Đại Học Khoa Học Tự Nhiên

KHOA Toán-Cơ-Tin học

BÁO CÁO GIỮA KỲ

Algorithm Mechanism Report

PARIS: Planning Algorithms for Reconfiguring Independent Sets

Môn học: Xây dựng phần mềm

Giảng viên hướng dẫn: Hoàng Anh Đức

Thành viên:

- Tạ Văn Sơn – MSSV: 22000124
- Nguyễn Công An – MSSV: 22000068
- Mai Văn Phương – MSSV: 22000115
- Đào Thanh Quang – MSSV: 22000118
- Trần Hoàng Đức – MSSV: 22000089
- Phùng Hữu Uy– MSSV: 22001654

Hà Nội, 2026

Mục lục

Ký hiệu và thuật ngữ

Ký hiệu / Thuật ngữ	**Ý nghĩa**
$G = (V, E)$	Đồ thị vô hướng với tập đỉnh V và tập cạnh E
$I \subseteq V$	Tập độc lập (independent set)
I_s, I_t	Tập độc lập ban đầu (start) và mục tiêu (target)
Token	Một "quân cờ" đặt trên đỉnh, đại diện cho một phần tử của tập độc lập
Token Jump	Phép di chuyển một token từ đỉnh này sang đỉnh khác sao cho cấu hình mới vẫn là independent set
$\rho = \langle I_0, \ldots, I_n \rangle$	Chuỗi reconfiguration
$\|\rho\|$	Độ dài chuỗi reconfiguration (số lần token jump)
ISR	Independent Set Reconfiguration
PDDL	Planning Domain Definition Language
SAS+	Biểu diễn trung gian của bài toán planning: $\langle \mathcal{V}, \mathcal{A}, I, G \rangle$
$\mathcal{V}$, $\mathcal{A}$	Tập biến trạng thái và tập hành động trong SAS+
pre(a), eff(a)	Tiền điều kiện và tác động của hành động a
Heuristic	Hàm ước lượng chi phí từ trạng thái hiện tại đến goal
Landmark	Fact hoặc hành động bắt buộc phải xảy ra trong mọi lời giải
BDD	Binary Decision Diagram
GBFS	Greedy Best-First Search
A*	Thuật toán tìm đường tối ưu kết hợp chi phí thực và heuristic
Portfolio	Chiến lược chạy tuần tự nhiều thành phần với time limit khác nhau
Counter Abstraction	Kỹ thuật trừu tượng hóa bằng cách đếm số token theo màu sắc
PSPACE-complete	Lớp độ phức tạp của bài toán ISR
CoRe 2022	1st Combinatorial Reconfiguration Challenge 2022

Bảng 1: Bảng ký hiệu và thuật ngữ

1 Giới thiệu tổng quan về PARIS

PARIS (*Planning Algorithms for Reconfiguring Independent Sets*) là một hệ thống solver được thiết kế để giải bài toán Independent Set Reconfiguration (ISR) bằng cách tiếp cận novel: **biến đổi bài toán ISR thành bài toán lập kế hoạch cổ điển** (classical planning), sau đó áp dụng các kỹ thuật tìm kiếm tiên tiến đã được phát triển bởi cộng đồng AI Planning.

PARIS được phát triển bởi Remo Christen, Salomé Eriksson, Michael Katz, Christian Muise, Alice Petrov, Florian Pommerening, Jendrik Seipp, Silvan Sievers và David Speck, công bố tại ECAI 2023 [**?**]. Tại CoRe Challenge 2022, PARIS đạt 4 giải nhất, 3 giải nhì, và 3 giải ba trên tổng số các track.

Pipeline gồm 5 bước chính:

2 Định nghĩa bài toán ISR

2.1 Đồ thị và tập độc lập

Cho đồ thị vô hướng $G = (V, E)$ với $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$.

Định nghĩa 1 (Independent Set): Một tập $I \subseteq V$ là *tập độc lập* nếu:

$$\forall u, v \in I : \{u, v\} \notin E$$

2.2 Mô hình Token Jump

Định nghĩa 2 (Token Jump): Một *token jump* là phép di chuyển một token từ đỉnh $v \in I$ sang đỉnh $u \notin I$ sao cho $I' = (I \setminus \{v\}) \cup \{u\}$ thỏa mãn:

$$\forall w \in I' : \{u, w\} \notin E$$

Token không cần di chuyển dọc theo cạnh – nó có thể "nhảy" đến bất kỳ đỉnh nào.

2.3 Bài toán ISR

Định nghĩa 3 (ISR): Cho $G = (V, E)$, I_s và I_t với $|I_s| = |I_t|$. Bài toán ISR yêu cầu xác định có tồn tại chuỗi reconfiguration:

$$\rho = \langle I_0 = I_s, I_1, \ldots, I_n = I_t \rangle$$

trong đó mỗi I_i là independent set và I_{i+1} thu được từ I_i bằng đúng một token jump.

Ví dụ: Xét đồ thị có cạnh $(1,2), (2,3), (3,4), (4,5), (1,5), (2,4)$. Với $I_s = \{1,3\}$ và $I_t = \{2,4\}$, chuỗi ngắn nhất là:

$$\{1,3\} \xrightarrow{1\to5} \{3,5\} \xrightarrow{3\to2} \{2,5\} \xrightarrow{5\to4} \{2,4\}$$

2.4 Độ phức tạp

Bài toán ISR (dưới token jump) là **PSPACE-complete** đối với đồ thị tổng quát [**?**]. Không gian trạng thái có thể đạt $O(2^{|V|})$ trong trường hợp xấu nhất, đòi hỏi các kỹ thuật tìm kiếm thông minh.

3 Mô hình trạng thái (State Model)

Từ góc độ planning:

- **Trạng thái:** Một token configuration – independent set $I \subseteq V$ hợp lệ với đúng $k = |I_s|$ token.
- **Trạng thái ban đầu:** I_s.
- **Trạng thái đích:** I_t.
- **Chuyển trạng thái:** Mỗi token jump hợp lệ là một cạnh trong reconfiguration graph.

4 Mã hóa sang PDDL và SAS+

4.1 Ngôn ngữ PDDL và biểu diễn SAS+

PDDL là ngôn ngữ chuẩn để mô tả bài toán lập kế hoạch. Biểu diễn SAS+ là dạng trung gian mà Fast Downward sử dụng: $\langle \mathcal{V}, \mathcal{A}, I, G \rangle$ với $\mathcal{V}$ là tập biến trạng thái, $\mathcal{A}$ là tập hành động, I là trạng thái ban đầu, và G là điều kiện đích.

4.2 Encoding Single (1 hành động)

Listing 1: PDDL Single Encoding – 1 action move

```
(:action move
 :parameters (?l1 ?l2 - loc)
 :precondition (and
   (tokened ?l1)        ; Diem nguon co token
   (free ?l2)           ; Diem dich khong co token
   (forall (?l3 - loc) (imply
     (and (not (= ?l1 ?l3)) (edge ?l2 ?l3))
     (free ?l3))))      ; Hang xom ?l2 phai trong
 :effect (and
   (not (tokened ?l1)) (free ?l1)
   (tokened ?l2) (not (free ?l2))))
```

Cách này sinh $O(n^2)$ ground actions và chậm khi grounding.

4.3 Encoding Split (2 hành động – cách PARIS thực sự dùng)

Listing 2: PDDL Split Encoding – pick + place

```
(:action pick
 :parameters (?l1 - loc)
 :precondition (and (handfree) (tokened ?l1))
 :effect (and (not (handfree)) (holding)
              (free ?l1) (not (tokened ?l1))))

(:action place
 :parameters (?l1 - loc)
 :precondition (and (holding) (free ?l1)
   (forall (?l2 - loc) (imply (edge ?l1 ?l2) (free ?l2))))
 :effect (and (not (holding)) (handfree)
              (not (free ?l1)) (tokened ?l1)))
```

Chỉ cần $2n$ ground actions. Mỗi token jump cần 2 bước (1 pick + 1 place), kết quả cần post-processing để ghép lại.

4.4 Encoding SAS+ trực tiếp

PARIS encode trực tiếp sang SAS+. Với instance ISR $\langle G = (V, E), I_s, I_t \rangle$:

Biến trạng thái:

$$\mathcal{V} = \{v_i \mid i \in V\} \cup \{\text{hand}\}$$

Mỗi v_i có miền $\{\texttt{free}, \texttt{occupied}\}$; biến hand cũng có miền $\{\texttt{free}, \texttt{occupied}\}$.

Trạng thái ban đầu và đích:

$$I = \{v_i \mapsto \texttt{occupied} \mid i \in I_s\} \cup \{v_i \mapsto \texttt{free} \mid i \notin I_s\} \cup \{\text{hand} \mapsto \texttt{free}\}$$

$$G = \{v_i \mapsto \texttt{occupied} \mid i \in I_t\} \cup \{v_i \mapsto \texttt{free} \mid i \notin I_t\} \cup \{\text{hand} \mapsto \texttt{free}\}$$

Hành động: Với mỗi đỉnh $v \in V$:
Hành động pick(v):

$$\text{pre}(\text{pick}(v)) = \{v \mapsto \texttt{occupied},\ \text{hand} \mapsto \texttt{free}\}$$

$$\text{eff}(\text{pick}(v)) = \{v \mapsto \texttt{free},\ \text{hand} \mapsto \texttt{occupied}\}$$

Hành động place(v):

$$\text{pre}(\text{place}(v)) = \{v \mapsto \texttt{free},\ \text{hand} \mapsto \texttt{occupied}\} \cup \{v' \mapsto \texttt{free} \mid \{v, v'\} \in E\}$$

$$\text{eff}(\text{place}(v)) = \{v \mapsto \texttt{occupied},\ \text{hand} \mapsto \texttt{free}\}$$

Ví dụ minh họa encoding: Với đồ thị 4 đỉnh, cạnh $(1,2), (2,3), (3,4)$, $I_s = \{1,3\}$, $I_t = \{2,4\}$:

Biến	**Trạng thái ban đầu**	**Trạng thái đích**	**Ý nghĩa**
v_1	`occupied`	`free`	Có token → trống
v_2	`free`	`occupied`	Trống → có token
v_3	`occupied`	`free`	Có token → trống
v_4	`free`	`occupied`	Trống → có token
hand	`free`	`free`	Bắt đầu và kết thúc không cầm token

Bảng 2: Encoding SAS+ cho ví dụ $I_s = \{1,3\}$, $I_t = \{2,4\}$

5 Các thuật toán tìm kiếm trong PARIS

PARIS sử dụng **sequential portfolio**: chạy tuần tự các thành phần với time limit riêng; thành phần nào tìm được kết quả thì dừng.

5.1 A* + Landmarks (tìm kiếm tối ưu)

A* là thuật toán tìm kiếm tối ưu theo hàm đánh giá:

$$f(n) = g(n) + h(n)$$

$g(n)$ là chi phí thực từ trạng thái ban đầu đến n; $h(n)$ là heuristic ước lượng chi phí từ n đến goal. A* đảm bảo tìm lời giải ngắn nhất nếu h là *admissible*.

> **Landmark:** Một landmark là một *fact* hoặc *action* bắt buộc phải xảy ra trong mọi lời giải hợp lệ. PARIS dùng h^1-landmarks và RHW-landmarks kết hợp uniform cost partitioning để đảm bảo admissibility.

Kết quả: **A* + Landmarks** tìm được lời giải tối ưu, sound và complete.

5.2 GBFS + Landmarks (tìm kiếm satisficing)

Greedy Best-First Search (GBFS) dùng:

$$f(n) = h(n)$$

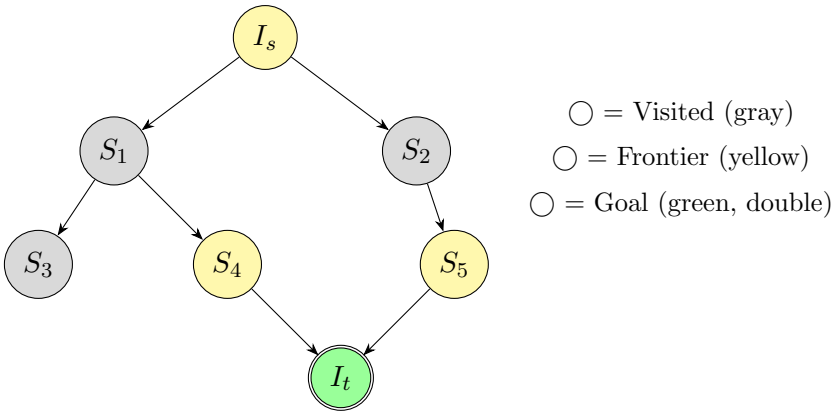
GBFS không xét $g(n)$, ưu tiên node có h nhỏ nhất. Nhanh hơn A* nhưng không đảm bảo tối ưu. Trong thực nghiệm, GBFS + Landmarks có **coverage cao nhất** trong tất cả các thành phần của PARIS.

5.3 Symbolic Search với BDD (tìm kiếm tối ưu)

Binary Decision Diagram (BDD) biểu diễn compact một tập trạng thái. Thay vì duyệt từng trạng thái riêng lẻ, symbolic search thao tác trên *tập trạng thái* bằng BDD:

- **Frontier** = tập trạng thái ở biên (BDD).
- Mỗi bước: áp dụng *image computation* – tính tập trạng thái kế tiếp của toàn bộ frontier cùng lúc.
- Tránh lặp: kiểm tra giao với tập đã thăm (phép $\cap$ trên BDD).

PARIS dùng **SymK** với thư viện BDD CUDD để chạy forward symbolic blind search. Ưu điểm: BDD biểu diễn hàng triệu trạng thái với bộ nhớ nhỏ nếu các trạng thái có cấu trúc.



Hình 1: Minh họa không gian tìm kiếm với BDD symbolic search

5.4 Counter Abstraction (phát hiện vô nghiệm)

Đây là kỹ thuật **mới** do PARIS đề xuất, mục tiêu phát hiện nhanh instance vô nghiệm mà không cần duyệt không gian trạng thái đầy đủ.

Bước 1 – Tô màu đỉnh:

Màu	Điều kiện	Ý nghĩa
Xanh lam (blue)	$v \in I_s$, $v \notin I_t$	Token ở start nhưng không ở goal
Đỏ (red)	$v \notin I_s$, $v \in I_t$	Token ở goal nhưng không ở start
Tím (purple)	$v \in I_s$, $v \in I_t$	Token ở cả hai
Trắng (white)	$v \notin I_s$, $v \notin I_t$	Không có token ở cả hai

Bảng 3: Quy tắc tô màu trong Counter Abstraction

Bước 2 – Trừu tượng hóa: Mỗi cấu hình token được ánh xạ sang abstract state $\langle c_1, c_2, \ldots, c_k \rangle$ trong đó c_i là **số lượng** token đang đứng trên các đỉnh màu i.

Bước 3 – Xây dựng abstract state space: Với mỗi abstract state $s = \langle c_1, \ldots, c_n \rangle$, di chuyển 1 token từ màu i sang màu j tạo ra $s' = \langle \ldots, c_i - 1, \ldots, c_j + 1, \ldots \rangle$.

Bước 4 – Phát hiện vô nghiệm: Nếu abstract goal **không reachable** từ abstract initial state, thì bài toán gốc **chắc chắn vô nghiệm**.

Tính đúng đắn: Đây là *sound abstraction* – nếu bài toán gốc có lời giải thì abstract version cũng có lời giải. Do đó: abstract unsolvable $\Rightarrow$ original unsolvable.

Độ phức tạp: Với 4 màu và k token, số abstract state tối đa là $O(k^4)$ – đa thức trong k, không phụ thuộc $|V|$. Counter abstraction chạy rất nhanh.

Ví dụ: Với $I_s = \{1, 2, 3\}$ (màu blue) và $I_t = \{4, 5, 6\}$ (màu red):

$$\text{Abstract initial} = \langle 3, 0 \rangle \quad \text{(3 blue, 0 red)}$$
$$\text{Abstract goal} = \langle 0, 3 \rangle \quad \text{(0 blue, 3 red)}$$
$$\text{Abstract path} : \langle 3, 0 \rangle \to \langle 2, 1 \rangle \to \langle 1, 2 \rangle \to \langle 0, 3 \rangle$$

6 Chiến lược Portfolio

6.1 Portfolio cho track Existent và Shortest

Bước	**Thành phần**	**Time limit**	**Lý do**
1	Counter Abstraction (lần 1)	10 giây	Phát hiện nhanh instance vô nghiệm đơn giản
2	Symbolic Search (SymK)	70 phút	Tìm lời giải tối ưu bằng BDD-based search
3	A* + Landmarks	70 phút	Tìm lời giải tối ưu bằng heuristic search
4	GBFS + Landmarks	70 phút	Tìm lời giải nhanh (satisficing), coverage cao
5	Counter Abstraction (lần 2)	14 giờ	Xử lý instance vô nghiệm khó

Bảng 4: Portfolio cho track Existent và Shortest [**?**]

6.2 Single-solver cho track Existent và Shortest

PARIS chạy **GBFS + Landmarks** trong 70 phút – thành phần có coverage cao nhất trong thực nghiệm.

6.3 Portfolio cho track Longest

1. **GBFS + Landmarks** (330 giây): tìm lời giải ban đầu làm lower bound.
2. **Symbolic Top-k Search** (70 phút): dùng SymK-LL để tìm và liệt kê các lời giải loopless theo thứ tự tăng dần, lọc lấy các lời giải dài hơn lower bound hiện tại.

7 Fast Downward và cơ sở hạ tầng

Tất cả các thành phần của PARIS được xây dựng trên nền tảng **Fast Downward** [**?**]:

- Fast Downward tiếp nhận input PDDL, dịch sang SAS+, rồi áp dụng các search algorithm.
- PARIS encode trực tiếp sang SAS+ (không qua PDDL) để tránh overhead của bước dịch.
- SymK dùng thư viện BDD **CUDD** để biểu diễn và thao tác tập trạng thái.

8 Phân tích output và kiểm chứng lời giải

8.1 Định dạng output chuẩn CoRe 2022

Listing 3: Ví dụ file output .out (reachable)

```
a YES
3 5
2 5
2 4
```

Listing 4: Ví dụ file output .out (unreachable)

```
a NO
```

8.2 Kiểm chứng nội bộ

Một lời giải hợp lệ phải thỏa mãn:

1. Trạng thái đầu là I_s, cuối là I_t.
2. Mỗi I_i là independent set: $\forall u, v \in I_i : \{u, v\} \notin E$.
3. Mỗi bước thay đổi đúng 1 đỉnh: $|I_i \triangle I_{i+1}| = 2$.
4. Không có trạng thái lặp lại (loopless).

9 Ví dụ end-to-end chi tiết

Input: Đồ thị 5 đỉnh với cạnh $(1,2),(2,3),(3,4),(4,5),(1,5),(2,4)$; $I_s = \{1,3\}$; $I_t = \{2,4\}$.

Bước 1 – Encoding SAS+: 6 biến ($v_1, \ldots, v_5$, hand), trạng thái ban đầu $\{v_1 = \text{occ}, v_3 = \text{occ}, \ldots\}$, goal $\{v_2 = \text{occ}, v_4 = \text{occ}, \ldots\}$. 10 actions: pick($v_i$) và place($v_i$) với $i = 1, \ldots, 5$.

Bước 2 – Chạy GBFS + Landmarks: Planner tìm được plan gồm 6 bước SAS+:

```
pick(1), place(5), pick(3), place(2), pick(5), place(4)
```

Bước 3 – Post-processing: Ghép cặp pick-place thành token jump:

1. `pick(1) + place(5)`: jump $1 \to 5$
2. `pick(3) + place(2)`: jump $3 \to 2$
3. `pick(5) + place(4)`: jump $5 \to 4$

Kết quả: Chuỗi $\{1,3\} \to \{3,5\} \to \{2,5\} \to \{2,4\}$, độ dài 3 – tối ưu.

10 Ưu điểm và hạn chế

10.1 Ưu điểm

- **Tái sử dụng công nghệ planning:** Toàn bộ hệ sinh thái planning được tận dụng mà không cần implement từ đầu.
- **Đa dạng chiến lược:** Portfolio kết hợp các thành phần bổ trợ nhau.
- **Tối ưu và satisficing:** A* và Symbolic Search đảm bảo lời giải tối ưu; GBFS cân bằng tốc độ và chất lượng.
- **Kết quả thực tế xuất sắc:** Giành phần lớn giải thưởng tại CoRe Challenge 2022.

10.2 Hạn chế

- **Chi phí bộ nhớ:** BDD và landmark computation tốn RAM với đồ thị lớn ($n > 1000$).
- **Grounding overhead:** Bước grounding vẫn chậm với đồ thị dày đặc.
- **Phụ thuộc Fast Downward:** Mọi cải tiến phụ thuộc vào hạ tầng của Fast Downward.

Tài liệu

[1] Naomi Nishimura. *Introduction to Reconfiguration. Algorithms*, 11(4):52, 2018. https://doi.org/10.3390/a11040052.

[2] Marcin Kamiński, Paul Medvedev, and Martin Milanič. *Complexity of independent set reconfigurability problems. Theoretical Computer Science*, 439:9–15, 2012.

[3] Paul Bonsma. *The complexity of rerouting shortest paths. Theoretical Computer Science*, 510:1–12, 2013.

[4] Robert A. Hearn and Erik D. Demaine. *PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theoretical Computer Science*, 343(1–2):72–96, 2005.

[5] Silvan Richter and Matthias Westphal. *The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research*, 39:127–177, 2010.

[6] Erez Karpas and Carmel Domshlak. *Cost-optimal planning with landmarks.* In *Proceedings of IJCAI 2009*, pp. 1728–1733.

[7] Julie Porteous, Laura Sebastia, and Jörg Hoffmann. *On the extraction, ordering, and usage of landmarks in planning.* In *Proceedings of ECP 2001*, pp. 37–48.

[8] Jörg Hoffmann and Bernhard Nebel. *The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[9] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. *A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] David Speck, Florian Geißer, Robert Mattmüller, and Álvaro Torralba. *Symbolic top-k planning.* In *Proceedings of AAAI 2020*, pp. 9967–9974.

[11] Álvaro Torralba, Vidal Alcázar, Peter Kissmann, and Stefan Edelkamp. *SymBA*: A symbolic bidirectional A* planner.* In *IPC-8 planner abstracts*, 2014.