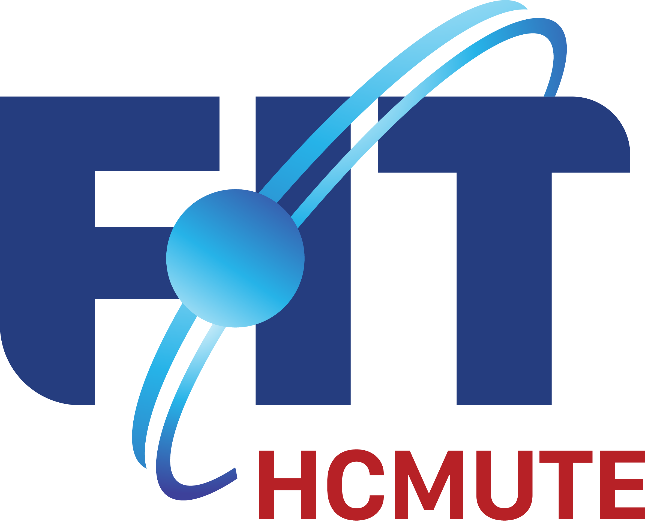
**Trường đại học sư phạm kỹ thuật TP.HCM**

**Khoa Công Nghệ Thông Tin**

**Bộ môn trí tuệ nhân tạo**



**TRẦN VŨ KHANH – 22110349**

**VŨ VĂN ĐỨC – 22110312**

**LÊ VĂN TÚ – 22110454**

***Đề Tài :***

**CONSTRAINT PROPAGATION**

**VÀ ỨNG DỤNG THUẬT TOÁN AC-3 GIẢI BÀI TOÁN SUDOKU**

**BÁO CÁO CUối KÌ**

**MÔN học: TRÍ TUỆ NHÂN TẠO \_ ARIN330585**

**Giáo viên hướng dẫn: THS. Lê Minh Tân**

Khóa 2022 – 2026

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

Họ và tên Sinh viên 1 : Trần Vũ Khanh MSSV 1: 22110349

Họ và tên Sinh viên 2 : Vũ Văn Đức MSSV 2: 22110312

Họ và tên Sinh viên 3 : Lê Văn Tú MSSV 3: 22110454

Ngành: Công nghệ Thông tin

Tên đề tài: Constraint propagation và ứng dụng thuật toán AC-3 giải bài toán Sudoku

Họ và tên Giáo viên hướng dẫn: THS. Lê Minh Tân

**NHẬN XÉT**

Về nội dung đề tài & khối lượng thực hiện:

1. Ưu điểm:

1. Khuyết điểm

1. Đề nghị cho bảo vệ hay không ?
2. Đánh giá loại :
3. Điểm :

Tp*. Hồ Chí Minh, ngày 03 tháng 05 năm 2024*

Giáo viên hướng dẫn

*(Ký & ghi rõ họ tên)*

**LỜI CẢM ƠN**

Đầu tiên, nhóm em xin gửi lời cảm ơn chân thành đến thầy Lê Minh Tân, giảng viên hướng dẫn của chúng em. Thầy đã tận tình giảng dạy, hướng dẫn chúng em để có thể hoàn thành bài tiểu luận này, đã truyền đạt cho chúng em những kiến thức độc đáo và vô cùng thú vị về trí tuệ nhân tạo. Đây chắc chắn là những kiến thức quý báu, là hành trang để nhóm em có thể vững bước sau này khi đi học và đi làm.

Bộ môn Trí tuệ nhân tạo là môn học vô cùng thú vị, bổ ích và có tính thực tế cao. Dù đã cố gắng hết sức mình, nhưng những thiếu sót và hạn chế trong bài của nhóm là điều không thể tránh khỏi. Chúng em rất mong nhận được đánh giá, nhận xét và lời góp ý chi tiết từ thầy và các bạn để củng cố năng lực cho từng cá nhân. Một lần nữa, chúng em xin cảm ơn thầy.

Nhóm chúng em xin chân thành cảm ơn!

Nhóm sinh viên thực hiện**.**

**MỤC LỤC**

[**1. CONSTRAINT PROPAGATION 1**](#_Toc165691815)

[**1.1. LOCAL CONSISTENCY 1**](#_Toc165691816)

[*1.1.1. Local consistency là gì?* 1](#_Toc165691817)

[*1.1.2. Node consistency* 1](#_Toc165691818)

[*1.1.3. Arc Consistency* 2](#_Toc165691819)

[*1.1.4. Path Consistency* 6](#_Toc165691820)

[*1.1.5. K – Consistency* 7](#_Toc165691821)

[**1.2. GLOBAL CONSTRAINT 7**](#_Toc165691822)

[*1.2.1. Global Constraint là gì?* 7](#_Toc165691823)

[*1.2.2. AllDifferent Constraints* 8](#_Toc165691824)

[*1.2.3. Resource Constraint* 8](#_Toc165691825)

[**2. ƯU ĐIỂM, NHƯỢC ĐIỂM VÀ NHỮNG ỨNG DỤNG CỦA CONSTRAINT PROPAGATION 9**](#_Toc165691826)

[**2.1. Ưu điểm 9**](#_Toc165691827)

[**2.2. Nhược điểm 10**](#_Toc165691828)

[**2.3. So sánh Global Constraints với Local Consistency 10**](#_Toc165691829)

[**2.4. Ứng dụng thực tiễn 10**](#_Toc165691830)

[**3. TRÒ CHƠI SUDOKU 11**](#_Toc165691831)

[**3.1. Giới thiệu trò chơi 11**](#_Toc165691832)

[**3.2. Sử dụng AC-3 giải quyết bài toán Sudoku 12**](#_Toc165691833)

[**3.3. Demo thuật toán giải Sudoku 13**](#_Toc165691834)

[**4. KẾT LUẬN 16**](#_Toc165691835)

[**5. TÀI LIỆU THAM KHẢO 18**](#_Toc165691836)

# 1. CONSTRAINT PROPAGATION

Constraint Propagation (lan truyền ràng buộc) là một phương pháp trọng yếu trong việc giải quyết các Bài toán Ràng Buộc (Constraint Satisfaction Problems - CSPs). Constraint Propagation là quá trình triệt tiêu các giá trị không khả thi cho các biến dựa trên các ràng buộc đã được định nghĩa. Khi một biến được thử nghiệm gán một giá trị mới, thuật toán sẽ tự động cập nhật miền giá trị của các biến còn lại, dựa trên mối quan hệ ràng buộc giữa chúng.

Constraint Propagation có thể được xem như một bước tiền xử lý, nơi mà kết quả là việc thu hẹp miền giá trị của các biến, từ đó tạo điều kiện thuận lợi cho các thuật toán tìm kiếm giải quyết CSPs một cách nhanh chóng và hiệu quả hơn. Trong một số trường hợp, Constraint Propagation thậm chí có thể tìm ra lời giải cho bài toán CSPs mà không cần phải áp dụng thêm bất kỳ giải thuật tìm kiếm nào khác.

## 1.1. LOCAL CONSISTENCY

### 1.1.1. Local consistency là gì?

Hãy tưởng tượng bài toán ràng buộc (CSP) như một bức tranh đồ thị, với mỗi biến được đại diện bởi một đỉnh, và mỗi ràng buộc nhị phân giữa hai biến được biểu diễn bằng một cung. Sự nhất quán cục bộ được định nghĩa là trạng thái mà trong đó một tập hợp con của các đỉnh có thể được gán giá trị sao cho tất cả các ràng buộc giữa chúng đều được thỏa mãn.

Constraint Propagation là một kỹ thuật được sử dụng để duy trì sự nhất quán cục bộ giữa các đỉnh. Giúp loại bỏ các giá trị không thể tồn tại trong lời giải cuối cùng, qua đó giảm thiểu không gian tìm kiếm. Có bốn loại nhất quán cục bộ chính được xem xét trong CSP: Node consistency, arc consistency, Path Consistency, K – Consistency.

### 1.1.2. Node consistency

Kỹ thuật nhất quán đơn giản nhất được gọi là nhất quán nút (node consistency) loại bỏ các giá trị không nhất quán với các ràng buộc đơn ngữ khỏi miền biến. Thuật toán nhất quán nút khá đơn giản: nó đi qua các biến và loại bỏ các giá trị không nhất quán với bất kỳ ràng buộc đơn nào trên biến [1].

Ví dụ: Cho một biến x với miền giá trị K = {2, 3, 4, 5, 6, 7, 8, 9}, và biến x phải là một số nguyên tố. Mục tiêu của thuật toán là loại bỏ các giá trị không nhất quán từ miền giá trị của biến x dựa trên ràng buộc về số nguyên tố. Trong trường hợp này, để biến x phải là một số nguyên tố, chúng ta chỉ cần giữ lại các giá trị trong miền giá trị của x là các số nguyên tố, và loại bỏ các giá trị khác. Đầu tiên, chúng ta sẽ kiểm tra từng giá trị trong miền giá trị của biến x. Nếu giá trị đó không phải là số nguyên tố, chúng ta sẽ loại bỏ nó khỏi miền giá trị của x.

Với ví dụ này, miền giá trị ban đầu của x là K = {2, 3, 4, 5, 6, 7, 8, 9}. Sau khi áp dụng thuật toán nhất quán nút, chúng ta sẽ loại bỏ các giá trị không phải là số nguyên tố, và miền giá trị mới của x sẽ là {2, 3, 5, 7}, với các giá trị 4, 6, 8, 9 bị loại bỏ. Bước tiền xử lý này giúp giảm số lượng giá trị trong miền giá trị của biến x, từ đó đơn giản hóa việc thực hiện các giai đoạn giải quyết vấn đề ràng buộc sau này.

### 1.1.3. Arc Consistency

Tính nhất quán cung (Arc Consistency) đảm bảo rằng mọi giá trị trong miền giá trị của một biến ( Xi ) phải thỏa mãn các ràng buộc nhị phân với mọi biến khác ( Xj ), nghĩa là cho mỗi giá trị trong ( di ), tồn tại ít nhất một giá trị trong ( dj ) sao cho cặp giá trị này tuân thủ ràng buộc nhị phân giữa ( Xi ) và ( Xj ). Một bài toán CSP được coi là thỏa mãn tính nhất quán cung khi mọi biến đều nhất quán cung với tất cả các biến khác.

Ví dụ, nếu ta có hai biến X1 và X2 với các miền giá trị là {1, 2, 3} và {2, 3, 4} tương ứng, và có một ràng buộc nhị phân yêu cầu X1 phải nhỏ hơn X2, thì tính nhất quán cung sẽ đảm bảo rằng mỗi giá trị trong miền giá trị của X1 phải có ít nhất một giá trị trong miền giá trị của X2 mà thỏa mãn điều kiện X1 nhỏ hơn X2 [2].

Thông thường, để đạt được tính nhất quán cung, ta lan truyền miền từ một biến sang biến khác và ngược lại, cập nhật các cung trong đồ thị. Điều này được gọi là sửa đổi cung (oriented arc revision). Lặp lại quá trình này cho đến khi không thể thực hiện thêm sửa đổi nào nữa. Ý tưởng này đã được tổng quát hóa để giải quyết CSP tùy ý và được gọi là AC-2. Một phiên bản hiệu quả hơn của thuật toán này, giữ một hàng đợi duy nhất các cung để kiểm tra lại, được gọi là AC-3 (Arc-Consistency 3) và có lẽ là thuật toán duy trì tính nhất quán được sử dụng rộng rãi nhất.

*Giới thiệu thuật toán AC-3 (Arc-Consistency 3), ý tưởng, giải thích và phân tích độ phức tạp*

Thuật toán AC-3, được phát triển bởi Alan Mackworth vào năm 1977, là một phương pháp hiệu quả để duy trì tính nhất quán cung trong CSP. Như đã trình bày ở phần trước, ý tưởng chính của AC-3 là kiểm tra nhất quán cung bằng các sử dụng một hàng đợi các cặp biến, loại bỏ các giá trị không khả thi từ miền giá trị của biến, qua đó giảm không gian tìm kiếm và tăng tốc độ giải quyết bài toán. Khi miền giá trị của một biến bị giảm xuống, tất cả các cặp biến có ràng buộc với biến đó sẽ được thêm vào hàng đợi để kiểm tra lại. Nếu miền giá trị của một biến trở nên rỗng, điều này chỉ ra rằng không tồn tại giải pháp nhất quán cho CSP. Ngược lại, nếu không còn cặp biến nào trong hàng đợi, CSP được coi là đã được giải quyết một phần.

Mã giả thuật toán AC-3:

A screenshot of a computer code

Description automatically generated

Hình 1. Mã giả thuật toán AC-3

Giải thích mã giả:

Bước 1: Khởi tạo hàng đợi (queue) ban đầu chứa tất cả các cung (arc) của CSP. Mỗi cung là một cặp biến (Xi, Xj) có ràng buộc với nhau.

Bước 2: Kiểm tra xem hàng đợi có rỗng không. Nếu không rỗng, lấy ra cung ở đầu hàng đợi và loại bỏ nó khỏi hàng đợi để bắt đầu kiểm tra. Giả sử cung này nối hai biến Xi và Xj.

Bước 3: Gọi hàm Revise để kiểm tra xem liệu có cần loại bỏ giá trị nào từ miền giá trị của Xi để đảm bảo tính nhất quán với Xj không. Với mỗi giá trị x trong miền giá trị của Xi (Di), kiểm tra xem có tồn tại giá trị y nào trong miền giá trị của Xj (Dj) thỏa mãn ràng buộc giữa Xi và Xj không. Nếu không tồn tại giá trị y nào thỏa mãn, hàm Revise trả về True biểu thị đã loại bỏ giá trị x khỏi Di.

Bước 4: Nếu hàm Revise đã loại bỏ ít nhất một giá trị từ Di, kiểm tra xem Di có rỗng không. Nếu Di rỗng, thuật toán kết thúc. Nếu Di không rỗng, đẩy tất cả các cung liên quan đến Xi (ngoại trừ cung {Xi, Xj}) vào hàng đợi. Các cung này có dạng (Xk, Xi), với Xk là các biến lân cận của Xi.

Bước 5: Lặp lại từ Bước 2 cho đến khi hàng đợi rỗng. Khi hàng đợi rỗng, điều này có nghĩa là tất cả các cung đã được kiểm tra tính nhất quán và không còn giá trị nào cần phải loại bỏ nữa. Thuật toán kết thúc.

*Độ phức tạp của giải thuật AC-3*

Trong trường hợp tồi tệ nhất, độ phức tạp của giải thuật AC-3 bằng O(ed3) [2].

Với (d) là miền giá trị tối đa và e là số cung của bài toán CSP. Độ phức tạp này bắt nguồn từ việc giải thuật AC-3 cần kiểm tra tính nhất quán cho mỗi cặp biến liên kết với một ràng buộc. Trong quá trình này, giải thuật AC-3 có thể phải xem xét tất cả các giá trị có thể của mỗi biến trong mỗi cặp, và do đó có thể phải thực hiện tối đa d2.Vì mỗi ràng buộc có thể liên quan đến hai biến, số lượng cung e trong mô hình ràng buộc có thể được coi là hai lần số lượng ràng buộc. Do đó, số lượng kiểm tra tổng cộng có thể lên đến O(ed2) .

Tuy nhiên, trong trường hợp xấu nhất, giải thuật AC-3 có thể cần thực hiện kiểm tra nhiều lần cho cùng một cặp biến. Cụ thể, nếu một giá trị bị loại bỏ khỏi miền giá trị của một biến, tất cả các cặp biến liên quan đến biến đó cần được kiểm tra lại. Do đó, trong trường hợp xấu nhất, mỗi cặp biến có thể cần được kiểm tra lại ( d ) lần, dẫn đến độ phức tạp thời gian là O(ed3).

*Bài toán minh họa*

Giả sử chúng ta có một bài toán ràng buộc với ba biến (A, B, C), với biến A, C có miền giá trị là {1, 2, 3} và B có miền giá trị là {1, 3}. Các ràng buộc được định nghĩa như sau: (A > B), (A > C) và (B < C).

A diagram of a network

Description automatically generated

Hình 2: Ảnh minh họa bài toán ràng buộc giữa ba biến A,B và C

Đầu tiên khởi tạo hàng đợi của chúng ta sẽ chứa: ( (A, B), (A, C), (B, C) ).

Xét cặp ( A, B): kiểm tra các giá trị của A = {1,2,3} ta thấy A = 1 không thể thỏa mãn (A > B), loại bỏ giá trị 1 khỏi miền giá trị của A, lúc này miền giá trị của A sẽ là A = {2, 3}, thêm các ràng buộc liên quan đến A vào cuối hàng đợi ( B, A), ( C, A)

A diagram of a network

Description automatically generated

Hình 3: Ảnh minh họa bài toán sau khi xét cặp ràng buộc (A, B)

Xét cặp (A, C): kiểm tra các giá trị của A = {2,3}, ta có A{2} > C{1} và A{3}> C{2}, A{3} > C{1} nên ràng buộc nhất quán.

Xét cặp (B, C): giá trị B{3} không thể bé hơn bất kì một giá trị nào trong miền giá trị của C, loại bỏ B{3} khỏi miền giá trị của B. Thêm vào hàng đợi các ràng buộc liên quan đến B: (A, B), (C, B).

Xét tiếp cặp (B, A): kiểm tra miền giá trị của B ={1}, ta thấy B{1} < A{2} nên ràng buộc nhất quán.

Tương tự quá trình sẽ duyệt qua hết hàng đợi. Khi hàng đợi không còn ràng buộc nào cần duyệt, thuật toán kết thúc, kết quả trả về A = {3}, B = {1}, C = {2}.

A diagram of a network

Description automatically generated

Hình 4: Ảnh minh họa kết quả bài toán ràng buộc giữa ba biến A,B và C

### 1.1.4. Path Consistency

Kỹ thuật hoạt động của Path Consistency (PC) dựa trên ý tưởng thu hẹp không gian biến bằng cách loại bỏ các giá trị không tương thích với các ràng buộc ẩn. Đây là một dạng của sự nhất quán cục bộ, nằm giữa nhất quán nút (node-consistency) và nhất quán cung (arc-consistency).

Path Consistency làm việc bằng cách xem xét mọi cặp biến và một biến thứ ba, kiểm tra xem có tồn tại một giá trị cho biến thứ ba sao cho mọi giá trị có thể của cặp biến đầu tiên đều thỏa mãn ràng buộc với biến thứ ba hay không. Nếu không tồn tại giá trị như vậy, giá trị không nhất quán sẽ bị loại bỏ khỏi miền của cặp biến.

Để đạt được Path Consistency, chúng ta cần thực hiện thuật toán PC trên mọi bộ ba biến (triple) trong mạng ràng buộc. Đối với mỗi bộ ba biến (X, Y, Z), chúng ta sẽ xem xét tất cả các cặp giá trị có thể (x, y) thuộc miền của X và Y. Nếu không có giá trị z nào trong miền của Z sao cho cả ba giá trị (x, y, z) đều thỏa mãn các ràng buộc giữa chúng, thì cặp giá trị (x, y) sẽ bị loại bỏ khỏi miền xem xét. Quá trình này được lặp lại cho đến khi không còn có thể loại bỏ thêm giá trị nào nữa, hoặc khi một miền của biến nào đó trở nên rỗng, điều này chỉ ra rằng mạng ràng buộc không thể giải được. PC giúp giảm thiểu số lượng kiểm tra ràng buộc trong quá trình giải quyết vấn đề và có thể dẫn đến việc phát hiện sớm các vấn đề không thể giải quyết, từ đó tiết kiệm thời gian và nguồn lực tính toán.

### 1.1.5. K – Consistency

K-Consistency là một khái niệm mở rộng của Path Consistency, áp dụng cho các nhóm biến có kích thước lớn hơn. Trong khi Path Consistency tập trung vào các bộ ba biến, K-Consistency có thể áp dụng cho bất kỳ nhóm nào gồm K biến [1]. K-Consistency thường được sử dụng trong các vấn đề có không gian tìm kiếm lớn hơn, nơi mà việc thu nhỏ miền tìm kiếm là thiết thực, giúp giải quyết vấn đề hiệu quả hơn.

Để đạt được K-Consistency, chúng ta cần xem xét mọi nhóm của K biến trong mạng ràng buộc và đảm bảo rằng, mọi nhóm con gồm K-1 biến, tồn tại ít nhất một giá trị trong miền của biến còn lại sao cho tất cả các ràng buộc liên quan đến nhóm con này đều được thỏa mãn [1]. Nếu điều này không đúng, loại bỏ các giá trị không nhất quán.

Với K bằng 2, chúng ta có Arc-Consistency. Với K bằng 3, 3-consistency tương đương với Path Consistency. Khi K tăng lên, mức độ của sự nhất quán cũng tăng lên, nhưng đồng thời, độ phức tạp tính toán cũng tăng lên đáng kể.

## 1.2. GLOBAL CONSTRAINT

### 1.2.1. Global Constraint là gì?

Global Constraint hay ràng buộc toàn cục là một khái niệm quan trọng trong lĩnh vực của Constraint Satisfaction Problems (CSPs), giúp tối ưu hóa quá trình giải quyết CSPs bằng cách giảm không gian tìm kiếm và thời gian tính toán. Khác với các ràng buộc cục bộ chỉ tác động lên một hoặc hai biến, Global Constraints tác động lên một tập hợp lớn các biến và thường biểu diễn một mẫu ràng buộc phức tạp hoặc một quy tắc chung mà nhiều biến cần tuân theo.

Nói cách khác, Global Constraint xem xét toàn bộ bài toán thay vì chỉ các biến cục bộ để suy luận về các giá trị khả thi. Điều này có thể dẫn đến việc loại bỏ nhiều giá trị không hợp lệ hơn so với các ràng buộc thông thường, tuy nhiên cũng thường phức tạp hơn để thiết lập và thực thi. Tuy là một công cụ mạnh mẽ trong Constraint Propagation nhưng nên được sử dụng thận trọng vì độ phức tạp của nó.

Trong lĩnh vực tối ưu hóa và lý thuyết đồ thị, Global Constraints đóng một vai trò quan trọng trong việc mô hình hóa và giải quyết các bài toán phức tạp. Chúng ta có thể xem xét trò chơi Sudoku. Trong trò chơi này, bạn có một lưới 9x9 được chia thành các ô nhỏ. Mục tiêu là điền các số từ 1 đến 9 vào lưới sao cho mỗi số chỉ xuất hiện một lần trong mỗi hàng, mỗi cột, và mỗi khối 3x3 không có số nào xuất hiện quá một lần.

Một ví dụ cho Global Constraints, bài toán cổ điển: Bài toán Người Bán Hàng Đi Lạc (Traveling Salesman Problem). Ở bài toán này, mục tiêu là tìm ra lộ trình ngắn nhất qua một số thành phố, sao cho mỗi thành phố chỉ được thăm một lần và người bán hàng kết thúc hành trình tại thành phố xuất phát. Đây là một ví dụ điển hình của Global Constraint có ràng buộc AllDifferent.

### 1.2.2. AllDifferent Constraints

AllDifferent Constraints (hoặc còn gọi là ràng buộc khác nhau) là một khái niệm quan trọng trong lĩnh vực Constraint Satisfaction Problems (CSPs). Điều này đảm bảo rằng tất cả các biến trong một nhóm cụ thể phải có giá trị khác nhau.

Quay lại với Bài toán trò chơi Sudoku. AllDifferent Constraints được áp dụng cho mỗi hàng, mỗi cột và mỗi khối 3x3 để đảm bảo rằng không có số nào lặp lại trong cùng một hàng, cột hoặc khối**.**

### 1.2.3. Resource Constraint

Cùng với AllDifferent Constraints, Resource Constraint (Ràng Buộc Tài Nguyên) là một loại ràng buộc toàn cục khác trong lĩnh vực giải quyết bài toán thỏa mãn ràng buộc (CSP). Ràng buộc này áp đặt giới hạn cho việc sử dụng tài nguyên chung giữa các biến số. Mục đích là đảm bảo tổng nhu cầu do các giá trị được gán trên các biến không vượt quá năng lực tài nguyên sẵn có.

Hãy tưởng tượng một bài toán lập lịch đơn giản sau, bạn có một hệ thống gồm một tập các tài nguyên (R1, R2, R3, R4) và các tiến trình (A, B, C). Mỗi tiếng trình cần một số lượng tài nguyên nhất định để thực thi.

|  |  |
| --- | --- |
| A diagram of a network  Description automatically generated | Tiến trình A cần tài nguyên R1, R3 để thực thi và đang yêu cầu tài nguyên R1 và R3.  Tiến trình B cần cả bốn loại tài nguyên, đang được cấp R1, R2, R3 và cần thêm R4  Tiến trình C cần tài nguyên R2, R4. Và đang được cấp tài nguyên R2, R4 |

Hình 5: Ảnh minh họa Đồ thị cấp phát tài nguyên

Ở ví dụ này, ta thấy tài nguyên R3 đang phải cung cấp cho hai tiến trình A và B. Một giai đoạn nào đó, các tiến trình A, B cùng yêu cầu một tài nguyên R3. Lúc này Resource Constraint sẽ đảm bảo tìm ra lộ trình phân bổ tài nguyên một cách hiệu quả sao cho không dẫn đến tắc nghẽn chương trình.

Resource Constraint giúp tối ưu hóa việc sử dụng tài nguyên, đồng thời giảm thiểu tắc nghẽn và đảm bảo thực hiện dự án suôn sẻ. tránh lãng phí và tối ưu hóa hiệu suất chương trình. Thường được sử dụng rộng rãi trong nhiều bài toán thỏa mãn ràng buộc như: Lập lịch (nhiệm vụ, cuộc họp, tài nguyên), Phân bổ tài nguyên (ngân sách, nhân sự), Đóng gói thùng (lắp các vật phẩm vào thùng chứa), Lập kế hoạch dự án (phụ thuộc lẫn nhau, khả dụng tài nguyên).

# 2. ƯU ĐIỂM, NHƯỢC ĐIỂM VÀ NHỮNG ỨNG DỤNG CỦA CONSTRAINT PROPAGATION

## 2.1. Ưu điểm

Tối ưu hóa không gian tìm kiếm: Như đã đề cập ở phần một, Constraint Propagation giúp giảm không gian tìm kiếm bằng cách loại bỏ các giá trị không hợp lệ, từ đó tăng hiệu suất giải quyết bài toán.

Biểu diễn quan hệ phức tạp: Global Constraints cho phép biểu diễn các quan hệ phức tạp giữa các biến một cách hiệu quả, giảm độ phức tạp của mô hình bài toán CSP.

Tăng cường hiệu quả giải quyết: Local Consistency giúp tăng cường hiệu quả giải quyết bài toán bằng cách đảm bảo mỗi phần của bài toán đều nhất quán với nhau.

Phát hiện sớm lỗi: Cả hai phương pháp Global Constraints và Local Consistency đều giúp phát hiện và loại bỏ sớm các giải pháp không hợp lệ, giảm thiểu thời gian và công sức không cần thiết.

## 2.2. Nhược điểm

Khó khăn trong việc thiết kế và triển khai: Việc thiết kế và xây dựng Constraint Propagation có thể phức tạp và tốn thời gian, đặc biệt đối với nhiều ràng buộc phức tạp.

Một số bài toán không thể mô hình hóa hiệu quả bằng Global Constraints, đặc biệt khi có sự phụ thuộc giữa các biến không thể mô tả qua các ràng buộc này.

Tốn bộ nhớ và tài nguyên tính toán: Local Consistency có thể đòi hỏi nhiều bộ nhớ và tài nguyên tính toán để duy trì nhất quán giữa các biến và ràng buộc.

Không phù hợp với mọi loại bài toán: Cả hai phương pháp đều có thể không phù hợp với một số loại bài toán phức tạp, đặc biệt là khi số lượng biến và ràng buộc lớn.

## 2.3. So sánh Global Constraints với Local Consistency

Tuy cùng thực hiện một nhiệm vụ, nhưng Global Constraints và Local Consistency lại có các hướng tiếp cận riêng, do đó điểm mạnh yếu là khác nhau. Vậy nên cần phải xem xét bài toán CSP mà đưa ra giải pháp hợp lý.

Global Constraints thường được áp dụng cho một tập lớn các biến và dùng để giải quyết các quan hệ phức tạp, trong khi Local Consistency thường tập trung vào việc duy trì nhất quán giữa các cặp biến hoặc nhóm nhỏ các biến. Cả hai đều giúp giảm không gian tìm kiếm nhưng Global Constraints có thể phức tạp hơn để thiết kế và triển khai so với Local Consistency, đặc biệt khi đối mặt với nhiều ràng buộc phức tạp. Song Local Consistency lại có thể đòi hỏi nhiều tài nguyên tính toán hơn Global Constraints.

## 2.4. Ứng dụng thực tiễn

Phương pháp Lan Truyền Ràng Buộc (Constraint Propagation) có nhiều ứng dụng thực tiễn trong các lĩnh vực khác nhau. Dưới đây là một số ứng dụng cụ thể:

Lập lịch: Trong lĩnh vực sản xuất và dịch vụ, Constraint Propagation được sử dụng để lập lịch sản xuất, phân bổ nguồn lực, và quản lý dự án, giúp tối ưu hóa việc sử dụng nguồn lực và thời gian.

Hệ thống kiểm soát thời gian thực: Trong các hệ thống tự động và kiểm soát, Constraint Propagation giúp đảm bảo rằng các hành động được thực hiện đúng thời gian và tuân theo các ràng buộc về thời gian và nguồn lực.

Xử lý ngôn ngữ tự nhiên (NLP): Constraint Propagation có thể sử dụng để kiểm tra tính nhất quán của các ràng buộc về ngữ pháp, ngữ nghĩa hoặc logic trong văn bản.

Giao diện người dùng đồ họa: Trong thiết kế giao diện người dùng, Constraint Propagation được sử dụng để đảm bảo rằng các yếu tố giao diện tuân theo các ràng buộc về kích thước, vị trí, và mối quan hệ với các yếu tố khác.

Phân đoạn hình ảnh dựa trên ràng buộc: Việc phân đoạn hình ảnh thành các vùng có chất lượng khác nhau (chẳng hạn như màu sắc, kết cấu hoặc hình dạng) có thể được coi là vấn đề CSP trong thị giác máy tính, trong đó các biến đại diện cho các vùng và các ràng buộc xác định mức độ tương tự hoặc không giống như các khu vực lân cận với nhau.[3]

Sudoku: Trò chơi giải đố nổi tiếng Sudoku có thể được mô hình hóa như một bài toán CSP, trong đó các biến đại diện cho các ô của lưới và các ràng buộc xác định quy tắc của trò chơi, chẳng hạn như cấm lặp lại cùng một số trong một hàng, cột hoặc khu vực.[3]

# 3. TRÒ CHƠI SUDOKU

## 3.1. Giới thiệu trò chơi

Sudoku là một trò chơi giải đố logic sử dụng các con số vô cùng phổ biến và được yêu thích trên toàn thế giới, có đa dạng câu đố từ dễ đến khó. Trò chơi này không đòi hỏi kiến thức toán học cao siêu mà chỉ cần sự tập trung, tư duy logic và óc suy luận nhạy bén để giải mã những ô số bí ẩn.

Sudoku bao gồm một lưới 9x9, được chia thành 9 hình vuông nhỏ 3x3. Mục tiêu của trò chơi là điền các số từ 1 đến 9 vào lưới sao cho mỗi hàng, mỗi cột và mỗi hình vuông 3x3 đều chứa tất cả các số từ 1 đến 9 mà không lặp lại.

|  |  |
| --- | --- |
| Câu đó Sudoku | Giải pháp |
| A sudoku game with numbers and letters  Description automatically generated | A square of numbers with black and orange squares  Description automatically generated with medium confidence |

Hình 6: Ảnh câu đố Sudoku và giải pháp

Được rồi, ở đây chúng tôi không đủ nghị lực để giải mã một ván chơi Sudoku. Hãy để Constraint Propagation giúp chúng ta làm điều đó.

## 3.2. Sử dụng AC-3 giải quyết bài toán Sudoku

Trò chơi Sudoku có thể được xem như một bài toán CSP có 81 biến số tương đường với 81 ô trên bảng Sudoku. Với mỗi biến số có một miền giá trị từ 1 đến 9. Đối với một bảng Sudoku tiêu chuẩn 9x9, có tổng cộng: 9 AllDifferent Constraints cho các hàng, 9 AllDifferent Constraints cho các cột và 9 AllDifferent Constraints cho các hình vuông 3x3. Thuật toán AC-3 có thể được sử dụng để chuyển đổi các ràng buộc Alldiff thành các ràng buộc nhị phân [4]. Ví dụ, một ràng buộc Alldiff cho một hàng hoặc một cột có thể được chia nhỏ thành các ràng buộc Diff giữa mỗi cặp biến số trong hàng hoặc cột đó. Một ràng buộc Alldiff có thể được biểu diễn như sau:



Mỗi ràng buộc Alldiff có thể được chuyển đổi thành 36 ràng buộc Diff (A,B), và với 27 Alldiff constraints trong một bảng Sudoku, điều này dẫn đến việc có tổng cộng 972 ràng buộc Diff (27 \* 36). Tuy nhiên, do mỗi cặp biến số có hai hướng ràng buộc (ví dụ: Diff(A,B) và Diff(B,A) là khác nhau), số lượng ràng buộc thực tế cần xem xét là 1944 (27\*36\*2).

Thuật toán AC-3 sẽ lặp đi lặp lại qua các cặp biến số này để đảm bảo tính nhất quán của cung, loại bỏ những giá trị không thỏa mãn từ miền giá trị của các biến số. Quá trình này tiếp tục cho đến khi không còn giá trị nào có thể loại bỏ hoặc khi bảng Sudoku được giải quyết hoàn toàn.

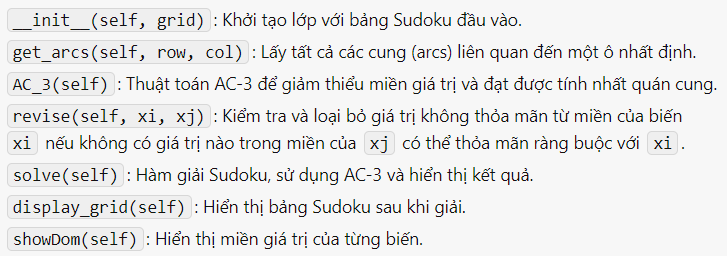
## 3.3. Demo thuật toán giải Sudoku

Sau đây là phần demo thuật toán giải Sudoku sử dụng thuật toán AC-3. Chương trình này sẽ giải quyết trò chơi Sudoku với bàn cờ tiêu chuẩn 9x9, dựa trên ý tưởng giảm thiểu miền giá trị của 81 biến số.

**Input**: Bản đồ vị trí các biến của Sudoku, với các biến có khoảng giá trị từ 1 đến 9. Biến 0 đại diện cho vị trí ô trống.

**Output**: Bản Sudoku sau khi đã giải, hoặc thông báo không thể giải được nếu thuật toán AC-3 không tìm ra lời giải.

Các hàm được định nghĩa trong Class SudokuSolver:



Giải thích Chương trình:

*def \_\_init\_\_(self, grid)*: Hàm khởi tạo của lớp SudokuSolver. Nó được thiết kế để nhận vào một ma trận trò chơi Sudoku và thực hiện các công việc lưu bảng Sudoku vào biến *self.grid*. Tính kích thước của bảng (*self.size*) và kích thước của mỗi khối trong Sudoku (*self.box\_size*).

Đồng thời chịu trách nhiệm khởi tạo miền giá trị (*self.domains*) cho mỗi ô trên bảng. Nếu ô đó đã có số, miền giá trị chỉ chứa số đó; nếu ô trống, miền giá trị sẽ là danh sách các số từ 1 đến 9. Tạo danh sách các cung (*self.arcs*) giữa các ô chưa được điền số, dựa trên các hàng, cột và hộp nhỏ. Danh sách cung *self.arcs* được xem là hạt nhân của của kĩ thuật AC-3.

|  |
| --- |
| def \_\_init\_\_(self, grid):          self.grid = grid          self.size = len(grid)          self.box\_size = int(self.size\*\*0.5)          self.domains = {              (r, c): list(range(1, self.size + 1)) if grid[r][c] == 0 else [grid[r][c]]              for r in range(self.size)              for c in range(self.size)          }          self.arcs = []          for i in range(self.size):              for j in range(self.size):                  if self.grid[i][j] == 0:                      self.arcs.extend(self.get\_arcs(i, j))                      self.arcs.extend(self.get\_arcs(j, i)) |

Hàm thứ hai, get\_arcs(self, row, col): Hàm này nhận vào một biến và trả về danh sách các cung liên quan đến biến (row, col). Một cung được định nghĩa là một cặp biến mà giữa chúng có ràng buộc. Ở đây là ràng buộc không được có số trùng nhau trong cùng một hàng, một cột hay một khối hộp 3x3 chứa vị trí chỉ định.

|  |
| --- |
| def get\_arcs(self, row, col):          arcs = []          for i in range(self.size):              if i != col:                  arcs.append(((row, col), (row, i)))          for j in range(self.size):              if j != row:                  arcs.append(((row, col), (j, col)))          start\_row, start\_col = self.box\_size \* (row // self.box\_size), self.box\_size \* (              col // self.box\_size          )          for i in range(start\_row, start\_row + self.box\_size):              for j in range(start\_col, start\_col + self.box\_size):                  if (i, j) != (row, col):                      arcs.append(((row, col), (i, j)))          return arcs |

*revise(self, xi, xj):* Hàm này kiểm tra và loại bỏ các giá trị không thỏa mãn từ miền của biến xi nếu không có giá trị nào trong miền của xj có thể thỏa mãn ràng buộc với xi. Nếu có sự thay đổi trong miền giá trị, hàm sẽ trả về True.

|  |
| --- |
| def revise(self, xi, xj):          revised = False          for x in self.domains[xi]:              if not any(x != y for y in self.domains[xj]):                  self.domains[xi].remove(x)                  revised = True          return revised |

Hàm *AC\_3(self)* thực hiện thuật toán AC-3 để đạt được tính nhất quán cung. Nó sẽ lặp qua danh sách các cung trong self.arcs và sử dụng hàm revise để giảm thiểu miền giá trị. Nếu miền giá trị của một biến nào đó trở nên rỗng, hàm sẽ trả về False, báo hiệu rằng bảng Sudoku không thể giải được. Nếu không, hàm sẽ trả về True khi tất cả các cung đều nhất quán.

|  |
| --- |
| def AC\_3(self):          while self.arcs:              (xi, xj) = self.arcs.pop(0)              if self.revise(xi, xj):                  if not self.domains[xi]:                      return False                  for xk in self.get\_arcs(\*xi):                      self.arcs.append((xk[1], xk[0]))          return True |

solve(self): Hàm này chịu trách nhiệm gọi AC\_3 để giải ma trận Sudoku. Nếu AC\_3 trả về True, hàm sẽ hiển thị bảng Sudoku đã giải. Nếu không, hàm sẽ thông báo rằng không thể giải được Sudoku bằng giải thuật AC-3 và gọi self.showDom để đưa ra miền giá trị hiện tại của các biến. Solve(self) hoàn toàn có thể được cải tiến, kết hợp với những giải thuật tìm kiếm như Backtracking tăng hiệu quả chương trình.

|  |
| --- |
| def solve(self):          if self.AC\_3():              print("Sudoku giải được:")              self.display\_grid()          else:              print("Kỹ thuật AC\_3 chưa giải được Sudoku này.")              self.showDom() |

Hàm *display\_grid(self)* được dùng để hiển thị bảng Sudoku. Trong khi đó, hàm *showDom(self)* sẽ hiển thị miền giá trị của từng ô trên bảng Sudoku. Điều này hữu ích để debug và xem quá trình giảm thiểu miền giá trị diễn ra như thế nào.

|  |
| --- |
| def display\_grid(self):          for r in range(self.size):              for c in range(self.size):                  if len(self.domains[(r, c)]) == 1:                      print(f"{self.domains[(r, c)][0]} ", end="")                  else:                      print("0 ", end="")              print()      def showDom(self):          for i in range(self.size):              for j in range(self.size):                  print(f"({i}, {j}) dom: {self.domains[(i, j)]}") |

Kết quả của Demo:

|  |  |
| --- | --- |
|  |  |

Hình 7: Ma trận Sudoku và giải pháp đưa ra

Sau khi thực hiện demo thuật toán AC-3 để giải Sudoku cho thấy khả năng giảm thiểu miền giá trị và đạt tính nhất quán là có hiệu quả.

Tuy nhiên, thuật toán AC-3 có thể không đủ để giải quyết hoàn toàn bài toán Sudoku vì nó chỉ loại bỏ những giá trị không thỏa mãn ràng buộc mà không cố gắng điền các giá trị vào bảng Sudoku. Do đó, một sự kết hợp giữa AC-3 với các phương pháp tìm kiếm khác sẽ mang lại hiệu quả tốt nhất.

# 4. KẾT LUẬN

Constraint Propagation là một phương pháp mạnh mẽ trong giải các bài toán ràng buộc (CSPs), giúp giải quyết các vấn đề phức tạp một cách hiệu quả. Phương pháp này loại bỏ các giá trị không phù hợp và thu hẹp không gian tìm kiếm, giúp tiếp cận nhanh chóng với giải pháp đúng đắn. Constraint Propagation được điều khiển bằng các thuật toán thông minh, đảm bảo sự nhất quán giữa các biến và ràng buộc, từ đó tạo ra hiệu quả giải quyết tốt. Phân loại của Constraint Propagation linh hoạt, từ các cấp độ đơn giản như Node Consistency đến Global Constraint, phù hợp với nhiều loại bài toán khác nhau. Mỗi cấp độ mang lại các lợi ích và hạn chế riêng, phù hợp với từng tình huống cụ thể.

Ưu điểm của Constraint Propagation bao gồm tối ưu hóa không gian tìm kiếm, giảm thiểu thời gian giải quyết, phát hiện lỗi sớm và giải quyết các bài toán phức tạp một cách hiệu quả.

Tuy nhiên, cần lưu ý rằng Constraint Propagation có thể gặp khó khăn trong việc triển khai cho các bài toán phức tạp, đòi hỏi nhiều tài nguyên tính toán, thời gian thực hiện lớn và không phù hợp với mọi loại bài toán.

Phương pháp này được ứng dụng rộng rãi trong nhiều lĩnh vực như lập kế hoạch sản xuất, quản lý dự án, hệ thống kiểm soát thời gian thực, xử lý ngôn ngữ tự nhiên, giao diện người dùng đồ họa và giải quyết các bài toán tổ hợp. Trong trò chơi Sudoku, Constraint Propagation đã chứng minh khả năng giải quyết thông qua việc giải mã các ô số bí ẩn, mang lại niềm vui chiến thắng và khẳng định sức mạnh của trí tuệ nhân tạo.

# 5. TÀI LIỆU THAM KHẢO

**[1] Theory and Practice of Constraint Propagation, J. Figwer, June 2001. Đường dẫn:** [**https://www.cs.nott.ac.uk/~pszrq/files/CP3.pdf**](https://www.cs.nott.ac.uk/~pszrq/files/CP3.pdf)

**[2] AC-3 algorithm, Đường dẫn:** [**https://en.wikipedia.org/wiki/AC-3\_algorithm**](https://en.wikipedia.org/wiki/AC-3_algorithm)

**[3] Constraint Satisfaction Problems (CSP) in Artificial Intelligence. Đường dẫn:** [**https://www.geeksforgeeks.org/constraint-satisfaction-problems-csp-in-artificial-intelligence/**](https://www.geeksforgeeks.org/constraint-satisfaction-problems-csp-in-artificial-intelligence/)

**[4] Arc Consistency Explained. Đường dẫn:** [**https://www.boristhebrave.com/2021/08/30/arc-consistency-explained/**](https://www.boristhebrave.com/2021/08/30/arc-consistency-explained/)