ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH


## TRÍ TUỆ NHÂN TẠO (CS106.P21)

# BÁO CÁO
# CÀI ĐẶT THUẬT TOÁN A* SEARCH CHO TRÒ CHƠI SOKOBAN

GIẢNG VIÊN HƯỚNG DẪN: TS. LƯƠNG NGỌC HOÀNG



| STT | Họ tên SV | MSSV |
|---|---|---|
| 1 | Trần Minh Tiến | 23521587 |

# Nội dung



# 1 So sánh hiệu suất giữa A* Search (Heuristic mẫu) và UCS

## 1.1 Bảng thời gian chạy và số nút đã được mở ra của hai thuật toán

| Level | A* Search (Heuristic mẫu) | | | UCS | | |
|---|---|---|---|---|---|---|
| | **Length** | **Time (sec)** | **Node** | **Length** | **Time (sec)** | **Node** |
| 1 | 13 | 0.022358 | 195 | 12 | 0.064532 | 2115 |
| 2 | 9 | 0.002006 | 54 | 9 | 0.011800 | 183 |
| 3 | 15 | 0.013104 | 78 | 15 | 0.099475 | 1465 |
| 4 | 7 | 0.004650 | 55 | 7 | 0.004975 | 174 |
| 5 | 22 | 0.118109 | 1013 | 20 | 87.930650 | 1355329 |
| 6 | 19 | 0.007519 | 222 | 19 | 0.016315 | 612 |
| 7 | 21 | 0.100055 | 1102 | 21 | 0.660047 | 17283 |
| 8 | 97 | 0.177774 | 2433 | 97 | 0.293787 | 5357 |
| 9 | 8 | 0.004554 | 69 | 8 | 0.015717 | 181 |
| 10 | 33 | 0.031619 | 208 | 33 | 0.031142 | 526 |
| 11 | 34 | 0.014197 | 290 | 34 | 0.019082 | 668 |
| 12 | 23 | 0.040144 | 687 | 23 | 0.115662 | 3220 |
| 13 | 31 | 0.171617 | 1935 | 31 | 0.220933 | 6202 |
| 14 | 23 | 3.958724 | 10783 | 23 | 3.752331 | 72422 |
| 15 | 105 | 0.212972 | 2243 | 105 | 0.378022 | 6328 |
| 16 | 42 | 0.282910 | 1604 | 34 | 21.561160 | 150971 |
| 17 | **x** | **x** | **x** | **x** | **x** | **x** |
| 18 | **x** | **x** | **x** | **x** | **x** | **x** |

Bảng 1: So sánh thuật toán A* Search (Heuristic mẫu) và UCS

## 1.2 Nhận xét

### 1.2.1 Hiệu suất về số lượng node mở rộng

- UCS mở rộng tất cả các node có chi phí nhỏ nhất trước. Điều này giúp đảm bảo tìm được lời giải tối ưu nhưng lại mở rộng nhiều trạng thái không cần thiết, đặc biệt trong môi trường có nhiều ngõ cụt hoặc cấu trúc phức tạp.
- A* sử dụng heuristic để ưu tiên các node có tiềm năng tốt hơn. Nếu heuristic đủ chính xác, A* sẽ mở rộng ít node hơn UCS vì nó tập trung vào những hướng có khả năng dẫn đến mục tiêu nhanh nhất.
- Trong bài toán Sokoban, A* có xu hướng mở rộng ít node hơn UCS nhờ heuristic Manhattan dẫn hướng hợp lý. Tuy nhiên, nếu có vật cản khiến heuristic đánh giá sai, A* có thể mở rộng một số node không cần thiết, đôi khi nhiều hơn UCS.

### 1.2.2 Hiệu suất về thời gian chạy

- UCS có thể rất chậm nếu số lượng trạng thái lớn, do nó phải mở rộng tất cả các trạng thái có cùng chi phí trước khi tiến xa hơn.
- A* thường nhanh hơn nếu heuristic đủ chính xác, vì nó ưu tiên đi theo những đường tốt hơn thay vì mở rộng toàn bộ trạng thái có cùng chi phí như UCS.
- A* thường nhanh hơn UCS trong các bài toán có đường đi rõ ràng nhờ heuristic giúp định hướng. Nhưng nếu heuristic không chính xác, A* có thể mất thời gian đánh giá các hướng đi không thực sự tối ưu, làm giảm lợi thế về thời gian.

### 1.2.3 Hiệu suất về bộ nhớ sử dụng

- Cả UCS và A* đều lưu trữ tất cả các trạng thái đã mở rộng và các trạng thái trong hàng đợi ưu tiên, nên đều có yêu cầu bộ nhớ lớn.
- A* có thể yêu cầu bộ nhớ ít hơn UCS nếu heuristic giúp nó mở rộng ít node hơn.
- Trong những bài toán lớn (bản đồ rộng, nhiều hộp), cả hai thuật toán đều có thể tiêu tốn lượng lớn bộ nhớ. Khi heuristic tốt, A* giảm số trạng thái mở rộng, giúp tiết kiệm bộ nhớ hơn UCS. Nếu heuristic kém, A* có thể tiêu tốn nhiều bộ nhớ hơn do lưu trữ nhiều trạng thái chưa cần thiết.

# 2 So sánh hiệu suất giữa A* Search (Heuristic cũ) và A* Search (Heuristic mới)

## 2.1 Heuristic mới

Heuristic cải tiến được thiết kế để nâng cao độ chính xác trong việc ước lượng khoảng cách từ trạng thái hiện tại đến đích, giúp thuật toán A* mở rộng ít trạng thái hơn và tìm đường tối ưu hơn.
Thay vì chỉ chọn cặp hộp - mục tiêu gần nhất tại từng bước như heuristic cũ, thuật toán mới sử dụng **Bài toán ghép cặp tối ưu (Hungarian Algorithm)**. Cách tiếp cận này đảm bảo tổng khoảng cách giữa hộp và mục tiêu là nhỏ nhất có thể trên toàn bộ không gian trạng thái.

### 2.1.1 Cách hoạt động

- Loại bỏ các hộp đã đặt đúng vị trí trên mục tiêu.
- Xây dựng **ma trận khoảng cách Manhattan** giữa các hộp còn lại và các mục tiêu chưa có hộp.
- Giải **bài toán ghép cặp tối ưu** bằng Thuật toán Hungarian để tìm cách ghép hộp với mục tiêu sao cho tổng khoảng cách là nhỏ nhất.
- Tính tổng khoảng cách từ kết quả ghép cặp làm giá trị heuristic.
- Cộng thêm khoảng cách từ người chơi đến hộp gần nhất để phản ánh mức độ dễ tiếp cận của trạng thái hiện tại.

### 2.1.2 Công thức toán học

$$h(n) = \min_{\pi \in S_k} \sum_{i=1}^{k} \text{Manhattan}(box_i, goal_{\pi(i)}) + \min_j \text{Manhattan}(player, box_j) \tag{1}$$

Trong đó:

- $k$ là số hộp chưa đặt đúng vị trí.
- $S_k$ là tập hợp tất cả các hoán vị có thể của $k$ mục tiêu.
- $\pi(i)$ là ánh xạ từ hộp $i$ đến mục tiêu được gán theo thuật toán Hungarian.
- $\text{Manhattan}(box_i, goal_{\pi(i)})$ là khoảng cách Manhattan giữa hộp $i$ và mục tiêu được gán tương ứng.
- $\min_j \text{Manhattan}(player, box_j)$ là khoảng cách Manhattan từ người chơi đến hộp gần nhất.

### 2.1.3 Ưu điểm so với heuristic trước đó

- Thuật toán Hungarian đảm bảo tổng khoảng cách nhỏ nhất, thay vì chỉ chọn cặp gần nhất theo từng bước như cách ban đầu.
- Giá trị heuristic chính xác hơn giúp thuật toán A* ít bị đánh lừa bởi các ước lượng kém, dẫn đến tìm đường nhanh hơn.
- Khi có nhiều vật cản hoặc mục tiêu nằm xa, heuristic này vẫn đảm bảo một chiến lược ghép cặp tối ưu, thay vì bị ảnh hưởng bởi thứ tự xuất hiện như heuristic cũ.
- Việc bổ sung khoảng cách từ người chơi đến hộp gần nhất giúp thuật toán đánh giá chính xác hơn trạng thái hiện tại.

## 2.2 Bảng thời gian chạy và số nút đã được mở ra của hai thuật toán

| Level | A* Search (Heuristic cũ) | | | A* Search (Heuristic mới) | | |
|---|---|---|---|---|---|---|
| | Length | Time (sec) | Node | Length | Time (sec) | Node |
| 1 | 13 | 0.022358 | 195 | 12 | 0.007143 | 102 |
| 2 | 9 | 0.002006 | 54 | 9 | 0.002691 | 45 |
| 3 | 15 | 0.013104 | 78 | 15 | 0.020619 | 109 |
| 4 | 7 | 0.004650 | 55 | 7 | 0.002515 | 40 |
| 5 | 22 | 0.118109 | 1013 | 20 | 0.114834 | 999 |
| 6 | 19 | 0.007519 | 222 | 19 | 0.012365 | 192 |
| 7 | 21 | 0.100055 | 1102 | 21 | 0.189106 | 1629 |
| 8 | 97 | 0.177774 | 2433 | 97 | 0.230911 | 2261 |
| 9 | 8 | 0.004554 | 69 | 8 | 0.007028 | 42 |
| 10 | 33 | 0.031619 | 208 | 33 | 0.023085 | 217 |
| 11 | 34 | 0.014197 | 290 | 34 | 0.019727 | 291 |
| 12 | 23 | 0.040144 | 687 | 23 | 0.045335 | 622 |
| 13 | 31 | 0.171617 | 1935 | 31 | 0.184624 | 1785 |
| 14 | 23 | 3.958724 | 10783 | 23 | 7.643445 | 13905 |
| 15 | 105 | 0.212972 | 2243 | 105 | 0.242587 | 2240 |
| 16 | 42 | 0.282910 | 1604 | 34 | 3.831626 | 9276 |
| 17 | **x** | **x** | **x** | **x** | **x** | **x** |
| 18 | **x** | **x** | **x** | **x** | **x** | **x** |

Bảng 2: So sánh thuật toán A* Search (Heuristic cũ) và A* Search (Heuristic mới)

## 2.3 Nhận xét

### 2.3.1 Hiệu suất về số lượng node mở rộng

- **Heuristic cũ:** Sử dụng cách ghép hộp với mục tiêu theo thứ tự xuất hiện, không đảm bảo tối ưu. Điều này có thể dẫn đến việc chọn những cặp hộp - mục tiêu không hợp lý, làm tăng số node mở rộng trong quá trình tìm kiếm.

- **Heuristic mới:** Áp dụng thuật toán Hungarian để tìm cách ghép hộp với mục tiêu sao cho tổng khoảng cách là nhỏ nhất. Nhờ đó, số lượng node mở rộng giảm đáng kể, đặc biệt trong những bài toán có nhiều vật cản hoặc mục tiêu nằm xa vị trí ban đầu của hộp.

### 2.3.2 Hiệu suất về thời gian chạy

- **Heuristic cũ:** Chỉ sử dụng khoảng cách Manhattan đơn giản để ghép cặp hộp - mục tiêu, giúp thời gian tính toán heuristic nhanh hơn. Tuy nhiên, do cách ghép không tối ưu, thuật toán A* phải mở rộng nhiều trạng thái hơn, dẫn đến tổng thời gian chạy lâu hơn.

- **Heuristic mới:** Việc giải bài toán tối ưu hóa ghép cặp bằng thuật toán Hungarian làm tăng thời gian tính toán heuristic. Tuy nhiên, do heuristic này giúp A* mở rộng ít trạng thái hơn, tổng thời gian chạy có thể giảm đáng kể. Đặc biệt trong các bài toán phức tạp với nhiều vật cản hoặc hộp nằm xa mục tiêu, lợi ích từ việc giảm số node mở rộng lớn hơn chi phí tính toán heuristic, giúp tìm lời giải nhanh hơn so với heuristic cũ.

### 2.3.3 Hiệu suất về bộ nhớ sử dụng

- **Heuristic cũ:** Chỉ tính toán khoảng cách Manhattan giữa từng hộp và mục tiêu theo thứ tự có sẵn, không yêu cầu thêm bộ nhớ để lưu trữ ma trận chi phí hay giải bài toán tối ưu. Tuy nhiên, vì số lượng trạng thái mở rộng nhiều hơn, thuật toán tìm kiếm có thể tiêu tốn nhiều bộ nhớ hơn để lưu trữ các trạng thái đã mở rộng.

- **Heuristic mới:** Cần thêm bộ nhớ để xây dựng ma trận chi phí và chạy thuật toán Hungarian. Tuy nhiên, do giúp thuật toán tìm kiếm mở rộng ít trạng thái hơn, tổng bộ nhớ sử dụng có thể ít hơn heuristic cũ, đặc biệt trong các bài toán lớn với nhiều hộp và mục tiêu.