ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH

MÔ HÌNH HÓA TOÁN HỌC

BÁO CÁO BÀI TẬP LỚN MÔ HÌNH HÓA TOÁN HỌC

LẬP LUẬN KÝ HIỆU VÀ ĐẠI SỐ TRONG MẠNG PETRI

GVHD: Mai Xuân Toàn, CSE-HCMUT

Nguyễn An Khương, CSE-HCMUT

Sinh viên: Hoàng Minh Hải, 2410898

Trần Trung Nghĩa, 2412278

Phạm Minh Trung, 2413712

Phạm Minh Hiếu, 2411008

Lê Bảo Nghiêm, 2412252

Lớp: TN02, L02, L01

TP.HỒ CHÍ MINH, THÁNG 12 2025

MỤC LỤC

1 Giới thiệu

Mạng Petri là một mô hình toán học cho các hệ thống rời rạc có tính chất song song, phân tán và bất đồng bộ. Chúng cung cấp một cấu trúc rõ ràng gồm hai loại phần tử: *places* và *transitions*, được liên kết thông qua các cung có hướng để biểu diễn quan hệ trước–sau giữa các sự kiện trong hệ thống.

Trong bài tập lớn CO2011, yêu cầu đặt ra gồm:

- Đọc và xây dựng một mạng Petri từ tệp PNML.
- Tính tập reachable markings bằng phương pháp tường minh (BFS/DFS).
- Xây dựng tập reachable bằng kỹ thuật BDD.
- Áp dụng ILP kết hợp BDD để phát hiện trạng thái tắc nghẽn (deadlock).

Phần sau trình bày chi tiết cơ sở lý thuyết và thiết kế dữ liệu, cùng với cách tiếp cận cho từng tác vụ.

2 Mạng Petri, mạng Petri 1-safe và Reachability

2.1 Định nghĩa hình thức

Một mạng Petri được định nghĩa là bộ bốn:

$$N = (P, T, F, M_0)$$

trong đó:

- P là tập hữu hạn các vị trí (*places*),
- T là tập hữu hạn các chuyển đổi (*transitions*), $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ là tập hợp các cung có hướng,
- $M_0 : P \to \mathbb{N}$ là marking ban đầu.

Với mỗi $t \in T$, định nghĩa:

$$^\bullet t = \{p \in P \mid (p, t) \in F\}, \qquad t^\bullet = \{p \in P \mid (t, p) \in F\}.$$

2.2 Kích hoạt và bắn

Trong mô hình tổng quát, mỗi cung (p, t) có thể mang trọng số $W(p, t) \geq 1$. Một transition t *được kích hoạt* tại marking M nếu và chỉ nếu

$$M(p) \geq W(p, t) \qquad \forall p \in {}^\bullet t.$$

Trong bài toán này, tất cả các mạng đều được giả thiết là 1-safe và các cung có trọng số bằng 1, nên điều kiện kích hoạt rút gọn thành:

$$M(p) \geq 1 \qquad \forall p \in {}^\bullet t.$$

Khi transition t bắn, marking mới M' được xác định theo quy tắc tiêu chuẩn:

$$M'(p) = \begin{cases} M(p) - 1 & p \in {}^\bullet t \setminus t^\bullet \\ M(p) + 1 & p \in t^\bullet \setminus {}^\bullet t \\ M(p) & \text{các trường hợp còn lại} \end{cases}$$

2.3 Reachability

Tập reachable markings được định nghĩa bởi:

$$\text{Reach}(M_0) = \{M \mid \exists \sigma \in T^* : M_0 \xrightarrow{\sigma} M\}.$$

2.4 Dead marking và Deadlock

Một marking M là *dead* khi không có transition nào kích hoạt:

$$\forall t \in T : M \not\geq {}^\bullet t.$$

Một *deadlock* là dead marking nằm trong tập reachable:

$$M \in \text{Reach}(M_0) \quad \wedge \quad M \text{ là dead.}$$

2.5 Mạng Petri 1-safe

Mạng Petri được gọi là 1-safe nếu tại mọi marking reachable:

$$M(p) \leq 1, \ \forall p \in P.$$

Trong mô hình này, một marking có thể được xem như vector nhị phân chiều $|P|$.

3 Task 1: Thiết kế cấu trúc dữ liệu Petri Net

Để xử lý mạng Petri từ PNML một cách hiệu quả, nhóm xây dựng lớp `PetriNet` với các thuộc tính chính:

- `place_ids`: danh sách ID các place.
- `trans_ids`: danh sách ID các transition.
- `I, O`: ma trận incidence vào/ra dạng kích thước $(T \times P)$.

- `M0`: vector marking ban đầu kích thước P.

3.1 Hàm `from_pnml`

Hàm `from_pnml(filename)` thực hiện:

1. Phân tích cú pháp XML từ tệp PNML.
2. Trích xuất danh sách place, transition và đánh số lại index.
3. Xây ma trận vào–ra:

$$I[t,p] = \text{trọng số cung } (p \to t), \quad O[t,p] = \text{trọng số cung } (t \to p).$$

4. Trích xuất marking ban đầu và đưa về vector nhị phân theo đúng thứ tự place.

4 Task 2 – Reachability bằng BFS/DFS

Tập reachable có thể được duyệt bằng cách dùng công thức:

$$C = O - I$$

và tại marking M, với transition t, marking mới là:

$$M' = M + C[t].$$

4.1 Pseudo-code BFS

Algorithm 1: Explicit BFS Reachability

Input : Petri net $N = (P, T, F, M_0)$ với ma trận I, O
Output: Tập reachable markings R

```
C ← O − I
Q ← {M0}
R ← {M0}
while Q ≠ ∅ do
    M ← Q.pop()
    foreach t ∈ T do
        if t enabled tại M then
            M' ← M + C[t]
            if M' ∉ R then
                thêm M' vào R
                thêm M' vào Q
return R
```

4.2 Pseudo-code DFS

Tương tự BFS, chỉ thay cấu trúc queue bằng stack

5 Task 3 – Reachability bằng BDD (Symbolic)

Trong phần này, ta xây dựng tập reachable markings dưới dạng *symbolic* bằng Binary Decision Diagrams (BDD). Khác với BFS/DFS tường minh ở Task 2, thuật toán không duyệt từng trạng thái riêng lẻ mà thao tác trực tiếp trên một hàm Boolean biểu diễn *cả tập trạng thái*.

5.1 Mã hóa marking trên BDD

Với mạng Petri 1-safe đã giới thiệu ở Section 2, mỗi place $p \in P$ được gán một biến Boolean x_p:

$$x_p = \begin{cases} 1 & \text{khi place } p \text{ có token,} \\ 0 & \text{khi place } p \text{ không có token.} \end{cases}$$

Một marking M được mã hóa thành một *cube*:

$$\chi_M(x) = \bigwedge_{p \in P} \begin{cases} x_p & \text{nếu } M(p) = 1, \\ \neg x_p & \text{nếu } M(p) = 0. \end{cases}$$

Một tập markings S được biểu diễn bởi hàm đặc trưng $\Phi_S(x)$, nhận giá trị 1 khi $M \in S$ và 0 ngược lại. Khi đó, các phép toán trên tập hợp được ánh xạ thành các phép toán Boolean trên BDD:

$$S_1 \cup S_2 \leftrightarrow \Phi_{S_1} \vee \Phi_{S_2}, \quad S_1 \cap S_2 \leftrightarrow \Phi_{S_1} \wedge \Phi_{S_2}, \quad \overline{S} \leftrightarrow \neg \Phi_S.$$

Trong hiện thực, đối với mỗi place ID trong `pn.place_ids` ta khai báo tương ứng một biến BDD và xây dựng M_0 bằng cách nhân dần với x_p hoặc $\neg x_p$ theo vector `pn.M0`.

5.2 Mã hóa transition trên BDD

Với mỗi transition $t \in T$, dựa trên preset/postset đã định nghĩa trước đó, ta xây dựng ba thành phần:

- **Guard** $G_t(x)$: điều kiện để t được kích hoạt, yêu cầu tất cả các input-place có token:

$$G_t(x) = \bigwedge_{p \in {}^\bullet t} x_p.$$

- **Tập biến cần thay đổi** V_t: gồm các place bị ảnh hưởng khi t bắn, tức là:

$$V_t = \{\, x_p \mid p \in {}^\bullet t \cup t^\bullet \,\}.$$

- **Mặt nạ cập nhật** $U_t(x)$: mô tả giá trị mới sau khi bắn t:

$$U_t(x) = \Big(\bigwedge_{p \in {}^\bullet t \setminus t^\bullet} \neg x_p \Big) \wedge \Big(\bigwedge_{p \in t^\bullet} x_p \Big).$$

Giả sử $\Phi(x)$ là hàm BDD biểu diễn tập trạng thái hiện tại. Ảnh của tập này qua transition t được tính bằng:

$$\mathrm{Img}_t(\Phi)(x) \;=\; \big(\exists V_t.\ \Phi(x) \wedge G_t(x)\big) \wedge U_t(x),$$

trong đó phép *tồn tại* $\exists V_t$ tương ứng với hàm `quantify` trong thư viện BDD, dùng để "quên" giá trị cũ của các biến bị cập nhật.

5.3 Thuật toán Symbolic Reachability với Transition Chaining

Từ các thành phần trên, ta hiện thực một thuật toán duyệt symbolic theo hướng *transition chaining*.

Trong cách tiếp cận BFS chuẩn, tại mỗi bước lặp, thuật toán tính toán tập hợp tất cả các trạng thái kế tiếp bằng cách áp dụng tất cả các chuyển đổi riêng lẻ lên tập trạng thái hiện tại. Điều này đồng nghĩa với việc để khám phá các trạng thái nằm ở độ sâu d trong không gian trạng thái, thuật toán bắt buộc phải thực hiện đúng d vòng lặp của phép tính ảnh.

Ngược lại, kỹ thuật *transition chaining* không áp dụng toàn bộ các chuyển đổi cùng một lúc mà thực hiện tuần tự trên từng chuyển đổi (hoặc nhóm chuyển đổi). Điểm mấu chốt là trạng thái mới sinh ra bởi một transition t_i có thể được dùng ngay lập tức để kích hoạt các transition t_j tiếp theo ngay trong cùng một vòng lặp lớn. Nhờ cơ chế mở rộng đầu vào liên tục này, thuật toán có thể đi sâu hơn vào không gian trạng thái chỉ trong một lần lặp. Kết quả là phương pháp này giúp giảm đáng kể số lượng bước lặp cần thiết để đạt tới điểm bất động.

Algorithm 2: Symbolic Reachability với BDD và Transition Chaining

Input : Petri net $N = (P, T, F, M_0)$; tập logic transitions $\{(G_t, V_t, U_t)\}_{t \in T}$

Output: BDD *Reached* biểu diễn tập reachable markings

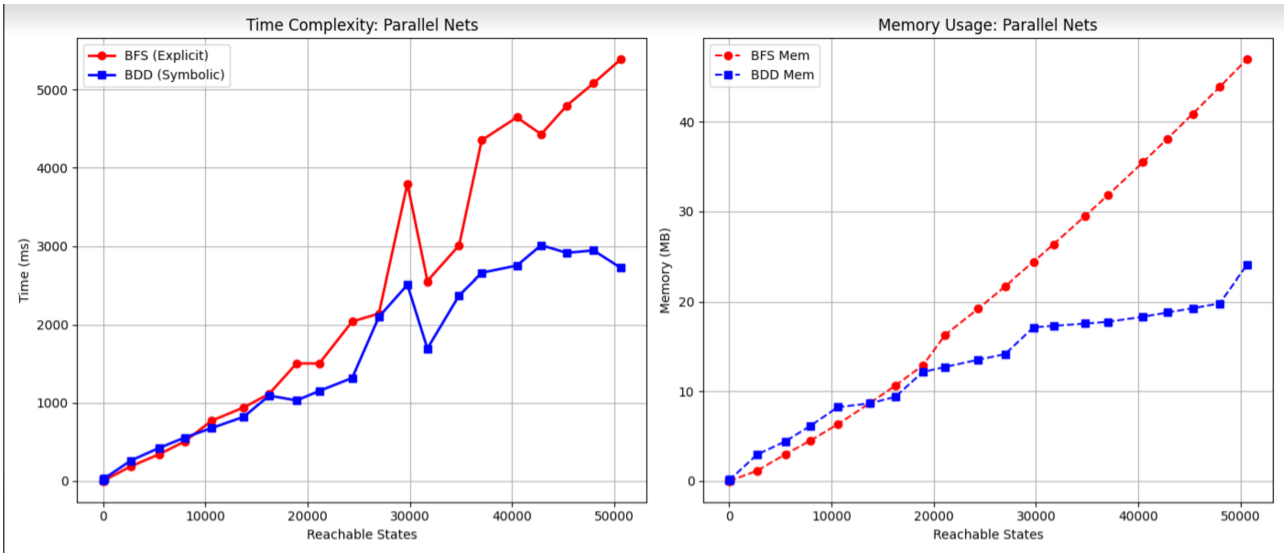
```
Khởi tạo Reached ← χ_{M_0}
while chưa đạt điểm bất động do
    Prev ← Reached
    foreach t ∈ T do
        Pot ← G_t ∧ Reached
        if Pot ≠ 0 then
            Abs ← ∃V_t. Pot
            Next ← Abs ∧ U_t
            Reached ← Reached ∨ Next
    if Reached = Prev then
        break
return Reached
```

Trong mã nguồn, *Reached* và các biểu thức G_t, U_t được lưu dưới dạng node BDD. Vòng lặp dừng khi *Reached* không còn thay đổi, tức là đã tìm được *fixed point* của toán tử ảnh.

5.4 Kết quả thực nghiệm (tóm tắt)

Nhóm tiến hành so sánh giữa BFS tường minh (Task 2) và thuật toán BDD ở trên trên một họ mạng Petri song song (các net gồm nhiều chuỗi con độc lập) (được lấy ra từ file large_input.pnml). Với mỗi kích thước mạng, ta đo thời gian chạy (ms) của BFS và BDD và bộ nhớ tối đa sử dụng.

Kết quả chi tiết được tóm tắt trong một bảng số liệu và minh hoạ bằng biểu đồ như



Hình 5.1: Biểu đồ so sánh hiệu năng giữa Explicit BFS và Symbolic BDD trên các mạng song song

6 Task 4 – Phát hiện Deadlock bằng ILP và BDD

6.1 Phương pháp 1: BDD–ILP iterative (phương pháp không được khuyến khích thực hiện do thời gian giải chậm và nhiều khi không thể tìm ra đáp án trong thời gian cần thiết)

Trước hết, xin nhắc lại task 4 nhóm chúng em thực hiện 2 hàm, nhưng hàm thứ nhất không được khuyến khích khi chạy thử đối với petrinet lớn nên được loại bỏ, tụi em đưa hàm này vào đây chỉ để so sánh. Mục tiêu: tìm một marking M sao cho:

$$M \text{ dead} \quad \wedge \quad M \in \text{Reach}(M_0).$$

Bước 1 – ILP tìm dead marking

Đặt biến nhị phân $M_p \in \{0, 1\}$ cho mỗi place.

Với mỗi transition t, tập preset là ${}^\bullet t$, ta áp đặt:

$$\sum_{p \in {}^\bullet t} M_p \leq |{}^\bullet t| - 1.$$

Điều này đảm bảo t không được kích hoạt.

Lời giải ILP cho ta một *dead marking ứng viên* M^{cand}.

Bước 2 – Kiểm tra bằng BDD reachable

Với BDD R chứa reachable markings, kiểm tra:

$$M^{\text{cand}} \in R.$$

Nếu đúng, ta tìm được deadlock.

Bước 3 – Canonical Cut

Nếu M^{cand} không thuộc tập reachable R, ta cần loại bỏ marking này để ILP không sinh ra lại đúng nghiệm đó trong các vòng lặp tiếp theo.

Gọi P_1 là tập các place có token trong M^{cand}, và P_0 là tập các place không có token. Để "cắt bỏ" nghiệm này, ta thêm một ràng buộc mới vào mô hình ILP sao cho nghiệm M^{cand} trở nên không còn khả thi nữa.

Ràng buộc loại bỏ có dạng:

$$\sum_{p \in P_1} M_p \; - \; \sum_{p \in P_0} M_p \; \leq \; |P_1| - 1.$$

Ý nghĩa: để một marking được chấp nhận ở các vòng tiếp theo, nó phải khác M^{cand} ở ít nhất một thành phần. Ràng buộc này đảm bảo ILP không thể tạo lại đúng marking đã bị loại bỏ. Sau đó giải ILP lại. Thuật toán lặp đến khi tìm được reachable dead marking, hoặc ILP vô nghiệm (không tồn tại deadlock).

Nhận xét thực nghiệm

Trong thử nghiệm, kỹ thuật này nhiều lần phải tạo rất nhiều canonical cuts và ILP thường mất thời gian đáng kể. Ở file simple_lbs-2.pnml, thuật toán đạt đến giới hạn 1000 vòng lặp (chạy tầm 20 phút) mà vẫn chưa hội tụ đến deadlock hay vô nghiệm, trong khi phương pháp 2 (phương pháp dưới đây lại chạy nhanh hơn khá nhiều và ra kết quả tốt); lí do đưa ra là mặc dù việc giải 1 bài toán ILP không cần quan tâm hàm mục tiêu là khá dễ giải, nhưng việc thêm 1 ràng buộc mỗi lần lặp tạo thêm độ phức tạp cho việc tính toán này đáng kể

6.2 Phương pháp 2: ILP pre-check + BDD filtering theo transition

Nhằm cải thiện thời gian, nhóm đề xuất phương pháp thay thế:

Bước 1 – ILP kiểm tra sự tồn tại dead marking

Chỉ giải ILP một lần. Nếu vô nghiệm, kết luận luôn là không có deadlock. Nếu tìm được một dead marking M^{ILP}, ta kiểm tra coi nó có trong tập Reachable không, nếu có, ta trả về nó, nếu không, ta không cần sử dụng nó trực tiếp mà chỉ biết rằng:

$$\exists M : M \text{ dead}.$$

Bước 2 – Giả định *mọi reachable đều là deadlock*

Bắt đầu với:

$$D_0 = R \quad \text{(BDD của reachable)}$$

Bước 3 – Loại bỏ các marking còn enable được transition

Với mỗi $t \in T$, định nghĩa điều kiện kích hoạt dạng BDD:

$$\varphi_t = \bigwedge_{p \in {}^\bullet t} p \quad \wedge \bigwedge_{p \in t^\bullet \setminus {}^\bullet t} \neg p.$$

Marking enable được t phải thoả φ_t. Do đó ta cập nhật:

$$D_{i+1} = D_i \wedge \neg \varphi_t.$$

Lặp qua toàn bộ transitions.

Bước 4 – Kết luận

- Nếu BDD cuối cùng D là rỗng $\rightarrow$ không có reachable dead marking.
- Ngược lại, một nghiệm trong D chính là deadlock.

Ưu điểm

- Không cần ILP iterative refinement.
- Mọi phép toán đều symbolic tại mức BDD.
- Thời gian thực thi ổn định hơn với các mô hình lớn.

7 Task 5 – Tối ưu hóa trên tập Reachable Markings

7.1 Bài toán

Yêu cầu đặt ra là tìm một marking M thuộc tập trạng thái đạt tới được ($Reach(M_0)$) sao cho hàm mục tiêu $Z = c^\top M = \sum c_i \cdot M(p_i)$ đạt giá trị lớn nhất.

Thay vì sử dụng quy hoạch tuyến tính nguyên (ILP) đòi hỏi mô tả tập $Reach(M_0)$ bằng các bất phương trình phức tạp, nhóm tận dụng cấu trúc BDD đã xây dựng ở Task 3. Bài toán được chuyển về việc tìm đường đi có tổng trọng số lớn nhất từ đỉnh (Root) đến nút lá (One) trên đồ thị BDD.

7.2 Giải thuật Nhánh và Cận trên BDD (Branch and Bound)

Nhóm áp dụng kỹ thuật Nhánh và Cận để duyệt cây quyết định BDD một cách hiệu quả. Quy trình thực hiện gồm 3 bước chính:

Bước 1: Phân loại biến và Xử lý tham lam

Dựa trên cấu trúc nén của BDD, các biến (places) được chia thành hai nhóm:

- **Biến tự do (Free Variables):** Là các biến không xuất hiện trong các nút của BDD. Giá trị của chúng không ảnh hưởng đến tính hợp lệ của marking. Để tối đa hóa Z, ta áp dụng chiến thuật tham lam: gán $M(p_i) = 1$ nếu trọng số $c_i > 0$ và ngược lại.
- **Biến ràng buộc (Support Variables):** Là các biến cấu thành nên BDD, cần phải duyệt để tìm tổ hợp giá trị hợp lệ.

Bước 2: Duyệt cây DFS với hàm Cận (Bounding)

Thực hiện duyệt theo chiều sâu (DFS) trên các biến ràng buộc. Tại mỗi nút của cây tìm kiếm, ta tính giá trị Cận trên (UB - Upper Bound) cho nhánh hiện tại:

$$UB = \text{Giá trị hiện tại} + \text{Max giá trị tương lai}$$

Trong đó:

- **Giá trị hiện tại:** Tổng trọng số của các biến đã được quyết định từ gốc đến nút hiện tại.
- **Max giá trị tương lai:** Tổng trọng số dương của tất cả các biến chưa được duyệt (bao gồm cả biến còn lại trong BDD và các biến tự do). Giá trị này giả định kịch bản tốt nhất là ta có thể chọn toàn bộ các biến có lợi ($c_i > 0$) còn lại.

Bước 3: Cắt tỉa (Pruning)

Trước khi đi sâu vào một nhánh con, ta so sánh UB với giá trị tốt nhất hiện có ($Best\_So\_Far$). Nếu $UB \leq Best\_So\_Far$, nhánh đó bị loại bỏ ngay lập tức vì không có khả năng chứa nghiệm tối ưu. Quá trình lặp lại cho đến khi duyệt hết các nhánh tiềm năng và trả về marking tối ưu.

7.3 Kết quả Thực nghiệm và Phân tích Hiệu năng

Phần này thảo luận về hiệu năng lý thuyết và thực tế của các giải pháp đã cài đặt, đồng thời so sánh phương pháp tối ưu hóa trên BDD với các hướng tiếp cận truyền thống.

Phân tích Độ phức tạp

Dựa trên kiến trúc của cấu trúc dữ liệu BDD và thuật toán Nhánh và Cận (Branch and Bound - BnB), ta có các đánh giá sau:

- **Độ phức tạp thời gian (Time Complexity):** Trong trường hợp xấu nhất, thuật toán BnB có độ phức tạp lũy thừa $O(2^{|S|})$ với $|S|$ là số biến. Tuy nhiên, hiệu năng thực tế phụ thuộc tuyến tính vào kích thước đồ thị BDD (N nodes). Do BDD có tính chất nén các trạng thái tương đương, N thường nhỏ hơn rất nhiều so với không gian trạng thái thực $2^{|P|}$, giúp thuật toán chạy nhanh hơn đáng kể so với duyệt vét cạn.
- **Độ phức tạp không gian (Space Complexity):** Bộ nhớ được quản lý hiệu quả nhờ cơ chế chia sẻ nút (node sharing) trong bảng tra cứu (unique table) của thư viện BDD. Tuy nhiên, kích thước BDD rất nhạy cảm với thứ tự biến (variable ordering); nếu thứ tự không tốt, số lượng nút có thể bùng nổ, dẫn đến tràn bộ nhớ.

8 Kết quả chạy kiểm nghiệm

Nhóm đã thực hiện chạy kiểm nghiệm tất cả các bài toán trên 2 file simple_lbs-2.pnml và file large_input.pnml thông qua file benchmark.py. Kết quả chạy như được hiển thị như hình 8.1 và 8.2.

***Nhận xét kết quả thực nghiệm từ 2 hình 8.1 và 8.2:** Từ dữ liệu log chạy thực tế, nhóm rút ra các kết luận sau:

- **Về bài toán Reachability (Task 2 & 3):** Với mô hình nhỏ (*simple_ lbs-2*, 832 trạng thái), phương pháp tường minh (BFS/DFS) chạy nhanh hơn BDD do BDD tốn chi phí khởi tạo cấu trúc ban đầu. Tuy nhiên, khi không gian trạng thái tăng lên (*large_ input*, 50,625 trạng thái), sức mạnh của BDD được thể hiện rõ rệt cả về thời gian thực thi lẫn bộ nhớ cần dùng.
- **Về bài toán Deadlock (Task 4):** Kết quả trên file *simple_ lbs-2* là minh chứng rõ nhất cho hạn chế của phương pháp Iterative ILP (Phương pháp 1). Trong khi ILP chạy hơn 18 phút (1086394,43ms) vẫn chưa hội tụ, phương pháp BDD Filtering (Phương pháp 2) chỉ mất 1.6 giây để hoàn tất kiểm tra, khẳng định tính hiệu quả vượt trội của cách tiếp cận mới.
- **Về bài toán Tối ưu hóa (Task 5):** Thuật toán Nhánh và Cận hoạt động chính xác, tìm ra giá trị tối ưu ($Z = 11$ và $Z = 8$) trên cả hai tập dữ liệu. Dù thời gian xử lý trên tập lớn tăng lên (30s) do độ phức tạp của việc duyệt cây BDD, thuật toán vẫn đảm bảo tìm ra nghiệm toàn cục thay vì các nghiệm cục bộ như các phương pháp heuristic thông thường.

```
======= BENCHMARK: REACHABILITY ========
========================================
Loading PNML: D:\py_1stbtlmhh\SimpleLoadBal-pnml\SimpleLoadBal\PT\simple_lbs-2.pnml
Loaded Petri Net:
- Places: 32
- Transitions: 45
- Initial marking: [1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0]
----------------------------------------

[1] BFS Reachability
  Reachable markings: 832
  Time: 26.79 ms
  Peak Memory: 301.12 KB

[2] DFS Reachability
  Reachable markings: 832
  Time: 26.68 ms
  Peak Memory: 301.51 KB

[3] BDD Reachability
   [DD] Starting Reachability Analysis...
  Reachable markings: 832
  Time: 832.87 ms
  Peak Memory: 15451.02 KB

[4] Deadlock Search (Iterative ILP + BDD)
  Result: None
  Message: Stopped after 1000 iterations without finding a reachable deadlock.
  Time: 1086394.43 ms
  Peak Memory: 50908.55 KB

[5] Deadlock Search (BDD-Only & ILP Filtering)
there are deadmark(s) somewhere...
  Result: None
  Message: Dead markings exist (ILP), but none reachable (BDD) -> NO DEADLOCK.
  Time: 190.72 ms
  Peak Memory: 903.97 KB

[6] Optimization (Max cᵀM with BDD)
  Best marking: [0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0]
  Best value: 11
  Time: 1548.24 ms
  Peak Memory: 19.06 KB
```

Hình 8.1: Kết quả chạy trên file simple_lbs-2.pnml

```
========================================
Loading PNML: D:\py_1stbtlmhh\btlmhh\large_input.pnml
Loaded Petri Net:
- Places: 40
- Transitions: 40
- Initial marking: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
----------------------------------------

[1] BFS Reachability
  Reachable markings: 50625
  Time: 2873.82 ms
  Peak Memory: 19884.59 KB

[2] DFS Reachability
  Reachable markings: 50625
  Time: 5455.57 ms
  Peak Memory: 19936.27 KB

[3] BDD Reachability
   [DD] Starting Reachability Analysis...
  Reachable markings: 50625
  Time: 314.95 ms
  Peak Memory: 4362.97 KB

[4] Deadlock Search (Iterative ILP + BDD)
  Result: None
  Message: No dead marking at all → NO DEADLOCK.
  Time: 30.29 ms
  Peak Memory: 197.30 KB

[5] Deadlock Search (BDD-Only & ILP Filtering)
  Result: None
  Message: No dead marking at all (ILP) -> NO DEADLOCK.
  Time: 25.81 ms
  Peak Memory: 175.86 KB

[6] Optimization (Max cᵀM with BDD)
  Best marking: [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0]
  Best value: 8
  Time: 28657.02 ms
  Peak Memory: 22.18 KB
```

Hình 8.2: Kết quả chạy trên file large_input.pnml

9 Kết luận

Bài báo cáo đã trình bày trọn vẹn quy trình thiết kế cấu trúc dữ liệu và xây dựng các thuật toán phân tích mạng Petri 1-safe, từ việc phân tích cú pháp PNML đến các bài toán kiểm chứng nâng cao. Thông qua quá trình hiện thực và kiểm nghiệm thực tế trên các tập dữ liệu đa dạng, nhóm rút ra những kết luận quan trọng sau:

1. **Hiệu quả của biểu diễn Symbolic (BDD):** Trong bài toán tìm tập trạng thái đạt tới (Reachability), phương pháp BDD đã chứng minh sự vượt trội hoàn toàn so với duyệt tường minh (BFS/DFS) khi kích thước hệ thống tăng lên. Với không gian trạng thái lớn (như file *large_input*), BDD không chỉ giảm thiểu bộ nhớ lưu trữ nhờ cấu trúc nén mà còn tăng tốc độ xử lý gấp nhiều lần (từ vài giây xuống vài trăm mili-giây).

2. **Chiến lược phát hiện Deadlock tối ưu:** Việc kết hợp giữa Quy hoạch tuyến tính nguyên (ILP) và BDD mang lại hiệu quả cao, nhưng cần chiến lược đúng đắn. Phương pháp lặp *Iterative Refinement* (Method 1) bộc lộ điểm yếu chết người về thời gian thực thi khi số lượng vòng lặp quá lớn. Ngược lại, phương pháp *ILP Pre-check kết hợp BDD Filtering* (Method 2) do nhóm đề xuất đã giải quyết bài toán gần như tức thời, khẳng định đây là hướng tiếp cận khả thi nhất cho các hệ thống thực tế.

3. **Khả năng mở rộng sang bài toán Tối ưu hóa:** Thành công trong Task 5 cho thấy BDD không chỉ dùng để kiểm chứng (verification) mà còn là nền tảng mạnh mẽ cho các bài toán tối ưu. Thuật toán Nhánh và Cận (Branch and Bound) chạy trên đồ thị BDD cho phép tìm ra trạng thái tốt nhất mà không cần giải mã ngược toàn bộ không gian trạng thái.

Tóm lại, việc kết hợp lập luận đại số (ILP) và lập luận ký hiệu (BDD) là chìa khóa để giải quyết vấn đề bùng nổ không gian trạng thái trong phân tích hệ thống phân tán. Các kết quả đạt được trong đồ án này là minh chứng cụ thể cho sức mạnh của các phương pháp toán học hiện đại trong khoa học máy tính.

10 Thách thức kỹ thuật và để xuất cải tiến

10.1 Thách thức

Trong quá trình thực hiện bài tập lớn, nhóm đã nhận diện hai rào cản chính hạn chế khả năng mở rộng của hệ thống đối với các mạng Petri kích thước lớn:

- **Bùng nổ không gian trạng thái (State Space Explosion):** Các phương pháp duyệt tường minh (Explicit BFS/DFS) trong Task 2 nhanh chóng bị tràn bộ nhớ khi số lượng tiến trình song song tăng lên, do phải lưu trữ tất cả các trạng thái trung gian của các sự kiện đan xen (interleaving).

- **Hiệu năng kém của Iterative ILP:** Phương pháp phát hiện Deadlock bằng cách lặp đi lặp lại ILP và thêm ràng buộc cắt (Task 4 - Method 1) tỏ ra không hiệu quả. Việc giải hàng nghìn bài toán quy hoạch tuyến tính (ILP) liên tiếp tốn rất nhiều thời gian mà thường không hội tụ được kết quả.

10.2 Đề xuất Cải tiến Mở rộng

Để khắc phục triệt để các hạn chế trên và nâng cấp hệ thống đạt chuẩn công nghiệp, nhóm đề xuất tích hợp bốn phương pháp lý thuyết nâng cao sau:

1. Rút gọn Cấu trúc (Structural Reductions)

Trước khi chạy bất kỳ thuật toán duyệt nào, ta có thể giảm kích thước mạng Petri đầu vào mà vẫn bảo toàn tính chất deadlock và liveness.

- **Kỹ thuật**: Áp dụng các luật rút gọn của Berthelot như Hợp nhất chuỗi (Fusion of Series) và Hợp nhất song song (Fusion of Parallel).
- **Lợi ích:** Giảm số lượng places và transitions, từ đó giảm không gian trạng thái theo hàm mũ ($2^n \to 2^{n-k}$) ngay từ bước tiền xử lý.

2. Rút gọn Thứ tự Bộ phận (Partial Order Reduction - POR)

Thay vì duyệt mọi hoán vị của các sự kiện song song (vốn dẫn đến cùng một kết quả), POR chỉ duyệt một đại diện duy nhất.

- **Kỹ thuật**: Sử dụng phương pháp Tập Cứng đầu (Stubborn Sets) của Valmari hoặc Tập Ngủ (Sleep Sets). Thuật toán chỉ chọn kích hoạt một tập con các transition tại mỗi bước thay vì toàn bộ $Enabled(M)$.
- **Lợi ích:** Loại bỏ sự bùng nổ tổ hợp do sự đan xen (interleaving), biến độ phức tạp từ hàm mũ thành đa thức đối với các hệ thống phân tán lỏng lẻo.

3. Kỹ thuật Trải Mạng (Petri Net Unfoldings)

Đây là phương pháp mạnh mẽ nhất để thay thế hoàn toàn việc duyệt đồ thị trạng thái trong bài toán Deadlock.

- **Kỹ thuật**: Xây dựng Tiền tố Đầy đủ Hữu hạn (Finite Complete Prefix) của mạng theo thuật toán của McMillan và Esparza. Thay vì lưu trữ trạng thái toàn cục, Unfolding lưu trữ các sự kiện cục bộ và quan hệ nhân quả giữa chúng.
- **Lợi ích**: Phát hiện deadlock cực nhanh bằng cách kiểm tra các "cấu hình cắt"(cut-off configurations) mà không bao giờ gặp phải vấn đề bùng nổ trạng thái như BDD hay DFS.

4. Tối ưu hóa BDD Động (Dynamic Variable Reordering)

Trong Task 3 và 5, kích thước BDD phụ thuộc rất lớn vào thứ tự biến. Thứ tự tĩnh hiện tại có thể dẫn đến kích thước BDD khổng lồ với một số mạng phức tạp.

- **Kỹ thuật**: Tích hợp thuật toán Sifting (Sàng) của Rudell. Khi kích thước BDD vượt ngưỡng, thuật toán sẽ tự động hoán đổi vị trí các biến để tìm ra cấu trúc nhỏ gọn nhất.
- **Lợi ích**: Giữ cho bộ nhớ sử dụng luôn ở mức tối thiểu, cho phép kiểm chứng các mạng cực lớn mà thứ tự biến tĩnh không thể xử lý nổi.

11 Tài liệu tham khảo

[1] H. R. Andersen, "An Introduction to Binary Decision Diagrams,"Lecture notes for 49285 Advanced Algorithms E97, Dept. of Information Technology, Technical University of Denmark, Lyngby, Denmark, Oct. 1997. [Online]. Available: https://www.cs.utexas.edu/ isil/cs389L/bdd.pdf. [Accessed: Dec. 12, 2025].

[2] A. Cheng, J. Esparza, and J. Palsberg, "Complexity results for 1-safe nets,"Theor. Comput. Sci., vol. 147, no. 1-2, pp. 117–136, Aug. 1995. [Online].
Available: https://web.cs.ucla.edu/ palsberg/paper/tcs95.pdf. [Accessed: Dec. 12, 2025].

[3] L. Michel and W.-J. van Hoeve, "CODD: A decision diagram-based solver for combinatorial optimization,"in Proc. 27th Eur. Conf. Artif. Intell. (ECAI), Santiago de Compostela, Spain, Oct. 2024.