.*

Chứng minh: Cho $\lambda \in \mathbb{R}^p_{\geq}$ và $\widehat{y} \in S(\lambda, Y)$. Khi đó:

$$\sum_{k=1}^{p} \lambda_k \widehat{y}_k \leq \sum_{k=1}^{p} \lambda_k y_k \quad \text{, với mọi } y \in Y$$

Giả sử rằng $\widehat{y} \notin Y_{wN}$. Khi đó tồn tại $y' \in Y$ với $y'_k < \widehat{y}_k$, k = 1,...,p. Do đó:

$$\sum_{k=1}^{p} \lambda_k y'_k < \sum_{k=1}^{p} \lambda_k \widehat{y}_k$$

do ít nhất một trong các trọng số λ_K phải dương. Điều vô lý này dẫn tới giả thiết sai và định lý đã được chứng minh.

Định lý (1.2.2) [1]: *Nếu Y là tập $\mathbb{R}^p_{\geq}$-lồi khi đó $Y_{wN} = S(Y)$*

Định lý (1.2.3) [1]: *Cho tập $Y \in \mathbb{R}^p$ khi đó $S(Y) \subset Y_N$.*

Bổ đề (1.2.4) [1]: *$Y_N \subset S_{(Y)}$ nếu Y là một tập $\mathbb{R}^p_{\leq}$-lồi.*

Bổ đề (1.2.5) [1]: *Giả sử rằng $\widehat{y}$ là nghiệm tối ưu của bài toán vô hướng hóa.*

$$\min_{x \in X} \sum_{k=1}^{p} \lambda_k f_k(x) \tag{1.1}$$

với $\lambda \in \mathbb{R}^p_{\leq}$. Khi đó, ta có các tính chất sau:

1. *Nếu $\lambda \in \mathbb{R}^p_{\leq}$ thì $\widehat{x} \in X_{wE}$*
2. *Nếu $\lambda \in \mathbb{R}^{\overline{p}}_{>}$ thì $\widehat{x} \in X_E$*
3. *Nếu $\lambda \in \mathbb{R}^p_{\leq}$ và $\widehat{x}$ là nghiệm tối ưu duy nhất của (1.1) khi đó $\widehat{x} \in X_{sE}$*

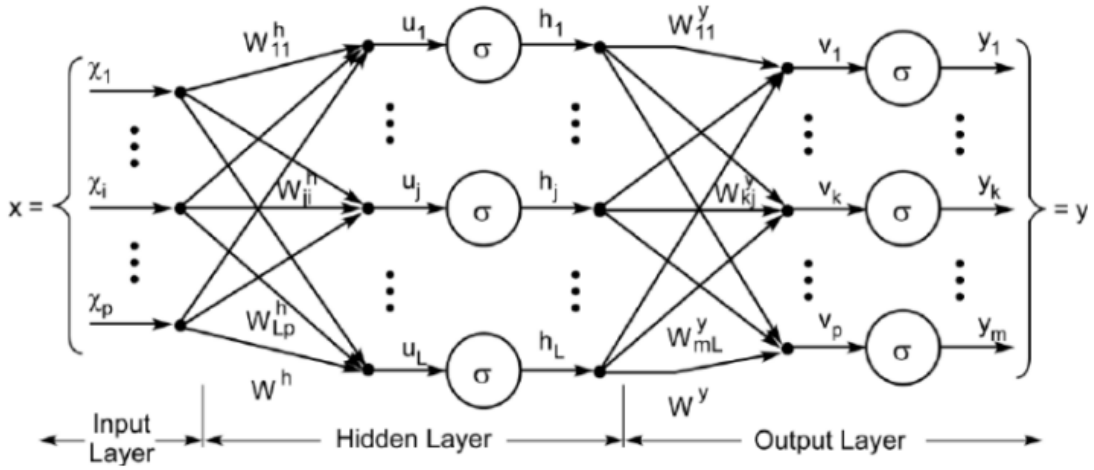
Bổ đề (1.2.6) [1]: *Cho X là tập lồi và f_k là hàm lồi , k = 1,...,p. Nếu $\widehat{x} \in X_{wE}$ thì tồn tại $\lambda \in \mathbb{R}^p_{\leq}$ sao cho $\widehat{x}$ là nghiệm tối ưu của bài toán (1.1).*

1.2 Mô hình mạng Neural

Trong phần này em sẽ thực hiện trình bày kiến thức cơ bản về mạng Neural và kiến thức cơ bản về Recurrent Neural Networks, kiến trúc LSTM, GRU và các kiến thức được tham khảo trong tài liệu và các kí hiệu được giải thích trong bảng kí hiệu.

1.2.1 Tổng quan mạng Neural

Mô hình Neural Network được xây dựng dựa trên mô hình tổng quát Multilayer Perceptrion - MLP. Một MLP là một ánh xạ $y = f*(x, \theta)$ với input là dữ liệu và output là giá trị dự đoán. Hàm f sẽ được xây dựng dựa trên các hàm tuyến tính, logistic, tanh,...Bằng việc lựa chọn các hàm thích hợp và dữ liệu đầu vào mà mô hình MLP có thể xấp xỉ được tham số θ và từ đó giúp máy tính xác định được đầu ra.



Hình 1.1: Kiến trúc cơ bản của MLP [6]

Kiến trúc cơ bản của Neural Network được xây dựng tương tự như hình 1.1 gồm có 3 thành phần sau:

1. Input Layer: là dữ liệu đầu vào.

2. Hidden Layer: Lớp ẩn tính toán trung gian và lưu trữ thông tin tại các Neural với các *activation function* khác nhau.

3. Output Layer: Lớp đầu ra là giá trị dự đoán.

Mỗi *layer* sẽ là một tầng của mạng, trong mỗi *layer* đều có thể có nhiều Neural. Các Neural được liên kết với nhau bằng kết nối có trọng số **w**. Nếu mỗi Neural tại lớp thứ *k+1* được liên kết với toàn bộ các Neural tại lớp thứ *k* ta gọi đó là *Fully-Connected* và nếu không được liên kết toàn bộ ta gọi đó là *Pooling-Connected*. Output của Neural ở lớp trước sẽ là Input của Neural ở lớp phía sau. Giá trị của Neural sẽ được tính bằng tổng tuyến tính của tích các trọng số với input layer cộng với bias hiệu chỉnh *sai số*, sau đó đưa kết thu được qua *activation function* thu được giá trị đầu ra tại mỗi Neural.

1.2.2 Đồ thị tính toán: Unfolding Computational Graphs

Xây dựng đồ thị tính toán mở *Unfolding Computational Graphs (UCG)* tương ứng cho thực hiện một chuỗi các sự kiện. Cấu trúc cổ điển của nó được biểu diễn như sau:

$$s^{(t)} = f(s^{(t-1)}, \theta)$$

với $s^{(t)}$ được gọi là trạng thái của hệ thống tại thời điểm t. Phương trình gọi là tái phát *recurrent* bởi giá trị của **s** tại thời điểm **t** sẽ phụ thuộc vào **s** tại thời điểm **t-1**.

Với hữu hạn bước lặp thì biểu đồ có thể được áp dụng với *t-1* lần. Như vậy, biểu thức có thể được biểu thị bởi đồ thị tính toán tuần hoàn có hướng truyền thống dưới đây:

Hình 1.2: Computational Graph cổ điển [6]

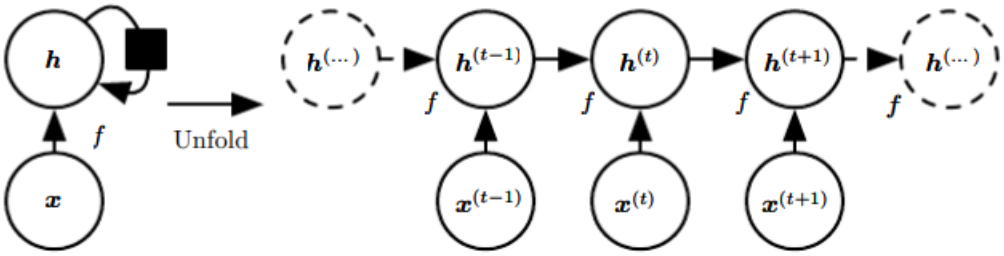
Mặt khác, ta có thể xem xét một hệ thống được tính toán bởi tín hiệu ngoài $x^{(t)}$.

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}, \theta) \tag{1.2}$$

Để chỉ ra trạng thái của đơn vị ẩn thì chúng ta sẽ viết lại phương trình (1.2) bằng sử dụng biến h thay thế cho s để biểu thị trạng thái

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \theta)$$

Reccurent Networks điển hình sẽ bổ sung thêm các đặc điểm kiến trúc như các lớp đầu ra đọc thông tin cho trạng thái để đưa ra dự đoán. Khi đào tạo *Reccurent Networks* sẽ đào tạo một nhiệm vụ dự đoán tương lai từ dữ liệu quá khứ. Mạng sử dụng $h^{(t)}$ như một loại mất mát toán tắt các khía cạnh liên quan đến nhiệm vụ của chuỗi đầu vào từ quá khứ đến thời điểm t. Nó xảy ra mất mát vì nó ánh xạ 1 chuỗi độ dài tùy ý $(x^{(t)}, x^{(t-1)}, x^{(t-2)}, ..., x^{(2)}, x^{(1)})$ đến một vecto $h^{(t)}$ có độ dài cố định.

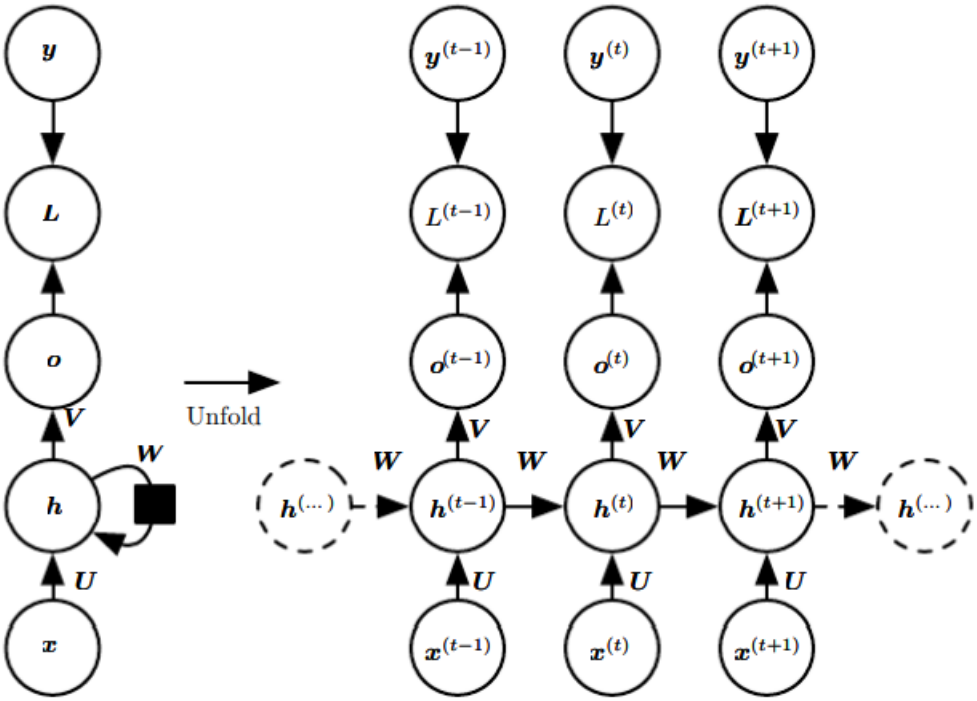


Hình 1.3: Unfolded computational graph [6]

Biểu đồ mô tả một cách rõ ràng về những gì các phép tính toán thực hiện đồng thời cũng minh họa thông tin cho quá trình lan truyền tiến *(forward)* (tính toán ouput và loss) và tính toán lan truyền ngược *(backpropagation)* cho độ dốc tính toán *(gradient)* bằng cách hiển thị rõ ràng đường đi dọc theo các luồng thông tin.

1.2.3 Recurrent Neural Networks

Xây dựng dựa trên ý tưởng chia sẻ tham số và cuộn biểu đồ. Chúng ta có thể thiết kế một mạng *Recurrent Neural* như dưới đây



Hình 1.4: Cấu trúc Recurrent Neural Networks [6]

Quan sát đồ thị 1.4 có thể nhận thấy rằng đồ thị biểu diễn tính toán

của mạng hồi quy ánh xạ một chuỗi đầu vào của giá trị **x** thành một chuỗi các giá trị đầu ra **o** tương ứng. Mất mát **L** đo khoảng cách của mỗi **o** so với **y** mục tiêu tương ứng. Giả sử hàm mất mát **L** được so sánh dựa trên tính toán $\widehat{y} = softmax(o)$ với mục tiêu **y**. Mạng RNN gồm các ma trận trọng số sau: ma trận trọng số **U** tính toán từ input **x** tạo ra các giá trị ẩn *hidden* **h** của mạng, các giá trị **h** sẽ được dùng làm input cho bước lặp tiếp theo trong *forward* mạng RNN và các giá trị **h** đó được kết nối với nhau thông qua ma trận trọng số **W**, việc tính toán từ **h** tới các giá trị output **o** của mạng thông qua ma trận trọng số **V** . Việc tính toán chi tiết được tiến hành như sau:

$$
\begin{aligned}
a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} \\
h^{(t)} &= \tanh(a^{(t)}) \\
o^{(t)} &= c + Vh^{(t)} \\
\widehat{y}^{(t)} &= \text{softmax}(o^{(t)})
\end{aligned}
$$

Trong đó, vecto **b** và **c** là *bias* độ lệch cùng với ma trận hệ số **U, W V** tương ứng với các kết nối *input-to-hidden, hidden-to-hidden, hidden-to-output*. Giả sử, chọn hàm mất mát $L^{(t)}$ là *negative log-likelihood* của $y^{(t)}$ cho chuỗi $x^{(1)}, ..., x^{(t)}$.

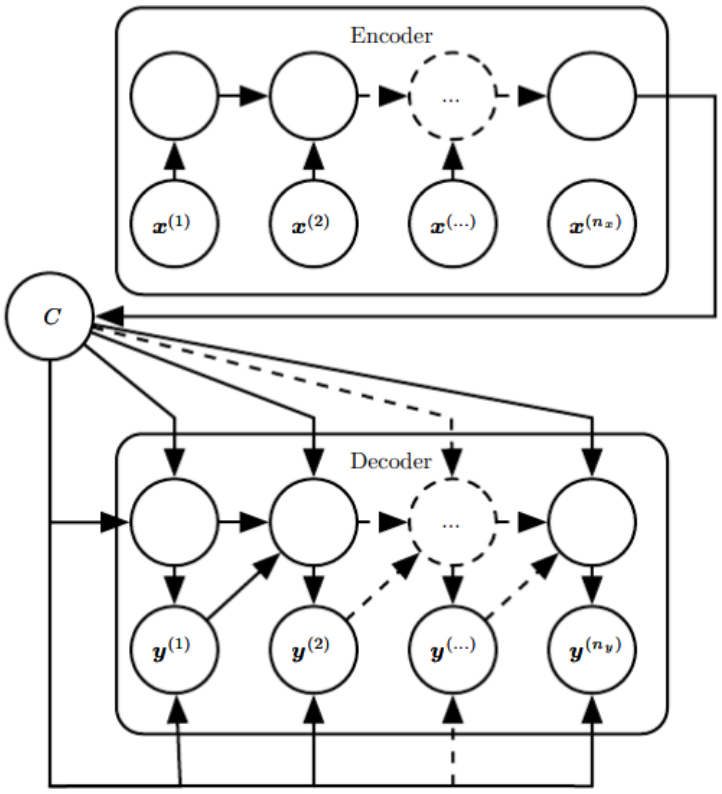
$$
\begin{aligned}
& L\big(\{x^{(1)}, ..., x^{(t)}\}, \{y^{(1)}, ..., y^{(t)}\}\big) \\
&= \sum_t L^{(t)} \\
&= -\sum_t logp_{model}\big(y^{(t)} | \{x^{(1)}, ..., x^{(t)}\}\big)
\end{aligned}
$$

Với p_{model} được tính qua mô hình khi so sánh giá trị mục tiêu $y^{(t)}$ với giá trị dự đoán $\widehat{y}^{(t)}$.

Kiến trúc Encoder-Decoder và Sequence to Sequence

Kiến trúc *Sequence-to-Sequence* của mạng RNN mô tả cho việc học tạo ra chuỗi output $(y^{(1)}, ..., y^{(n_y)})$ khi biết trước được chuỗi input $(x^{(1)}, x^{(2)}, ..., x^{(x_n)})$. Đây là sự kết hợp của khối Encoder đọc dữ liệu đầu và khối Decoder tạo ra chuỗi output đầu ra hoặc tính xác suất tại một sự kiện nào đó. Lớp ẩn

cuối cùng của Encoder được dùng để tính một biến (context) cố định C được trích xuất ra từ chuỗi đầu vào để làm input cho khối Decoder.

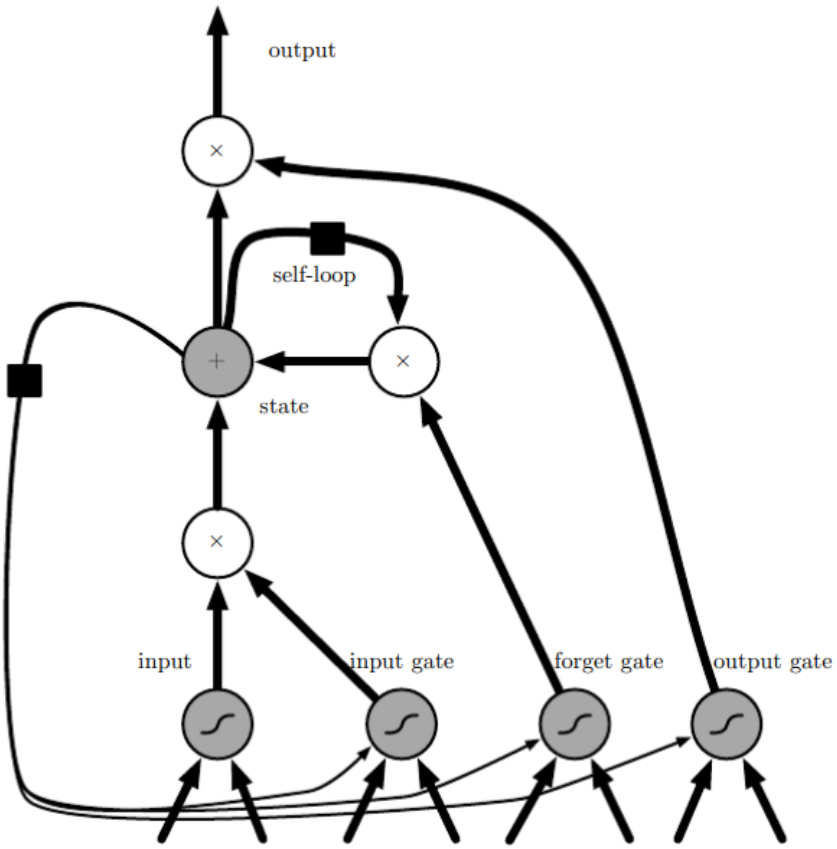


Hình 1.5: Khối kiến trúc Encoder-Decoder [6]

1.2.4 Sequence Modeling: Mạng LSTM và GRU

Mạng LSTM (Long-short term memory)

Về mặt lý thuyết thì mạng RNN có thể đưa thông tin từ *layer* trước đến các *layer* phía sau. Nhưng thực tế là thông tin chỉ mang được qua một số lượng state nhất định, điều đó dẫn đến hiện tượng *vanishing gradient*, hay nói cách khác mạng RNN sẽ chỉ đọc các *state* gần đó. Đây là hiện tượng *short term memory*. Với bài toán dịch văn bản trong xử lý ngôn ngữ tự nhiên thì input của mạng là những câu văn dài và ta cần thông tin của những câu văn trước đó để output đầu ra dịch đúng ngữ cảnh nhất. Chính vì vậy, để khắc phục hiện tượng *vanishing gradient* một mạng RNN cải tiến ra đời chính là mạng *Long-short term memory LSTM*.



Hình 1.6: Kiến trúc mạng LSTM [6]

Điểm đặc biệt trong cấu trúc tế bào của mạng LSTM là nó có cơ chế *self-loop* để làm giảm hiện tượng *vanishing gradient*, tế bào LSTM sẽ quyết định tỷ lệ lưu giữ thông tin của các state trước đó, các tham số của *self-loop* được điều khiển thông qua cổng *(forget gate)* $f_i^{(t)}$ (ứng với bước lặp thứ t và tế bào thứ i), cổng này sẽ điều chỉnh tỷ lệ quên thông tin trước trong khoảng từ 0 tới 1 thông qua hàm *Sigmoid*:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

trong đó, $x^{(t)}$ là vector input hiện tại và $h^{(t)}$ là vector *hidden layer* hiện tại chứa output của toàn bộ tế bào LSTM trước đó. b^f, U^f và W^f lần lượt là tham số *bias* của mạng, trọng số của dữ liệu đầu vào và trọng số cho tầng cổng quên *(forget gates)*. Thành phần quan trọng nhất của *State unit* $s_i^{(t)}$ lưu chữ thông tin của mạng. Khi đó trình cập nhật giá trị của mạng được diễn ra như sau:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\Big(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\Big)$$

với b, U, W lần lượt là hệ số của *bias*, tham số dữ liệu input và tham số *reccurent* của tế bào LSTM. Ta thấy giá trị của $s_i^{(t)}$ được cập nhật cùng với giá trị của tầng cổng quên $f_i^{(t)}$. Giá trị *external input gate* $g_i^{(t)}$ được tính toán tương tự như tầng cổng quên với các tham số riêng để tính toán tỷ lệ ảnh hưởng của dữ liệu đầu vào ở bước thứ t là $x_j^{(t)}$ và thông tin từ *state* trước đó là $h_j^{(t-1)}$:

$$g_i^{(t)} = \sigma\Big(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\Big)$$

Giá trị output $h_i^{(t)}$ của tế bào LTSM cũng có thể bị loại bỏ thông qua cổng *output gate* $q_i^{(t)}$ cũng được tính thông qua hàm sigmoid:

$$h_i^{(t)} = tanh\big(s_i^{(t)}\big) q_i^{(t)}$$

$$q_i^{(t)} = \sigma\Big(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\Big)$$

có chứa các tham số b^o, U^o, W^o lần lượt là *bias*, trọng số input và trọng số của tầng cổng quên đối với kết quả output dự đoán của các *state* trước đó.

Mạng GRU (Gated Recurrent Unit)

Mạng GRU là một loại mạng RNN cũng được thiết kế để khắc phục hiện tượng *vanishing gradient*. Mạng GRU có kiến trúc tương tự mạng LSTM nhưng có sự thay đổi trong cách thiết kế tế bào *cell*, cụ thể là thay đổi cách thiết lập hàm để "quên" thông tin từ các state trước đó cũng như cách tính trọng số trong việc sử dụng thông tin từ các *state* trước đó:

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + \big(1 - u_i^{(t-1)}\big) \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}\right)$$

với u biểu thị cho *cổng cập nhật (update gate)* và r biểu thị cho *cổng reset*. Các cổng này được thiết lập tính toán như sau:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}\right)$$

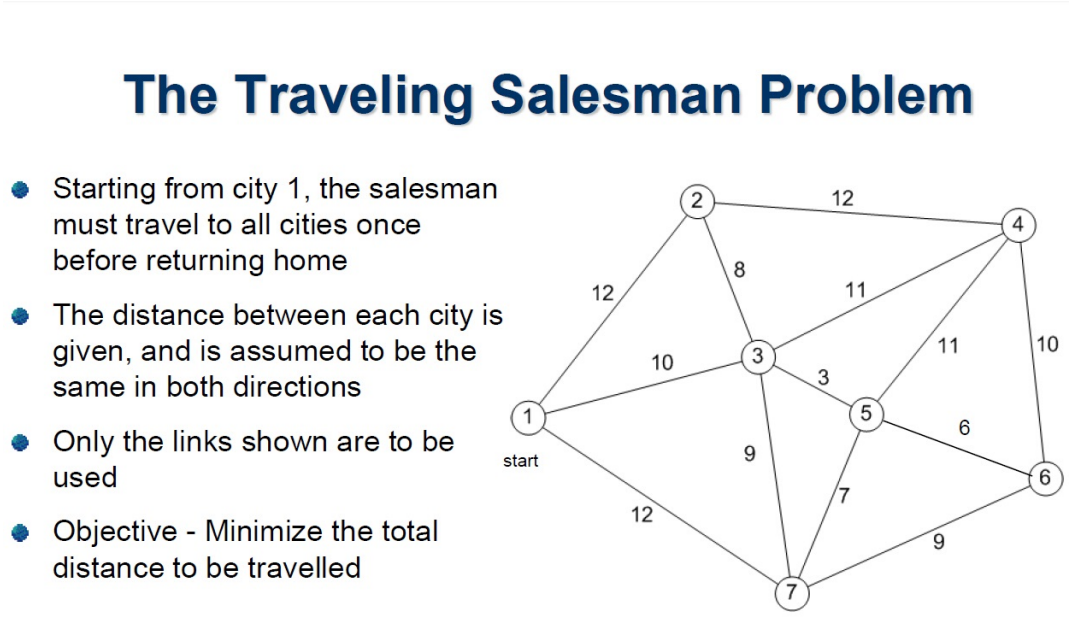
$$r_i^{(t)} = \sigma\left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}\right)$$

Cổng *reset* cũng như cổng cập nhật có thể bỏ qua thông tin của các *state* trước đó, cổng cập nhật có cơ chế tương tự như tầng cổng quên, cổng *reset* sẽ điều khiển phần thông tin nào của *state* trước đó được dùng để sử dụng trong việc tính toán của state này.

1.3 Bài toán tối ưu tổ hợp đã mục tiêu

1.3.1 Bài toán người đi du lịch (TSP)

Bài toán TSP là một bài toán lâu đời và có nhiều ứng dụng trong thực tế, đặc biệt là đối với ngành khoa học dữ liệu và phân tích tài chính. Các công ty vận tải sẽ thu thập thông tin những hành trình đã hoàn thành của nhân viên giao hàng và tiến hành phân tích dữ liệu từ đó đưa ra các tuyến đường ngắn hơn cho nhân viên thông qua việc sử dụng các mô hình Học máy để phân tích dữ liệu.



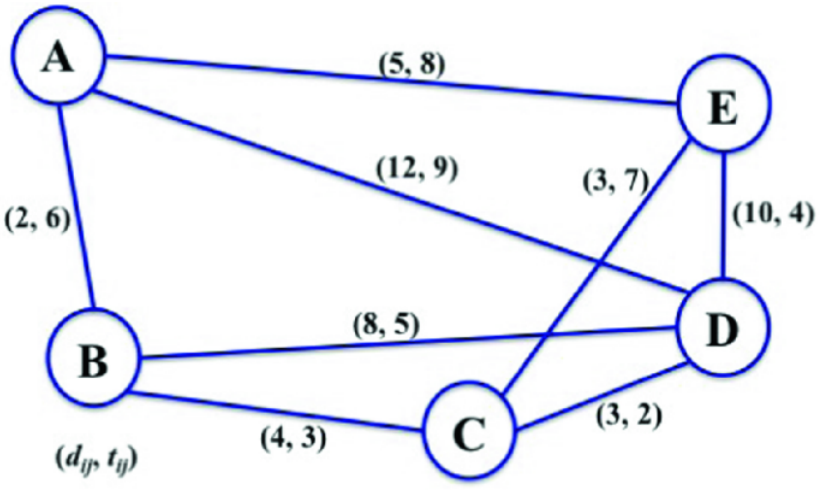
Hình 1.7: Mô tả bài toán TSP [9]

Bài toán TSP được phát biểu cụ thể như sau: Cho trước một đồ thị n đỉnh {0, 1,..., n - 1} và giá trị trọng số $C_{i,j}$ trên mỗi cặp đỉnh i, j, ta cần tìm một chu trình đi qua tất cả các cạnh của đồ thị, mỗi đỉnh đúng một lần sao cho tổng các trọng số trên chu trình đó là nhỏ nhất. Đây là bài toán thuộc lớp NP-Hard nên không tồn tại thuật toán tất định thời gian đa thức hiện có để giải bài toán này.

Đặc điểm chung của các tuật toán giải đúng bài toán thường sẽ tốn chi phí rất lớn, đặc biệt đối với số thành phố n $>$ 20. Do đó, các mô hình *Học máy* đang tiếp cận vấn đề này với mong muốn giản chi phí tạo ra để giải quyết bài toán trên. Trong báo cáo này sẽ tiếp cận một hướng học máy là Reinforcement Learning kết hợp với Deep Learning để giải bài toán khó hơn TSP là toán toán TSP đa mục tiêu (Multi-objective Travelling Salesman Problem).

1.3.2 Bài toán người du lịch đa mục tiêu (MOTSP)

Bài toán MOTSP được xét với nhiều *chi phí* khác nhau chứ không đơn thuần chỉ là chi phí khoảng cách giữa 2 thành phố mà bài toán TSP đã mô tả.



Hình 1.8: Mô tả bài toán MOTSP [10]

Bài toán MOTSP được mô tả một cách cụ thể như sau: cho một đồ thị n đỉnh độc lập, việc di chuyển từ đỉnh i đến j sẽ tốn m hàm chi phí khác nhau. Yêu cầu bài toán là tìm ra một chu trình sao cho tổng m hàm chi

phí là nhỏ nhất.

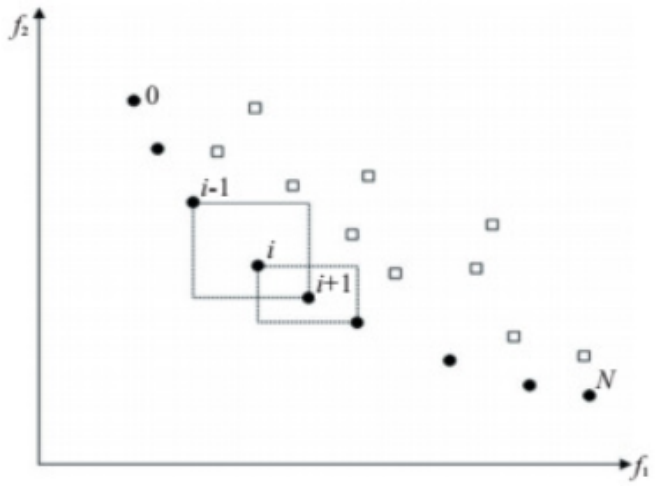
1.3.3 Giới thiệu thuật toán tiến hóa giải bài toán tối ưu tổ hợp đa mục tiêu

Thuật toán NSGA-II

NSGA-II là phương pháp giải bài toán tối ưu hóa đa mục tiêu sử dụng giải thuật di truyền kết hợp sắp xếp nghiệm không trội. Các đặc điểm nổi bật của thuật toán nằm ở: phương pháp dùng để gán độ thích nghi, cách duy trì quần thể ưu tú, cách tiếp cận nhằm đa dạng hóa quần thể.

Để gán độ thích nghi cho một cá thể ta gán cho cá thể thứ hạng theo các biên không trội mà cá thể đó thuộc vào dựa vào sắp xếp không trội *(Non-dominated sorting)*. Độ thích nghi của một cá thể sau khi sắp xếp được xác định như sau:

- Nghiệm hay cá thể nào nằm trên biên thứ nhất - R1 thì có độ thích nghi cao nhất và tất cả các nghiệm này được gán là hạng thứ 1.
- Tất cả các nghiệm nằm trên cùng một biên chứa các nghiệm không trội thì có cùng độ thích nghi và chúng có cùng thứ hạng.

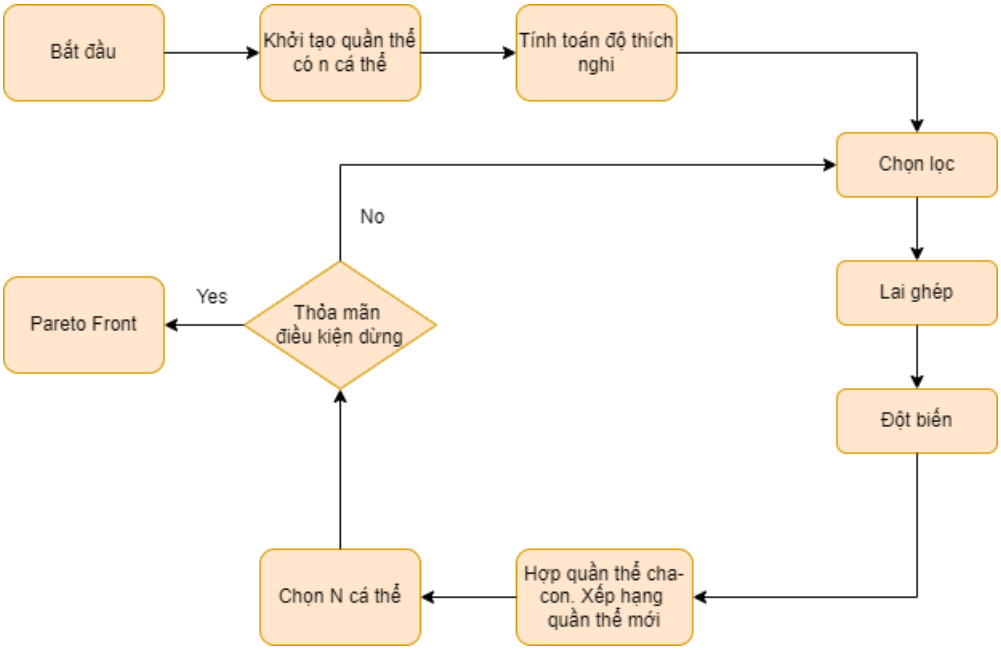


Hình 1.9: Khoảng cách quy tụ [20]

Để xem xét các nghiệm liền kề với nó trên cùng biên không trội có gần nó hay không, ta sẽ sử dụng độ đo theo khoảng cách quy tụ (Crowding Distance). Khoảng cách quy tụ của cá thể hay nghiệm nằm trên một biên

là chiều dài trung bình các cạnh của một hình hộp.

Xây dựng lược đồ thuật toán NSGA-II:



Hình 1.10: Sơ đồ thuật toán NSGA-II [1]

Các toán tử trong NSGA-II

Toán tử lai ghép SBX: [15] Quy trình tính toán cá thể con $x_i^{(1,t+1)}$ và $s_i^{(2,t+1)}$ từ cá thể cha mẹ $x_i^{(1,t)}$ và $s_i^{(2,t)}$ được mô tả như sau. Một hệ số chênh lệch β là tỷ lệ sự khác biệt tuyệt đối của hai cá thể con và hai cá thể cha mẹ được tính bằng:

$$\beta = \left| \frac{s_i^{(2,t+1)} - s_i^{(1,t+1)}}{s_i^{(2,t)} - s_i^{(1,t)}} \right| \tag{1.3}$$

Trước tiên, một số u ngẫu nhiên được sinh trong đoạn [0,1]. Sau đó sử dụng một hàm phân phối xác suất dưới đây để tìm $\overline{\beta}$ sao cho diện tích dưới đường cong phân phối từ 0 đến $\overline{\beta}$ bằng với u:

$$P(\beta) = \begin{cases} 0.5(n+1)\beta^n, & \beta \leq 1 \\ 0.5(n+1)\dfrac{1}{\beta^{n+2}}, & \beta > 1 \end{cases} \tag{1.4}$$

n là một số thực không âm bất kỳ. Giá trị n lớn cho xác suất cao hơn để tạo các giá trị con gần cha mẹ và giá trị nhỏ của n cho phép các điểm ở xa

được chọn làm cá thể con. Sau khi thu được $\overline{\beta}$ từ phân phối trên, các cá thể con được sinh ra như sau:

$$x_i^{(1,t+1)} = 0.5\Big[(1+\overline{\beta})x_i^{(1,t)} + (1-\overline{\beta})x_i^{(2,t)}\Big]$$

$$x_i^{(2,t+1)} = 0.5\Big[(1-\overline{\beta})x_i^{(1,t)} + (1+\overline{\beta})x_i^{(2,t)}\Big]$$

Hai cá thể con đối xứng nhau về các nghiệm cha mẹ, giúp tránh sự thiên vị đối với bất kỳ cá thể cha mẹ cụ thể nào. Một khía cạnh thú vị khác của toán tử chéo này là đối với một phân phối xác suất cố định nếu các giá trị cha mẹ ở xa nhau thì có thể tạo ra các giá trị con ở xa nhau, nhưng nếu các giá trị của cha mẹ gần nhau, các giá trị con sẽ gần nhau. Ban đầu khi các cá thể được khởi tạo là ngẫu nhiên, điều này cho phép khảo sát toàn bộ không gian tìm kiếm, nhưng khi các cá thể có xu hướng hội tụ do tác động của các toán tử di truyền, các cá thể ở xa không được sinh ra, do đó tập trung tìm kiếm vào một vùng hẹp.

Toán tử đột biến: [22] Đối với một thành phần của vectơ x, giá trị hiện tại của biến được thay đổi thành giá trị lân cận bằng cách sử dụng một phân phối xác suất, một yếu tố nhiễu loạn được xác định như sau:

$$\delta = \frac{c-p}{\Delta_{max}} \tag{1.5}$$

trong đó c là giá trị đột biến và Δ_{max} là đại lượng cố định, đại diện cho nhiễu tối đa cho phép ở giá trị cha mẹ p. Tương tự với toán tử lai ghép, giá trị đột biến được tính toán với phân phối xác suất phụ thuộc vào hệ số nhiễu δ như sau:

$$P(\delta) = 0.5(n+1)(1-|\delta|)^n \tag{1.6}$$

Phân phối xác suất trên thỏa mãn với $\delta \in (-1, 1)$ và n là một số thực không âm. Để tạo một giá trị đột biến, một số ngẫu nhiên u được tạo trong khoảng (0, 1). Sau đó, tính hệ số nhiễu loạn $\overline{\delta}$ tương ứng với u bằng cách sử dụng phân phối xác suất ở trên:

$$\delta = \begin{cases} (2u)^{\frac{1}{\eta+1}} - 1, & u < 0.5 \\ 1 - [2(1-u)]^{\frac{1}{\eta+1}}, & u \geq 0.5 \end{cases} \tag{1.7}$$

Khi đó giá trị đột biến được tính toán như sau:

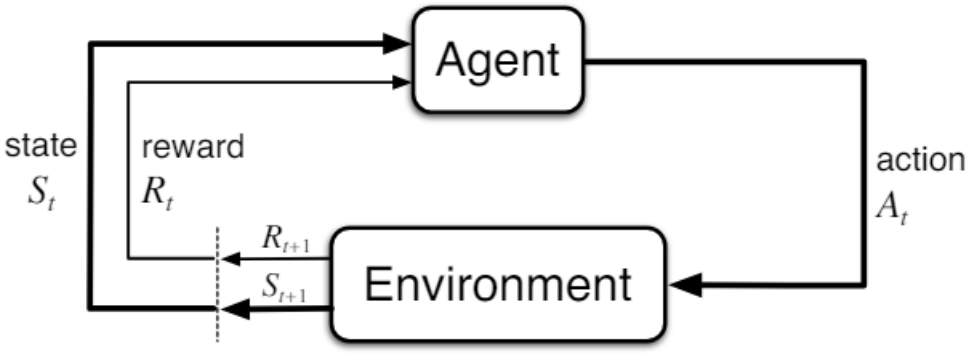
$$c = p + \overline{\delta}\Delta_{max} \tag{1.8}$$

Độ phức tạp của thuật toán với M là số hàm mục tiêu, N là kích cỡ quần thể [5] ta có: Thuật toán sắp xếp không trội có độ phức tạp là $O(M(2N)^2)$ Thuật toán tính khoảng cách quy tụ có độ phức tạp là $O(M(2N)log(2N))$. Sắp xếp theo khoảng cách quy tụ có độ phức tạp là $O(2Nlog(2N))$ Như vậy độ phức tạp tổng thể của thuật toán NSGA-II trong một lần tạo quần thể là $O(MN^2)$.

1.4 Reinforcement Learning

1.4.1 Mô hình tổng quan về Reinforcement Learning

Học tăng cường *Reinforcement Learning* là phương thức học xuyên suốt quá trình mà tác nhân quan sát trong một môi trường có sự biến đổi liên tục. Ta có thể quan sát một mô hình cơ bản của *RL* cùng với các định nghĩa liên quan:



Hình 1.11: Mô hình học tăng cường cơ bản [7]

- *Environment:* là môi trường mà tác nhân sẽ hoạt động.
- *Agent (Tác nhân):* Thực hiện quan sát môi trường đồng thời đưa ra hành động phù hợp để tối đa hóa phần thưởng thu được.
- *Reward (Phần thưởng):* Tổng phần thưởng mà tác nhân đạt được khi thực hiện một hành động hay một chuỗi các hành động trong môi trường.
- *State (Trạng thái):* Trạng thái của môi trường mà *Agent* nhận được.

- *Policy (Chính sách):* Tác nhân sẽ quan sát chính sách với những hành động phù hợp có thể đưa ra trong môi trường nhằm đạt được mục tiêu.
- *Episode:* một chuỗi các trạng thái và hành động cho đến trạng thái kết thúc: $s_1, a_1, s_2, a_2, ..., s_T, a_T$.
- *Accumulative Reward (phần thưởng tích lũy):* tổng phần thưởng tích lũy từ trạng thái đầu tiên đến trạng thái cuối cùng.

Như vậy, có thể hiểu rằng học tăng cường là quá trình học từ sự biến động của môi trường, đồng thời tích lũy phần thưởng đạt được với mục tiêu tối đa.

1.4.2 Quá trình quyết định Markov

Một nhiệm vụ học tăng cường thỏa mãn tính chất Markov được gọi là Quá trình quyết định *Markov*, hoặc *MDP* [7]. Nếu không gian trạng thái và hành động là hữu hạn, thì nó được gọi là quá trình quyết định Markov hữu hạn *(MDP hữu hạn)*. MDP hữu hạn đặc biệt quan trọng đối với lý thuyết học tăng cường. Một quá trình quyết định Markov bao gồm 5 thành phần như sau: (S, A, P, R, γ). Mỗi thành phần được mô tả như sau:

- S là tập hữu hạn biểu thị tất cả các trạng thái của quá trình. Ký hiệu trạng thái tại thời điểm t của quá trình là S_t.
- A là tập hữu hạn các hành động. Tại mội trạng thái $s \in S$ ta có hữu hạn hành động tại s là A_s.
- $P(s'|s, a)$ là xác suất chuyển sang trạng thái s' tại thời điểm t + 1 với $S_t = s$ và $A_t = a$.
- $R(s', a, s)$ là phần thưởng nhận được khi thực hiện hành động a_t từ trạng thái s_t đến trạng thái s_{t+1}.
- γ là độ hao hụt phần thưởng giữa hiện tại và tương lai.

Ta định nghĩa, chính sách π là một phương án lựa chọn hành động mà quá trình tuân theo. Chính sách này bao gồm 2 loại sau:

- Xác định: Khi trạng thái của tác nhân nhận được là $s \in S$ thì hành động được chọn tiếp theo là $\pi(s)$.

- Ngẫu nhân: xác suất lựa chọn hành động $a \in A_s$ là $\pi(a|s)$.

Một mục tiêu mà bài toán đặt ra cho *quá trình quyết định Markov* là việc xây dựng một chính sách π nhằm tối ưu phần thưởng dài hạn hay còn được gọi là *lợi nhuận*, trong đó công thức của lợi nhuận là G_t với hao hụt γ là:

$$G_t = R_t + \gamma R_{t+1} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.9}$$

G_t là tổng tất cả các phần thưởng nhận được kể từ trạng thái s_t đến tương lai, với $0 < \gamma < 1$ thì các phần thưởng càng xa hiện tại sẽ càng bị *giảm giá (discount)* nhiều và ngược lại.

1.4.3 Hàm giá trị (Value Function)

Các thuật toán RL sẽ chỉ định cách mà tác tử thay đổi chính sách dựa vào chính những trải nghiệm (các bước đi và hành động trong quá khứ) của nó. Trong một số trường hợp, chiến lược có thể là một hàm hoặc bảng tra cứu đơn giản. Trong một số trường hợp khác, chiến lược có thể liên quan đến tính toán mở rộng.

Chính sách π là ánh xạ từ mỗi trạng thái $s \in S$. Kí hiệu $v^\pi(s)$ được định nghĩa kỳ vọng lợi nhuận khi bắt đầu từ trạng thái s và đi theo chính sách π:

$$v^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \tag{1.10}$$

Ta gọi hàm v_π là hàm giá trị trạng thái cho chính sách π. Tương tự, chúng ta xác định giá trị của việc thực hiện hành động a ở trạng thái s theo chính sách π, kí hiệu $q_\pi(s, a)$ là kỳ vọng lợi nhuận khi bắt đầu từ trạng thái s, chọn hành động a và sau đó tuân theo chính sách π:

$$q^{\pi}(s|a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}R_{t+k+1}\middle|S_t = s, A_t = a\right] \tag{1.11}$$

Chúng ta gọi q_π là hàm giá trị hành động cho chính sách π:

Mặt khác, với mọi $s \in S$, ta có:

$$\begin{aligned}
v_{\pi}(s) &= \mathbb{E}_{\pi}\big[G_t|S_t = s\big] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}R_{t+k+1}\middle|S_t = s\right] \\
&= \sum_{a}\pi(a|s)\sum_{s'}\sum_{r}p(s', r|s, a)\left[r + \gamma\mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}R_{t+k+2}\middle|S_{t+1} = s'\right]\right] \\
&= \sum_{a}\pi(a|s)\sum_{s',r}p(s', r|s, a)\Big[r + \gamma v_{\pi}(s')\Big]
\end{aligned}$$

Phương trình trên được gọi là phương trình *Bellman* và sẽ là phương trình cốt lõi cho việc cập nhật các hàm giá trị trong các thuật toán tối ưu chính sách của bài toán Reinforcement Learning.

1.4.4 Một số phương pháp Policy Gradient

Trong phần này, chúng ta quan tâm đến việc số hóa một *Policy* thuận lợi cho việc lựa chọn các hành động của tác nhân trong mỗi trạng thái nhận được. Chúng ta cũng có thể hiểu rằng mình sẽ phải xem xét đến phương pháp học tham số chính sách dựa trên gradient của một số hàm độ đo hiệu suất đối với tham số chính sách. Các phương pháp này đều tìm cách tối đa hóa hiệu suất vì vậy mà việc cập nhật phương án của *Policy* dựa trên hướng tăng của gradient trong hàm hiệu suất [7]. Giả sử θ là vecto tham số chính sách. Thì $J(\theta)$ sẽ là hàm hiệu suất chính sách. Ta xây dựng được công thức cập nhật xấp xỉ của vecto θ như sau:

$$\theta_{t+1} = \theta_t + \alpha \bigtriangledown J(\theta_t)$$

với, $\bigtriangledown J(\theta_t)$ là một ước lượng ngẫu nhiên mà kỳ vọng của nó xấp xỉ gradient của độ đo hiệu suất đối với θ_t. Các phương pháp tuân theo cấu trúc chung

này được gọi là chiến lược gradient *(policy gradient methods)*

Policy approximation

Mục tiêu của xấp xỉ chính sách $\pi_\theta(s, a)$ là ta cần tìm tham số tối ưu θ, do đó ta cần một hàm có khả năng đánh giá độ tốt của chính sách π_θ đang được ước lượng bởi các giá trị thuộc vector θ. Ta xét với môi trường chia theo trạng thái (tập) tức là có tồn tại trạng thái kết thúc vào cuối mỗi tập. Một hàm dùng để đánh giá độ tốt của chính sách π_θ là hàm lợi nhuận theo tập *Episodic-return objective:*

$$\begin{aligned} J_G(\theta) &= \mathbb{E}_{S_0 \sim d_0, \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right] \\ &= \mathbb{E}_{S_0 \sim d_0, \pi_\theta}\big[G_0\big] \\ &= \mathbb{E}_{S_0 \sim d_0}\big[\mathbb{E}_{\pi\theta}[G_t | S_t = S_0]\big] \\ &= \mathbb{E}_{S_0 \sim d_0}\big[v_{\pi\theta}(S_0)\big] \end{aligned}$$

với d_0 là phân phối xác suất biểu diễn trạng thái bắt đầu. Giá trị của $J_G(\theta)$ chính là kỳ vọng của hàm giá trị trạng thái bắt đầu từ S_0.

Policy Gradient

Đối với phương pháp này ta cần tìm ra một tham số θ để cực đại hàm $J(\theta)$. Một cách tiếp cận hiệu quả đó là phương pháp *stochastic gradient descent* phương pháp hướng tăng ngẫu nhiên. Ý tưởng phương pháp này là tăng giá trị đạo hàm của $J(\theta)$:

$$\triangle\theta = \alpha \bigtriangledown_\theta J(\theta)$$

với $\bigtriangledown_\theta J(\theta)$ là gradient của chính sách được định nghĩa như sau:

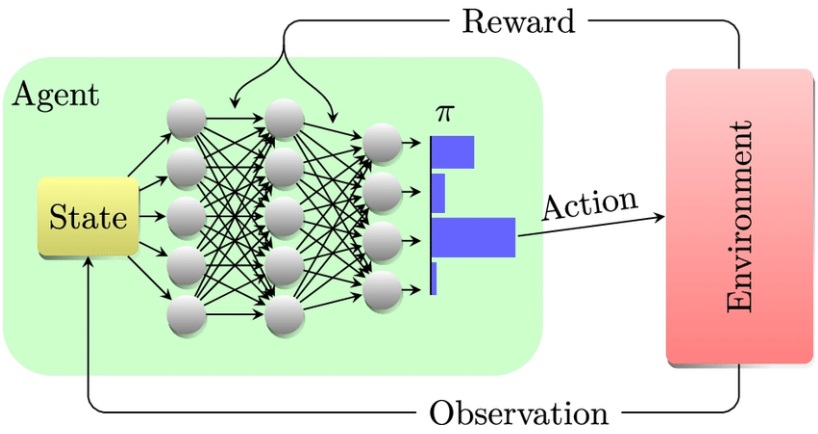
$$\bigtriangledown_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

và α là tham số thể hiện bước nhảy của thuật toán. Công việc còn lại là tính toán $\bigtriangledown_\theta J(\theta)$ dựa trên hàm khả vi θ. Ta có:

$$\bigtriangledown_\theta J(\theta) = \bigtriangledown_\theta \mathbb{E}_{\pi_\theta}[R]$$

1.4.5 Deep Reinforcement Learning

Mô hình *Học sâu tăng cường* là sự kết hợp của *Học sâu* và *Học tăng cường* với đặc điểm chính của mô hình là thay vì lưu trữ các cặp trạng thái - hành động và phần thưởng thì ta sẽ mô hình hóa các thông tin đó dưới dạng các mạng Neural. Việc làm này giúp giảm thiểu dung lượng lưu trữ trong trường hợp số lượng trạng thái và hành động lớn, cũng như tăng tốc độ huấn luyện của mô hình.



Hình 1.12: Deep Reinforcement Learning [4]

Để có thể giải các bài toán điều khiển tối ưu trên thực tế, các phương pháp Học tăng cường cần có một phương pháp lưu trữ hiệu quả số lượng trạng thái lớn của môi trường cũng như cần tăng độ hiểu quả tính toán trong việc học giá trị từ các trạng thái. Chính vì lý do đó, các hướng tiếp cận xấp xỉ sử dụng các mạng Học sâu ra đời với đặc điểm:

- Ánh xạ các trạng thái s đến các hàm mô tả đặc trưng $\phi(s)$.
- Tiến hành tham số hóa các hàm đặc trưng $v_\theta(\phi)$.
- Cập nhật tham số θ dẫn tới cập nhật các hàm giá trị cho chính sách $v_\pi(s) \sim v_\theta(\phi(s))$.

Trong bài báo cáo này, tác giả đề xuất giải quyết MOP bằng phương pháp DRL dựa trên khuôn khổ đơn giản nhưng hiệu quả (DRL-MOA). Chiến lược phân rã được áp dụng để phân tách MOP thành một số bài toán con. Mỗi bài toán con được mô hình hóa như một mạng lưới thần kinh. Khi đó các tham số mô hình của tất cả các vấn đề con được tối ưu hóa cộng tác theo đến chiến lược truyền tham số dựa trên vùng lân cận và thuật toán huấn luyện Actor-Critic. Dưới đây là *ứng dụng Deep Reinforcement Learning trong giải bài toán TSP* được trình bày chi tiết trong phần 2 của Đồ Án.

Chương 2

Ứng dụng Deep Reinforcement Learning giải bài toán MOTSP

Tại chương này, em sẽ trình bày về cách giải bài toán TSP và MOTSP bằng Deep Reinforcement Learning, cụ thể là việc xây dựng Pointer Network cùng việc huấn luyện Actor và Critic nhằm tìm ra Policy tối ưu hay tìm ra được đường đi tiêu tốn ít chi phí di chuyển nhất giữa các thành phố. Hướng giải bài toán TSP bằng DRL được tham khảo từ [11] và hướng giải bài toán MOTSP bằng DRL được tham khảo từ [12].

2.1 Deep Reinforcement Learning giải bài toán TSP

Xét bài toán TSP trong không gian Euclidean 2 chiều. Cho một đồ thị đầu vào biểu diễn như là một chuỗi n thành phố trong không gian 2 chiều như sau: s = $\{x_i\}_{i=1}^n$ với $x_i \in \mathbb{R}^2$. Bài toán yêu cầu chúng ta cần giải quyết là tìm ra một *chu trình* sao cho mỗi thành phố chỉ được thăm đúng một lần và chu trình đó có tổng độ dài là nhỏ nhất. Hay nói cách khác, chúng ta đang cần tìm ra chuỗi π và xác định độ dài của chuỗi. Ta định nghĩa độ dài 1 chu trình của π như sau:

$$L(\pi|s) = ||x_{\pi(n)} - x_{\pi(1)}||_2 + \sum_{i=1}^{n-1} ||x_{\pi(i)} - x_{\pi(i+1)}||_2$$

với $||.||$ là chuẩn l_2. Ta cần xác định tham số của chính sách ngẫu nhiên $P(\pi|s)$ sao cho khi cho dữ liệu đầu vào là chuỗi s thì chính sách đó gán cho các chu trình có độ dài ngắn xác suất cao xảy ra và ngược lại. Pointer

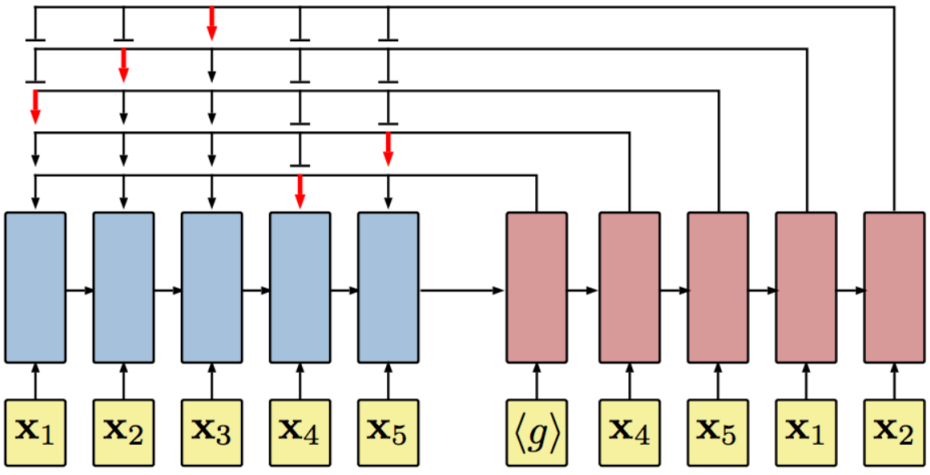
Network sẽ sử dụng công thức tích sự kiện để biểu diễn xác suất xảy ra của 1 chu trình như sau:

$$P(\pi|s) = \prod_{i=1}^{n} P(\pi(i)|\pi(< i), s)$$

Kiến trúc Pointer Network

Ý tưởng chính của Pointer Network: Mạng Pointer sẽ tiến hành tham số hóa các thành phần sau:

- θ : là bộ tham số mạng Pointer cần tối ưu bằng cách sử dụng mô hình không dựa trên model mà dựa vào các quy tắc (model-free policy-based Reinforcement Learning).
- $\pi(i)$: là thành phố đi tới ở bước thứ i được quy định bởi chính sách π.
- $\pi_i \leftarrow x_j$: Gán thành phố x_j vào chu trình ở bước thứ i theo chính sách π.
- L: là độ dài chu trình, trong bài toán DRL thì độ dài giữa hai thành phố chính là phần thưởng nhận được, độ dài hai thành phố càng ngắn thì hàm phần thưởng có giá trị càng lớn.
- $P(\pi(i)|\pi(< i), s)$: là xác suất xảy ra việc thăm quan các thành phố theo chính sách π.
- π : là một chính sách và mạng Pointer qua huấn luyện sẽ học được chính sách tối ưu.
- s : là môi trường hay chính là input đầu vào gồm tọa độ 2 chiều của các thành phố.



Hình 2.1: Kiến trúc Pointer Network [11]

Mục đích của mạng Pointer là để học được bộ tham số $P(\pi(i)|\pi(< i), s)$ với chỉ số i = 1,...,n bằng cách tối ưu tham số θ.

Mạng Pointer sẽ gồm 2 khối mạng *RNN* là Encoder dùng để đọc chuỗi input và Decoder dùng để dự đoán chuỗi output chính là thứ tự đi qua các thành phố. Cả 2 khối trên đều bao gồm các mạng LSTM, cụ thể như sau:

- **Khối Encoder:** được dùng để đọc chuỗi đầu vào s, mỗi lần 1 thành phố và chuyển nó thành chuỗi các trạng thái nhớ ẩn *(latent memory)* dưới dạng $\{enc_i\}_{i=1}^{n}$ với $enc_i \in \mathbb{R}^d$. Đầu vào cho mạng Encoder ở bước thứ i là một mảng d chiều ứng với thành phố x_i. Bước biển đổi này thông qua một hàm biến đổi *(Embedding)* tuyến tính được dùng cho mọi bước lặp của mạng biến đổi từ tọa độ x_i 2 chiều sang mảng d chiều.
- **Khối Decoder:** Khối Decoder cũng lưu trữ các trạng thái nhớ ẩn dưới dạng $\{dec_i\}_{i=1}^{n}$ với $dec_i \in \mathbb{R}^d$ và tại mỗi bước lặp i mạng sử dụng một cơ chế trỏ *pointing mechanism* để sinh ra một phân phối đưa ra xác suất thăm các thành phố tiếp theo. Khi một thành phố đã được thăm thì nó sẽ được chuyển thành input cho bước lặp tiếp theo của khối Decoder. Input dữ liệu đầu tiên của khối Decoder, được ký hiệu bởi $\langle g \rangle$ là một vector d chiều được coi như tham số huấn luyện của mạng Neural chứa thông tin tóm gọn của khối Encoder.

Cơ chế trỏ là đặc trưng của mạng *Pointer* giúp đưa ra phân phối xác suất thăm các thành phố cho từng bước lặp. Đồng thời, để tăng hiệu suất tính phân phối xác suất cho mô hình bằng việc kết hợp mức độ mô hình "quan tâm" hay trỏ tới một phần của dữ liệu input thì mạng *Pointer* sẽ sử dụng thêm hàm *glimpse* được trình bày ở phần dưới đây.

Cơ chế trỏ và cơ chế chú ý

Cơ chế trỏ được biểu thị thông qua hàm *attention* nhận input đầu vào là 1 vector truy vấn $q = dec_i \in \mathbb{R}^d$ (dữ liệu trạng thái ẩn của khối Decoder) và 1 dãy các vector tham chiếu ref $= \{enc_1, ..., enc_k\}$ với $enc_i \in \mathbb{R}^d$ (dữ liệu trạng thái ẩn của khối Encoder).

Cơ chế trỏ: Việc tính toán của nó được tham số hóa qua 2 ma trận *attention* $W_{ref}, W_q \in \mathbb{R}^{d\times d}$ và một vecto *attention* $v \in \mathbb{R}^d$ được tính như sau:

$$u_i = \begin{cases} v^T.tanh(W_{ref}.ref_i + W_q.q) & \text{nếu } \ i \neq \pi(j) \ \text{ với mọi } \ i < j \\ -\infty & \text{nếu ngược lại} \end{cases}$$

Hàm *attention* được định nghĩa như sau:

$$A(ref, q, W_{ref}, W_q, v) \triangleq softmax(u)$$

Hàm này có tác dụng tính xác suất thăm thành phố $\pi(j)$ được quy định bởi chính sách π cụ thể như sau:

$$P(\pi(j)|\pi(< j), s) \triangleq A(enc_{1:n}, dec_j)$$

Việc đặt trường hợp $u_i = -\infty$ biểu thị mô hình của chúng ta chỉ trỏ tới các thành phố chưa được thăm trong chu trình giúp thỏa mãn yêu cầu của bài toán TSP.

Cơ chế chú ý: Hàm *attention* sẽ tiến hành dự đoán phân phối xác suất thăm các thành phố được biểu thị qua A(ref,q) khi có k truy vấn. Đồng thời, phân phối này được tính thông qua hàm *glimpse* với sự kết hợp các tham chiếu ref_i biểu thị tỷ lệ mô hình "quan tâm" tới truy vấn q. Hàm *Glimpse G(ref,q)* nhận dữ liệu input truyền vào như hàm *attention* A và được tham số hóa bởi các ma trận $W^g_{ref}, W^g_q \in \mathbb{R}^{d\times d}$ và $v^g \in \mathbb{R}^d$ và được tính toán như sau:

$$P = A(ref, q, W^g_{ref}, W^g_q, v^g)$$
$$G(ref, q, W^g_{ref}, W^g_q, v^g) \triangleq \sum_{i=1}^{k} ref_i P_i$$

Hàm *glimpse* G được tính thông qua tổ hợp tuyến tính trọng số các vector tham chiếu với xác suất P_i được tính thông qua hàm *attention* từ đó hàm *glimpse* giúp mô hình tính tỷ lệ "quan tâm" tới truy vấn q. Hàm này có thể được sử dụng nhiều lần trên cùng bộ tham chiếu ref :

$$g_0 \triangleq q$$
$$g_l \triangleq G(ref, g_{l-1}, W^g_{ref}, W^g_q, v^g)$$

Giá trị vector g_l được nạp vào hàm *attention* $A(ref, q, W^g_{ref}, W^g_q, v^g)$ để tính xác suất cho cơ chế trỏ, quá trình thực nghiệm chỉ ra rằng việc sử dụng cơ chế tham dự làm tăng hiệu suất của mô hình.

Quy tắc luyện Actor-Critic

Việc huấn luyện mô hình Pointer để giải bài toán TSP theo kiểu Học có giám sát *supervised* thường tốn chi phí cực lớn trong việc gán nhãn dữ liệu, việc gán nhãn các chu trình tốt với đồ thị dày với nhiều đỉnh là điều cực kỳ tốn kém chính vì vậy ta cần huấn luyện mô hình mạng *Pointer* bằng các phương pháp *Học không giám sát unsupervised [11]* và tiến hành huấn luyện theo phương pháp *model-free policy-based* cụ thể là phương pháp *policy gradient* để huấn luyện mạng Pointer đưa ra tham số θ tối ưu. Hàm mục tiêu của mô hình là hàm trung bình độ dài chu trình khi cho trước chuỗi input s, được định nghĩa như sau:

$$J(\theta|s) = \mathbb{E}_{\pi \sim p_\theta(.|s)} L(\pi|s) \tag{2.1}$$

Trong suốt quá trình huấn luyện, các chuỗi input được lấy mẫu từ phân phối S, do đó hàm tổng mục tiêu huấn luyện được tính như sau:

$$J(\theta) = \mathbb{E}_{s \sim S} J(\theta|s)$$

Theo thuật toán REINFORCE [18] đạo hàm của (2.1) được tính như sau:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi \sim p_\theta(.|s)}[(L(\pi|s) - b(s)) \nabla_\theta log p_\theta(\pi|s)]$$

Ta chọn hàm *baseline* b(s) là dạng trung bình tăng theo hàm mũ (exponential moving average) của các phần thưởng mà mạng thu được để biểu diễn chính sách được huấn luyện tốt dần theo thời gian huấn luyện. Việc sử dụng hàm baseline để xấp xỉ độ dài trung bình của chu trình $\mathbb{E}_{\pi \sim p_\theta(.|s)} L(\pi|s)$ bằng thực nghiệm đã chứng tỏ tăng độ hiệu quả cho mô hình. Do đó, chúng ta sẽ xây dựng thêm mạng *Critic* được tham số hóa bởi θ_v để xấp xỉ độ dài chu trình trung bình đang được xét tới bởi chính sách p_0 khi cho trước chuỗi dữ liệu s. Mạng *critic* được huấn luyện bởi phương pháp *stochastic gradient descent* thông qua hàm trung bình bình phương sai số *(MSE)* hay là hàm đánh giá sai số chuẩn l_2 được tính giữa giá trị hàm baseline $b_{\theta_v}(s)$ mà mạng *Critic* đang dự đoán và độ dài thật sự của chu trình được tính thông qua chính sách gần nhất vừa được lấy mẫu, cụ thể:

$$L(\theta_v) = \frac{1}{B}\sum_{i=1}^{B} ||b_{\theta_v}(s_i) - L(\pi_i|s_i)||_2^2 \qquad \textit{với } B \textit{ là mẫu i.i.d}$$

Kết hợp quá trình lấy mẫu cho s, π xây dựng *Pointer network* và mạng *Critic* qua T bước huấn luyện cũng như sử dụng phương pháp *policy gradient* để huấn luyện mô hình thì ta thu được quy tắc luyện *Actor-Critic* được trình bày tại **Thuật toán 2.1**:

Thuật toán 2.1 *Actor-Critic training [11]*

procedure *TRAIN(training set S, number of training steps T, batch size B)*
    Initialize pointer network params θ
    Initialize critic network params θ_v
    for $t \leftarrow 1$ *to* T ***do***
        $s_i \sim SAMPLEINPUT(S)$ *for* $i \in \{1, ..., B\}$
        $\pi_i \sim SAMPLESOLUTION(p_\theta(.|s_i))$ *for* $i \in \{1, ..., B\}$
        $b_i \leftarrow b_{\theta_v}(s_i)$ *for* $i \in \{1, ..., B\}$
        $g_\theta \leftarrow \frac{1}{B}\sum_{i=1}^{B}(L(\pi_i|s_i) - b_i)\nabla_\theta log p_\theta(\pi_i|s_i)$
        $\mathcal{L}_v \leftarrow \frac{1}{B}\sum_{i=1}^{B} ||b_i - L(\pi_i)||_2^2$
        $\theta \leftarrow ADAM(\theta, g_\theta)$
        $\theta_v \leftarrow ADAM(\theta_v, \nabla_{\theta_v}, \mathcal{L}_v)$
    end for
    return θ
end procedure

2.2 Deep Reinforcement Learning giải bài toán MOTSP

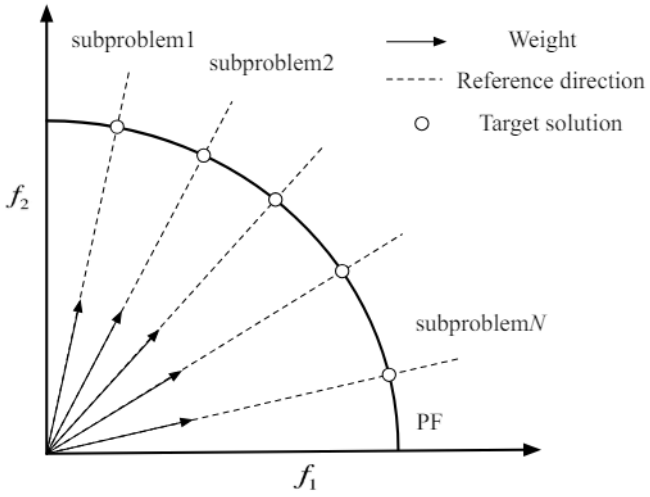
Trong phần này, em xin trình bày mô thức chung để giải bài toán MOTSP đó là DLR-MOA. Mô thức này gồm hai bước chính là sử dụng phương pháp phân rã (decomposition) để chuyển bài toán tối ưu đa mục tiêu thành các bài toán con và mỗi bài toán con đó sẽ được mô hình hóa bằng một mạng neural. Các tham số của các mạng neural sau đó được kết hợp lại thông qua chiến lược tham số lân cận (neighborhood-based parameter transfer) và cuối cùng được huấn luyện thông qua quy tắc luyện Actor-Critic của Học tăng cường.

2.2.1 Mô thức của bài toán DRL-MOA

Một mô thức được sử dụng cho thuật toán toán DRL-MOA bao gồm 2 chiến lược kết hợp đồng thời với nhau:

Chiến lược phân rã

Chiến lược phân rã là một chiến lược thường được dùng trong việc giải các bài toán tối ưu đa mục tiêu. Ý tưởng chính của chiến lược này là chuyển bài toán tối ưu đa mục tiêu ban đầu thành các bài toán vô hướng con và giải lần lượt các bài toán con đó và kết hợp nghiệm Pareto của các bài toán con này để thu được Pareto Front của bài toán ban đầu. Một cách tiếp cận đó là thực hiện vô hướng hóa để thực hiện giải bài toán MOTSP. Thực hiện xây dựng bộ vecto trọng số $\lambda^1, \lambda^2, ..., \lambda^n$ cho n bài toán con sau khi phân rã. Cụ thể, $\lambda^j = (\lambda_1^j, ..., \lambda_M^j)^T$, với M biểu diễn số mục tiêu cần tối ưu.



Hình 2.2: Lược đồ phân rã [12]

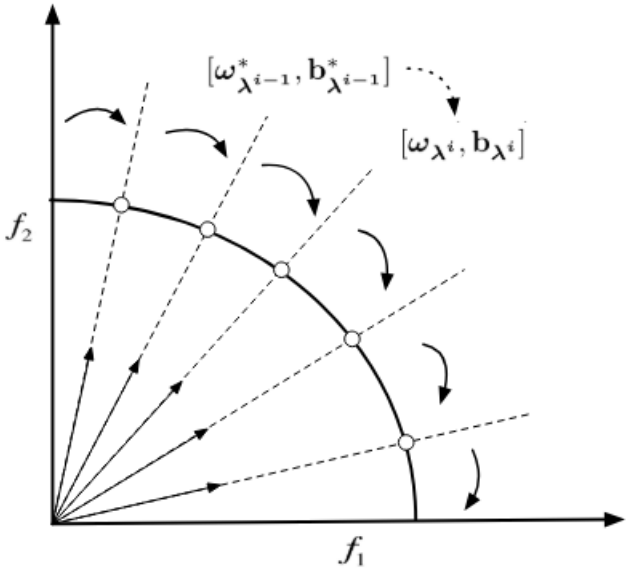
Bài toán tối ưu đa mục tiêu ban đầu được phân rã thành N bài toán tối ưu vô hướng con bằng phương pháp vô hướng hóa. Hàm mục tiêu của bài toán con thứ j được biểu diễn [21] như sau:

$$\min g^{ws}(x|\lambda_i^j) = \overset{M}{_{i=1}}\lambda_i^j f_i(x)$$

Cuối cùng, đường cong nghiệm *Pareto Front* được xác định bằng cách kết hợp nghiệm tối ưu của N bài toán con này.

Chiến lược tham số lân cận (Neighborhood-based parameter-transfer strategy)

Tiến hành giải N bài toán con vô hướng bằng cách mô hình chúng dưới dạng các mạng Neural. Bằng các chứng minh và thực nghiệm từ [21] thì việc giải các bài toán con cho nghiệm tối ưu gần nhau nếu như các vector trọng số liền kề. Do đó, một bài toán con có thể được giải khi biết trước nghiệm tối ưu từ bài toán con lân cận của nó.



Hình 2.3: Chiến lược tham số lân cận [12]

Cụ thể, bài toán con trong công việc này được mô hình hóa Là một mạng nơ-ron, các tham số của mô hình mạng của bài toán con thứ (i-1) biểu diễn dưới dạng $[\omega_{\lambda^{i-1}}, b_{\lambda^{i-1}}]$. Ký hiệu $[\omega^*, b^*]$ biểu thị tham số của mạng Neural đã tối ưu và $[\omega, b]$ biểu thị cho các tham số chưa tối ưu. Giả sử bài toán i - 1 đã được giải và cho ra nghiệm tối ưu xấp xỉ khi đó tham số $[\omega^*_{\lambda^{i-1}}, b^*_{\lambda^{i-1}}]$ được dùng làm tham số ban đầu khi bắt đầu huấn luyện mạng Neural tương ứng cho bài toán con thứ i. Bằng thực nghiệm đã chỉ ra rằng chiến lược tham số lân cận giúp giải N bài toán tối ưu con nhanh chóng hiệu quả hơn phương pháp giải lần lượt N bài toán con đó. Thuật toán giải bài toán MOTSP theo mô thức chung của thuật toán DRL-MOA được mô tả:

Thuật toán 2.2 *General Framework of DRL-MOA [12]*

Input: *The model of the subproblem* $\mathcal{M}$ = ***[w,b]***, *weight vectors* $\lambda^1, ..., \lambda^N$

Output: *The optimal model* $\mathcal{M}^* = [w^*, b^*]$

$[w_{\lambda^1}, b_{\lambda^1}] \leftarrow Random\_Initialize$
for $i \leftarrow 1 : N$ ***do***
    if $i == 1$ ***then***
        $[w^*_{\lambda^1}, b^*_{\lambda^1}] \leftarrow Actor\_Critic([w_{\lambda^1}, b_{\lambda^1}], g^{ws}(\lambda^1))$
    else
        $[w_{\lambda^i}, b_{\lambda^i}] \leftarrow [w^*_{\lambda^{i-1}}, b^*_{\lambda^{i-1}}]$
        $[w^*_{\lambda^i}, b^*_{\lambda^i}] \leftarrow Actor\_Critic([w_{\lambda^i}, b_{\lambda^i}], g^{ws}(\lambda^i))$
    end if
end for
return $[w^*, b^*]$
Given inputs of the MOP, the PF can be directly calculated by $[w^*, b^*]$.

Khi đã giải được N bài toán con thì đường cong nghiệm *Pareto Front* sẽ được giải thông qua việc chạy forward mô hình một lần, đồng thời với bộ tham số tối ưu của mô hình thì ta không cần phải huấn luyện lại mô hình cho các *test case* khác hay các đồ thị mới giúp tiết kiệm chi phí tính toán.

2.2.2 Mô hình hóa các bài toán con MOTSP

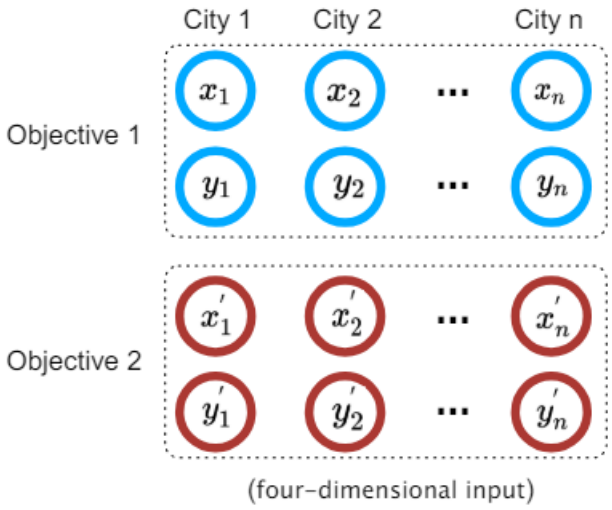
Mạng Pointer được điều chỉnh để mô hình hóa cho các bài toán vô hướng con đồng thời quy tắc luyện *Actor-Critic* sẽ được sử dụng. Ta cần tìm chu trình đi qua n thành phố được ký hiệu dưới dạng 1 hoán vị tuần hoàn p, để tối ưu M hàm mục tiêu khác nhau, hàm chi phí *(cost function)* sẽ được phát biểu như sau:

$$\min\ z_k(p) = \sum_{i=1}^{n-1} c^k_{p(i),p(i+1)} + c^k_{p(n),p(1)}, \quad \text{k = 1, 2, 3,..., M}$$

với $c^k_{p(i),p(i+1)}$ là hàm chi phí thứ k_{th} của việc chuyển từ thành phố $p(i)$ đến thành phố $p(i+1)$ theo chính sách p.

Mô tả Input-Output dữ liệu

Mô hình sẽ có dữ liệu đầu vào là $X = \{s^i, i = 1, ..., n\}$ với n là số thành phố. Mỗi phần tử s^i được biểu diễn dưới dạng chuỗi $\{s^i = (s^i_1, ..., s^i_M)\}$ Trong đó, s^j_i là thuộc tính của thành phố thứ i được dùng để tính hàm chi phí thứ j, luôn luôn $s^i_1 = (x_i, y_i)$ biểu diễn cặp tọa độ (x, y) của thành phố thứ i và được dùng để tính độ dài giữa hai thành phố. Lấy TSP hai mục tiêu làm ví dụ, trong đó cả hai hàm chi phí đều được xác định bởi hàm *Euclide distance [16]*. Trực quan dữ liệu đầu vào bài toán MOTSP 2 mục tiêu như sau:



Hình 2.4: Mô tả Input dữ liệu [12]

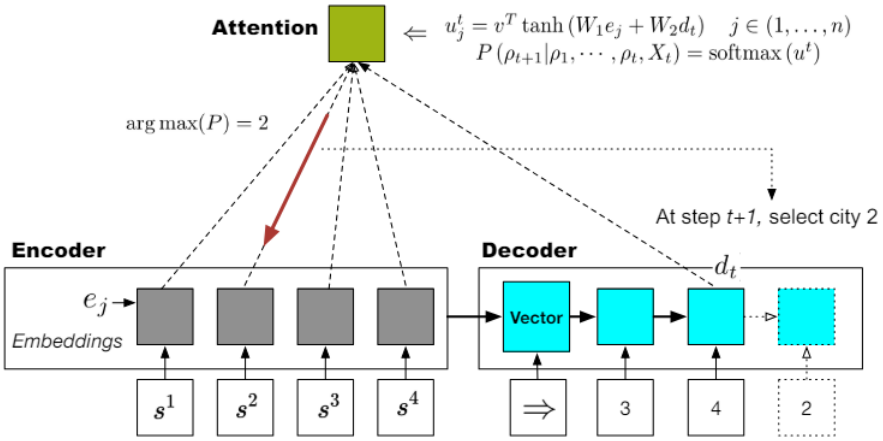
Trong trường hợp có 2 hàm mục tiêu, ta có biểu diễn cấu trúc input dữ liệu. Đầu ra của mô hình là các tổ hợp chu trình di chuyển giữa các thành phố như sau: $Y = \{p_1, ..., p_n\}$. Để ánh xạ Input X đến Output Y thì xác suất được tính thông qua tích các sự kiện như sau:

$$P(Y|X) = \prod_{t=1}^{n} P(p(t+1)|p(1), ..., p(t), X_t)$$

Thành phố đầu tiên được chọn ngẫu nhiên là p_1. Ở mỗi bước lặp decode t = 1, 2, ... , ta chọn $p(t_1$ từ thành phố Xt và các thành phố đã thăm $p(1), ..., p(t)$, ta sử dụng cấu trúc mạng *Sequence-to-Sequence* gồm 2 khối *Encoder* và *Decoder*, Khối *Encoder* sẽ đọc chuỗi input đầu vào và tổng hợp tạo ra 1 vector ngữ cảnh *(context)* tổng hợp tri thức ban đầu và chuyển

vào cho khối *Decoder* để từ đó khối *Decoder* sẽ tạo ra chuỗi mong muốn, đối với bài toán MOTSP là chu trình p tối ưu.

Kiến trúc Pointer Network kết hợp khối Encoder và Decoder



Hình 2.5: Cơ chế chú ý kết hợp Encoder và Decoder tạo ra xác suất lựa chọn tiếp theo thành phố. [12]

Cơ chế chú ý

Cơ chế này có tác dụng tính xem độ quan tâm của mô hình tới dữ liệu input đầu vào trong việc sử dụng dữ liệu để dự đoán thành phố ở bước *decode* thứ t. Thành phố càng được "quan tâm" bởi mô hình thì càng dễ được chọn. Cụ thể việc tính toán như sau:

$$u_j^t = v^T.tanh(W_1 e_j + W_2 d_t) \qquad j \in \overline{1, n}$$
$$P(p_{t+1}|p_1, ..., p_t, X_t) = softmax(u^t)$$

trong đó sử dụng các tham số học như vector v và hai ma trận trọng số tương ứng với dữ liệu trạng thái ẩn của khối Encoder và Decoder là W_1, W_2 là các tham số học của mô hình. Dữ liệu trạng thái ẩn d_t là chìa khóa trong việc tính xác suất $P(p_{t+1}|p_1, ..., p_t, X_t)$ do nó lưu trữ dữ liệu từ các bước lặp trước $p_1, ..., p_t$.

Trong quá trình huấn luyện, thành phố tiếp theo được chọn khi có xác suất cao nhất được tính theo cơ chế tham dự tuy nhiên mô hình vẫn sẽ chọn thành phố tiếp theo thông qua việc chọn mẫu từ một phân phối xác suất biểu diễn tỷ lệ thăm thành phố. Quá trình này mang ý nghĩa khám phá *exploration* trong *Reinforcement Learning*.

Khối Encoder

Ở đây, cách thiết kế khối Encoder có sự khác biệt hơn khi sử dụng 1 khối tích chập 1 chiều *1-D convolution layer* để chuyển vecto đầu vào thành 1 vecto nhiều chiều d_h (cụ thể $d_h = 128$). Đầu ra của khối Encoder là môt $n \times d_h$ vector. Việc chỉ sử dụng 1 lớp tích chấp 1 chiều giúp tiết kiệm chi phí tính toán do thứ tự các thành phố trong chuỗi Input không có ý nghĩa nên ta không cần dùng các khối RNN để đọc chuỗi Input đầu vào mà chỉ cần 1 khối tích chập để lưu trữ thông tin. Tham số của khối tích chập 1-D được dùng cho toàn bộ các thành phố.

Khối Decoder

Khối Decoder có tác dụng chuyển thông tin từ chuỗi thành phố đã có $p(1), p(2), ..., p(t)$ để tiến hành chọn ra thành phố $p(t + 1)$. Do đó, khối Decoder cần sử dụng 1 mạng RNN để chắt lọc thông tin từ chuỗi đã dự đoán và input để dự đoán thành phố tiếp theo sẽ tới. Ở đây sử dụng khối GRU để tiết kiệm chi phí tính toán do GRU vẫn giảm được hiện tượng *vanishing gradient* và cũng có ít tham số cần luyện hơn khối LSTM. Mạng Pointer sẽ sử dụng thông tin của các trạng thái nhớ ẩn lưu trữ dữ liệu từ khối Encoder là $e_!, e_2, ..., e_n$ và thông tin của các trạng thái nhớ ẩn lưu trữ dự liệu tại bước thứ t là dt của khối Decoder. Việc tính toán được mô hình thực hiện thông qua cơ chế *Attention mechanism* được trình bày ở trên.

Huấn luyện Actor-Critic

Mô hình huấn luyện sẽ gồm 2 khối Actor và Critic [17]. Nhiệm vụ của khối Actor là chọn các thành phố tối ưu cho từng bước lặp theo chính sách ϵ-greedy của Học tăng cường mang ý nghĩa cho việc khai thác *exploitation.* Khối Critic đóng vai trò tối ưu cho toàn bộ chính sách p mang ý nghĩa tối ưu dài hạn và đóng vai tròn cho quá trình khám phá *exploration.* Cụ thể như sau:

- Khối Actor tính xác suất thăm thành phố tiếp theo. Khối Actor được mô hình hóa bằng *Pointer Network.*
- Khối Critic đánh giá phần thưởng trung bình thu được trong quá trình huấn luyện mạng và được mô hình hóa cũng bởi *Pointer Network.*

Trong quá trình huấn luyện, ta sẽ tiến hành tạo ra các phân phối xác suất $\{\Phi_{N_1}, ..., \Phi_{N_M}\}$ để dữ liệu được chọn từ các mẫu này. Trong đó, N biểu diễn cho các đặc trưng khác nhau của dữ liệu input, ví dụ như tọa độ hay độ an toàn của các thành phố. Trong trường hợp cụ thể như bài toán MOTSP 2 mục tiêu cần tối ưu là N_1, N_2 đều là các đặc trưng biểu thị tọa độ của thành phố được biểu diễn dưới dạng tọa độ Euclide thì chúng có thể có phân phối đều $[0, 1] \times [0, 1]$.

Hai mạng *Actor* và *Critic* được tham số hóa thông qua hai tham số lần lượt là θ và ϕ. Để huấn luyện 2 mạng này thì trong quá trình *training* N ví dụ *(instance)* hay dữ liệu input biểu thị cho N bài toán con được lấy mẫu từ các phân phối $\{\Phi_{N_1}, ..., \Phi_{N_M}\}$. Với mỗi ví dụ, ta sử dụng khối Actor với tham số θ hiện tại để tạo ra chu trình qua n thành phố và tính giá trị phần thưởng thu được. Sau đó sử dụng *policy gradient* để cập nhật lại tham số θ. $V(X_0^k, \phi)$ là phần thưởng xấp xỉ cho bài toán con thứ k được tính bởi khối *Critic*.

Thuật toán 2.3 *Actor-Critic training algorithm [12]*

Input: $\theta, \phi \leftarrow$ *initialized parameters given in Algorithm 1*

Output: *The optimal parameters* θ, ϕ

for *iteration* $\leftarrow$ *1,2,...* ***do***
  generate T problem instances from $\{\Phi_\infty, ..., \Phi_\mathcal{M}\}$ *for the MOTSP*
  for $k \leftarrow 1, ..., T$ ***do***
    $t \leftarrow 0$
    while *not terminated* ***do***
      select the next city p_{t+1}^k *according to* $P(p_{t+1}^k | p_1^k, ..., p_t^k, X_t^k)$
      Update X_t^k *to* X_{t+1}^k *by leaving out the visited cities*
    end while
    compute the reward R^k
  end for
  $d\theta \leftarrow \frac{1}{N} \sum_{k=1}^{N} (R^k - V(X_0^k, \phi)) \nabla_\theta log P(Y^k | X_0^k)$
  $d\phi \leftarrow \frac{1}{N} \sum_{k=1}^{N} \nabla_\phi (R^k - V(X_0^k, \phi))^2$
  $\theta \leftarrow \theta + \eta d\theta$
  $\phi \leftarrow \phi + \eta d\phi$
end for

Khi tất cả các mô hình của các bài toán con đã được huấn luyện, các giải pháp tối ưu Pareto có thể được đưa ra trực tiếp bởi một sự lan truyền tiến đơn giản của các mô hình. Thời gian độ phức tạp (time complexity) thuật toán DRL-MOA để giải bài toán MOTSP xấp xỉ là $O(Nnd_h^2)$ với N bài toán con. Vì quá trình lan truyền tiến khối Encode-Decoder trong mạng Neural có thể khá nhanh nên các giải pháp có thể luôn luôn giải được trong một thời gian hợp lý.

Chương 3

Cài đặt chương trình và trình bày kết quả thử nghiệm

3.1 Cài đặt mô hình

Trong chương này em sẽ trình bày tiến trình cài đặt thuật toán DRL-MOA và thuật toán NSGA-II đồng thời so sánh kết quả hội tụ của 2 thuật toán trên qua đồ thị.

3.1.1 Cài đặt thuật toán DRL-MOA

Để có thể triển khai thuật toán DRL-MOA trên thực tế, ta cần thực hiện hai nhiệm vụ:

1. Thiết kế khối Actor-Critic.
2. Khởi tạo dữ liệu "không nhãn" do ta đang tiến hành huấn luyện mạng theo kiểu học không giám sát.

Việc tiến hành huấn luyện 2 khối *Actor* và *Critic* đều sử dụng phương pháp tối ưu *Adam [14]* với hệ số học *learning rate* $\eta = 0.0001$ và độ dài 1 *batch* là 200. Việc khởi tạo tham số ban đầu cho mạng được tiến hành khởi tạo theo phương pháp *khởi tạo Xavier [19]*.

Thành phần khối Actor-Critic trong mạng Neural

Thông số chi tiết về kích thước các khối trong mạng GRU được xây dựng như một khối tích chập 1 chiều được mô tả dưới đây:

Actor network(Pointer Network)	
Encoder:	1D-Conv(D_{input}, 128, kernel size=1, stride=1)
Decoder:	GRU(hidden size=128, number of layer=1)
	Attention(No hyper parameters)
Critic network	
1D-Conv(D_{input}, 128, kernel size=1, stride =1)	
1D-Conv(128, 20, kernel size=1, stride =1)	
1D-Conv(20, 20, kernel size=1, stride =1)	
1D-Conv(20, 1, kernel size=1, stride =1)	

Hình 3.1: Kích thước tham số mô hình [12]

Khởi tạo dữ liệu

Trong báo cáo này, tiến hành cài đặt mô hình và thử nghiệm với ví dụ trộn *Mixed instances* để thử nghiệm khả năng thích ứng của mô hình với cấu trúc input dữ liệu khác nhau, cụ thể là dữ liệu 3 chiều với số thành phố cần đi qua lần lượt là 40, 100, 150, 200 thành phố. Mô hình sẽ tiến hành tối ưu 2 hàm mục tiêu với hàm mục tiêu thứ nhất là độ dài giữa hai thành phố được đo bằng cặp tọa độ *(x, y)* ứng với 2 chiều đầu tiên của dữ liệu, hàm mục tiêu thứ hai là đo độ cao giữa hai thành phố h ứng với chiều dữ liệu cuối cùng.

Tập dữ liệu huấn luyện tham số 2 khối *Actor-Critic* được lưu qua đường link sau: *Data Trainning Set Actor-Critic*

Tập dữ liệu huấn luyện *training set* sẽ được thiết kế dựa trên việc lấy mẫu từ mẫu phân phối đều trên đoạn [0, 1]. Do mô hình được luyện theo kiểu không giám sát nên quá trình huấn luyện chỉ cần dữ liệu 3 chiều mà chúng ta tự tạo mà *không* ảnh hưởng tới độ hiệu quả của mô hình và hàm phần thưởng được tính toán dựa trên khoảng cách *Euclide* giữa 2 thành phố và độ cao giữa 2 thành phố và không cần nhãn của dữ liệu.

Tập dữ liệu kiểm tra *Test set* sẽ được lấy từ hai bộ dữ liệu chuyên dụng cho bài toán TSB là *kroA* và *kroB* trong thư viện *TSPLIB [8]*, chúng được dùng để xây dựng các ví dụ *instances* cho biết khoảng cách *Euclide* giữa 2 thành phố A và B. Chiều cao của các thành phố sẽ được sinh ngẫu nhiên.

Môi trường và phần cứng chạy chương trình

1. Cả code nguồn cho quá trình *Trainning* cũng như *Testing* đều được thực hiện trên *Jupyter Notebook* của *Kaggle*.
2. Phần cứng bao gồm: CPU Ram 29GB, Disk space 73GB, GPU P100 memory 16GB, Output file size 19.5GB.
3. Ngôn ngữ lập trình: *Python.*
4. Source Code *Trainning Process*: *Trainning MOTSP*
5. Source Code *Testing Process*: *Testing MOTSP*

3.1.2 Cài đặt thuật toán NSGA-II

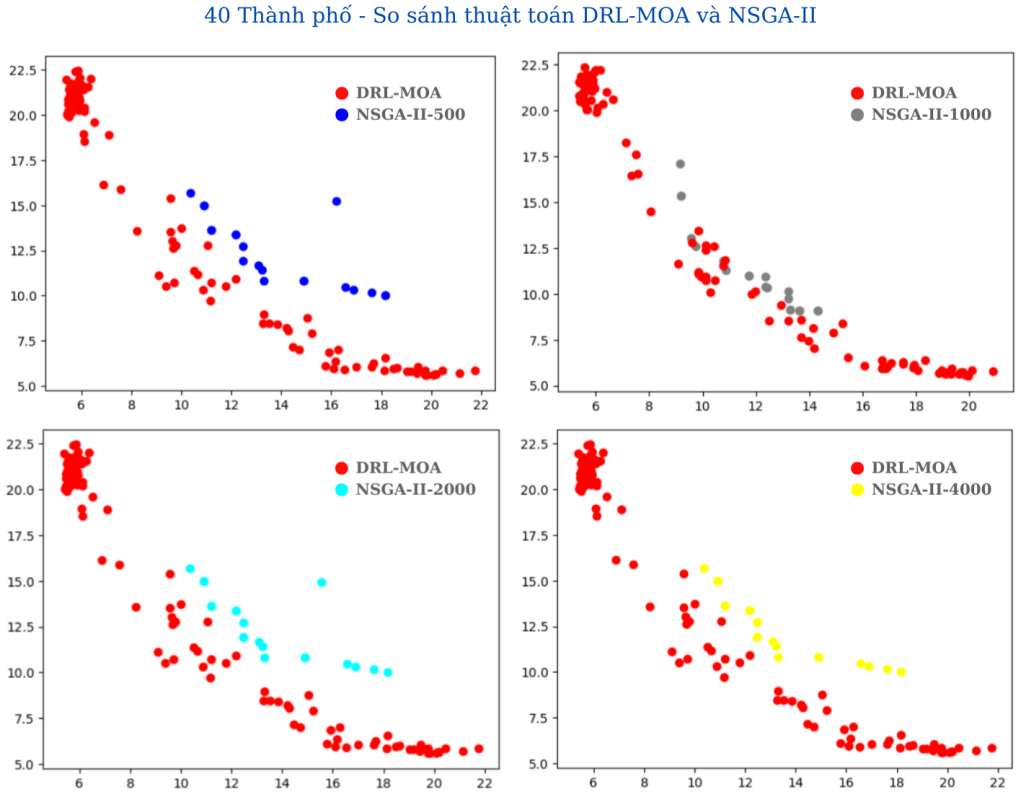
Thuật toán ứng dụng NSGA-II và framework cài đặt cho thuật toán được tham khảo từ [2], [13].

Thuật toán NSGA-II là một trong những thuật toán tiến hóa được dùng cho bài toán tối ưu đa mục tiêu và cũng đã để lại nhiều kết quả tốt trong quá trình nghiên cứu các phương pháp giải bài toán MOP. Chính vì vậy, đối với bài báo cáo này, em sẽ tiến hành cài đặt thuật toán NSGA-II để thực hiện so sánh mức độ hiệu quả của nó so với thuật toán DRL-MOA.

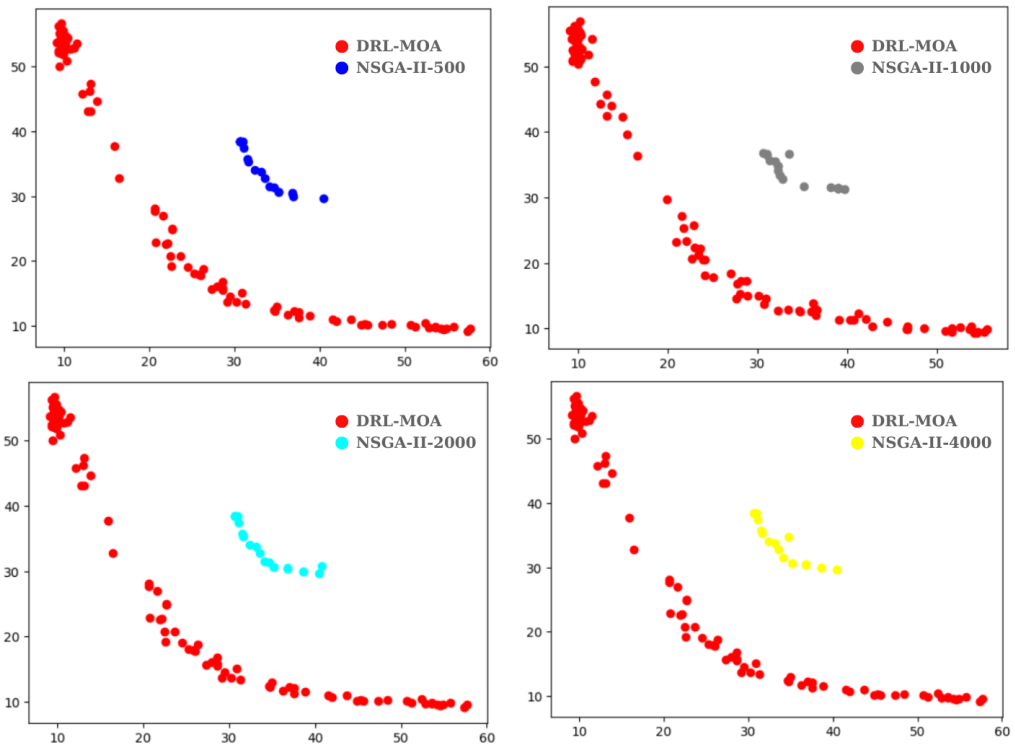
Vì thuật toán sẽ sử dụng đến ma trận kề vì vậy mà em tiến hành cài đặt thuật toán này ngay phía sau *Testing* của thuật toán DRL-MOA để tiện cho việc so sánh. Source Code *NSGA-II: Testing MOTSP JupyterNB Task 12*

3.2 Kết quả thử nghiệm và đánh giá so sánh mô hình

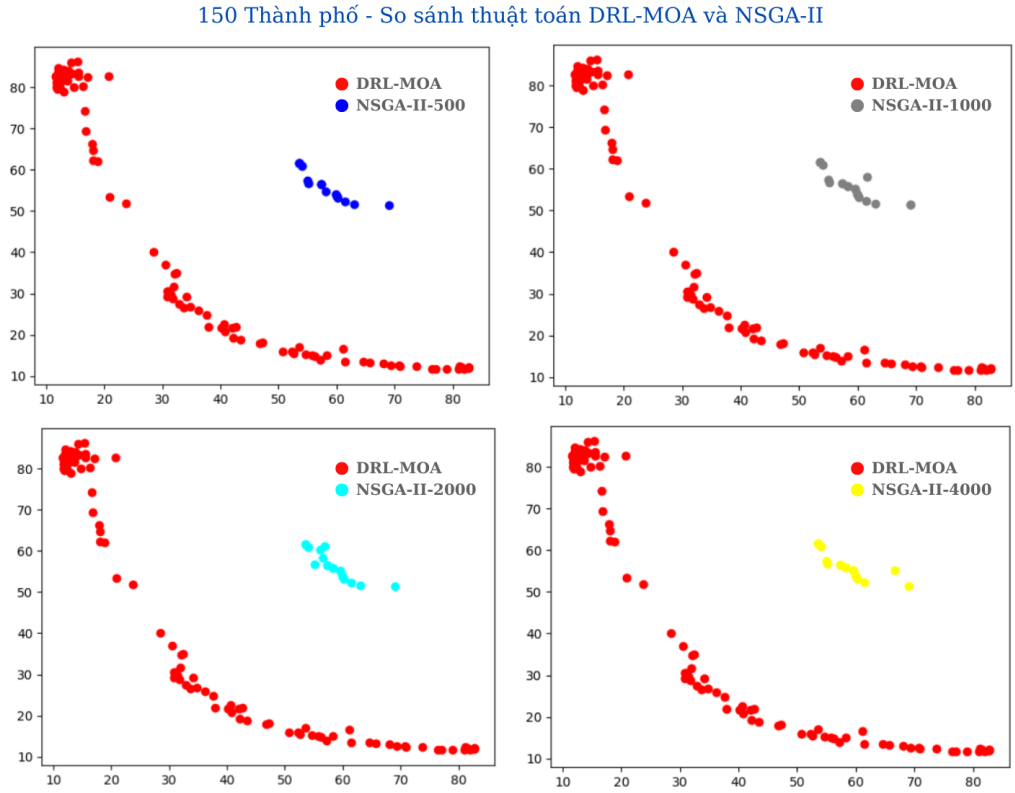
Trong báo cáo này, tác giả sẽ tiến hành cài đặt lại mô hình và tính toán trong trường hợp *Mixed instances* của dữ liệu với 2 hàm mục tiêu cần tối ưu. Mô hình sẽ tìm *Pareto Front* trong các trường hợp đồ thị có 40, 100, 150, 200 đỉnh (hay chính là số thành phố cần đi qua). Thuật toán DRL-MOA sẽ được so sánh với hai loại thuật toán NSGA-II. Thuật toán NSGA-II sẽ được thực thi với số lần lặp lần lượt là 500, 1000, 2000, 4000. Dưới đây là kết quả so sánh hai thuật toán.



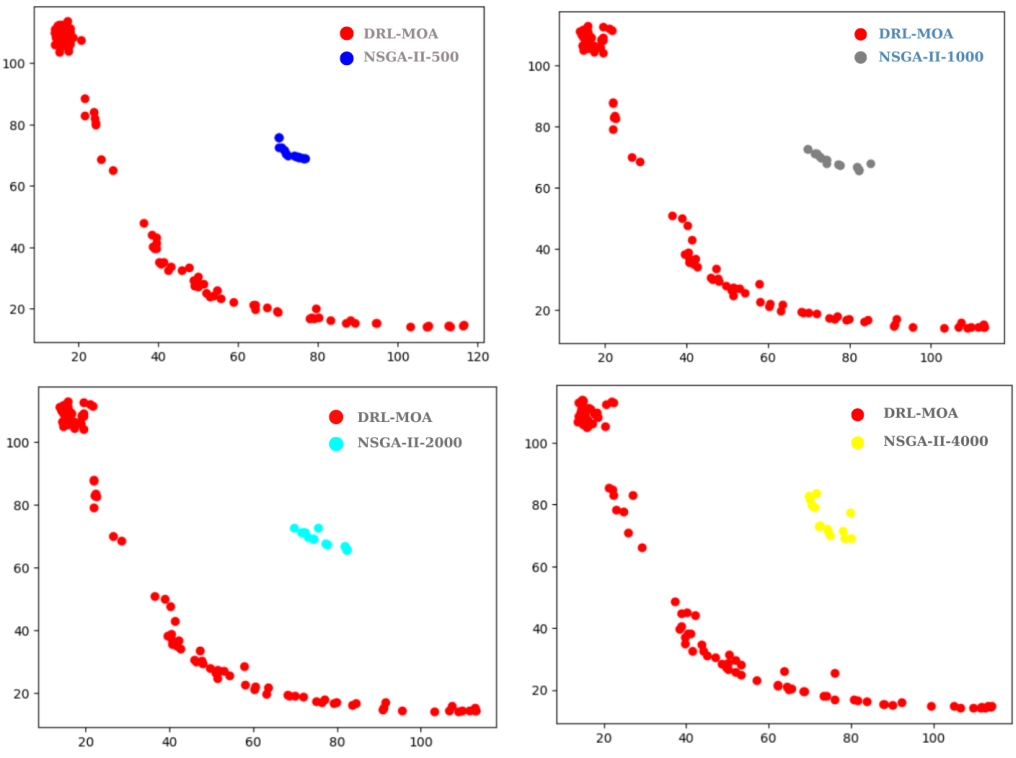
Hình 3.2: Kết quả so sánh giữa hai thuật toán trong trường hợp 40 thành phố



Hình 3.3: Kết quả so sánh giữa hai thuật toán trong trường hợp 100 thành phố



Hình 3.4: Kết quả so sánh giữa hai thuật toán trong trường hợp 150 thành phố



Hình 3.5: Kết quả so sánh giữa hai thuật toán trong trường hợp 200 thành phố

Quan sát so sánh trên có thể đánh giá được rằng đối với trường hợp 40 thành phố, cả 2 thuật toán đều cho ra *Pareto Front* tốt, có sự tương đồng về tính hội tụ giữa 2 thuật toán. Tuy nhiên khi số bài toán tăng dần tức đồ thị trở lên dày hơn thì dù qua quá trình lặp lớn 4000 vòng lặp thì thuật toán NSGA-II cho kết quả hội tụ khá kém trong khi đó đường cong Pareto được tạo ra bởi thuật toán DRL-MOA đều tốt hơn cả về độ hội tụ và tính đa dạng *(diversity)*. Có thể quan sát một cách trực quan từ đồ thị thấy rằng khi với trường hợp 100 thành phố đã có sự khác biệt đáng kể về đường cong *Pareto* tuy nhiên thuật toán NSGA-II vẫn cho thấy được đường cong ở mức nhỏ nhưng không quá kém. Khi số thành phố tăng lên 150 và 200 là kết quả chứng minh rõ nét nhất về sự khác biệt giữa DRL-MOA và NSGA-II, số lần lặp tăng lên rất nhiều nhưng mức độ hội tụ từ NSGA-II là quá kém trong những trường hợp phức tạp này. Từ đó khẳng định độ hiệu quả cao của thuật toán DRL-MOA trong việc giải bài toán MOTSP.

Kết luận

Đồ án đã đạt được mục tiêu đề ra

Kết quả của đồ án

1. Trình bày nội dung lý thuyết về tối ưu đa mục tiêu, *mạng Neural* và *Học tăng cường.*
2. Xây dựng các phương pháp giải bài toán TSP và phiên bản nâng cấp hơn là MOTSP.
3. Đồ án đã đưa ra các kết quả thực nghiệm và đánh giá mô hình sau quá trình cài đặt các mô hình *Học máy* trên thực tế.

Kỹ năng đạt được

1. Đọc và nghiên cứu các báo cáo khoa học chuyên ngành, tài liệu tham khảo uy tín trên thế giới.
2. Viết đồ án và báo cáo có trình tự, rành mạch, cụ thể.
3. Rèn luyện khả năng tư duy cho một mô hình lớn.
4. Kỹ năng code cũng trở lên thành thạo và tối ưu hơn.

Hướng phát triển của đồ án trong tương lai

Trong thời gian tới, em sẽ tiếp tục nghiên cứu một số bài toán áp dụng tới *Học máy* đặc biệt là với *Deep Reinforcement Learning* để xây dựng ý tưởng cho các đề tài sau:

- Giải các bài toán tối ưu tổ hợp đa mục tiêu khác như: bài toán *Knapsack*, bài toán *Capacitated*, *vehicle routing*,...
- Nghiên cứu các xây dựng cách thiết kế hàm mục tiêu mới, các kiến trúc mạng neural mới nhằm tăng hiệu suất của quy tắc luyện *Actor-Critic.*

- Xây dựng một số hệ thống *Reinforcement Learning BoardGame* kết hợp đào tạo *Agent* phục vụ cho mục đích đưa ra quyết định tức thời trong bài toán chứng khoán.
- Xây dựng mô hình *(Multi-Objective Reinforcement Learning)* để giải các bài toán tổ hợp động cho mạng viễn thông.

Tài liệu tham khảo

Tiếng Việt

[1] Trần Ngọc Thăng. (2017). *"Phương pháp giải một số bài toán tối ưu đa mục tiêu và ứng dụng"*. Luận án tiến sĩ toán học. ms:62460112.

Tiếng Anh

[2] B. A. Beirigo & A. G. dos Santos. (2016). *"Application of NSGA-II framework to the travel planning problem using real-world travel data," in 2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 746–753.

[3] Christos H Papadimitriou and Kenneth Steiglitz. (1977). *"On the complexity of local search for the traveling salesman problem"*. SIAM Journal on Computing 6.1. pp. 76–83.

[4] Dalla Pozza, N., Buffoni, L., Martina, S., & Caruso, F. (2022). *"Quantum reinforcement learning: the maze problem"*. Quantum Mach Intell 4: 11.

[5] Deb, K., Agrawal, S., Pratap, A.,& Meyarivan, T. (2000). *"A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II"*. In Parallel Problem Solving from Nature PPSN VI: 6th International Conference Paris, France, September 18–20, 2000 Proceedings 6 (pp. 849-858). Springer Berlin Heidelberg.

[6] [Ebook] Ian Goodfellow, Yoshua Bengio, Aaron Courville. (2016). *"Deep-Learning-The-MIT-Press"*.

[7] [Ebook] Richard S.Sutton & Andrew G.Barto. (2014-2015). *"Reinforcement Learning: An Introduction, Second edition, in progress"*.

[8] Gerhard Reinelt. (1991). *"TSPLIB-A traveling salesman problem library"*. ORSA journal on computing 3.4, pp. 376–384.

[9] Harinath Selvaraj. (2018, December). *"The Unsolved Travelling salesmen problem"*.

[10] Ibrahim A. Hameed. (2019). *"Multi-objective Solution of TravelingSalesman Problem with Time"*, pp. 4.

[11] Irwan Bello et al. *"Neural combinatorial optimization with reinforcement learning"*. In: arXiv preprint arXiv:1611.09940 (2016).

[12] Kaiwen Li , Tao Zhang & Rui Wang. *"Deep Reinforcement Learning for Multi-objective"*. In: IEEE transactions on cybernetics 51.6 (2020), pp. 3103–3114.

[13] K. Deb, A. Pratap, S. Agarwal, & T. Meyarivan. (2002).*"A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197.

[14] Kingma, D. P., & Ba, J. (2014). *"Adam: A method for stochastic optimization"*. arXiv preprint arXiv:1412.6980.

[15] Sharma, D., Kumar, A., Deb, K., & Sindhya, K. (2007, September). *"Hybridization of SBX based NSGA-II and sequential quadratic programming for solving multi-objective optimization problems"*. In 2007 IEEE congress on evolutionary computation (pp. 3003-3010). IEEE.

[16] T. Lust and J. Teghem. (2010). *"The multiobjective traveling salesman problem: a survey and a new approach."* in Advances in Multi-Objective Nature Inspired Computing. Springer. pp. 119–141.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. *"Asynchronous methods for deep reinforcement learning"*. International conference on machine learning. pp. 1928-1937.

[18] Williams, R. J. (1992). *"Simple statistical gradient-following algorithms for connectionist reinforcement learning"*. Machine learning, 8, 229-256.

[19] Xavier Glorot and Yoshua Bengio. (2010).*"Understanding the difficulty of training deep feedforward neural networks"*. Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings. pp. 249–256.

[20] Xinwu Yang, Guizeng You, Chong Zhao, Mengfei Dou & Xinian Guo. *"An Improved multi-objective genetic algorithm based on orthogonal design and adaptive clustering pruning strategy"*, fig.3, pp. 6.

[21] Zhang, Q., & Li, H. (2007). *"MOEA/D: A multiobjective evolutionary algorithm based on decomposition"*. IEEE Transactions on evolutionary computation, 11(6), 712-731.

[22] Zhengrui Zhang, Fei Wu* & Aonan Wu. *"Research on Multi-Objective Process Parameter Optimization Method in Hard Turning Based on an Improved NSGA-II Algorithm"*, pp. 4.

Chỉ mục