Lecture Note: Introduction to Artificial Intelligence
Bài Giảng: Nhập Môn Trí Tuệ Nhân Tạo

Nguyễn Quản Bá Hồng*

Ngày 17 tháng 5 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Lecture Note: Introduction to Artificial Intelligence – Bài Giảng: Nhập Môn Trí Tuệ Nhân Tạo.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/lecture/NQBH_introduction_AI_lecture.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/lecture/NQBH_introduction_AI_lecture.tex.
- *Codes*:
  - C++: https://github.com/NQBH/advanced_STEM_beyond/tree/main/AI/C++.
  - Python: https://github.com/NQBH/advanced_STEM_beyond/tree/main/AI/Python.

Mục lục

1 Basic

1.1 Search problems – Các bài toán tìm kiếm

Resources – Tài nguyên.

1. Wikipedia/search problem.

In computational complexity theory & computability theory, a *search problem* is a computational problem of finding an *admissible* answer for a given input value, provided that such an answer exists. In fact, a search problem is specified by a binary relation R where xRy iff "y is an admissible answer given x". Search problems frequently occur in graph theory & combinatorial optimization, e.g. searching for matchings, optional cliques, & stable sets in a given undirected graph.

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: https://nqbh.github.io/. GitHub: https://github.com/NQBH.

– Trong lý thuyết độ phức tạp tính toán và lý thuyết khả năng tính toán, 1 bài toán tìm kiếm là 1 bài toán tính toán tìm 1 câu trả lời có thể chấp nhận được cho 1 giá trị đầu vào nhất định, với điều kiện là câu trả lời như vậy tồn tại. Trên thực tế, 1 bài toán tìm kiếm được chỉ định bởi 1 quan hệ nhị phân R trong đó xRy nếu và chỉ nếu "y là 1 câu trả lời có thể chấp nhận được cho x". Các bài toán tìm kiếm thường xảy ra trong lý thuyết đồ thị và tối ưu hóa tổ hợp, ví dụ như tìm kiếm các phép khớp, các nhóm tùy chọn và các tập ổn định trong 1 đồ thị vô hướng nhất định.

An algorithm is said to solve a search problem if, for every input value x, it returns an admissible answer y for x when such an answer exists; otherwise, it returns any appropriate output, e.g., "not found" for x with no such answer.

Definition 1 (Search problem)**.** *If R is a binary relation s.t.* field$(R) \subseteq \Gamma^+$ *& T is a Turing machine, then T calculates f if:*

- *if x is s.t. there is some y s.t. $R(x, y)$ then T accepts x with output z s.t. $R(x, z)$ (there may be multiple y, & T need only find 1 of them).*
- *If x is s.t. there is no y s.t. $R(x, y)$ then T rejects x.*

The graph of a partial function is a binary relation, & if T calculates a partial function then there is at most 1 possible output.

A R can be viewed as a *search problem*, & a Turing machine which calculates R is also said to solve it. Every search problem has a corresponding decision problem, namely $L(R) = \{x; \exists y,\ R(x, y)\}$. This definition can be generalized to n-ary relations by any suitable encoding which allows multiple strings to be compressed into 1 string (e.g., by listing them consecutively with a delimiter).

1.2 Gradient – Độ dốc

Resources – Tài nguyên.

1. [Tiệ25]. Vũ Hữu Tiệp. *Machine Learning Cơ Bản*. Chap. 12: Gradient Descent.

Ví dụ 1 ([Tiệ25], p. 160)**.** *Xét hàm số $f(x) = x^2 + 5\sin x$, $f \in C(\mathbb{R}$ có đạo hàm $f'(x) = 2x + 5\cos x$. Giả sử xuất phát từ 1 điểm x_0, quy tắc cập nhật tại vòng lặp thứ t là*

$$x_{t+1} = x_t - \eta f'(x_t) = x_t - \eta(2x_t + 5\cos x_t).$$

Codes:

- *Python:*

```
import math
import numpy as np

# f(x) = x^2 + 5sin x
def f(x):
    return x**2 + 5*np.sin(x)

def df(x): # derivative f'(x) of f(x)
    return 2*x + 5 * np.cos(x)

x = float(input("x = "))
print("f(x) = ", f(x))
print("df(x) = ", df(x))

tol = 1e-3 # tolerance: just a small number

def gradient_descent(x0, eta): # x0: starting point, eta: learning rate
    x = [x0]
    for i in range(100):
        x_new = x[-1] - eta*df(x[-1]) # x_new: x_{t+1}, x[-1]: x_t
        if abs(df(x_new)) < tol:
            break
        x.append(x_new)
    return(x, i)

x0 = float(input("x0 = "))
eta = float(input("eta = "))
if eta <= 0:
    print("error: eta must be positive!")
else:
    print(gradient_descent(x0, eta))
```

Bài toán 1. *Xét hàm số* $f(x) = x^3 + 3x^2 + 5\sin x - 7\cos x + \sqrt{2}e^{-2x}$. *Viết chương trình* C/C++, Python *để: (a) Tính hàm* $f(x), f'(x)$ *với* $x \in \mathbb{R}$ *được nhập từ bàn phím. (b) Viết hàm gradient descent theo công thức*

$$x_{t+1} = x_t - \eta f'(x_t),$$

với $\eta \in (0, \infty)$ *được gọi là* tốc độ học (learning rate).

Chứng minh. Dễ thấy $f(x)$ là 1 hàm liên tục trên $\mathbb{R}$, i.e., $f \in C(\mathbb{R}$, & có đạo hàm $f'(x) = 3x^2 + 6x + 5\cos x + 7\sin x - 2\sqrt{2}e^{-2x}$.
Code Python:

```
# f1(x) = x^3 + 3x^2  + 5sin x - 7cos x + sqrt{2}e^{-2x}
def f1(x):
    return x**3 + 3*x**2 + 5*np.sin(x) - 7*np.cos(x) + np.sqrt(2)*np.exp(-2*x)

def df1(x):
    return 3*x**2 + 6*x + 5*np.cos(x) + 7*np.sin(x) - 2*np.sqrt(2)*np.exp(-2*x)

x = float(input("x = "))
print("f(x) = ", f(x))
print("df(x) = ", df(x))

tol = 1e-3 # tolerance: just a small number

def gradient_descent_f1(x0, eta): # x0: starting point, eta: learning rate
    x = [x0]
    for i in range(100):
        x_new = x[-1] - eta*df1(x[-1]) # x_new: x_{t+1}, x[-1]: x_t
        if abs(df1(x_new)) < tol:
            break
        x.append(x_new)
    return(x, i)

x0 = float(input("x0 = "))
eta = float(input("eta = "))
if eta <= 0:
    print("error: eta must be positive!")
else:
    print(gradient_descent_f1(x0, eta))
```

□

Remark 1. *Có thể tham khảo các công thức tính đạo hàm ở Wikipedia/tables of derivatives.*

Bài toán 2. *Xét hàm số* $f(x,y) = 2x^3y^2 + \dfrac{\sqrt{x^3}}{y} + \sin(x^2y) + e^{\cos(xy^2)}$. *Viết chương trình* C/C++, Python *để: (a) Tính hàm* $f(x,y), \nabla f(x,y)$ *với* $x, y \in \mathbb{R}$ *được nhập từ bàn phím. (b) Viết hàm gradient descent cho 2 trường hợp:*

$$(x_{t+1}, y_{t+1}) = (x_t, y_t) - \eta \nabla f(x_t, y_t),$$

or

$$\begin{cases} x_{t+1} = x_t - \boldsymbol{\alpha} \cdot \nabla f(x_t, y_t) = x_t - \alpha_1 \partial_x f(x_t, y_t) - \alpha_2 \partial_x f(x_t, y_t), \\ y_{t+1} = y_t - \boldsymbol{\beta} \cdot \nabla f(x_t, y_t) = x_t - \beta_1 \partial_x f(x_t, y_t) - \beta_2 \partial_x f(x_t, y_t), \end{cases}$$

Python:

```
# f(x,y) = 2x^3y^2 + sqrt(x^3)/y + sin(x^2y) + e^{cos(xy^2)}

def f(x, y):
    return 2*x**3*y**2 + np.sqrt(x**3)/y + np.sin(x**2 * y) + np.exp(np.cos(x * y**2))

def grad_f(x, y):
    df_dx = 6*x**2 * y**2 + (3/2) * x**0.5 / y + 2*x*y * np.cos(x**2 * y) - y**2 * np.sin(x * y**2) * np.exp(np.
    df_dy = 4*x**3 * y - np.sqrt(x**3) / y**2 + x**2 * np.cos(x**2 * y) - 2*x*y * np.sin(x * y**2) * np.exp(np.c
    return np.array([df_dx, df_dy])

x = float(input("x = "))
```

```
y = float(input("y = "))
print("f(x,y) = ", f(x,y))
print("grad f(x,y) = ", grad_f(x,y))
```

1.3 Minimax

Resources – Tài nguyên.

1. Wikipedia/minimax.

Minimax (sometimes *Minmax, MM, or saddle point*) is a decision rule used in AI, decision theory, game theory, statistics, & philosophy for *minimizing* the possible loss for a worse case (maximum loss) scenario. When dealing with gains, it is referred to as "maximin" – to maximize the minimum gain. Originally formulated for several-player zero-sum game theory, covering both the cases where players take alternate moves & those where they make simultaneous moves, it has also been extended to more complex games & to general decision-making in the presence of uncertainty.

– *Minimax* (đôi khi là *Minmax, MM*, hoặc *điểm yên ngựa*) là 1 quy tắc quyết định được sử dụng trong trí tuệ nhân tạo, lý thuyết quyết định, lý thuyết trò chơi, thống kê và triết học để giảm thiểu tổn thất có thể xảy ra cho 1 kịch bản xấu nhất (tổn thất tối đa). Khi giải quyết các khoản lợi nhuận, nó được gọi là "maximin" – để tối đa hóa lợi nhuận tối thiểu. Ban đầu được xây dựng cho lý thuyết trò chơi tổng bằng không của nhiều người chơi, bao gồm cả các trường hợp mà người chơi thực hiện các nước đi thay thế và các trường hợp mà họ thực hiện các nước đi đồng thời, nó cũng đã được mở rộng sang các trò chơi phức tạp hơn và ra quyết định chung khi có sự không chắc chắn.

2 Heuristic Algorithms – Các Thuật Giải Heuristic

- **heuristic** [a] (formal) heuristic teaching or education encourages you to learn by discovering things for yourself.
- **heuristics** [n] [uncountable] (formal) a method of solving problems by finding practical ways of dealing with them, learning from past experience.

2.1 Heuristic (Computer Science) – Tự tìm tòi (Khoa Học Máy Tính)

Resources – Tài nguyên.

1. Wikipedia/heuristic (CS).

In mathematical optimization & CS, *heuristic* (from Greek "I find, discover") is a technique designed for problem solving more quickly when classic methods are too slow for finding an exact or approximate solution, or when classic methods fail to find any exact solution in a search space. This is achieved by trading optimality, completeness, accuracy or precision for speed. In a way, it can be considered a shortcut.

– Trong tối ưu hóa toán học và khoa học máy tính, heuristic (từ tiếng Hy Lạp "Tôi tìm thấy, khám phá") là 1 kỹ thuật được thiết kế để giải quyết vấn đề nhanh hơn khi các phương pháp cổ điển quá chậm để tìm ra giải pháp chính xác hoặc gần đúng, hoặc khi các phương pháp cổ điển không tìm thấy bất kỳ giải pháp chính xác nào trong không gian tìm kiếm. Điều này đạt được bằng cách đánh đổi tính tối ưu, tính hoàn chỉnh, tính chính xác hoặc độ chính xác để lấy tốc độ. Theo 1 cách nào đó, nó có thể được coi là 1 lối tắt.

A *heuristic function*, also simply called a *heuristic*, is a function that ranks alternatives in serch algorithms at each branching step based on available information to decide which brand to follow. E.g., it may approximate the exact solution.

– Một hàm heuristic, hay còn gọi đơn giản là heuristic, là 1 hàm xếp hạng các phương án thay thế trong thuật toán tìm kiếm tại mỗi bước phân nhánh dựa trên thông tin có sẵn để quyết định nhánh nào sẽ theo. Ví dụ, nó có thể xấp xỉ giải pháp chính xác.

Motivation of heuristic. The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.

– **Động lực của phương pháp tìm kiếm.** Mục tiêu của phương pháp tìm kiếm là đưa ra 1 giải pháp trong 1 khung thời gian hợp lý đủ tốt để giải quyết vấn đề đang xét. Giải pháp này có thể không phải là giải pháp tốt nhất trong tất cả các giải pháp cho vấn đề này hoặc có thể chỉ đơn giản là xấp xỉ giải pháp chính xác. Nhưng nó vẫn có giá trị vì việc tìm ra nó không đòi hỏi quá nhiều thời gian.

Heuristics may produce results by themselves, or they may be used in conjunction with optimization algorithms to improve their efficiency (e.g., they may be used to generate good seed values).

– Phương pháp tìm kiếm có thể tự tạo ra kết quả hoặc có thể được sử dụng kết hợp với các thuật toán tối ưu hóa để cải thiện hiệu quả của chúng (ví dụ: chúng có thể được sử dụng để tạo ra các giá trị hạt giống tốt).

Results about NP-hardness in theoretical computer science make heuristics the only viable option for a variety of complex optimization problems that need to be routinely solved in real-world applications.

– Kết quả về độ khó NP trong khoa học máy tính lý thuyết khiến phương pháp tìm kiếm trở thành lựa chọn khả thi duy nhất cho nhiều vấn đề tối ưu hóa phức tạp cần được giải quyết thường xuyên trong các ứng dụng thực tế.

Heuristics underlie the whole field of AI & the computer simulation of thinking, as they may be used in situations where there are no known algorithms.

– Phương pháp tìm kiếm là nền tảng cho toàn bộ lĩnh vực AI & mô phỏng suy nghĩ bằng máy tính, vì chúng có thể được sử dụng trong những tình huống không có thuật toán nào được biết đến.

Simpler problem. 1 way of achieving the computational performance gain expected of a heuristic consists of solving a simpler problem whose solution is also a solution to the initial problem.

– *Bài toán đơn giản hơn.* 1 cách để đạt được hiệu suất tính toán mong đợi của 1 phương pháp tìm kiếm là giải 1 bài toán đơn giản hơn mà giải pháp của nó cũng là giải pháp cho bài toán ban đầu.

Example 1 (Search)**.** *An example of heuristic making an algorithm faster occurs in certain search problems. Initially, the heuristic tries every possibility at each step, like the full-space search algorithm. But it can stop the search at any time if the current possibility is already worse than the best solution already found. In such search problems, a heuristic can be used to try good choices 1st so that bad paths can be eliminated early, see Wikipedia/alpha-beta pruning. In the case of best-1st search algorithms, e.g.* A* *search, the heuristic improves the algorithm's convergence while maintaining its correctness as long as the heuristic is admissible.*

– 1 ví dụ khác về phương pháp tìm kiếm giúp thuật toán nhanh hơn xảy ra trong 1 số bài toán tìm kiếm nhất định. Ban đầu, phương pháp tìm kiếm thử mọi khả năng ở mỗi bước, giống như thuật toán tìm kiếm toàn không gian. Nhưng nó có thể dừng tìm kiếm bất kỳ lúc nào nếu khả năng hiện tại đã tệ hơn giải pháp tốt nhất đã tìm thấy. Trong các bài toán tìm kiếm như vậy, phương pháp tìm kiếm có thể được sử dụng để thử các lựa chọn tốt trước để các đường dẫn xấu có thể bị loại bỏ sớm (xem cắt tỉa alpha–beta). Trong trường hợp các thuật toán tìm kiếm tốt nhất trước, chẳng hạn như tìm kiếm A**, phương pháp tìm kiếm cải thiện sự hội tụ của thuật toán trong khi vẫn duy trì tính chính xác của nó miễn là phương pháp tìm kiếm được chấp nhận.*

Example 2 (Newell & Simon: heuristic search hypothesis)**.** *In their Turing Award acceptance speech,* Allen Newell & Herbert A. Simon *discuss the heuristic search hypothesis: a physical symbol system will repeatedly generate & modify known symbol structures until the created structure matches the solution structure. Each following step depends upon the step before it, thus the heuristic search learns what avenues to pursue & which ones to disregard by measuring how close the current step is to the solution. Therefore, some possibilities will never be generated as they are measured to be less likely to complete the solution.*

– Trong bài phát biểu nhận giải thưởng Turing, Allen Newell và Herbert A. Simon thảo luận về giả thuyết tìm kiếm theo phương pháp heuristic: 1 hệ thống ký hiệu vật lý sẽ liên tục tạo ra và sửa đổi các cấu trúc ký hiệu đã biết cho đến khi cấu trúc được tạo ra khớp với cấu trúc giải pháp. Mỗi bước tiếp theo phụ thuộc vào bước trước đó, do đó tìm kiếm theo phương pháp heuristic tìm hiểu những con đường nào cần theo đuổi và những con đường nào cần bỏ qua bằng cách đo lường mức độ gần của bước hiện tại với giải pháp. Do đó, 1 số khả năng sẽ không bao giờ được tạo ra vì chúng được đo lường là ít có khả năng hoàn thành giải pháp hơn.

A heuristic method can accomplish its task by using search trees. However, instead of generating all possible solution branches, a heuristic selects branches more likely to produce outcomes than other branches. It is selective at each decision point, picking branches that are more likely to produce solutions.

– 1 phương pháp heuristic có thể hoàn thành nhiệm vụ của mình bằng cách sử dụng cây tìm kiếm. Tuy nhiên, thay vì tạo ra tất cả các nhánh giải pháp khả thi, 1 phương pháp heuristic sẽ chọn các nhánh có nhiều khả năng tạo ra kết quả hơn các nhánh khác. Nó có tính chọn lọc tại mỗi điểm quyết định, chọn các nhánh có nhiều khả năng tạo ra giải pháp hơn.

Example 3 (Antivirus software – Phần mềm diệt virus)**.** *Antivirus software often uses heuristic rules for detecting viruses & other forms of malware. Heuristic scanning looks for code &/or behavioral patterns common to a class or family of viruses, with different sets of rules for different viruses. If a file or executing process is found to contain matching code patterns &/or to be performing that set of activities, then the scanner infers that the file is infected. The most advanced part of behavior–based heuristic scanning is that it can work against highly randomized self-modifying/mutating (polymorphic) viruses that cannot be easily detected by simpler string scanning methods. Heuristic scanning has the potential to detect future viruses without requiring the virus to be 1st detected somewhere else, submitted to the virus scanner developer, analyzed, & a detection update for the scanner provided to the scanner's users.*

– Phần mềm diệt vi-rút thường sử dụng các quy tắc heuristic để phát hiện vi-rút và các dạng phần mềm độc hại khác. Quét heuristic tìm kiếm mã và/hoặc các mẫu hành vi phổ biến đối với 1 lớp hoặc họ vi-rút, với các bộ quy tắc khác nhau cho các loại vi-rút khác nhau. Nếu phát hiện thấy 1 tệp hoặc quy trình thực thi có chứa các mẫu mã trùng khớp và/hoặc đang thực hiện bộ hoạt động đó, thì trình quét sẽ suy ra rằng tệp đó đã bị nhiễm. Phần tiên tiến nhất của quét heuristic dựa trên hành vi là nó có thể hoạt động chống lại các vi-rút tự sửa đổi/đột biến (đa hình) ngẫu nhiên cao mà không thể dễ dàng phát hiện bằng các phương pháp quét chuỗi đơn giản hơn. Quét heuristic có khả năng phát hiện vi-rút trong tương lai mà không cần phải phát hiện vi-rút trước ở nơi khác, gửi cho nhà phát triển trình quét vi-rút, phân tích và cung cấp bản cập nhật phát hiện cho trình quét cho người dùng trình quét.

2.1.1 Pitfalls of heuristic – Những cạm bẫy của phương pháp tìm kiếm

Some heuristics have a strong underlying theory; they are either derived in a top-down manner from the theory or are arrived at based on either experimental or real world data. Others are just rules of thumb based on real-world observation or experience without even a glimpse of theory. The latter are exposed to a larger number of pitfalls.

– Một số phương pháp tìm kiếm có lý thuyết cơ bản mạnh mẽ; chúng được suy ra theo cách từ trên xuống từ lý thuyết hoặc được đưa ra dựa trên dữ liệu thực nghiệm hoặc dữ liệu thực tế. Những phương pháp khác chỉ là các quy tắc kinh nghiệm dựa trên quan sát hoặc kinh nghiệm thực tế mà không hề có 1 chút lý thuyết nào. Những phương pháp sau có nhiều cạm bẫy hơn.

When a heuristic is reused in various contexts because it has been to "work" in 1 context, without having been mathematically proven to meet a given set of requirements, it is possible that the current data set does not necessarily represent future data sets (see overfitting) & that purported "solutions" turn out to be akin to noise.

– Khi 1 phương pháp tìm kiếm được tái sử dụng trong nhiều bối cảnh khác nhau vì nó được coi là "có hiệu quả" trong 1 bối cảnh, nhưng chưa được chứng minh về mặt toán học là đáp ứng được 1 tập hợp các yêu cầu nhất định, thì có khả năng là tập dữ liệu hiện tại không nhất thiết đại diện cho các tập dữ liệu trong tương lai (xem quá khớp) và các "giải pháp" được cho là giống như nhiễu.

Statistical analysis can be conducted when employing heuristics to estimate the probability of incorrect outcomes. To use a heuristic for solving a search problem or a knapsack problem, it is necessary to check that the heuristic is admissible. Given a heuristic function $h(v_i, v_g)$ meant to approximate the true optimal distance $d^\star(v_i, v_g)$ to the goal node v_g in a directed graph G containing n total nodes or vertices labeled $v_0, v_1, \ldots, v_n$, "admissible" means roughly that the heuristic underestimates the cost to the goal or formally that $h(v_i, v_g) \leq d^\star(v_i, v_g)$, $\forall(v_i, v_g)$ where $i, g \in \{0, 1, \ldots, n\}$.

– Phân tích thống kê có thể được tiến hành khi sử dụng phương pháp tìm kiếm để ước tính xác suất của các kết quả không chính xác. Để sử dụng phương pháp tìm kiếm để giải quyết bài toán tìm kiếm hoặc bài toán ba lô, cần phải kiểm tra xem phương pháp tìm kiếm đó có thể chấp nhận được hay không. Với 1 hàm phương pháp tìm kiếm $h(v_i, v_g)$ có nghĩa là xấp xỉ khoảng cách tối ưu thực sự $d^\star(v_i, v_g)$ đến nút đích v_g trong đồ thị có hướng G chứa tổng cộng n nút hoặc đỉnh được gắn nhãn $v_0, v_1, \ldots, v_n$, "có thể chấp nhận được" có nghĩa là phương pháp tìm kiếm đó ước tính thấp chi phí đến đích hoặc chính thức là $h(v_i, v_g) \leq d^\star(v_i, v_g)$, $\forall(v_i, v_g)$ trong đó $i, g \in \{0, 1, \ldots, n\}$.

If a heuristic is not admissible, it may never find the goal, either by ending up in a dead end of graph G or by skipping back & forth between 2 nodes v_i, v_j where $i, j \neq g$.

– Nếu 1 phương pháp tìm kiếm không được chấp nhận, nó có thể không bao giờ tìm thấy mục tiêu, hoặc là kết thúc ở ngõ cụt của đồ thị G hoặc bằng cách bỏ qua & tiến giữa 2 nút v_i, v_j trong đó $i, j \neq g$.

Some types of heuristics:

1. Constructive heuristic.
2. Metaheuristic: Methods for controlling & tuning basic heuristic algorithms, usually with usage of memory & learning.

   – Siêu thuật toán tìm kiếm: Phương pháp kiểm soát & điều chỉnh các thuật toán tìm kiếm cơ bản, thường sử dụng bộ nhớ & học tập.
3. Matheuristics: Optimization algorithms made by the interoperation of metaheuristics & mathematical programming (MP) techniques.

   – Thuật toán tìm kiếm Toán học: Các thuật toán tối ưu hóa được tạo ra bằng cách kết hợp các kỹ thuật siêu thuật toán và kỹ thuật Toán Tối Ưu (MP).
4. Reactive search optimization: Methods using online ML principles for self-tuning of heuristics.

   – Tối ưu hóa tìm kiếm phản ứng: Phương pháp sử dụng nguyên lý học máy trực tuyến để tự điều chỉnh phương pháp tìm kiếm.

2.2 Admissible heuristics – Các phương pháp tìm kiếm có thể chấp nhận được

Resources – Tài nguyên.

1. Wikipedia/admissible heuristic.

In CS, specifically in algorithms related to pathfinding, a heuristic function is said to be *admissible* if it never overestimates the cost of reaching the goal, i.e., the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path. I.e., it should act as a lower bound.

– Trong khoa học máy tính, đặc biệt là trong các thuật toán liên quan đến tìm đường, 1 hàm heuristic được cho là có thể chấp nhận được nếu nó không bao giờ ước tính quá cao chi phí để đạt được mục tiêu, tức là chi phí mà nó ước tính để đạt được mục tiêu không cao hơn chi phí thấp nhất có thể từ điểm hiện tại trên đường đi. Nói cách khác, nó sẽ hoạt động như 1 giới hạn dưới.

It is related to the concept of consistent heuristics. While all consistent heuristics are admissible, not all admissible heuristics are consistent.

– Nó liên quan đến khái niệm về phương pháp tìm kiếm nhất quán. Trong khi tất cả các phương pháp tìm kiếm nhất quán đều có thể chấp nhận được, không phải tất cả các phương pháp tìm kiếm được chấp nhận đều nhất quán.

$$\text{consistent heuristic} \not\Longleftrightarrow \text{admissible heuristic}$$

Search algorithms. An admissible heuristic is used to estimate the cost of reaching the goal state in an informed search algorithm. In order for a heuristic to be admissible to the search problem, the estimated cost must always be $\leq$ the actual cost of reaching the goal state. The search algorithm uses the admissible heuristic to find an estimated optimal path to the goal state from the current node. E.g., in A* search the evaluation function, where n is the current node, is $f(n) = g(n) + h(n)$, where $f(n)$: the evaluation function, $g(n)$: the cost from the start node to the current node, & $h(n)$: estimated cost from current node to goal. $h(n)$ is calculated using the heuristic function. With a non-admissible heuristic, the A* algorithm could overlook the optimal solution to a search problem due to an overestimation in $f(n)$.

– **Thuật toán tìm kiếm.** Một phương pháp tìm kiếm có thể chấp nhận được được sử dụng để ước tính chi phí đạt được trạng thái mục tiêu trong 1 thuật toán tìm kiếm có thông tin. Để 1 phương pháp tìm kiếm có thể chấp nhận được đối với bài

toán tìm kiếm, chi phí ước tính phải luôn bằng $\leq$ chi phí thực tế để đạt được trạng thái mục tiêu. Thuật toán tìm kiếm sử dụng phương pháp tìm kiếm có thể chấp nhận được để tìm đường dẫn tối ưu ước tính đến trạng thái mục tiêu từ nút hiện tại. Ví dụ, trong tìm kiếm A*, hàm đánh giá, trong đó n là nút hiện tại, là $f(n) = g(n) + h(n)$, trong đó $f(n)$: hàm đánh giá, $g(n)$: chi phí từ nút bắt đầu đến nút hiện tại, & $h(n)$: chi phí ước tính từ nút hiện tại đến mục tiêu. $h(n)$ được tính toán bằng cách sử dụng hàm phương pháp tìm kiếm. Với 1 phương pháp tìm kiếm không được chấp nhận, thuật toán A* có thể bỏ qua giải pháp tối ưu cho 1 bài toán tìm kiếm do ước tính quá cao trong $f(n)$.

Formulation. n is a node, h is a heuristic, $h(n)$ is cost indicated by h to reach a goal from n, $h^\star(n)$ is the optimal cost to reach a goal from n. Then $h(n)$ is *admissible* if $h(n) \leq h^\star(n)$, $\forall$ node n.

Construction. An admissible heuristic can be derived from a relaxed version of the problem, or by information from pattern databases that store exact solutions to subproblems of the problem, or by using inductive learning methods.

– 1 phương pháp tìm kiếm có thể chấp nhận được có thể được rút ra từ phiên bản đơn giản của bài toán, hoặc từ thông tin từ cơ sở dữ liệu mẫu lưu trữ các giải pháp chính xác cho các bài toán con của bài toán, hoặc bằng cách sử dụng các phương pháp học quy nạp.

2.2.1 Optimality proof of admissible heuristics

If an admissible heuristic is used in an algorithm that, per iteration, progresses only the path of lowest evaluation (current cost + heuristic) of several candidate paths, terminates the moment its exploration reaches the goal &, crucially, never closes all optimal paths before terminating (something that's possible with A* search algorithm if special case isn't taken), then this algorithm can only terminate on an optimal path. To see why, consider the following proof by contradiction:

– Nếu 1 phương pháp tìm kiếm có thể chấp nhận được được sử dụng trong 1 thuật toán, theo mỗi lần lặp, chỉ tiến triển theo đường đánh giá thấp nhất (phương pháp tìm kiếm chi phí hiện tại +) của 1 số đường ứng viên, kết thúc ngay khi quá trình khám phá của nó đạt đến mục tiêu &, quan trọng là không bao giờ đóng tất cả các đường tối ưu trước khi kết thúc (điều này có thể thực hiện được với thuật toán tìm kiếm A* nếu không áp dụng trường hợp đặc biệt), thì thuật toán này chỉ có thể kết thúc trên 1 đường tối ưu. Để biết lý do, hãy xem xét bằng chứng phản chứng sau:

Assume such an algorithm managed to terminate on a path T with a true cost T_{true} greater than the optimal path S with true cost S_{true}. I.e., before terminating, the evaluated cost of T was $\leq$ the evaluated cost of S (or else S would have been picked). Denote these evaluated costs $T_{\text{eval}}, S_{\text{eval}}$, resp. The above can be summarized as follows,

$$S_{\text{true}} < T_{\text{true}},\ T_{\text{eval}} \leq S_{\text{eval}}.$$

If our heuristic is admissible it follows that at this penultimate step $T_{\text{eval}} = T_{\text{true}}$ because any increase on the true cost by the heuristic on T would be inadmissible & the heuristic cannot be negative. On the other hand, an admissible heuristic would require that $S_{\text{eval}} \leq S_{\text{true}}$ which combined with the above inequalities gives us $T_{\text{eval}} < T_{\text{true}}$ & more specifically $T_{\text{eval}} \neq T_{\text{true}}$. As $T_{\text{eval}}, T_{\text{true}}$ cannot be both equal & unequal our assumption must have been false & so it must be impossible to terminate on a more costly than optimal path.

– Giả sử 1 thuật toán như vậy đã kết thúc trên 1 đường dẫn T với chi phí thực T_{true} lớn hơn đường dẫn tối ưu S với chi phí thực S_{true}. Tức là, trước khi kết thúc, chi phí được đánh giá của T bằng $\leq$ chi phí được đánh giá của S (nếu không thì S đã được chọn). Ký hiệu các chi phí được đánh giá này là $T_{\text{eval}}, S_{\text{eval}}$, tương ứng. Có thể tóm tắt những điều trên như sau,

$$S_{\text{true}} < T_{\text{true}},\ T_{\text{eval}} \leq S_{\text{eval}}.$$

Nếu phương pháp heuristic của chúng ta có thể chấp nhận được thì theo đó tại bước áp chót này $T_{\text{eval}} = T_{\text{true}}$ vì bất kỳ sự gia tăng nào về chi phí thực tế của phương pháp heuristic trên T đều không thể chấp nhận được & phương pháp heuristic không thể là số âm. Mặt khác, 1 phương pháp heuristic có thể chấp nhận được sẽ yêu cầu $S_{\text{eval}} \leq S_{\text{true}}$ kết hợp với các bất đẳng thức trên cho chúng ta $T_{\text{eval}} < T_{\text{true}}$ & cụ thể hơn là $T_{\text{eval}} \neq T_{\text{true}}$. Vì $T_{\text{eval}}, T_{\text{true}}$ không thể vừa bằng nhau & không bằng nhau nên giả định của chúng ta phải sai & do đó không thể kết thúc trên 1 đường dẫn tốn kém hơn đường dẫn tối ưu.

Although an admissible heuristic can guarantee final optimality, it is not necessarily efficient.

– Mặc dù 1 phương pháp tìm kiếm có thể chấp nhận được có thể đảm bảo tính tối ưu cuối cùng, nhưng nó không nhất thiết phải hiệu quả.

2.3 Consistent heuristic

Resources – Tài nguyên.

1. Wikipedia/consistent heuristic.

In the study of path-finding problems in AI, a heuristic function is said to be *consistent*, or *monotone*, if its estimate is always $\leq$ the estimated distance from any neighboring vertex to the goal, plus the cost of reaching that neighbor.

Formally, for every node N & each successor P of N, the estimated cost of reaching the goal from N is $\leq$ the step cost of getting to P plus the estimated cost of reaching the goal from P. I.e.:

$$h(N) \leq c(N, P) + h(P),\ h(G) = 0,$$

where

- h is the consistent heuristic function
- N is any node in the graph
- $P$is any descendant of N

2.4 Roster problem – Bài toán phân công

Dạng toán 1. *Cài đặt & đánh giá thực nghiệm 1 thuật giải heuristic cho bài toán phân công công việc (đơn giản), & thuật giải cải tiến.*

Bài toán 3 (Roster – Bài toán phân công đơn giản)**.** *1 đề án gồm $n \in \mathbb{N}^\star$ công việc & các việc sẽ được thực hiện bởi $m \in \mathbb{N}^\star$ máy như nhau. Giả sử biết thời gian để 1 máy thực hiện việc thứ i là t_i. Yêu cầu: Tìm phương án phân công sao cho thời gian hoàn thành toàn bộ công việc là thấp nhất.*

Input. *m: số máy, n: số việc, dãy $t[0], \ldots, t[n-1]$ với $t[i]$: thời gian để 1 máy thực hiện việc i.*

Output. *Bảng phân công tối ưu.*

Sample.

roster.inp	roster.out
3 10 4 9 5 2 7 6 10 8 7 5	

Thuật giải cho bài toán phân công đơn giản – Pseudocode.

Mathematical analyse – Phân tích Toán học. Gọi n công việc là $w_1, \ldots, w_n$ (w: work), gọi m máy là $M_1, \ldots, M_m$ (các máy này có công suất làm việc như nhau). Yêu cầu của bài toán: Phân hoạch tập $\{t_i\}_{i=1}^n$ thành m tập con $T_1, \ldots, T_m$ lần lượt có số phần tử là $n_1, \ldots, n_m$, i.e., $|T_i| = n_i$, $\forall i = 1, \ldots, m$. □

Computer Science analyse – Phân tích Tin học. □

```
for (i = 0; i < n; i++) {
    chọn việc i chưa phân công có thời gian thực hiện cao nhất;
    chọn máy m có thời gian làm việc thấp nhất;
    bố trí việc i cho máy m;
}
```

- Input: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/Python/roster.inp.
- Output: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/Python/roster.out.
- Python: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/Python/roster.py.

```
m, n = map(int, input().split())
t = [int(x) for x in input().split()]
d = [0] * m # devices/machines's current accomplished time
t.sort(reverse = True) # descending order
for i in range(n):
    current_max_work = t[0] # current longest work
    t.pop(0) # remove current longest work
    # print(t)
    d.sort() # ascending order
    # print(d)
    d[0] += int(current_max_work) # laziest device takes longest work
    # print(d)
print(max(d))
```

Các bước `print` để mô phỏng quá trình giao công việc cho các máy để tiện hình dung, không bắt buộc.

Bài toán 4 (Extended roster – Bài toán phân công mở rộng)**.** *Có $n \in \mathbb{N}^\star$ công việc & $m \in \mathbb{N}^\star$ máy không đồng nhất. Biết thời gian máy i làm việc j là $t_{ij} = t[i][j]$. Yêu cầu: Lập bảng phân công tối ưu.*

Input. *m: số máy, n: số việc, array 2 chiều $t[i][j]$: thời gian để máy i thực hiện việc j.*

Output. *Bảng phân công tối ưu.*

Sample.

extended_roster.inp	extended_roster.out
3 8 4 5 4 10 8 6 12 8 7 5 7 3 9 7 9 5 10 6 7 12 10 6 5 7	

Cách phát biểu khác của bài toán phân công mở rộng. Có $n \in \mathbb{N}^\star$ công việc sẽ được phân công cho $m \in \mathbb{N}^\star$ người thực hiện, mỗi việc được phân công cho 1 người. Giả sử ta biết thời gian $t_{ij} = t[i][j]$ cần để người thứ i thực hiện công việc thứ j, $\forall i = 1, \ldots, m$, $\forall j = 1, \ldots, n$. Tìm 1 phương pháp phân công sao cho thời gian hoàn thành tất cả các công việc là thấp nhất.

2.5 Bài toán tô màu đồ thị – Graph coloring problem

Bài toán 5 (Bài toán tô màu các đỉnh đồ thị – Graph coloring problem)**.** *Có 1 đồ thị vô hướng đơn giản. Ta muốn tìm cách tô màu cho các đỉnh của đồ thị sao cho 2 đỉnh cạnh nhau phải có màu khác nhau. Yêu cầu: Tìm phương án tô sao cho số màu sử dụng là ít nhất.*

Input. *Đồ thị vô hướng đơn giản.*

Output. *Mỗi đỉnh tô màu gì.*

1 thuật giải heuristic. Sử dụng nguyên lý thứ tự:

```
for (i = 0; i < n; i++) {
    chọn đỉnh s chưa tô có d[s] lớn nhất;
    chọn màu: ưu tiên tô đỉnh s bằng 1 trong các màu đã sử dụng, nếu không được thì sử dụng màu mới;
    sau khi tô màu cho đỉnh s: với mỗi đỉnh x cạnh, giảm d[x]; ???
}
```

$d[x]$: số đỉnh cạnh x mà chưa tô màu. ???

2.6 Shortest path problem – Bài toán đường đi ngắn nhất

Bài toán 6. *Cài đặt & thử nghiệm* A**. So sánh với Dijkstra nếu được.*

Input. $G = (V, E)$ *có trọng số dương, đỉnh xuất phát* a*, đỉnh mục tiêu* z*. Thông tin bổ sung:* $h(x)$*: ước lượng khoảng cách từ* a *đến mục tiêu* z*.*

Output. *Đường đi ngắn nhất shortest path SP từ* a *đến* z*.*

2.7 Traveling salesman problem (TSP) – Bài toán người bán hàng du lịch

Resources – Tài nguyên.

1. Wikipedia/traveling salesman problem.

An example of approximation is described by Jon Bentley for solving the traveling salesman problem (TSP):

Problem 1 (Original Traveling Salesman Problem (TSP))**.** *Given a list of cities & the distances between each pair of cities, what is the shortest possible route that visits each city exactly once & returns to the origin city?*

– Cho 1 danh sách các thành phố & khoảng cách giữa mỗi cặp thành phố, đâu là tuyến đường ngắn nhất có thể đi qua mỗi thành phố đúng 1 lần & quay trở lại thành phố ban đầu?

so as to select the order to draw using a pen plotter. TSP is known to be NP-hard so an optimal solution for even a moderate size problem is difficult to solve. Instead, the greedy algorithm can be used to give a good but not optimal solution (it is an approximation to the optimal answer) in a reasonably short amount of time. The greedy algorithm heuristic says to pick whatever is currently the best next step regardless of whether that prevents (or even makes impossible) good steps later. It is a heuristic in the sense that practice indicates it is a good enough solution, while theory indicates that there are better solutions (& even indicates how much better, in some cases).

– Jon Bentley đã đưa ra 1 ví dụ về phép xấp xỉ để giải bài toán người bán hàng du lịch (TSP): để chọn thứ tự vẽ bằng bút vẽ. TSP được biết là NP-khó nên giải pháp tối ưu cho ngay cả bài toán có kích thước vừa phải cũng khó giải. Thay vào đó, thuật toán tham lam có thể được sử dụng để đưa ra giải pháp tốt nhưng không tối ưu (là phép xấp xỉ với câu trả lời tối ưu) trong 1 khoảng thời gian khá ngắn. Thuật toán tham lam nói rằng hãy chọn bất kỳ bước tiếp theo nào hiện là tốt nhất bất kể điều đó có ngăn cản (hoặc thậm chí khiến không thể) thực hiện các bước tốt sau này hay không. Đây là thuật toán theo nghĩa là thực hành chỉ ra rằng đó là giải pháp đủ tốt, trong khi lý thuyết chỉ ra rằng có những giải pháp tốt hơn (& thậm chí chỉ ra tốt hơn bao nhiêu, trong 1 số trường hợp).

Problem 2 (Traveling Salesman Problem (TSP))**.** *The traveling salesman must visit every city in this territory exactly once & then return to the starting point; given the cost of travel between all cities, how should he plan his itinerary for minimum total cost of the entire tour?*

TSP $\in$ NP-Complete.

Remark 2 (Approximate TSP by GAs)**.** *We shall discuss a single possible approach to approximate the TSP by genetic algorithms (GAs).*

2.8 15 Puzzle Problem

Resources – Tài nguyên.

1. Wikipedia/admissible heuristic.

2. Wikipedia/15 puzzle.

2 different examples of admissible heuristics apply to the Wikipedia/15 puzzle problem:

- Hamming distance
- Manhattan distance

The Hamming distance is the total number of misplaced tiles. It is clear that this heuristic is admissible since the total number of moves to order the tiles correctly is at least the number of misplaced tiles (each tile not in place must be moved at least once). The cost (number of moves) to the goal (an ordered puzzle) is at least the Hamming distance of the puzzle.

– Khoảng cách Hamming là tổng số ô bị đặt sai vị trí. Rõ ràng là phương pháp tìm kiếm này có thể chấp nhận được vì tổng số lần di chuyển để sắp xếp các ô đúng ít nhất bằng số ô bị đặt sai vị trí (mỗi ô không đúng vị trí phải được di chuyển ít nhất 1 lần). Chi phí (số lần di chuyển) đến đích (một câu đố có thứ tự) ít nhất bằng khoảng cách Hamming của câu đố.

The Manhattan distance of a puzzle is defined as:

$$h(n) = \sum_{\text{all tiles}} \text{distance}(\text{tile}, \text{correct position}).$$

Consider the puzzle below in which the player wishes to move each tile s.t. the numbers are ordered. The Manhattan distance is an admissible heuristic in this case because every tile will have to be moved at least the number of spots in between itself & its correct position.

– Xem xét câu đố bên dưới trong đó người chơi muốn di chuyển từng ô theo thứ tự các số. Khoảng cách Manhattan là 1 phép thử có thể chấp nhận được trong trường hợp này vì mỗi ô sẽ phải được di chuyển ít nhất số điểm giữa nó & vị trí chính xác của nó.

Bài toán 7 (Tính khoảng cách Hamming & khoảng cách Manhattan)**.** *(a) Cho 1 dãy hoán vị của* [15] *theo thứ tự sắp xếp trái sang phải, trên xuống dưới. Tính khoảng cách Hamming & khoảng cách Manhattan của hoán vị này.*

Input. *Dòng 1 chứa số bộ test* $t \in \mathbb{N}^\star$*. Với* t *dòng tiếp theo, mỗi dòng chứa đúng 1 hoán vị* $\{a_n\}_{n=1}^{15} = a_1, a_2, \ldots, a_{15}$ *của* [15].

Output. *Khoảng cách Hamming & khoảng cách Manhattan của hoán vị* $\{a_n\}_{n=1}^{15}$.

Sample.

15_puzzle.inp	15_puzzle.out
1 4 6 3 8 7 12 9 14 15 13 1 5 2 10 11	$h(n) = 36$

(b) Mở rộng bài toán từ 15 *thành* $n \in \mathbb{N}^\star$.

3 Miscellaneous

Tài liệu

[Tiệ25] Vũ Khắc Tiệp. *Machine Learning Cơ Bản*. 2025, p. 422.