**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**Đề tài**

**CONSTRAINT PROPAGATION**

**MÔN HỌC: TRÍ TUỆ NHÂN TẠO**

**GVHD: ThS. Lê Minh Tân**

**MÃ LỚP: ARIN330585\_22\_2\_02**

|  |  |
| --- | --- |
| ***Nhóm sinh viên thực hiện:*** |  |
| Nguyễn Trung Phiên | 21110593 |
| Ngô Quang Nghĩa | 21110559 |
| Lê Văn Vũ | 21110943 |
| Nguyễn Chí Thanh | 21110644 |
| Nguyễn Phạm Mạnh Hóa | 21110885 |
| Nguyễn Thành Nhơn | 21110907 |
| Đinh Quang Anh | 21110863 |

**TP Hồ Chí Minh , tháng 5 năm 2023**

**BẢNG PHÂN CÔNG NHIỆM VỤ**

**Nhóm sinh viên thực hiện: Nhóm 06 – Lớp thứ 7, Tiết 09-12**

|  |  |  |  |
| --- | --- | --- | --- |
| **Họ và tên** | **Mã số sinh viên** | **Nhiệm vụ** | **Tỉ lệ hoàn thành** |
| Nguyễn Trung Phiên | 21110593 | Code demo, thuyết trình, viết 3.2.2 | 100% |
| Ngô Quang Nghĩa | 21110559 | Viết 2.1.1, 2.1.2.1, 2.1.2.3, chỉnh sửa định dạng | 100% |
| Lê Văn Vũ | 21110943 | Viết chương 3.1, 3.2.1 | 100% |
| Nguyễn Chí Thanh | 21110644 | Viết từ 2.1.2.3 đến 2.1.2.5, làm slide | 100% |
| Nguyễn Phạm Mạnh Hóa | 21110885 | Viết mở đầu, kết luận, cảm ơn, chương 1 | 100% |
| Nguyễn Thành Nhơn | 21110907 | Viết 2.2 | 100% |
| Đinh Quang Anh | 21110863 | Viết 2.1.3, 2.1.4 | 100% |

**Nhận xét của giảng viên**

*Ngày tháng năm 2023*

*Giảng viên chấm điểm*

**ĐÁNH GIÁ**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Nội dung | **Điểm** | Cơ sở đánh giá |
| Báo cáo cuối kỳ | | | |
| 1 | Hình thức: Bìa, lời cảm ơn, lời mở đầu, mục lục, canh lề, đúng font, trang đánh giá & chấm điểm, chú thích, danh mục tài liệu tham khảo, ít lỗi chính tả,... | …/2 |  |
| 2 | Trình bày đầy đủ nội dung. Lựa chọn ứng dụng phù hợp với đề tài. | …/3 |  |
| 3 | Chạy được ít nhất 1 demo phù hợp nội dung báo cáo. | …/2 |  |
| 4 | Lập luận thuyết phục. Đưa ra đồ thị, thống kê, phân tích, kết luận đánh giá đa chiều. | .../1.5 |  |
| 5 | Tham khảo nhiều tài liệu nghiên cứu. Các tài liệu được đánh số, chú thích rõ ràng. | .../1.5 |  |
|  | Tổng: | .../10 |  |
|  | **Trưởng nhóm ký, ghi họ tên** | **Giảng viên ký, ghi họ tên** | |

Ghi chú: ……………………………………………………………………………..

**LỜI CẢM ƠN**

Trong quá trình tìm hiểu nghiên cứu thực hiện đề tài tiểu luận môn Trí Tuệ Nhân Tạo, nhóm chúng em đã nhận được sự hỗ trợ nhiệt tình chỉ bảo tận tâm từ rất nhiều người, từ bạn bè, anh chị và thầy cô. Vì thế nhóm chúng em xin gửi lời cảm ơn chân thành nhất đến với các cá nhân và tổ chức sau đây: Đầu tiên, chúng em xin cảm ơn thầy Lê Minh Tân – người hướng dẫn, giảng dạy của môn học này đã cung cấp cho chúng em kiến thức cơ bản lẫn cả nâng cao trong lĩnh vực Trí Tuệ Nhân Tạo và cũng đã hỗ trợ chúng em vô cùng nhiệt tình trong quá trình thực hiện tiểu luận. Chúng em cũng muốn bày tỏ lòng biết ơn của mình đến các nhà nghiên cứu, các chuyên gia trong lĩnh vực này, những người đã cống hiến cuộc đời mình, thanh xuân mình để tạo ra các công nghệ và thuật toán đột phá, thay đổi nhiều góc nhìn khác nhau trong Trí Tuệ Nhân Tạo, điều đó giúp cho cuộc sống của nhân loại trở nên ngày tốt đẹp và hiện đại hơn. Ngoài ra, chúng em cũng không thể quên được sự hỗ trợ của những người bạn, anh chị xung quanh trong quá trình học tập và thực hiện đề tài này. Cuối cùng, chúng em xin chân thành cảm ơn Trường Đại Học Sư Phạm Kỹ Thuật Thành Phố Hồ Chí Minh và các nhà tài trợ, các công ty công nghệ đã cung cấp điều kiện và nguồn lực để chúng em có thể hoàn thành tiểu luận này một cách chỉnh chu nhất. Một lần nữa, chúng em xin gửi lời cảm ơn sâu sắc và mong rằng những đóng góp, nghiên cứu của chúng em sẽ mang lại giá trị tốt đẹp, hữu ích cho cộng đồng khoa học và công nghệ.

**DANH MỤC TỪ VIẾT TẮT**

**Các từ viết tắt trong báo cáo**

* CSP: constraint sactifaction problem
* CSPs**:** constraint sactifaction problems
* AC: arc consistency
* DAC: directional arc consistency
* GAC: generalized arc consistency
* MAC: Maintaining arc consistency

**MỤC LỤC**

[LỜI MỞ ĐẦU 1](#_Toc134038409)

[Chương 1: Tổng quan về Constraint Propagation 2](#_Toc134038410)

[1.1. Bài toán thỏa mãn thỏa mãn ràng buộc là gì ? 2](#_Toc134038411)

[1.1.1. Định nghĩa bài toán thỏa mãn ràng buộc 2](#_Toc134038412)

[1.1.2. Giải pháp cho bài toán thỏa mãn ràng buộc 3](#_Toc134038413)

[1.2. Lan truyền ràng buộc và các khái niệm liên quan 3](#_Toc134038414)

[1.2.1. Constraint Propagation là gì? 4](#_Toc134038415)

[1.2.2. Kỹ thuật giảm thiểu vấn đề (Problem reduction techniques) 5](#_Toc134038416)

[1.2.2.1. Giảm thiểu vấn đề (Problem reduction) là gì ? 5](#_Toc134038417)

[1.2.2.2. Chiến lược giảm thiểu vấn đề 6](#_Toc134038418)

[1.2.3. Giới thiệu về Local Consistency 7](#_Toc134038419)

[1.2.3.1. Tương thích (consistency) là gì? 7](#_Toc134038420)

[1.2.3.2. Tương thích cục bộ (Local Consistency) 7](#_Toc134038421)

[Chương 2:  Local Consistency và Global constraint 8](#_Toc134038422)

[2.1. Local Consistency 8](#_Toc134038423)

[2.1.1. Node consistency 8](#_Toc134038424)

[2.1.2. Arc consistency 10](#_Toc134038425)

[2.1.2.1. Giới thiệu về Arc Consistency 10](#_Toc134038426)

[2.1.2.2.  Thuật toán AC-3 11](#_Toc134038427)

[2.1.2.3. Directional arc consistency (DAC) 16](#_Toc134038428)

[2.1.2.4. Generalized arc consistency (GAC) 17](#_Toc134038429)

[2.1.3. Path consistency 18](#_Toc134038430)

[2.1.4. K-consistency 20](#_Toc134038431)

[2.2. Global constraints 22](#_Toc134038432)

[2.2.1. Giới thiệu về Global constraints 22](#_Toc134038433)

[2.2.2. Ràng buộc AllDiffs 23](#_Toc134038434)

[2.2.3. Resource Constraint 24](#_Toc134038435)

[Chương 3: Ưu nhược điểm và ứng dụng của Constraint Propagation 26](#_Toc134038436)

[3.1. Ưu nhược điểm và ứng dụng của Constraint Propagation 26](#_Toc134038437)

[3.1.1. Ưu điểm 26](#_Toc134038438)

[3.1.2. Nhược điểm 26](#_Toc134038439)

[3.1.3. Ứng dụng của constraint propagation 27](#_Toc134038440)

[3.2. Ứng dụng constraint Propagation để giải quyết trò chơi kakuro 28](#_Toc134038441)

[3.2.1. Giới thiệu về trò Kakuro 28](#_Toc134038442)

[3.2.2. Giải quyết bài toán 30](#_Toc134038443)

[3.2.2.1. Định nghĩa bài toán dưới dạng bài toán thỏa mãn ràng buộc: 30](#_Toc134038444)

[3.2.2.2. Input và output của bài toán: 31](#_Toc134038445)

[3.2.2.3. Các bước giải quyết bài toán 32](#_Toc134038446)

[3.2.3. Thực nghiệm chứng minh sự hiệu quả của constraint propagation trong giải quyết bài toán ràng buộc 35](#_Toc134038447)

[LỜI KẾT LUẬN 38](#_Toc134038448)

[TÀI LIỆU THAM KHẢO 39](#_Toc134038449)

### **LỜI MỞ ĐẦU**

Trong thực tế luôn tồn tại những bài toán lớn, phức tạp cùng với nhiều ràng buộc khác nhau, việc tìm ra lời giải chính xác hoặc tối ưu của bài toán thường là rất khó khăn. Trong nhiều trường hợp, ta không thể tìm ra lời giải bằng cách sử dụng các kỹ thuật tính toán truyền thống mà ta cần sử dụng những phương pháp khác để giải quyết bài toán. Một trong những kỹ thuật đó chính là constraint propagation.

Constraint propagation là một cơ chế giải quyết bài toán bằng cách áp dụng các ràng buộc và thông tin mô tả về các giới hạn của bài toán để loại bỏ các khả năng không hợp lệ. Các ràng buộc có thể được biểu diễn bằng các phương trình, các giới hạn, hoặc các quy tắc khác. Ví dụ, trong bài toán xếp hậu, các quân hậu được đặt trên bàn cờ sao cho không có hai quân hậu nào cùng hàng, cùng cột, hoặc cùng đường chéo. Để giải quyết bài toán này, ta có thể sử dụng constraint propagation bằng cách loại bỏ các khả năng không hợp lệ khi đặt quân hậu, dựa trên các ràng buộc trên.

Cơ chế constraint propagation giúp giảm thiểu số lượng các khả năng phải kiểm tra và tối ưu hóa quá trình giải quyết bài toán. Điều này giúp tiết kiệm thời gian, tài nguyên và đạt được lời giải chính xác hoặc tối ưu của bài toán một cách nhanh chóng và hiệu quả.

Ngoài ra, constraint propagation còn được áp dụng trong nhiều lĩnh vực khác như lập lịch, lập kế hoạch, ... Các thuật toán constraint propagation cũng được sử dụng rộng rãi trong các hệ thống thông minh nhân tạo để giải quyết các bài toán phức tạp.

Trong chủ đề này, chúng ta sẽ tìm hiểu sâu hơn về cơ chế constraint propagation, cách nó hoạt động và các ứng dụng của nó trong các lĩnh vực khác nhau.

**Chương 1: Tổng quan về Constraint Propagation**

**1.1. Bài toán thỏa mãn thỏa mãn ràng buộc là gì ?**

Các bài toán thỏa mãn ràng buộc (CSPs) là các vấn đề toán học được định nghĩa là một tập hợp các đối tượng có trạng thái phải thỏa mãn một số ràng buộc hoặc giới hạn. CSP biểu diễn các thực thể trong một bài toán như một tập hợp đồng nhất các ràng buộc hữu hạn đối với các biến, được giải bằng các phương pháp thỏa mãn ràng buộc. Trong CSP, chúng ta có một tập hợp các biến với các miền đã biết và một tập hợp các ràng buộc áp đặt các hạn chế đối với các giá trị mà các biến đó có thể nhận. Nhiệm vụ của chúng ta là gán một giá trị cho mỗi biến để chúng thỏa mãn tất cả các ràng buộc. [CSP thường thể hiện độ phức tạp cao](https://en.wikipedia.org/wiki/Complexity_of_constraint_satisfaction), đòi hỏi sự kết hợp giữa phương pháp phỏng [đoán](https://en.wikipedia.org/wiki/Heuristics) và [tìm kiếm tổ](https://en.wikipedia.org/wiki/Combinatorial_search) hợp để có thể được giải quyết trong một khoảng thời gian hợp lý. Các bài toán thỏa mãn ràng buộc (CSPs) thường được sử dụng để mô hình hóa các vấn đề trong thế giới thực trong AI, chẳng hạn như lập lịch trình, lập kế hoạch sản xuất và nhiều vấn đề khác.[[1]](#footnote-1)

**1.1.1. Định nghĩa bài toán thỏa mãn ràng buộc**

Một vấn đề thỏa mãn ràng buộc bao gồm ba thành phần X, D và C:

* X là một tập hợp các biến, {X1, …, Xn}
* D là một tập hợp các miền, {D1, …, Dn}, một miền cho mỗi biến.
* C là một tập hợp các ràng buộc chỉ định các kết hợp giá trị cho phép.

Một miền Di bao gồm một tập hợp các giá trị cho phép {v1, …, vn} cho biến Xi. Ví dụ, một biến Boolean sẽ có miền {true, false}. Các biến khác nhau có thể có các miền khác nhau với các kích cỡ khác nhau.

Mỗi ràng buộc Cj bao gồm một cặp {scope, rel}, trong đó scope là một bộ các biến tham gia vào ràng buộc và rel là một quan hệ xác định các giá trị mà các biến đó có thể nhận. Một quan hệ có thể được biểu diễn dưới dạng một tập hợp rõ ràng của tất cả các bộ giá trị thỏa mãn ràng buộc, hoặc dưới dạng một hàm mà có thể tính toán xem một bộ có phải là thành viên của mối quan hệ hay không. Ví dụ: nếu X1 và X2 cả hai đều có miền {1,2,3}, thì ràng buộc nói rằng X1 phải lớn hơn X2 có thể được viết là ((X1,X2),{(3,1),(3,2),(2,1)}) hoặc như ((X1,X2),X1>X2).[[2]](#footnote-2)

**1.1.2. Giải pháp cho bài toán thỏa mãn ràng buộc**

Các bài toán thỏa mãn ràng buộc sẽ được giải quyết thông qua các phép gán (assignment) các giá trị vào các biến {Xi = vi, …, Xn = vn }. Một phép gán không vi phạm bất kỳ ràng buộc nào được gọi là phép gán nhất quán hoặc hợp lệ. Phép gán hoàn chỉnh là phép gán trong đó mỗi biến được gán một giá trị. Phép gán một phần là phép gán mà một số biến không được gán. Giải pháp cho một bài toán thỏa mãn ràng buộc là một phép gán hoàn chỉnh và hợp lệ. Bên cạnh đó, giải pháp một phần cho bài toán là một phép gán một phần và hợp lệ. 2

**1.2. Lan truyền ràng buộc và các khái niệm liên quan**

Khi giải quyết một bài toán tối ưu hóa ràng buộc, chúng ta cần tìm giá trị của các biến sao cho các ràng buộc được đáp ứng. Tuy nhiên, với những bài toán lớn, việc giải quyết các ràng buộc này trở nên rất phức tạp. Constraint Propagation giải quyết vấn đề này bằng cách sử dụng các ràng buộc đã biết để suy luận giá trị của các biến còn lại. Quá trình này được thực hiện bằng cách sử dụng các thuật toán thông minh để đưa ra các giả định và loại trừ các giá trị không hợp lệ cho các biến. Các giá trị hợp lệ được truyền đi và sử dụng để giải quyết các ràng buộc khác. Nhờ vậy không gian tìm kiếm các giải pháp tiềm năng có thể được giảm đáng kể, dẫn đến việc giải quyết vấn đề hiệu quả hơn. Như vậy, sự lan truyền ràng buộc đóng vai trò như một công cụ mạnh mẽ để giải quyết các CSPs phức tạp, giúp tìm ra các giải pháp chính xác và hiệu quả hơn.[[3]](#footnote-3)

**1.2.1. Constraint Propagation là gì?**

Constraint Propagation (sự lan truyền ràng buộc) là quá trình giảm miền giá trị của một biến thông qua việc áp dụng ràng buộc, thông tin này được truyền đạt cho các ràng buộc khác liên quan đến cùng một biến. Những ràng buộc này có thể sử dụng miền giá trị đã giảm này để loại bỏ các giá trị từ các miền giá trị của các biến khác, và truyền đạt thông tin này qua mạng ràng buộc và tiếp tục cho đến khi tìm thấy một giải pháp hoặc xác định rằng không có giải pháp nào tồn tại. Đây một kỹ thuật quan trọng trong lĩnh vực ràng buộc (Constraint Programming) để giải quyết các bài toán có ràng buộc, giúp tối ưu hóa quá trình giải quyết bài toán, giảm thiểu thời gian và tài nguyên tính toán. Nó được dựa trên ý tưởng rằng khi một ràng buộc được áp dụng cho một biến, thì ràng buộc đó cũng áp dụng cho các biến khác mà có liên quan đến nó.[[4]](#footnote-4)

Constraint Propagation thường được sử dụng kết hợp với các thuật toán tìm kiếm, chẳng hạn như thuật toán tìm kiếm lan truyền ràng buộc (Constraint Propagation Search) hoặc thuật toán quay lui (Backtracking Search), để tìm ra các giải pháp tối ưu hoặc tối ưu gần nhất cho các bài toán CSPs.

**1.2.2. Kỹ thuật giảm thiểu vấn đề (Problem reduction techniques)**

Problem reduction techniques là các kỹ thuật giảm kích thước vấn đề nhằm biến đổi CSP thành các vấn đề tương đương nhưng có thể dễ dàng giải quyết hơn bằng cách giảm kích thước của miền và ràng buộc trong các CSP.

Hai vấn đề CSP là tương đương nếu chúng có các tập hợp biến và giải pháp giống nhau. Một vấn đề CSP P1 được giảm thiểu thành một vấn đề P2 khi:

* P1 tương đương với P2
* Miền của các biến trong P2 là tập con của các biến trong P1
* Ràng buộc trong P2 ít nhất cũng phải bằng hoặc hạn chế hơn những ràng buộc trong P1.

Các điều kiện này đảm bảo rằng một giải pháp cho P2 cũng là một giải pháp cho P1[[5]](#footnote-5)

**1.2.2.1. Giảm thiểu vấn đề (Problem reduction) là gì ?**

Giảm thiểu vấn đề là một lớp kỹ thuật để chuyển đổi một CSP thành các vấn đề có thể dễ dàng giải quyết hơn hoặc được nhận ra là không thể giải quyết được. Mặc dù giảm kích thước vấn đề đơn thuần không tạo ra các giải pháp, nhưng nó có thể cực kỳ hữu ích khi được sử dụng kết hợp với các phương pháp tìm kiếm hoặc tổng hợp vấn đề. Giảm thiểu vấn đề đóng một vai trò rất quan trọng trong việc giải quyết CSP. Chúng ta xem các ràng buộc như là tập hợp các nhãn (label), giảm thiểu vấn đề có nghĩa là loại bỏ các nhãn từ các ràng buộc đó, những nhãn đó không xuất hiện trong bất kỳ bộ giải pháp nào. Nếu các ràng buộc được xem như là các hàm, thì việc giảm thiểu vấn đề có nghĩa là sửa đổi các hàm ràng buộc.[[6]](#footnote-6)

**1.2.2.2. Chiến lược giảm thiểu vấn đề**

Problem reduction bao gồm hai nhiệm vụ có thể thực hiện:

* Loại bỏ các giá trị dư thừa khỏi các miền của các biến .Ví dụ: DX={1,2,3,4,5,6}. Nếu các ràng buộc là x < 4 và x < 6. Giá trị 4, 5, 6 là dư thừa. Ngoài ra, lưu ý rằng Ràng buộc x < 6 là dư thừa.
* Siết chặt các ràng buộc sao cho có ít hơn các nhãn thỏa mãn chúng, tức là loại bỏ nhãn dư thừa. Ví dụ: Xét ba biến A, B và C. Giả sử rằng tồn tại các ràng buộc nhị phân giữa các biến. Xét một nhãn (<A,a>,<C,c>) thỏa mãn các ràng buộc trên A và C. Đây có thể là nhãn ghép dư thừa, Nếu chúng ta không thể gán bất kỳ giá trị nào b đến B sao cho (<A,a>,<B,b>) hoặc (<B,b>,<C,c>) thỏa mãn.[[7]](#footnote-7)

Nếu miền của bất kỳ biến hoặc bất kỳ ràng buộc nào được thu gọn thành một tập rỗng, thì ta có thể kết luận rằng vấn đề là không thể giải được. Việc giảm thiểu vấn đề phải có khả năng nhận ra các giá trị và các nhãn dư thừa. Các vấn đề được giảm thiểu có thể dễ giải quyết hơn vì các lý do sau đây. Thứ nhất, miền của các biến trong vấn đề được giảm thiểu không lớn hơn các miền trong vấn đề gốc. Điều này giúp ta có ít giá trị hơn để xem xét. Thứ hai, các ràng buộc trong vấn đề giảm thiểu ít nhất là chặt hơn so với các ràng buộc trong vấn đề gốc. Điều này có nghĩa là ta cần xem xét ít nhãn hơn trong vấn đề giảm thiểu.[[8]](#footnote-8)

### **1.2.3. Giới thiệu về Local Consistency**

### **1.2.3.1. Tương thích (consistency) là gì?**

Trong CSPs, thỏa mãn (satisfy) là khái niệm được sử dụng để chỉ việc tìm ra một giải pháp thỏa mãn tất cả các ràng buộc. Các biến được gán giá trị để thỏa mãn các ràng buộc được đưa ra. Nếu một giá trị được gán cho một biến không thỏa mãn các ràng buộc, thì giá trị đó không được chấp nhận và phải chọn giá trị khác.Trong khi đó tương thích (consistency) là khái niệm dùng để chỉ một tính chất của ràng buộc. Ràng buộc được gọi là tương thích nếu nó có thể được đáp ứng bởi các giá trị hợp lệ cho các biến liên quan. Một ràng buộc không tương thích sẽ không thể đáp ứng được, do đó giải pháp của CSPs phải đảm bảo tất cả các ràng buộc là tương thích.

**1.2.3.2. Tương thích cục bộ (Local Consistency)**

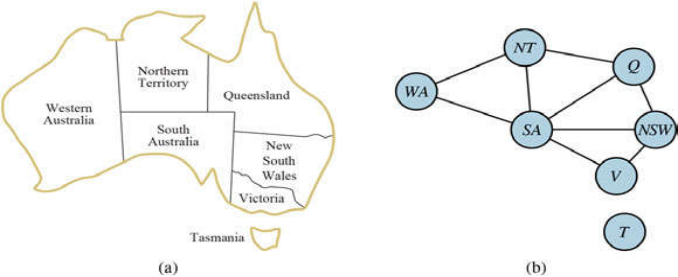
Local consistency là một yếu tố quan trọng trong các thuật toán suy luận và tìm kiếm trong trí tuệ nhân tạo và quan trọng trong các bài toán tối ưu, phân tích cú pháp, và bài toán đồ thị. Local consistency được đảm bảo khi tất cả các giá trị có thể có của các biến đều hợp lệ và đúng đắn trong các ràng buộc đã cho. Tất cả các hình thức của tính Local consistency đều có thể được thực hiện bằng việc truyền đạt ràng buộc, điều này có thể làm giảm miền giá trị của biến và tập hợp các phân bổ thỏa mãn một ràng buộc và có thể đưa ra các ràng buộc mới, từ đó giảm bớt số lượng các giá trị cần xét và cải thiện tính hiệu quả của các thuật toán.Khi việc lan truyền ràng buộc tạo ra một miền trống hoặc một ràng buộc không thỏa mãn, vấn đề ban đầu là không thỏa mãn.[[9]](#footnote-9)

**Chương 2:  Local Consistency và Global constraint**

**2.1. Local Consistency**

Trong bài toán thỏa mãn ràng buộc, local consistency (tương thích cục bộ) là một khái niệm quan trọng trong constraint propagation, chúng được sử dụng để làm cho các giá trị có thể được gán cho các biến trong một ràng buộc tương thích với nhau. Mục đích là để giảm không gian tìm kiếm và đồng thời làm cho bài toán dễ dàng giải quyết hơn. Local Consistency xác định các thuộc tính mà các bài toán ràng buộc phải đáp ứng sau khi lan truyền ràng buộc. Có nhiều loại local consistency được sử dụng như là Node Consistency (tương thích nút), Arc Consistency(tương thích cạnh),  Path Consistency(tương thích đường dẫn), K-Consistency.

**2.1.1. Node consistency**



*Hình 1. bài toán tô màu đồ thị*

Một biến đơn tương ứng với một nút trong đồ thị CSP được xem node-consistent (tương thích nút) nếu như tất cả các giá trị trong miền giá trị của biến đó phù hợp với các ràng buộc đơn liên quan đến nút đó. Trong đó ràng buộc đơn là một loại ràng buộc liên quan đến một biến duy nhất. Nó chỉ định một hạn chế đối với các giá trị có thể có mà biến có thể nhận mà không cần tham chiếu đến bất kỳ biến nào khác. Để đảm bảo tương thích nút cho một biến thì nếu có giá trị nào đó không phù hợp với ràng buộc đơn áp dụng lên nút thì sẽ lập tức loại bỏ khỏi miền giá trị của biến đó. Ngoài ra, node-consistent là một trong những phương pháp quan trọng và hiệu quả để giải quyết các các bài toán ràng buộc trong lý thuyết đồ thị cụ thể là bài toán CSP. Việc đảm bảo node consistency cho tất cả các biến trong đồ thị CSP sẽ giúp tăng cường tính khả thi, đúng đắn và đồng thời giảm đi thời gian tìm kiếm, từ đó cải thiện hiệu suất công việc.[[10]](#footnote-10)

Ví dụ: trong bài toán tô màu bản đồ nước Úc như trên hình (1) thì <(SA),SA!=green> hoặc (SA!=green) được xem là một ràng buộc đơn. Trong bài toán này, trường hợp SA với miền giá trị gồm 3 màu {red, green, blue} chúng ta có thể làm cho nó tương thích nút bằng cách loại bỏ đi màu green ra khỏi miền giá trị ban đầu, từ đó SA sẽ có một miền giá trị mới đó là {red, blue}. Từ đó thấy rằng một đồ thị được xem là tương thích nút nếu tất cả các nút trong đồ thị đó đều tương thích nút. Độ phức tạp của thuật toán Node Consistency là O(d\*n), trong đó n là số lượng biến hay còn gọi là số lượng nút trong đồ thị và d chính là kích thước miền giá trị tối đa.10

Vậy việc áp dụng Node Consistency có ưu nhược điểm như thế nào khi giải quyết các bài toán?  Node Consistency giúp tìm ra các giá trị không khả thi của biến ngay từ đầu, giúp giảm thiểu thời gian và tài nguyên tính toán cho việc giải quyết bài toán. Nhưng Node Consistency cũng có nhược điểm nhất định đó là chỉ xử lý được các ràng buộc tại một nút và không thể xử lý các ràng buộc phức tạp liên quan đến nhiều nút.

**2.1.2. Arc consistency**

**2.1.2.1. Giới thiệu về Arc Consistency**

Trong một bài toán ràng buộc CSP thì một biến trong đó được xem là **“arc-consistent”** hay còn được gọi là “tương thích cạnh” nếu như mỗi giá trị trong miền giá trị đó đều thỏa mãn các ràng buộc nhị phân của biến đó. Cụ thể hơn, nếu Xi là arc-consistent (tương thích cạnh) đối với một biến khác Xj, thì với mỗi giá trị trong miền giá trị Di hiện tại của Xi sẽ phải tồn tại một giá trị nào đó có trong miền giá trị Dj của Xj nhằm để thỏa mãn ràng buộc nhị phân trên cạnh (Xi, Xj)12. Để đảm bảo một biến là tương thích cạnh thì ta cần phải loại bỏ những giá trị mà không tương thích ra khỏi miền giá trị của biến đó. Ngoài ra, cần phải biết ký hiệu (X🡪Y) được gọi là một cạnh trong đồ thị ràng buộc, khi mà giải quyết các bài toán ràng buộc  thì việc đưa ra định nghĩa về sự tương thích cạnh sẽ không có tính đối xứng, tức là nếu như cạnh (X🡪Y) là tương thích thì không có nghĩa cạnh (Y🡪X) cũng tương thích và ngược lại.[[11]](#footnote-11)

Ví dụ khi áp dụng arc-consistent vào bài toán tô màu bản đồ nước Úc như trên hình (1). Giả sử đến thời điểm nào đó thì miền giá trị của SA chỉ còn là {red, blue}, còn miền giá trị của WA chỉ còn là {red}. Nếu mà ta chọn red cho SA và chọn red cho WA, thì lúc này sẽ không tương thích do nó đã vi phạm ràng buộc là hai biến liền kề nhau có cùng một màu, cùng một giá trị. Như vậy để đảm bảo sự tương thích giữa hai biến liền kề nhau thì cách đơn giản nhất là loại bỏ đi giá trị red ra khỏi danh sách các giá trị hợp lệ đối với biến SA, thì lúc này SA chỉ còn blue và WA thì có miền giá trị là red nên đã thỏa mãn tương thích cạnh giữa SA và WA. Ngoài ra sau khi loại bỏ red ra khỏi danh sách các giá trị hợp lệ của biến SA thì ta cũng phải cần xem xét lại các cạnh liền kề với SA như ( NT🡪SA ), ( NSW🡪SA ), ( Q🡪SA), ( V🡪SA ) sao cho tất cả các cạnh đều phù thỏa mãn ràng buộc.[[12]](#footnote-12) Bên cạnh đó còn một ví dụ khác khi áp dụng arc-consistent vào bài toán như sau, giả sử cho hai biến x, y với ràng buộc nhị phân là x < y. Nếu x thuộc khoảng giá trị [2, 6] và y thuộc khoảng giá trị [3, 7], thì ta nói rằng x thương thích cạnh với y vì với mỗi giá trị trong miền giá trị của x ta luôn tìm được ít nhất một giá trị trong miền giá trị của y sao cho hai giá trị này thỏa mãn ràng buộc nhị phân trên 2 biến đó là x < y. Nhưng nếu x thuộc khoảng [2, 7] và y thuộc khoảng [3, 7] thì x sẽ không tương thích cạnh với y vì với x = 7 thÌ không tìm được giá trị trong y thỏa ràng buộc x < y.

**2.1.2.2.  Thuật toán AC-3**

Thuật toán phổ biến nhất để giải quyết vấn đề arc consistency đó là AC-3 được phát minh bởi Mackworth vào năm 1977. Và để làm cho các biến tương thích cạnh thì AC-3 cần phải duy trì một hàng đợi cho các cạnh để xem xét, sửa đổi. Nói chung AC-3 được dùng để giảm đi số lượng các giá trị cần xét. Ban đầu, hàng đợi bao gồm tất cả các cạnh trong CSP (mỗi ràng buộc nhị phân sẽ gồm hai cạnh tương ứng với hai hướng). Sau đó, AC-3 sẽ lấy ra một cạnh tùy ý ( Xi, Xj ) từ hàng đợi và làm cho Xi tương thích cạnh với Xj. Nếu ngay bước này không làm thay đổi miền giá trị Di thì thuật toán sẽ di chuyển sang cạnh tiếp theo. Nhưng nếu nó làm thay đổi tức là làm cho miền giá trị Di trở nên nhỏ hơn thì chúng ta thêm hàng đợi cho tất cả các cạnh (Xk, Xi), tại đó Xk được xem biến lân cận với Xi. Chúng ta cần phải làm điều đó bởi vì sự thay đổi miền giá trị Di có thể cho phép giảm thêm số lượng các giá trị cần xem xét trong Dk ngay cả khi chúng ta đã xem Xk trước đó. Ngoài ra, nếu miền giá trị Di trở nên rỗng thì chúng ta kết luận rằng trong toàn bộ bài toán CSP không có một giải pháp tương thích nào cả và AC-3 có thể ngay lập tức trả về giá trị lỗi. Nếu không thì chúng ta sẽ tiếp tục kiểm tra, cố gắng xóa các giá trị từ miền giá trị của các biến cho đến khi không còn một cạnh nào tồn tại trong hàng đợi nữa. Ngay tại thời điểm đó, chúng ta chỉ còn lại một CSP tương đương với CSP gốc nhưng CSP tương thích cạnh sẽ tìm kiếm một cách nhanh chóng hơn, đơn giản hơn bởi vì các biến bây giờ sẽ có miền giá trị nhỏ hơn. Trong một số trường hợp, nó giải quyết hoàn toàn các vấn đề bằng cách giảm kích thước miền giá trị của tất cả các biến về 1 nhưng cũng sẽ có trường hợp không tìm được giải pháp nào nếu giảm kích thước của một số miền giá trị về 0.[[13]](#footnote-13)

Tóm lại, thuật toán AC-3 là một trong những thuật toán quan trọng và hiệu quả để giải quyết vấn đề arc consistency trong CSP. Thuật toán này đem lại độ hiệu quả cao vì nó cải thiện được sự khả thi của bài toán, độ chính xác của bài toán cũng được đảm bảo và có thể áp dụng cho nhiều bài toán CSP khác nhau như giải quyết bài toán Sudoku, bài toán tìm kiếm đường đi tối ưu,…

Ví dụ: khi ta xem xét các biến đi kèm với miền giá trị như sau A{1,2,3,4}, B{1,2,3,4}, C{1,2,3,4} và ràng buộc nhị phân giữa các biến trên là A<B và A>C. Sau khi thực hiện theo thuật toán AC-3 để đảm bảo tương thích cạnh giữa các biến thì ta có được miền giá trị của các biến là A={2,3}, B{3,4}, C={1,2} và quá trình giảm các miền giá trị này được biểu diễn như sau[[14]](#footnote-14):

**QUEUE ARC             DOMAIN**              {(A,B), (B,A), (A,C), (C,A)}                                                        {(B,A), (A,C), (C,A)}             (A,B)     A={1,2,3,4}   {(A,C), (C,A)}              (B,A)   B={1,2,3,4} {(C,A)}               (A,C)   A={1,2,3} {(B,A), (C,A)}                     Thêm (B,A) để kiểm tra         {(C,A)}               (B,A) B={2,3,4} {} (C,A) C={1,2,3,4}

* **Mã giả thuật toán AC-3 [[15]](#footnote-15)**

**Hàm** AC-3(csp) **trả về** false nếu tìm thấy một sự không tương thích và trả về true nếu ngược lại

queue 🡸 1 queue chứa tất cả các cạnh trong CSP

**Trong khi** queue không rỗng **thì thực hiện**

(Xi, Xj ) 🡸 Pop(queue)

**Nếu** sự thay đổi REVISE(csp,Xi,Xj) xảy ra **thì**

**Nếu** kích thước miền giá trị Di = 0 **thì trả về** false

**Với mỗi** biến Xk trong danh sách các biến lân cận của Xi - {Xj} **thì thực hiện**

Thêm cạnh (Xk, Xi) vào queue

**Trả về** True

**Hàm** REVISE(csp, Xi, Xj) **trả về** true nếu chúng ta sửa đổi miền giá trị của Xi

revised 🡸 false

**Với mỗi** mỗi phần tử x trong miền giá trị Di  **thì thực hiện**

**Nếu** không có giá trị y nào trong miền giá trị Dj cho phép cặp (x,y) thỏa mãn ràng buộc giữa Xi và Xj **thì**

Loại bỏ x ra khỏi tập giá trị Di

revised 🡸 true

**Trả về** giá trị revised

* **Đánh giá thuật toán AC-3 [[16]](#footnote-16)**

**Input:** csp nhị phân với các thành phần X, D, C

**Output:** False nếu phát hiện 1 sự không tương thích và True nếu ngược lại

**Complexity:** Độ phức tạp không gian là O(c) bởi vì thuật toán chỉ lưu trữ các cạnh trên đồ thị ràng buộc. Còn đối với độ phức tạp thời gian trong trường hợp xấu nhất là O(c\*d3) bởi vì khi giả sử một CSP có n biến, mỗi biến có kích thước miền giá trị tối đa là d và với c ràng buộc nhị phân hay còn là số cung trên đồ thị. Mỗi một cạnh (Xk, Xi) chỉ có thể được chèn vào hàng đợi chỉ d lần do Xi chỉ có nhiều nhất giá trị d để xóa. Việc kiểm tra tính tương thích của một cung có thể được hoàn thành trong O(d2) thời gian nên từ đó ta có được O(c\*d3) tổng thời gian trong trường hợp xấu nhất.

* **Các phiên bản cải tiến của thuật toán AC-3[[17]](#footnote-17)**

Thuật toán AC-3 có độ phức tạp của thuật toán còn khá cao, đây được xem là một hạn chế khá lớn và nó có thể trở nên khá chậm chạp khi xử lý các bài toán lớn, phức tạp Để khắc phục vấn đề trên thì đã có nhiều bản cải tiến của thuật toán AC-3 được đưa ra như là AC-4, AC-6, AC-2001. Các phiên bản này chủ yếu tập trung vào việc cải thiện các phương pháp thực hiện kiểm tra tính khả thi và loại bỏ các ràng buộc không khả thi. Điều này giúp tăng tốc độ xử lý của thuật toán và cải thiện hiệu suất công việc một cách đáng kể. Tuy nhiên mỗi phiên bản cải tiến sẽ có ưu điểm và nhược điểm riêng, nên việc lựa chọn phiên bản nào để giải quyết bài toán sao cho phù hợp cũng là một bước vô cùng quan trọng.

**AC-4:** Hai nhà phát minh Mohr và Henderson đã cảm thấy rằng thuật toán AC-3 không phải là một thuật toán tối ưu để giải quyết bài toán lan truyền ràng buộc nên đã đề xuất ra thuật toán AC-4 để cải thiện độ phức tạp thời gian một cách đáng kể. Ý tưởng phát triển của AC-4 rất khác so với AC-3 vì nó được dùng để lưu trữ nhiều thông tin hơn. Nếu AC-3 chỉ thực hiện được số lượng công việc tối thiểu trong một lần gọi hàm REVISE, thì bây giờ AC-4 chỉ cần đảm bảo được rằng tất cả các giá trị còn lại của xi đều tương thích với c trong đó c được coi là một ràng buộc đối với các biến liên kết với nhau. Thuật toán AC-4 đạt được sự tương thích cạnh với độ phức tạp thời gian là O(c\*d2) và độ phức tạp không gian là O(c\*d2).

**AC-6:** Hai nhà phát minh Bessiere và Cordier đã đề xuất thuật toán AC-6, đây chính là một sự kết hợp giữa tính lười của AC-3 tức là nó chỉ tính toán khi cần thiết giúp tối ưu hóa việc xử lý ràng buộc và tính đầy đủ thông tin của AC-4. Ở đây AC-6 có độ phức tạp thời gian giống như AC-4 nhưng sẽ dừng tìm kiếm hỗ trợ cho một giá trị trên một ràng buộc ngay khi đã tìm thấy ra được giá trị đầu tiên. Ngoài ra, AC-6 còn duy trì một cấu trúc dữ liệu nhẹ hơn so với AC-4. Thuật toán AC-6 đạt được sự tương thích cạnh với độ phức tạp thời gian là O(c\*d2) và độ phức tạp không gian là O(c\*d)

**AC-2001:** Thuật toán đạt được độ tối ưu bằng cách lưu trữ giá trị nhỏ nhất cho mỗi giá trị trên mỗi ràng buộc, tương tự như như thuật toán AC-6. Thuật toán AC-2001 đạt được sự tương thích cạnh với độ phức tạp trong thời gian là O(c\*d2) và độ phức tạp không gian là O(c\*d).

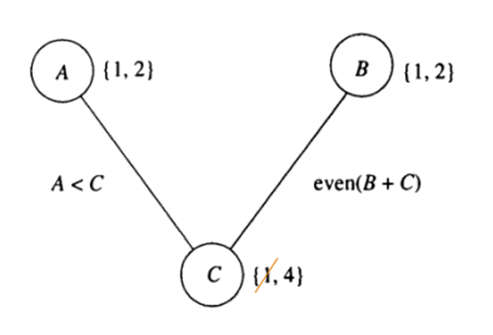
**2.1.2.3. Directional arc consistency (DAC)**

Directional arc consistency được là một bản mở rộng của arc consistency, được sử dụng trong các bài toán ràng buộc nhiều chiều. Nó kiểm tra tính hợp lệ của mỗi giá trị trong miền của biến dựa trên các ràng buộc chỉ theo hướng từ biến này đến các biến khác. Nó đảm bảo rằng các giá trị này sẽ đủ để thỏa mãn các ràng buộc theo hướng đó.

Một CPS được gọi Directional arc consistency (DAC) theo thứ tự của các biến X1,X2,X3,X4…Xn khi và chỉ khi với mọi Xi là arc-consistent với mỗi Xj sao cho j > i.[[18]](#footnote-18)

* **So sánh giữa directional arc consistency và arc consistency**

DAC mạnh hơn AC trong quá trình tìm kiếm vì nó xem xét các ràng buộc theo một hướng nhất định, điều này có thể giúp loại bỏ các giá trị không nhất quán sớm hơn. Trong khi đó AC chỉ xem xét các ràng buộc theo cả hai hướng mà không tính đến tính hướng của các ràng buộc. Tuy nhiên, xét về tính đảm bảo tương thích cạnh thì AC mạnh hơn DAC vì AC đảm bảo mọi giá trị trong một miền nhất quán với mọi giá trị khác, bất kể chúng có hướng như thế nào, trong khi DAC chỉ đảm bảo tính nhất quán theo một hướng nhất định.

Ví dụ:  Chạy DAC theo cả hai hướng ACB và BCA, theo hướng BCA thì nó chỉ xóa 1 ra khỏi miền giá trị của C. Tuy nhiên kết quả thu được thì vẫn không đảm bảo tương thích cạnh vì lúc này cạnh (B 🡺 C) không nhất quán do giá trị 1 trong miền giá trị của B không tương thích với bất kì giá trị nào còn lại trong miền giá trị của C.

*Hình 2. Ví dụ về DAC theo cả hai hướng thì không đảm bảo tương thích cạnh cho bài toán*

**2.1.2.4. Generalized arc consistency (GAC)**

* **Định nghĩa và ví dụ[[19]](#footnote-19)**

Generalized arc consistency (GAC) là một phương pháp mở rộng của khái niệm arc consistency để xử lý các ràng buộc n-ary thay vì chỉ là các ràng buộc nhị phân (binary constraints), còn được gọi là hyper-arc consistency.

Một biến Xi là generalized arc consistent với ràng buộc n-ary nếu đối với mỗi giá trị v trong miền giá trị của Xi thì đều tồn tại bộ giá trị là phần tử của ràng buộc, bộ này chứa tất cả các giá trị được lấy từ miền giá trị của các biến tương ứng bị ảnh hưởng bởi ràng buộc, và có thành phần Xi của nó bằng v.

Ví dụ: Nếu tất cả các biến đều có miền giá trị {0, 1, 2, 3}, với ràng buộc bậc ba X < Y < Z. Ta có biểu diễn ràng buộc bậc 3 lại như sau ((X, Y, Z), ((0, 1, 2), (1, 2, 3)). Để làm cho biến X đạt được tính tương thích với ràng buộc bậc 3 chúng ta cần phải loại bỏ 2 và 3 khỏi miền giá trị của X vì trong các bộ giá trị của ràng buộc bậc 3 thì không có bộ nào có giá trị là 2 hoặc 3 ở vị trí của biến X.

* **So sánh generalized arc consistency (GAC) và arc consistency (AC)**

AC chỉ áp dụng được cho ràng buộc nhị phân, trong khi GAC có thể áp dụng cho rất nhiều loại ràng buộc khác nhau. GAC có thể giải quyết được các ràng buộc áp dụng cho nhiều biến, trong khi AC chỉ có thể giải quyết được các ràng buộc nhị phân, hạn chế trong việc giải nhiều bài toán phức tạp. GAC tốn nhiều tài nguyên hơn so với AC, do đó AC thường được sử dụng khi giải quyết các vấn đề đơn giản. GAC cũng có thể không hiệu quả nếu ràng buộc quá phức tạp hoặc quá lớn.

**2.1.3. Path consistency**

Path consistency (tính nhất quán trên đường đi) là một thuật toán trong giải quyết ràng buộc trong đó mỗi ràng buộc được áp dụng trên một path (đường đi) giữa hai biến để đảm bảo tính toàn vẹn của ràng buộc. Path consistency thắt chặt các ràng buộc nhị phân bằng cách sử dụng các ràng buộc ngầm được suy ra bằng cách xem xét bộ ba biến. Một tập hợp hai biến {Xi,Xj}  được xem là path-consistent (tương thích đường) đối với biến thứ ba nếu với mọi phép gán {Xi=a, Xj=b} phù hợp với các ràng buộc, tồn tại một phép gán Xm thỏa mãn các ràng buộc trên {Xi, Xm} và {Xm, Xj}.[[20]](#footnote-20)

Hãy xét thuật toán path consistency cho bài toán tô màu bản đồ Australia chỉ với hai màu đỏ và xanh. Chúng ta sẽ làm cho tập hợp {WA, SA} tương thích đường với NT. Chúng ta bắt đầu bằng cách liệt kê các phép gán nhất quán cho tập hợp. Trong trường hợp này, chỉ có 2 trường hợp là {WA=red, SA=blue} hoặc {WA=blue, SA=red}. Chúng ta có thể thấy rằng với cả hai phép gán này, NT không thể có màu red hoặc blue (vì nó sẽ mâu thuẫn với WA hoặc SA). Bởi vì không có lựa chọn hợp lệ nào cho, nên chúng ta loại bỏ cả hai phép gán và cuối cùng chúng ta không có phép gán hợp lệ nào cho. Do đó, ta có thể kết luận rằng không thể có giải pháp nào cho vấn đề trên.24

Bên cạnh đó, Montanari đề xuất path consistencynhư sau :

* Trong đại số quan hệ:Ri,j ⊆ πi,j(Ri,m ⨝Dm ⨝ Rm,j)
* Để đạt được tính nhất quán của đường dẫn:
* Ri,j’← Ri,j∩ π i,j(Ri,m ⨝ Dm ⨝ Rm,j)
* Ri,j← Ri,j’

Thuật toán trên được gọi là PC-2. Nếu tất cả đường dẫn có độ dài 2 được làm nhất quán, thì tất cả đường dẫn có độ dài bất kỳ đều nhất quán [Montanari 1974], vì vậy không cần xem xét đường dẫn dài hơn. Điều này được gọi là path consistency bởi vì người ta có thể coi nó giống như nhìn vào một đường dẫn từ Xi đến Xj với Xm ở giữa.[[21]](#footnote-21)

Ví dụ cho thuật toán của Montanari là xét mạng N với các biến x1, x2, x3, miền D(x1) = D(x2) = D(x3) = {1, 2} và C = {x1 = x2, x2 = x3}. N không nhất quán về đường đi vì cả ((x1, 1),(x3, 2)) hay ((x1, 2),(x3, 1)) đều không thể mở rộng thành giá trị x2 thỏa mãn cả c12 và c23. Mạng N′ = ( X, D, C ∪ {x1 = x3}) là đường nhất quán.[[22]](#footnote-22)

Mặt khác khi so sánh giữa arc consistency và path consistency, ta thấy path consistency mạnh hơn ac consistency vì nó kiểm tra tính khả thi của các ràng buộc trên các đường đi dài hơn, chứ không chỉ giữa các cặp biến. Tuy nhiên, path consistency có độ phức tạp tính toán cao hơn so với ac consistency. Vì vậy, khi giải quyết các bài toán ràng buộc lớn và số lượng ràng buộc không quá lớn, arc consistency có thể là một lựa chọn tốt hơn. Việc lựa chọn thuật toán nào phụ thuộc vào độ phức tạp của bài toán và số lượng ràng buộc cần được kiểm tra.

**2.1.4. K-consistency**

K-consistency là một thuật toán trong giải quyết ràng buộc , ám chỉ tính toàn vẹn của ràng buộc trên tập hợp K biến. Tập hợp K biến được gọi là K-tuple. Tính *K-consistency* được định nghĩa như sau: Một CSP là k-consistency nếu đối với bất kỳ tập hợp k-1 biến nào đối với bất kỳ phép gán tương thích nào cho các biến đó thì sẽ luôn có một giá trị tương thích có thể được gán cho biến thứ k còn lại. 1-consistency phát biểu rằng, cho tập hợp rỗng, ta có thể làm cho bất kỳ tập hợp nào của một biến trở nên tương thích, hay nói cách khác ta gọi là node consistency (tương thích nút). Tương tự như thế thì 2-consistency giống như arc consistency. Đối với đồ thị ràng buộc nhị phân, 3-consistency cũng giống như path consistency.[[23]](#footnote-23)

Giả sử một mạng bao gồm các biến x1, x2, x3 với các miền D(x1) = D(x2) = D(x3) = {1, 2} và một ràng buộc duy nhất c(x1, x2, x3) = {(1, 1, 1),(2, 2, 2)}. Mạng này nhất quán về cạnh vì nó không chứa bất kỳ ràng buộc nhị phân nào. Nó không nhất quán 3 vì việc khởi tạo (x1 = 1, x2 = 2), là một nhất quán cục bộ, không thể mở rộng nhất quán cho x3. Tính nhất quán 3 tạo ra ba ràng buộc nhị phân c12 = {(1, 1),(2, 2)}, c23 = {(1, 1),(2, 2)} và c13 = {(1, 1),( 2, 2)}[[24]](#footnote-24).

Một bài toán về thỏa mãn ràng buộc là strongly k-consistent nếu nó là k-consistent và cũng là (k-1)-consistent, (k-2)-consistent,… tiếp tục thỏa mãn đến 1-consistent. Giả sử chúng ta có một bài toán về thỏa mãn ràng buộc với n nút và làm cho nó strongly n-consistent (nghĩa là strongly k-consistent với k=n). Khi đó ta có thể giải bài toán như sau: Đầu tiên, chúng ta chọn một giá trị tương thích cho X1. Sau đó, ta phải đảm bảo rằng có thể chọn một giá trị cho X2 vì đồ thị là 2-consistent, tương tự với X3 vì nó là 3-consistent, ... Với mỗi biến Xi, ta chỉ cần tìm các giá trị d trong tập xác định để tìm giá trị phù hợp với X1,...,Xi-1. Tổng thời gian chạy chỉ là O(n2\*d).

Ví dụ, ta xét mạng với các biến x1, . . . , x6, miền giá trị bằng {1, 2} và C={c1(x1, x2, x3, x4), c2(x2, x3, x4, x5), x2 = x6, x6 ≠ x3}, với c1 = {(1112), (1121), (1211), (2122), (2212), (2221)} và c2 = {(1112), (1211), (1222), (2112), (2122), ( 2222)}. Nếu chúng ta áp dụng 4-consistency trên x2, x3, x4 đối với x1 và x5, chúng ta rút ra ràng buộc c3(x2, x3, x4) = {(121),(122),(211),(212)}. Tính nhất quán 3 trên x2, x3 với x4 tạo ra ràng buộc c4(x2, x3) = {(12),(21)}. Bằng cách áp dụng tính nhất quán 3 đầu tiên cho x2, x3 với x6, ràng buộc c4 sẽ được tạo ra trước c3. Vì vậy, c3 sẽ không bao giờ được tạo bởi vì tất cả các bộ dữ liệu của nó đã không phù hợp với c4[[25]](#footnote-25).

Vậy lý do tại sao 2 - consistency (hay arc consistency) thường được sử dụng phổ biến ?

* Độ phức tạp thời gian: 2-consistency có độ phức tạp thời gian tuyến tính O(n2), trong đó n là số lượng biến trong ràng buộc. Điều này có nghĩa là 2-consistency có thể được áp dụng trên rất nhiều bài toán ràng buộc với số lượng biến lớn mà vẫn đảm bảo thời gian chạy tốt.
* Hiệu quả: 2-consistency là một phương pháp giải ràng buộc đơn giản, dễ hiểu và dễ triển khai. Nó cũng có độ hiệu quả cao trong việc giảm số lượng giá trị có thể cho mỗi biến, giúp tăng tốc quá trình giải quyết ràng buộc.
* Tính toàn vẹn: 2-consistency đảm bảo tính toàn vẹn của ràng buộc trên từng cặp biến, tức là cho phép loại bỏ các giá trị không hợp lệ với ràng buộc đó. Điều này giúp cải thiện chất lượng của giải pháp tìm được.
* Tính ứng dụng: 2-consistency có thể được áp dụng trong nhiều lĩnh vực khác nhau, bao gồm lập lịch, tối ưu hóa và trí tuệ nhân tạo. Nó là một phương pháp cơ bản và quan trọng trong giải quyết ràng buộc.

**2.2. Global constraints[[26]](#footnote-26)**

**2.2.1. Giới thiệu về Global constraints**

Global constraints (ràng buộc toàn cục) là một ràng buộc liên quan đến một số biến tùy ý (nhưng không nhất thiết là tất cả các biến). Các ràng buộc toàn cục xảy ra thường xuyên trong các bài toán thực tế như tối ưu hóa ràng buộc (constraint optimization), lập lịch, quy hoạch tài nguyên, quy hoạch tuyến tính,… và có thể được xử lý bằng các thuật toán chuyên dụng. Các ràng buộc toàn cục này có tác dụng liên kết các biến với nhau và giúp giảm thiểu số lượng giải pháp không hợp lệ, tăng tốc độ và chính xác của các phương pháp giải quyết bài toán.  Ví dụ: trong bài toán xếp hình puzzle, một global constraint có thể được sử dụng để đảm bảo rằng các khối hình phải được đặt vào các vị trí khác nhau, không được chồng lên nhau. Hoặc trong bài toán tối ưu lập lịch sản xuất, một global constraint có thể được sử dụng để đảm bảo rằng tổng thời gian thực hiện các công việc trên mỗi máy không vượt quá một ngưỡng nhất định.

Các global constraints thường được cung cấp sẵn trong các thư viện tối ưu và được sử dụng phổ biến trong các lĩnh vực như tối ưu hóa tổ hợp, giải quyết bài toán ràng buộc, lập lịch sản xuất, lập lịch thời gian, điều phối chuyển động robot và nhiều lĩnh vực khác trong đời sống.

**2.2.2. Ràng buộc AllDiffs**

Đây là một ràng buộc phổ biến trong các bài toán tối ưu hóa, được sử dụng để đảm bảo rằng các biến nhận các giá trị khác nhau. Nó được áp dụng cho các bài toán tối ưu hóa như xếp lịch, lập lịch, tối ưu hóa địa chỉ mạng,.…

Ràng buộc AllDifferent đảm bảo rằng tất cả các biến được liên kết với ràng buộc này phải có các giá trị khác nhau. Vì vậy, trong trường hợp của các bài toán xếp lịch hoặc lập lịch, ràng buộc AllDifferent sẽ đảm bảo rằng mỗi công việc chỉ được thực hiện một lần và không có hai công việc nào cùng được thực hiện tại cùng một thời điểm.

Một hình thức phát hiện sự không nhất quán đơn giản cho các ràng buộc AllDiffs hoạt động như sau: nếu m biến có liên quan đến ràng buộc và nếu chúng có n giá trị khác biệt có thể có cùng nhau và m>n, thì ràng buộc không thể được thỏa mãn. Ví dụ: giả sử chúng ta có ba biến A,B,C và chúng có thể nhận các giá trị sau: biến A có thể nhận giá trị 1 hoặc 2, biến B có thể nhận giá trị 2 hoặc 3, biến C có thể nhận giá trị 3 hoặc 4. Tổng cộng, có n = 4 giá trị khác nhau mà các biến có thể nhận. Nhưng vì có m = 3 biến, nó sẽ không thể đảm bảo rằng các giá trị của chúng là khác nhau. Do đó, ràng buộc AllDiffs không thể được thỏa mãn trong trường hợp này.

Điều này dẫn đến thuật toán đơn giản sau: Đầu tiên, loại bỏ bất kỳ biến nào trong ràng buộc có miền đơn lẻ và xóa giá trị của biến đó khỏi các miền giá trị của các biến còn lại. Lặp lại cho đến khi không còn biến đơn lẻ nào. Nếu tại bất kỳ thời điểm nào mà một miền giá trị rỗng hoặc có nhiều biến hơn số giá trị miền giá trị còn lại, thì một sự mâu thuẫn đã được phát hiện.

Ví dụ: thuật toán trên có thể phát hiện sự không thống nhất trong phép gán{WA = red, NSW = red}. Lưu ý rằng các biến SA, NT và Q được kết nối hiệu quả bởi ràng buộc Alldiffs vì mỗi cặp phải có hai màu khác nhau. Sau khi áp dụng AC-3 với phần gán màu ban đầu là {WA = red, NSW = red}, miền giá trị của các biến SA, NT và Q đều bị thu hẹp xuống chỉ còn {green, blue}. Điều này có nghĩa là ta có ba biến nhưng chỉ có hai màu, vì vậy ràng buộc Alldiffs bị vi phạm. Do đó trong một số trường hợp, việc áp dụng một thuật toán đơn giản để kiểm tra tính nhất quán của ràng buộc bậc cao có thể hiệu quả hơn so với việc áp dụng tính nhất quán qua tương đương với một tập hợp các ràng buộc nhị phân.

**2.2.3. Resource Constraint**

Resource Constraint (ràng buộc tài nguyên) thường được gọi là Atmost constraint.Ví dụ: Trong bài toán lập kế hoạch, đặt P1,P2,P3,P4 biểu thị số lượng nhân sự được phân công cho mỗi nhiệm vụ trong bốn nhiệm vụ. Ràng buộc rằng không quá  10 nhân viên được chỉ định tổng cộng được viết là: Atmost (10,P1,P2,P3,P4). Chúng ta có thể phát hiện sự không nhất quán đơn giản bằng cách kiểm tra tổng các giá trị nhỏ nhất của các miền hiện tại. Ví dụ: Nếu mỗi biến có miền {3,4,5,6}, ràng buộc Atmost (10,P1,P2,P3,P4) không thể được thỏa mãn. Vì nếu tất cả các biến P1,P2,P3,P4 có giá trị nhỏ nhất bằng 3, tổng của chúng sẽ là 12, lớn hơn giới hạn 10 của ràng buộc Atmost. Do đó, không có bất kỳ bộ gán nào cho P1,P2,P3,P4 có thể thỏa mãn ràng buộc này. Chúng ta cũng có thể thực thi tính nhất quán bằng cách xóa giá trị tối đa của bất kỳ miền nào nếu giá trị đó không nhất quán với giá trị tối thiểu của các miền khác. Do đó, nếu mỗi biến trong ví dụ trên có miền {2,3,4,5,6}, thì các giá trị 5 và 6 có thể bị xóa khỏi miền.

Đối với các bài toán lớn giới hạn tài nguyên với các giá trị số nguyên, chẳng hạn như các bài toán hậu cần liên quan đến việc di chuyển hàng nghìn người trên hàng trăm phương tiện, thường không thể biểu diễn miền của mỗi biến dưới dạng một tập hợp số nguyên lớn và giảm dần tập hợp đó theo tính nhất quán, các phương pháp kiểm tra. Thay vào đó, các miền được đại diện bởi các giới hạn trên và dưới và được quản lý bằng cách truyền giới hạn. Ví dụ: trong bài toán lập lịch trình của hãng hàng không, giả sử có hai chuyến bay, F1 và F2, trong đó các máy bay có sức chứa tương ứng là 165 và 385.

Các miền ban đầu cho số lượng hành khách trên các chuyến bay F1 và F2 sau đó là: D1 = [0, 165] và D2 = [0, 385].

Bây giờ, giả sử chúng ta có ràng buộc bổ sung là hai chuyến bay cùng nhau phải chở 420 người: F1 + F2 = 420. Với ràng buộc này, ta có thể áp dụng kỹ thuật propagating bounds constraints để giảm miền giá trị của F1 và F2.

Với chuyến bay F1, ta có miền giá trị là {0,1,…,165} dựa trên sức chứa của máy bay. Tuy nhiên, với ràng buộc F1 + F2 = 420, ta có thể suy ra rằng miền giá trị của F2 phải nằm trong khoảng {255,…385}, bởi vì nếu F1 đạt giá trị lớn nhất là 165 thì F2 phải đạt giá trị nhỏ nhất là 255 để thỏa mãn ràng buộc F1 + F2 = 420.

Tương tự, với chuyến bay F2, ta có miền giá trị là {0,1,…,385}, nhưng vì ràng buộc F1 + F2 = 420 nên ta có thể suy ra rằng miền giá trị của F1 phải nằm trong khoảng {35,…,165}, bởi vì nếu F2 đạt giá trị lớn nhất là 385 thì F1 phải đạt giá trị nhỏ nhất là 35 để thỏa mãn ràng buộc F1 + F2 = 420.

D1 = [35, 165] và D2 = [255, 385].

Chúng ta nói rằng một CSP (problem ràng buộc) là bounds-consistent nếu đối với mỗi biến X, và cả giá trị lower-bound và upper-bound của X, đều tồn tại một giá trị của biến Y mà thỏa mãn ràng buộc giữa X và Y cho mỗi biến Y. Việc bounds propagation như vậy được sử dụng rộng rãi trong các bài toán ràng buộc thực tế.

# **Chương 3: Ưu nhược điểm và ứng dụng của Constraint Propagation**

## **3.1. Ưu nhược điểm và ứng dụng của Constraint Propagation**

### **3.1.1. Ưu điểm**

Một trong những ưu điểm chính của lan truyền ràng buộc là nó có thể làm giảm không gian tìm kiếm bằng cách giảm các miền giá trị của các biến và cắt tỉa các nhánh dẫn đến ngõ cụt. Điều này có thể làm cho vấn đề dễ giải quyết hơn và cải thiện hiệu suất của thuật toán của bạn. Ví dụ: nếu sử dụng lan truyền ràng buộc để gán màu cho bản đồ, có thể thấy rằng một số khu vực chỉ còn một màu có thể có và ta có thể gán màu đó mà không cần khám phá thêm. Một ưu điểm khác của lan truyền ràng buộc là nó giúp tìm ra các giải pháp tổng quát hơn. Ví dụ: nếu sử dụng lan truyền ràng buộc để giải câu đố Sudoku, ta có thể phát hiện ra rằng một số ô thuộc về một tập hợp con có thể được giải độc lập với phần còn lại.[[27]](#footnote-27)

### **3.1.2. Nhược điểm**

Một trong những nhược điểm chính của lan truyền ràng buộc là tốn nhiều bộ nhớ do trong quá trình giải bài toán cần phải lưu các thông tin về ràng buộc, các giá trị hợp lệ cũng như các giá trị đã loại bỏ. Ta có thể phải kiểm tra và cập nhật các miền và ràng buộc của nhiều biến, đồng thời thực hiện nhiều suy luận logic. Điều này có thể đặc biệt tốn kém nếu các ràng buộc là phi tuyến tính hoặc toàn cục. Bên cạnh đó, các thuật toán của kỹ thuật lan truyền ràng buộc có thể rất khó và phức tạp để thực hiện. Một hạn chế nữa của lan truyền ràng buộc là tính linh hoạt thấp do chỉ có thể giải quyết các bài toán ràng buộc với các ràng buộc được xác định trước đó. 31

### **3.1.3. Ứng dụng của constraint propagation**

Constraint propagation là công cụ hữu ích trong thiết kế thuật toán, được áp dụng trong nhiều lĩnh vực và ứng dụng khác nhau. Tô màu bản đồ là một ví dụ: bằng cách giảm các miền của mỗi khu vực dựa trên màu của các khu vực lân cận, có thể gán màu cho bản đồ sao cho không có hai khu vực liền kề có cùng màu. Tương tự, constraint propagation có thể được dùng để giải các bài toán mật mã và trò chơi N-queens. Đối với bài toán mật mã, giảm miền của mỗi chữ cái dựa trên các ràng buộc số học. Trong bài toán N-queens, dùng lan truyền ràng buộc để đặt N quân hậu trên bàn cờ N x N sao cho không có hai quân hậu tấn công lẫn nhau bằng cách giảm miền của mỗi hàng và cột dựa trên các ràng buộc chéo. [[28]](#footnote-28)

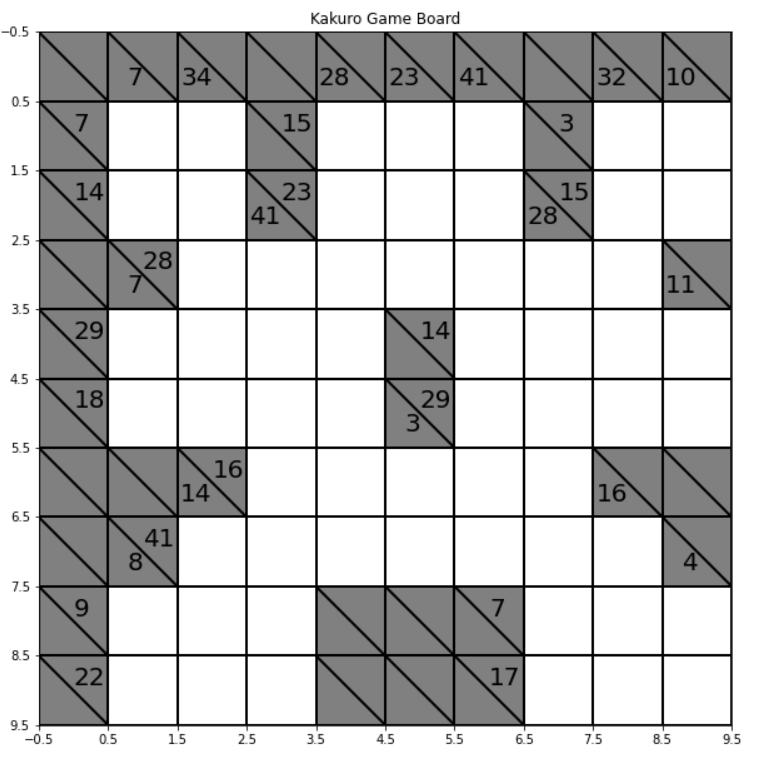
Dưới đây là một số ứng dụng khác của Constraint Propagation:

* Lập lịch và quản lý tài nguyên: Constraint propagation có thể được sử dụng để lập lịch và quản lý tài nguyên trong các dự án phức tạp, chẳng hạn như lập lịch công việc, lập lịch sản xuất, quản lý nguồn lực, và quản lý hệ thống phân phối.
* Tối ưu hóa lịch trình: Constraint propagation có thể được sử dụng để tối ưu hóa lịch trình cho các dịch vụ giao hàng, xe buýt, máy bay, hoặc các ứng dụng giao thông công cộng khác. Thuật toán này giúp đưa ra các lựa chọn lịch trình hợp lệ dựa trên các ràng buộc, chẳng hạn như thời gian hoàn thành công việc, giới hạn tốc độ, và khả năng đối tượng.
* Lập kế hoạch sản xuất: Constraint propagation cung cấp các công cụ hỗ trợ quản lý lịch trình sản xuất, đồng bộ hóa công việc giữa các máy móc, dự báo nguồn lực cần thiết và đưa ra lịch trình sản xuất hiệu quả. Giúp cải thiện hiệu suất sản xuất, giảm thiểu thời gian chờ đợi và đạt được tối đa hóa lợi nhuận nâng cao chất lượng sản xuất.
* Lập kế hoạch tài nguyên tự động: Constraint propagation cũng có thể được sử dụng để lập kế hoạch tài nguyên tự động, chẳng hạn như lập kế hoạch dành cho hệ thống điện, nước, ga, hoặc hệ thống mạng. Đưa ra lịch trình tối ưu cho việc sử dụng tài nguyên, đồng thời đáp ứng các ràng buộc về an toàn, hiệu suất giải quyết bài toán.

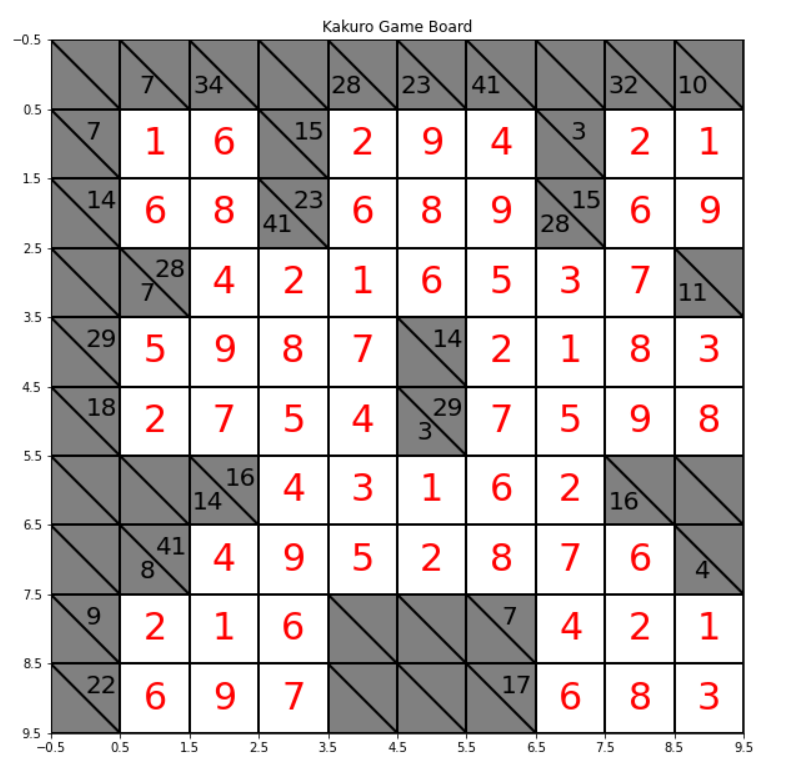
## **3.2. Ứng dụng constraint Propagation để giải quyết trò chơi kakuro**

### **3.2.1. Giới thiệu về trò Kakuro**

Kakuro là một trò chơi trí tuệ, một loại câu đố logic xuất phát từ Nhật Bản, hiện nay nó phổ biến khắp thế giới với tên gọi tiếng anh là Cross Sums. Kakuro là một loại trò chơi ô số, với các ô xếp theo hàng ngang và hàng dọc đan xen nhau, gọi là các khối. Các ô màu đen chứa một dấu gạch chéo từ phía trên bên trái đến phía dưới bên phải và một số ở một hoặc cả hai nửa của ô. Số trong các ô đen gọi là manh mối, số này cho bạn biết nó là tổng của các số trong các ô trắng của khối đó. Mục tiêu của câu đố là chèn một chữ số từ 1 đến 9 vào mỗi ô trắng để tổng các số trong mỗi mục nhập khớp với manh mối liên quan đến nó và sẽ không một số nào được xuất hiện hai lần trong bất kỳ khối nào[[29]](#footnote-29). Và câu đố Kakuro được biểu diễn như hình sau:



*Hình 3. Câu đố Kakuro 10x10*



*Hình 4. Lời giải cho câu đố kakuro 10x10*

* + 1. **Giải quyết bài toán**

**3.2.2.1. Định nghĩa bài toán dưới dạng bài toán thỏa mãn ràng buộc[[30]](#footnote-30):**

**Variables** : Xem xét bài toán này gồm 3 loại biến

* Biến rỗng : tương ứng với các ô gạch chéo nhưng không có manh mối điền vào.
* Biến giá trị: tương ứng với các ô trắng
* Biến ràng buộc: tương ứng với các ô gạch chéo nhưng có manh mối điền vào. Được chia làm hai loại là biến ràng buộc trên hàng ngang nếu manh mối nằm ở góc trên bên phải của ô ở và biến ràng buộc trên cột dọc nếu manh mối nằm ở góc dưới bên trái của ô.

🡺Ta xét tập các biến của bài toán này gồm hai loại biến là biến giá trị và biến ràng buộc.

**Domains**:

* Đối với biến giá trị thì sẽ có miền giá trị là {1, 2, 3, … 9}. Nhưng vì tổng các số trong 1 khối phải bằng với manh mối (giá trị tổng của các ô trong khối). Nên ta sẽ khéo léo giảm miền giá trị của biến giá trị thành {1,2,3, … min (9, giá trị tổng tương ứng) }. Trường hợp biến này nhận nhiều miền giá trị khác nhau thì miền giá trị cuối cùng của biến sẽ là giao của các miền giá trị đó với nhau.
* Đối với biến ràng buộc thì miền giá trị là 1 tập hợp chứa các chỉnh hợp chập k của tập các số từ 1 đến 9 sao cho tổng của các số trong mỗi chỉnh hợp bằng với manh mối tương ứng của biến ràng buộc. Ở đây k sẽ là số ô trong 1 khối mà biến ràng buộc này ảnh hưởng lên.

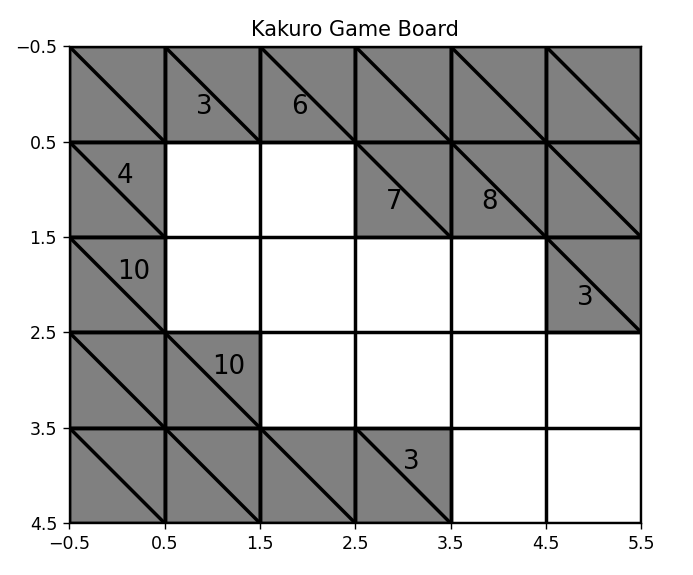
**Constraints:**

* Bài toán ràng buộc này chỉ gồm các ràng buộc nhị phân giữa hai biến đó là biến giá trị và biến ràng buộc.
* Ràng buộc nhị phân sẽ kiểm tra xem giá trị gán cho 1 biến giá trị có bằng với giá trị ở vị trí tương ứng trong 1 chỉnh hợp (1 giá trị của biến ràng buộc ) hay không.

### **3.2.2.2. Input và output của bài toán[[31]](#footnote-31):**

**Input:** Đầu vào sẽ là hai ma trận horizontal và vertical. Trong hai ma trận này, những ô có giá trị ‘# ‘ đại diện cho những ô trống không chứa giá trị trong bảng game kakuro, những ô có giá trị 0 đại diện cho những ô chứa giá trị. Đối với những ô có giá trị khác ‘#’ và khác 0 thì nếu ô đó thuộc ma trận horizontal đồng nghĩa với ô đó là ô ràng buộc trên hàng ngang, tương tự trên ma trận vertical thì ô đó là ô ràng buộc trên cột dọc và giá trị tại ô đó chính là manh mối.

**Ví dụ:** cho một câu đố kakuro



*Hình 5. Câu đố kakuro 5x6*

🡺Câu đố trên sẽ được biểu diễn tương ứng bởi hai ma trận:

Horizontal = [['#', '#', '#', '#', '#', '#'],

[4, 0, 0, '#', '#', '#'],

[10, 0, 0, 0, 0, '#'],

['#', 10, 0, 0, 0, 0],

['#', '#', '#', 3, 0, 0]]

Vertical = [['#', 3, 6, '#', '#', '#'],

['#', 0, 0, 7, 8, '#'],

['#', 0, 0, 0, 0, 3],

['#', '#', 0, 0, 0, 0],

['#', '#', '#', '#', 0, 0]]

**Ouput:** Đầu ra của bài toán là 1 phép gán các giá trị cho các biến mà thỏa mãn tất cả các ràng buộc nhị phân trong bài toán.

### **3.2.2.3. Các bước giải quyết bài toán**

**B1**: Chuyển bài toán thành bài toán thỏa mãn ràng buộc

**B2**: Trước khi tìm kiếm, sử dụng AC-3 để giảm miền giá trị của các biến

**B3**: Trong quá trình tìm kiếm, sử dụng Backtracking kết hợp với MAC (maintaining arc consistency)

**B4**: Đưa ra phép gán giá trị cho các biến và kết thúc

* **Chi tiết về các thuật toán dùng để giải quyết bài toán**

Bên cạnh thuật toán AC-3 đã được trình bày ở phần trên thì ta sẽ xem xét thuật toán MAC và thuật toán backtracking kết hợp với MAC

* Inferences là 1 sự suy luận, nếu AC-3 được sử dụng để giảm miền giá trị của các biến trước khi bắt đầu tìm kiếm thì inferences được sử dụng để suy ra miền giá trị bị giảm của các biến lân cận với biến vừa được gán giá trị trong suốt quá trình tìm kiếm .Inferences mà ta sử dụng để giải quyết bài toán này đó là MAC (maintaining arc consistency).Ý tưởng thuật toán MAC (maintaining arc consistency): sau khi một biến Xi được gán một giá trị , Hàm MAC sẽ gọi AC-3, nhưng thay vì dùng 1 queue mà ban đầu chứa tất cả các cạnh của bài toán thỏa ràng buộc, thì chúng ta chỉ khởi tạo queue ban đầu chứa các cạnh (Xj, Xi ) với Xj là các biến chưa được gán giá trị và là lân cận của Xi .[[32]](#footnote-32)
* Thuật toán Backtracking kết hợp với MAC. Mục đích của việc kết hợp này là sau khi dùng backtracking để thử gán giá trị cho một biến thì ta sử dụng tương thích cạnh để thu hẹp miền giá trị của các biến lân cận mà chưa được gán giá trị từ đó giảm không gian tìm kiếm của bài toán. Vì vậy việc tìm kiếm giải pháp cho bài toán sẽ nhanh chóng hơn. Trong quá trình tìm kiếm thì nếu như áp dụng MAC mà phát hiện bất kì biến nào có miền giá trị bị giảm thành tập rỗng thì sẽ quay lui ngay. 32
* **Mã giả của thuật toán MAC**: 33

**Hàm** MAC (csp, variable, assignment) **trả về** false nếu phát hiện một sự không tương thích và true nếu ngược lại

Queue 🡸 1 queue chứa các cạnh (Xj, Xi ) với Xj là các biến lân cận của Xi và chưa được gán giá trị.

**Trả về** AC\_3 (csp, queue)

* **Mã giả của thuật toán Backtracking kết hợp với MAC**:[[33]](#footnote-33)

**Hàm** BACKTRACKING-SEARCH-WITH-MAC (csp) **trả về** một giải pháp hoặc None

**Trả về** BACKTRACKING\_WITH\_MAC (csp, {})

**Hàm** BACKTRACKING\_WITH\_MAC (csp, assignment) **trả về** một giải pháp hoặc None

**Nếu** assignment đã đầy đủ **thì trả về** assignment

Var 🡸 SELECT-UNASSIGNED-VARIABLE(csp, assignment)

**Với mỗi** value **trong** ORDER-DOMAIN-VALUES(csp, var) **thì**

**Nếu** value tương thích với assignment **thì**

Thêm cặp {var = value} vào assignment

Miền giá trị của Var 🡸 {value}

inferences 🡸 MAC (csp, var, assignment)

**Nếu** inferences khác false **thì**

Result 🡸 BACKTRACKING\_WITH\_MAC (csp, assignment)

**Nếu** result khác None **thì trả về** resutlt

Khôi phục miền giá trị của bài toán

Loại bỏ cặp { var = value } khỏi assignment

**Trả về** None

* + 1. **Thực nghiệm chứng minh sự hiệu quả của constraint propagation trong giải quyết bài toán ràng buộc**

Tiến hành chạy thực nghiệm với các 5 bài toán kakuro có kích thước từ 5x6 (21 biến) đến 12x10 (117 biến) với ba hướng giải quyết: chỉ sử dụng backtrcking search, sử dụng AC-3 và backtracking search, sử dụng AC-3 và backtracking search kết hợp với MAC.

Thực nghiệm được tiến hành trên máy tính laptop với bộ xử lý core i5 2.42 GHz, 8GB RAM, hệ điều hành Windows. Toàn bộ code thực nghiệm được biên dịch và chạy trên zeppelin.Tiến hành chạy mỗi bài toán 10 lần với mỗi hướng giải quyết, thu thập số liệu và lập được bảng như sau:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Puzzle** | **X** | **Y** | **C** | **V** | **Backtracking time (s)** | **AC-3 + Backtracking time(s)** | **AC-3 +Backtracking**  **+ MAC time (s)** |
| 1 | 5 | 6 | 9 | 12 | 0.029153490 | 0.005269670 | 0.008548951 |
| 2 | 7 | 7 | 20 | 28 | --------- | 0.075520753 | 0.068440103 |
| 3 | 8 | 8 | 23 | 36 | ---------- | 1.192771291 | 0.151578378 |
| 4 | 10 | 10 | 36 | 62 | ---------- | ---------- | 240.7447957754135 |
| 5 | 12 | 10 | 44 | 73 | --------- | ---------- | 15.012227392196655 |

*Bảng 1. Thể hiện X: số hàng, Y: số cột, C: số biến ràng buộc, V: là số biến giá trị. Giá trị thời gian ở mỗi ô là thơi gian tìm kiếm trung bình để tìm ra giải pháp*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Puzzle** | **X** | **Y** | **C** | **V** | **Count Backtracking** | **Count AC-3**  **+**  **Backtracking** | **Count AC-3**  **+ Backtracking +**  **MAC** |
| 1 | 5 | 6 | 9 | 12 | 1498 | 36 | 21 |
| 2 | 7 | 7 | 20 | 28 | ---------- | 7721 | 51 |
| 3 | 8 | 8 | 23 | 36 | ---------- | 141149 | 64 |
| 4 | 10 | 10 | 36 | 62 | ---------- | ---------- | 12449 |
| 5 | 12 | 10 | 44 | 73 | ---------- | ---------- | 803 |

*Bảng 2. Thể hiện X: số hàng, Y: số cột, C: số biến ràng buộc, V: là số biến giá trị. Giá trị đếm ở mỗi ô là số lần đệ quy ( gọi hàm backtrack) trung bình khi giải quyết bài toán.*

Với những ô có chuỗi ký tự “--------" có nghĩa là khi ta sử dụng hướng giải quyết trên cột tương ứng để giải quyết game kakuro trên hàng tương ứng thì đã xảy ra hiện tượng tràn bộ nhớ (stack overflow).

Dựa theo 2 bảng số liệu trên ta có nhận xét rằng: Nếu chỉ sử dụng backtracking thì ta chỉ giải quyết được puzzle số 1 và sẽ xảy ra ra tràn bộ nhớ khi cố gắng dùng nó để giải quyết các puzzle về sau vì số lần gọi đệ quy quá nhiều do phải tìm kiếm trên không gian tìm kiếm lớn. Nếu ta sử dụng AC-3 trước khi backtracking thì kết quả sẽ khả quan hơn khi ta giải quyết được thêm 2 puzzle nữa, nhưng vẫn gây ra tràn bộ nhớ khi giải quyết 2 puzzle cuối. Cuối cùng khi sử dụng backtracking + MAC sau khi đã sử dụng AC-3 thì ta đã giải quyết được hết 5 puzzle. Rõ ràng thuật toán tìm kiếm backtracking không phải là thuật toán tìm kiếm tối ưu để giải quyết bài toán này vì dễ gây ra tràn bộ nhớ khi bài toán có kích thước lớn và phức tạp hơn. Nhưng ở đây ta chỉ bàn về sự hiệu quả khi áp dụng các phương pháp thu hẹp miền giá trị của các biến để giải quyết các bài toán ràng buộc nên ta sẽ bỏ qua việc đánh giá về độ hiệu quả của giải thuật tìm kiếm.

Qua nhận xét trên ta có thể kết luận rằng việc sử dụng AC-3 để giảm miền giá trị trước khi tìm kiếm sẽ giúp giảm không gian tìm kiếm, giúp việc tìm kiếm giải pháp nhanh hơn và cũng giảm thiểu khả năng gây tràn bộ nhớ khi gọi đệ quy nhiều lần. Sau đó việc tiếp tục duy trì tương thích cạnh trong quá trình tìm kiếm khi sử dụng MAC cũng giúp giảm thiểu không gian tìm kiếm rõ rệt. Vì vậy Constraint propagation nói chung hay các thuật toán AC-3, MAC nói riêng khá hiệu quả trong việc giảm miền giá trị của các biến, giảm thiểu không gian tìm kiếm của bài toán, giúp cho việc tìm kiếm ra giải pháp của bài toán nhanh chóng và hiệu quả hơn.

# **LỜI KẾT LUẬN**

Như vậy, bài báo đã giới thiệu về khái niệm cơ bản của constraint propagation và những ứng dụng của nó trong việc giải quyết các bài toán ràng buộc. Nói chung, constraint propagation là một kỹ thuật quan trọng để giải quyết vấn đề ràng buộc, bài báo cáo cũng giới thiệu các loại constraint propagation phổ biến nhất như forward checking, arc consistency, path consistency. Các phương pháp này có thể được sử dụng để giải quyết các bài toán ràng buộc trong nhiều lĩnh vực khác nhau, từ tự động hóa thiết kế đến quy hoạch tài nguyên và lập lịch. Mặc khác, ngoài những ưu điểm như giúp tăng hiệu suất và tối ưu xử lí các bài toán phức tạp thì nó cũng kèm theo không ít những nhược điểm như tốn bộ nhớ, tính linh hoạt thấp. Bên cạnh đó, bài báo cũng lưu ý rằng, mặc dù các phương pháp constraint propagation có thể hữu ích trong giải quyết các vấn đề ràng buộc, chúng vẫn không đảm bảo sẽ tìm được lời giải tối ưu nhất cho mọi bài toán và cần phải được áp dụng một cách hợp lí để đạt tối đa hiệu quả mà chúng có thể mang lại.

# **TÀI LIỆU THAM KHẢO**

1/ Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education

2/ F.Rossi, P.van Beek and T.Walsh (2006), Chapter 3, *Handbook of Constraint Programming*, Elsevier

3/ Edward Tsang (1996), *Foundations of Constraint Satisfaction*, Academic Press Limited

4/ Maria Simi, (28/09/2017), *AI Fundamentals: Constraints Satisfaction Problems*, Truy cập ngày 23/04/2023 – 30/04/2023, từ https://elearning.di.unipi.it/pluginfile.php/15224/mod\_resource/content/2/CSP-2.pdf

5/ Nguyễn Nhật Quang (2019-2020), *Trí Tuệ Nhân Tạo*, Truy cập ngày 25/04/2023, từ <https://huuvinhfit.files.wordpress.com/2015/01/chuong-5-thoa-man-rang-buoc.pdf>

6/ Wikipedia*(04/04/2023)*, *Constraint Satisfaction Problem,* truy cập ngày 24/04/2023, từhttps://en.wikipedia.org/wiki/Constraint\_satisfaction\_problem

7/ *Contraint Satisfaction Problems and Techniques*, Truy cập ngày 26/04/2023, từ https://uweb.engr.arizona.edu/~ece566/class/lecture-notes/Module-II/Mod-II-Lec-5.pdf

8/ IBM Corporation (25/11/2021), *Contraint propagation*, Truy cập ngày 27/04/2023, từ <https://www.ibm.com/docs/en/icos/20.1.0?topic=optimizer-constraint-propagation>

9/ *Local Consistency – Uses of Local Consistency*, Truy cập ngày 27/04/2023, từ <https://www.liquisearch.com/local_consistency/uses_of_local_consistency>

10/ Linkedin, *What are the advantages and challenges of using constraint propagation in CSPs*, truy cập ngày 28/04/2023,từ <https://www.linkedin.com/advice/1/what-advantages-challenges-using-constraint-propagation>

11/ Wikipedia *(18/01/2023)*, *Kakuro*, truy cập ngày 24/04/2023 từ <https://en.wikipedia.org/wiki/Kakur>o

12/ Prathamnagpure (31/07/2022), *Kakuro-Solver*, truy cập ngày 30/04/2023 - 02/05/2023 từ <https://github.com/prathamnagpure/Kakuro-Solver>

**DANH MỤC HÌNH ẢNH, BẢNG BIỂU**

**Danh mục hình ảnh**

[*Hình 1. Bài toán tô màu đồ thị* 8](#_Toc134046481)

[*Hình 2. Ví dụ về DAC theo cả hai hướng vẫn không đảm bảo tương thích cạnh* 17](#_Toc134046482)

[*Hình 3. Câu đố Kakuro 10x10* 29](#_Toc134046483)

[*Hình 4. Lời giải cho câu đố kakuro 10x10* 29](#_Toc134046484)

[*Hình 5. Câu đố kakuro 5x 6* 31](#_Toc134046485)

**Danh mục bảng biểu**

[*Bảng 1. Thể hiện X: số hàng, Y: số cột, C: số biến ràng buộc, V: là số biến giá trị. Giá trị thời gian ở mỗi ô là thơi gian tìm kiếm trung bình để tìm ra giải pháp* 36](#_Toc134046882)

[*Bảng 2. Thể hiện X: số hàng, Y: số cột, C: số biến ràng buộc, V: là số biến giá trị. Giá trị đếm ở mỗi ô là số lần đệ quy ( gọi hàm backtrack) trung bình khi giải quyết bài toán* 36](#_Toc134046883)

1. Wikipedia*(04/04/2023)*, *Constraint Satisfaction Problem*, truy cập ngày 24/04/2023, từhttps://en.wikipedia.org/wiki/Constraint\_satisfaction\_problem [↑](#footnote-ref-1)
2. Stuart J.Russell and Pter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-2)
3. Maria Simi, (28/09/2017), *AI Fundamentals: Constraints Satisfaction Problems*, Retrieved 23/04/2023 – 30/04/2023, from https://elearning.di.unipi.it/pluginfile.php/15224/mod\_resource/content/2/CSP-2.pdf [↑](#footnote-ref-3)
4. IBM Corporation (25/11/2021), *Contraint propagation*, Truy cập ngày 27/04/2023, từ <https://www.ibm.com/docs/en/icos/20.1.0?topic=optimizer-constraint-propagation> [↑](#footnote-ref-4)
5. Maria Simi, (28/09/2017), *AI Fundamentals: Constraints Satisfaction Problems*, Retrieved 23/04/2023 – 30/04/2023, from https://elearning.di.unipi.it/pluginfile.php/15224/mod\_resource/content/2/CSP-2.pdf [↑](#footnote-ref-5)
6. Edward Tsang (1996), *Foundations of Constraint Satisfaction*, Academic Press Limited [↑](#footnote-ref-6)
7. *Contraint Satisfaction Problems and Techniques*, Truy cập ngày 26/04/2023, từ https://uweb.engr.arizona.edu/~ece566/class/lecture-notes/Module-II/Mod-II-Lec-5.pdf [↑](#footnote-ref-7)
8. Edward Tsang (1996), *Foundations of Constraint Satisfaction*, Academic Press Limited [↑](#footnote-ref-8)
9. *Local Consistency – Uses of Local Consistency*, Truy cập ngày 27/04/2023, từ <https://www.liquisearch.com/local_consistency/uses_of_local_consistency> [↑](#footnote-ref-9)
10. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-10)
11. Maria Simi, (28/09/2017), *AI Fundamentals: Constraints Satisfaction Problems*, Retrieved 23/04/2023 – 30/04/2023, from https://elearning.di.unipi.it/pluginfile.php/15224/mod\_resource/content/2/CSP-2.pdf [↑](#footnote-ref-11)
12. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-12)
13. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-13)
14. Maria Simi, (28/09/2017), *AI Fundamentals: Constraints Satisfaction Problems*, Retrieved 23/04/2023 – 30/04/2023, from https://elearning.di.unipi.it/pluginfile.php/15224/mod\_resource/content/2/CSP-2.pdf [↑](#footnote-ref-14)
15. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-15)
16. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-16)
17. F.Rossi, P.van Beek and T.Walsh (2006), Chapter 3, *Handbook of Constraint Programming*, Elsevier [↑](#footnote-ref-17)
18. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach* (4th edition), Pearson Education [↑](#footnote-ref-18)
19. Maria Simi, (28/09/2017), *AI Fundamentals: Constraints Satisfaction Problems*, Retrieved 23/04/2023 – 30/04/2023, from https://elearning.di.unipi.it/pluginfile.php/15224/mod\_resource/content/2/CSP-2.pdf [↑](#footnote-ref-19)
20. Stuart J.Russell and Peter Norvig (2021), Chapter 6*, Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-20)
21. Maria Simi, (28/09/2017), *AI Fundamentals: Constraints Satisfaction Problems*, Retrieved 23/04/2023 – 30/04/2023, from <https://elearning.di.unipi.it/pluginfile.php/15224/mod_resource/content/2/CSP-2.pdf> [↑](#footnote-ref-21)
22. F.Rossi, P.van Beek and T.Walsh (2006), Chapter 3, *Handbook of Constraint Programming*, Elsevier [↑](#footnote-ref-22)
23. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-23)
24. F.Rossi, P.van Beek and T.Walsh (2006), Chapter 3, *Handbook of Constraint Programming*, Elsevier [↑](#footnote-ref-24)
25. F.Rossi, P.van Beek and T.Walsh (2006), Chapter 3, *Handbook of Constraint Programming*, Elsevier [↑](#footnote-ref-25)
26. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-26)
27. Linkedin, *What are the advantages and challenges of using constraint propagation in CSPs*, truy cập ngày 28/04/2023,từ https://www.linkedin.com/advice/1/what-advantages-challenges-using-constraint-propagation [↑](#footnote-ref-27)
28. Linkedin, *What are the advantages and challenges of using constraint propagation in CSPs*, truy cập ngày 28/04/2023,từ https://www.linkedin.com/advice/1/what-advantages-challenges-using-constraint-propagation [↑](#footnote-ref-28)
29. Wikipedia *(18/01/2023)*, *Kakuro*, truy cập ngày 24/04/2023 từ <https://en.wikipedia.org/wiki/Kakur>o [↑](#footnote-ref-29)
30. Prathamnagpure (31/07/2022), *Kakuro-Solver*, truy cập ngày 30/04/2023 - 02/05/2023 từ <https://github.com/prathamnagpure/Kakuro-Solver> [↑](#footnote-ref-30)
31. Prathamnagpure (31/07/2022), *Kakuro-Solver*, truy cập ngày 30/04/2023 - 02/05/2023 từ <https://github.com/prathamnagpure/Kakuro-Solver> [↑](#footnote-ref-31)
32. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-32)
33. Stuart J.Russell and Peter Norvig (2021), Chapter 6, *Artificial Intelligence: A Modern Approach (4th edition)*, Pearson Education [↑](#footnote-ref-33)