ỨNG DỤNG CỦA LẬP TRÌNH TẬP TRẢ LỜI ĐỂ BIỂU DIỄN VÀ THỰC THI MỘT SỐ BÀI TOÁN CỦA LÝ THUYẾT ĐỒ THỊ

Trương Công Tuấn

Khoa Công nghệ thông tin, Trường Đại học Khoa học, Đại học Huế

Email: tctuan@husc.edu.vn

Ngày nhận bài: 6/3/2024; ngày hoàn thành phản biện: 7/3/2024; ngày duyệt đăng: 7/3/2024

TÓM TẮT

Trong các thập kỷ đã qua, việc nghiên cứu lập trình logic đã đạt được nhiều thành tựu quan trọng cả về lý thuyết và ứng dụng. Lập trình logic được áp dụng rộng rãi để biểu diễn tri thức và xử lý thông tin trong nhiều lĩnh vực. Hiện nay, các chương trình logic với ngữ nghĩa tập trả lời đã được đưa ra như một mô hình mới cho việc áp dụng các kỹ thuật lập trình khai báo để biểu diễn tri thức. Trong bài báo này, chúng tôi trình bày ứng dụng của lập trình tập trả lời để biểu diễn và thực thi một số các bài toán của lý thuyết đồ thị.

Từ khóa: Lập trình logic, tập trả lời, ngữ nghĩa tập trả lời.

1. MỞ ĐẦU

Trong hơn ba thập kỷ qua, việc nghiên cứu lập trình logic (LP - Logic Programming) đã đạt được nhiều thành tựu quan trọng cả về lý thuyết và ứng dụng để biểu diễn tri thức [1, 3, 5, 11, 13]. Lập trình logic là một dạng lập trình khai báo, nó là một mẫu hình lập trình dựa trên logic toán học trong các mối quan hệ và các suy luận, cho phép biểu diễn các bài toán trong nhiều lĩnh vực bằng các chương trình logic và đưa ra các cơ chế lập luận suy diễn rất hiệu quả. Các chương trình được viết trong các ngôn ngữ lập trình logic là các tập hợp quy tắc logic, biểu diễn các sự kiện và các quy tắc về một vấn đề nào đó trong thực tế và được thực thi bởi các hệ thống lập trình logic, chẳng hạn DLV, SWI-Prolog,…

Lập trình logic là một thành phần của lĩnh vực rộng lớn là Trí tuệ nhân tạo (AI). Trong số các mô hình khác nhau của nó, lập trình tập trả lời (ASP – Answer Set Programming) đã nổi lên như một cách tiếp cận hấp dẫn và mạnh mẽ [4, 9, 13, 14]. Các chương trình logic với ngữ nghĩa tập trả lời đã được đưa ra như một mô hình mới cho việc áp dụng các kỹ thuật lập trình khai báo. Trong tiếp cận này, một bài toán sẽ được giải quyết bằng cách tạo ra một chương trình logic mà các tập trả lời của chương trình

cung cấp câu trả lời cho bài toán. ASP là một phương pháp lập trình hướng tới các bài toán tìm kiếm tổ hợp. Trong một bài toán như vậy, mục tiêu là tìm ra lời giải giữa một số lượng lớn nhưng hữu hạn các khả năng. Các bài toán tìm kiếm tổ hợp thường phát sinh trong khoa học và công nghệ, và ASP đã tìm thấy các ứng dụng trong các lĩnh vực đa dạng – trong tin sinh học, trong robot, trong khám phá không gian, trong ngành công nghiệp dầu khí và nhiều lĩnh vực khác. Logic là một thành phần thiết yếu của AI và ASP có ý nghĩa sâu sắc trong lĩnh vực này. Khả năng xử lý lý luận không đơn điệu, thông tin không chắc chắn và các vấn đề phức tạp, ASP góp phần vào tầm quan trọng của nó trong AI. ASP đã được Hiệp hội vì sự tiến bộ của Trí tuệ nhân tạo – Tạp chí về Trí tuệ nhân tạo công nhận vào năm 2016, khi Tạp chí AI này xuất bản một số đặc biệt về lập trình tập trả lời [15]. Hiện nay, ASP đã và đang được tiếp tục nghiên cứu phát triển mở rộng, các chương trình logic trong ASP có thể bao gồm nhiều dạng quy tắc khác nhau, chẳng hạn các quy tắc có chứa bản số và trọng số [6, 8], các quy tắc có ràng buộc [2, 10, 14],…

Bài báo này nhằm mục đích trình bày một số ứng dụng của lập trình tập trả lời trong việc biểu diễn tri thức thông qua một số bài toán của lý thuyết đồ thị. Phần còn lại của bài báo được tổ chức như sau, phần 2 trình bày về cú pháp của các dạng quy tắc của chương trình logic cùng với ngữ nghĩa tập trả lời. Phần 3 trình bày một số bài toán để minh họa việc ứng dụng lập trình logic trong việc biểu diễn tri thức. Cuối cùng, kết luận được nêu trong phần 4.

2. CÚ PHÁP VÀ NGỮ NGHĨA TẬP TRẢ LỜI CỦA CHƯƠNG TRÌNH LOGIC

Phần này chỉ trình bày tóm tắt cú pháp cần thiết, chi tiết đầy đủ hơn có thể xem trong [1, 7, 9].

2.1. Cú pháp các quy tắc

Định nghĩa 2.1 (Quy tắc) [1] Một *quy tắc* là công thức có dạng:

$$p \leftarrow q_1, \ldots, q_m, \ldots, not\ q_{m+1}, \ldots, not\ q_n \qquad (n \geq m \geq 0) \qquad (1)$$

trong đó p, q_i là các nguyên tố có đối là hằng hoặc biến, p được gọi là phần đầu và $q_1, \ldots, q_m, \ldots, not\ q_{m+1}, \ldots, not\ q_n$ là phần thân của quy tắc. Dấu phẩy trong phần thân quy tắc là ký hiệu thay cho phép hội ($\wedge$). *Quy tắc nền* là quy tắc không chứa biến.

Ý nghĩa của quy tắc có dạng (1) là nếu thân quy tắc nhận giá trị *true* thì đầu sẽ nhận giá trị *true*. Quy tắc với phần thân rỗng:

$$p \leftarrow$$

sẽ được gọi là *sự kiện*. Ký hiệu mũi tên có thể không viết. Ý nghĩa của sự kiện là nó luôn nhận giá trị *true*.

Định nghĩa 2.2 (Ràng buộc) [2] Một *ràng buộc* là công thức có dạng:

$$\leftarrow L_1, \ldots, L_m \qquad (n \geq 0) \qquad (2)$$

trong đó vế trái mặc định là *false*, L_1, ..., L_m là các literal (nguyên tố hoặc phủ định của nguyên tố) có đối là hằng hoặc biến. Ý nghĩa của ràng buộc (2) là L_1, ..., L_m không thể đồng thời nhận giá trị *true*.

Phần tiếp theo sẽ trình bày một số mở rộng các quy tắc và ràng buộc.

Định nghĩa 2.3 (Literal điều kiện) [6] Một *literal điều kiện* ◎◎có dạng $L{:}A$, trong đó L và A là các nguyên tố. ◎ là ký hiệu của tập các literal nền $L(a)$ sao cho $A(a)$ là true.

Ví dụ 1.1 Với p, q là các vị từ một ngôi, literal điều kiện $p(X){:}q(X)$ là tập hợp $\{p(a) \mid q(a)$ là *true*$\}$.

Trong các ứng dụng của ASP, một số lượng lớn các bài toán có thể được giải quyết bằng chương trình logic thông thường, nhưng đôi khi bài toán có một số ràng buộc mà một chương trình logic thông thường phải mất nhiều quy tắc để có thể giải quyết được hoặc không thể giải quyết được. Hạn chế này có thể được khắc phục bởi việc mở rộng của chương trình logic bằng cách cho phép các quy tắc có chứa các ràng buộc bản số, trọng số [6, 8, 13]. Trong phần này chúng tôi chỉ đề cập đến các ràng buộc bản số.

Định nghĩa 2.4 (Ràng buộc bản số) [6] Một *ràng buộc bản số C* là một biểu thức có dạng như sau:

$$L \leq \{l_1, \ldots, l_n\} \leq U \qquad (1)$$

trong đó L và U là hai số nguyên, lần lượt là chặn dưới và chặn trên của ràng buộc C, mỗi l_i là một literal hoặc literal điều kiện. Các giá trị L và U có thể bỏ qua, trong trường hợp này chặn dưới được xem là -∞ và chặn trên là ∞.

Định nghĩa 2.5 (Tính thỏa mãn của ràng buộc bản số) [6] Cho C là một ràng buộc bản số có dạng:

$$L \leq \{l_1, \ldots, l_n\} \leq U$$

và S là một tập các nguyên tố nền. Lúc đó S được gọi là *thỏa C* nếu tổng các literal trong tập $\{l_1, \ldots, l_n\}$ được thỏa bởi S là thuộc khoảng $[L,U]$.

Định nghĩa 2.6 (Quy tắc chọn) [6] *Quy tắc chọn* là công thức có dạng:

$$\{A_1, .., A_m\} \leftarrow \text{◎}_1, \ldots, \text{◎}_n \qquad (n \geq 0) \qquad (2)$$

trong đó A_i là các nguyên tố, ◎$_i$ là các ràng buộc bản số.

Quy tắc (2) có ý nghĩa là nếu thân quy tắc là *true* thì mọi tập con của các nguyên tố trong đầu quy tắc cũng là *true*. Nếu một nguyên tố không xuất hiện trong đầu của bất kỳ quy tắc nào mà làm thân là *true* thì nó sẽ là *false*.

Định nghĩa 2.7 (Quy tắc ràng buộc bản số) [6] *Quy tắc ràng buộc bản số* là công thức có:

$$C_0 \leftarrow C_1, \ldots, C_n$$

trong đó C_i là các ràng buộc bản số, $i = 1,\ldots, n$ ($n \geq 0$).

Định nghĩa 2.8 (Ràng buộc mở rộng) [2] Một *ràng buộc mở rộng* là quy tắc có dạng:

$$false \leftarrow C_1, \ldots, C_n$$

và thường được viết: $\leftarrow C_1, \ldots, C_n$

2.2. Ngữ nghĩa tập trả lời của chương trình logic

Phần này chỉ trình bày ngữ nghĩa tập trả lời của chương trình logic có chứa các quy tắc và ràng buộc thông thường. Chi tiết về ngữ nghĩa tập trả lời của chương trình logic chứa các quy tắc và ràng buộc mở rộng có thể xem trong [6, 8].

Định nghĩa 2.9 (Chương trình logic) *Chương trình logic* là một tập hữu hạn khác rỗng các quy tắc và có thể chứa các ràng buộc. *Chương trình logic nền* là chương trình logic chỉ gồm các quy tắc nền.

Định nghĩa 2.10 (Phép biến đổi Gelfond – Lifschitz) [8] Cho P là chương trình logic, I là một thể hiện tùy ý của P. *Phép biến đổi Gelfond – Lifschitz* thực hiện hai bước sau đây trên $ground(P)$:

1. Loại bỏ các quy tắc của $ground(P)$ có chứa literal âm *not* q trong thân của nó, với $q \in I$.
2. Loại bỏ tất cả literal âm trong các thân của các quy tắc còn lại của $ground(P)$.

Chương trình logic nền chứa các quy tắc còn lại, ký hiệu là P^I, P^I được gọi là *chương trình thu hẹp* của P theo I.

Nhận xét: Chương trình thu hẹp P^I là chương trình logic nền không chứa ký hiệu phủ định, nó là chương trình logic dương, vì vậy P^I có mô hình nhỏ nhất [1].

Định nghĩa 2.11 (Tập trả lời) [4] Một thể hiện I của chương trình logic P được gọi là *tập trả lời* của P nếu I trùng với mô hình nhỏ nhất của P^I.

Định nghĩa 2.12 (Ngữ nghĩa tập trả lời) [4] *Ngữ nghĩa tập trả lời* của chương trình logic P là tập các tập trả lời của P, ký hiệu ASP(P).

Ví dụ 1.2 Xem chương trình logic P gồm các quy tắc sau đây:

$$p(1, 2) \leftarrow$$
$$q(X) \leftarrow p(X, Y), \text{not } q(Y)$$

Xét thể hiện $I = \{q(2)\}$. Lúc đó, thực hiện phép biến đổi Gelfond – Lifschitz, ta nhận được chương trình P^I bao gồm các quy tắc:

$$p(1,2) \leftarrow$$
$$q(1) \leftarrow p(1,1)$$
$$q(2) \leftarrow p(2,1)$$

Mô hình nhỏ nhất của P^I là $\{p(1,2)\}$, nó khác với I. Vì vậy I không phải là tập trả lời của P.

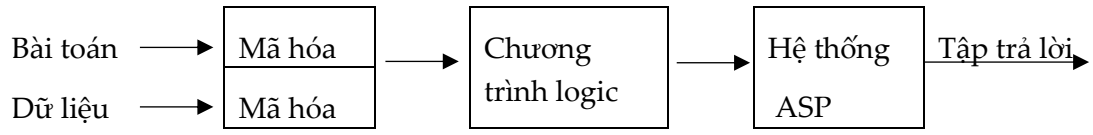
Bây giờ, ta xét tiếp thể hiện $J = \{p(1,2), q(1)\}$. Lúc đó, thực hiện phép biến đổi Gelfond – Lifschitz ta nhận được chương trình P^J gồm các quy tắc:

$$p(1,2) \leftarrow$$
$$q(1) \leftarrow p(1,2)$$
$$q(2) \leftarrow p(2,2)$$

Mô hình nhỏ nhất của chương trình P^J là $\{p(1,2), q(1)\}$ trùng với J. Vậy $J = \{p(1,2), q(1)\}$ là tập trả lời của P.

3. MỘT SỐ BÀI TOÁN ỨNG DỤNG CỦA LẬP TRÌNH TẬP TRẢ LỜI

Trong phần này sẽ trình bày việc ứng dụng lập trình tập trả lời để biểu diễn một số bài toán kinh điển của lý thuyết đồ thị, việc thực thi để tìm các câu trả lời cho các bài toán này sẽ được thực hiện bởi các hệ thống lập trình logic thông dụng như DLV, SWI-Prolog, Clingo,…Sau đây là sơ đồ biểu diễn và thực thi bài toán bằng lập trình tập trả lời:



Hình 1. Sơ đồ biểu diễn và thực thi bài toán bằng lập trình tập trả lời

Các bài toán trình bày trong phần này sẽ được thực thi bằng hệ thống lập trình logic Clingo, đây là hệ thống lập trình logic với ngữ nghĩa tập trả lời và là một thành phần của Hệ thống ASP Potassco (Potsdam Answer Set Solving Collection) được phát triển tại Đại học Potsdam, Đức., đường link của hệ thống này: https://potassco.org/clingo.

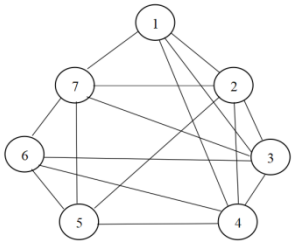
3.1 Bài toán sắc số của đồ thị

Bài toán sắc số của đồ thị được phát biểu như sau: Cho $G = \langle V, E \rangle$ là một đồ thị vô hướng. Hãy tô màu các đỉnh của đồ thị, sao cho hai đỉnh kề nhau không cùng một màu và số màu được sử dụng là ít nhất. Số màu ít nhất có thể sử dụng để tô màu các đỉnh của đồ thị được gọi là *sắc số* của đồ thị đó.

Bài toán sắc số có nhiều ứng dụng trong thực tế [7], chúng ta có thể giải quyết các bài toán khác nhau về lập lịch, lập kế hoạch, phân bổ công việc, xây dựng nhóm,… bằng việc dùng các chương trình logic để biểu diễn. Ta sẽ minh họa bài toán sắc số qua một số bài toán lập lịch thi như sau:

Bài toán Lập lịch thi: Hãy lập lịch thi trong một trường đại học sao cho không có sinh viên nào có hai môn thi cùng một thời điểm và số cuộc thi là ít nhất.

Có thể giải bài toán lập lịch thi bằng mô hình đồ thị, với các đỉnh là các môn thi, có một cạnh nối hai đỉnh nếu có sinh viên phải thi cả hai môn được biểu diễn bằng hai đỉnh này. Thời điểm thi của mỗi môn được biểu thị bằng các màu khác nhau. Để minh họa, giả sử có 7 môn thi cần xếp lịch thi. Các môn học được đánh số từ 1 tới 7 và các cặp môn thi sau có chung sinh viên: 1 và 2, 1 và 3, 1 và 4, 1 và 7, 2 và 3, 2 và 4, 2 và 5, 2 và 7, 3 và 4, 3 và 6, 3 và 7, 4 và 5, 4 và 6, 5 và 6, 5 và 7, 6 và 7. Đồ thị biểu diễn cho bài toán này như sau:



Hình 2. Đồ thị *G* minh họa cho bài toán Lập lịch thi

Để biểu diễn bài toán này bằng chương trình logic, trước hết ta sẽ định nghĩa các nguyên tố cùng với ý nghĩa được gán cho những nguyên tố này: ký hiệu *vertex* là vị từ 1-ngôi và *vertex*(*X*) để chỉ X là một đỉnh, *edge* là vị từ 2-ngôi và *edge*(*X*,*Y*) để chỉ cạnh từ đỉnh *X* đến đỉnh *Y*, *size*: là vị từ 1-ngôi và *size*(*N*) để chỉ *N* là số đỉnh của đồ thị, *color* là vị từ 2-ngôi và *color*(*X*,*C*) để chỉ màu của đỉnh *X* là màu C. Do đồ thị chỉ có 7 đỉnh nên số màu tối đa có thể sử dụng là 7, ta sẽ đánh số thứ tự màu là 1, 2,…7. Lúc đó, chương trình logic biểu diễn bài toán này như sau:

vertex(*1..7*).

edge(*1,2*) *edge*(*1,3*) *edge*(*1,4*) *edge*(*1,5*) *edge*(*1,6*) *edge*(*1,7*)

edge(2,1) *edge*(2,3) *edge*(2,4) *edge*(2,5) *edge*(2,7)

edge(3,1) *edge*(3,2) *edge*(3,4) *edge*(3,6) *edge*(3,7)

edge(4,1) *edge*(4,2) *edge*(4,3) *edge*(4,5) *edge*(4,6)

edge(5,2) *edge*(5,4) *edge*(5,6) *edge*(5,7)

edge(6,3) *edge*(6,4) *edge*(6,5) *edge*(6,7)

edge(7,1) *edge*(7,2) *edge*(7,3) *edge*(7,5) *edge*(7,6)

size(*N*) ← *N* = #*count*{*X* : *vertex*(*X*)} (1)

1 {*color* (*X*,*C*) : *C* =1.. *N*} 1 ← *vertex* (*X*), *size*(*N*) (2)

← *edge* (*X*,*Y*), *color* (*X*,*C*), *color* (*Y*,*C*) (3)

#*minimize*{1,*C* : *color*(*X*,*C*)} (4)

Ý nghĩa của quy tắc (1): Đồ thị có kích thước là N

Ý nghĩa của quy tắc (2): Mỗi đỉnh *X* của đồ thị có duy nhất một màu *C*

Ý nghĩa của ràng buộc (3): Hai đỉnh liền kề không cùng màu.

Ý nghĩa của ràng buộc (4): Số màu để tô cho các đỉnh là nhỏ nhất

Tiến hành thực thi chương trình này bằng các hệ thống lập trình logic, chẳng hạn Clingo ta nhận được 1 tập trả lời như sau:

{color(2,6), color(1,7), color(3,5), color(4,4), color(7,4), color(5,5), color(6,6)}

Với kết quả này thì số màu ít nhất cần dùng là 4 (tương ứng các màu có số thứ tự là 4, 5, 6, 7) nên sẽ có 4 cuộc thi được tổ chức.

Một số các bài toán thực tế có thể chuyển về bài toán sắc số:

Bài toán phân chia tần số: Giả sử có *n* đài phát truyền hình, ta cần phân chia các kênh truyền hình cho các đài phát sao cho không có hai đài phát nào cách nhau không quá *k* km lại dùng cùng một kênh và số kênh là ít nhất.

Để giải bài toán này, có thể xây dựng một đồ thị có các đỉnh là các đài phát, hai đỉnh được nối với nhau bằng một cạnh nếu chúng ở cách nhau không quá *k* km. Kênh truyền hình của mỗi đài được biểu thị bằng các màu khác nhau. Như vậy bài toán này được đưa về bài toán sắc số của đồ thị.

Bài toán xây các dựng kho lưu trữ hóa chất: Cần tổ chức các kho để lưu trữ các hóa chất, biết rằng có một số loại hóa chất không thể để cạnh nhau vì chúng có thể kết hợp gây nổ. Ta có thể giải quyết bài toán này bằng cách xây dựng một đồ thị mà các đỉnh của đồ thị đại diện cho các loại hóa chất khác nhau. Đối với mỗi cặp hóa chất có thể phát nổ nếu kết hợp lại, có một cạnh giữa các đỉnh tương ứng. Số màu của đồ thị này là số

lượng các khu vực lưu trữ khác nhau cần thiết để không có hai hóa chất nào trộn lẫn nổ được lưu trữ cùng nhau.

Bài toán phân chuồng nuôi thú: Một vườn bách thú muốn xây dựng chuồng tự nhiên để trưng bày các con thú. Một số loại thú sẽ ăn thịt các con thú khác nếu có cơ hội. Có thể dùng mô hình đồ thị và tô màu đồ thị như trên để xác định số chuồng khác nhau cần có và cách nhốt các con thú vào các chuồng thú tự nhiên này.

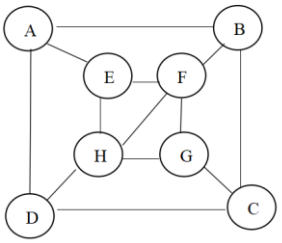
3.2. Bài toán tìm clique cực đại

Bài toán clique là một bài toán kinh điển của lý thuyết đồ thị và có nhiều ứng dụng. Nhiều vấn đề trong các mạng xã hội, sinh học, tài chính,… được giải quyết thông qua bài toán tìm kiếm các clique trong đồ thị [7]. Clique cũng được sử dụng như một công cụ phân cụm hay phân loại dữ liệu.

Một clique trong đồ thị vô hướng G là tập các đỉnh V' (V' là tập con của tập các đỉnh V của G) thoả mãn: với mỗi cặp đỉnh thuộc V' luôn tồn tại một cạnh của G nối chúng. Do vậy, một đồ thị con được tạo ra từ V' sẽ là một đồ thị đầy đủ. Kích thước của một clique là số đỉnh của nó. Bài toán đặt ra là xác định xem có tồn tại hay không một clique với kích thước cực đại. Ta minh họa qua bài toán sau:

Bài toán tìm mối quan hệ quen biết của những người trong một mạng xã hội: Một mạng xã hội thường có tập dữ liệu của những người tham gia và người ta mong muốn phân tích dữ liệu từ những người có mối quan hệ quen biết với nhau nhằm phục vụ việc nghiên cứu và tối ưu hoá hoạt động của họ. Bài toán đặt ra là tìm các nhóm người tham gia lớn nhất trong mạng xã hội có quan hệ quen biết với nhau.

Để giải quyết bài toán này, trước hết ta xây dựng một đồ thị G = ⟨V, E⟩, trong đó tập đỉnh V biểu thị cho hồ sơ dữ liệu của các người tham gia mạng xã hội và tập cạnh E thể hiện mối quan hệ quen biết lẫn nhau giữa hai người. Lúc đó bài toán đặt ra có thể chuyển về bài toán tìm clique cực đại của đồ thị G, nghĩa là tìm các tập đỉnh $V' \subseteq V$ cực đại sao cho hai đỉnh bất kỳ trong V' đều có cạnh nối giữa 2 đỉnh đó. Để minh họa, giả sử có đồ thị G = ⟨V, E⟩ như hình sau:



Hình 3. Đồ thị *G* minh họa cho bài toán tìm clique cực đại

Chương trình logic biểu diễn bài toán này như sau:

vertex(*a*) *vertex*(*b*) *vertex*(*c*)

vertex(*d*) *vertex*(*e*) *vertex*(*f*)

edge(*a*,*b*) *edge*(*b*,*c*) *edge*(*c*,*d*) *edge*(*d*,*a*)

edge(*e*,*f*) *edge*(*f*,*g*) *edge*(*g*,*h*) *edge*(*h*,*e*)

edge(*a*,*e*) *edge*(*b*,*f*) *edge*(*d*,*h*) *edge*(*f*,*h*)

{*clique*(*X*)} ← *vertex* (*X*)} (1)

← *clique*(*X*), *clique*(*Y*), *X!=Y*, *not edge*(*X*,*Y*), *not edge*(*Y*,*X*) (2)

#maximize {1,*X* : *clique*(*X*)} (5)

Trong chương trình này: *vertex* là vị từ 1-ngôi và *vertex*(*X*) để chỉ X là một đỉnh. *edge* là vị từ 2-ngôi và *edge*(*X*,*Y*) để chỉ cạnh từ đỉnh *X* đến đỉnh *Y*. *clique*: là vị từ 1-ngôi và *clique*(*X*) để đỉnh *X* thuộc vào *clique*.

Ý nghĩa của quy tắc (1): Các đỉnh thuộc clique là đỉnh của đồ thị. Ràng buộc (2): Loại bỏ các tập con có 2 đỉnh không liền kề. Ràng buộc (5) để tính số đỉnh cực đại của clique.

Tiến hành thực thi chương trình này bằng Hệ thống Clingo ta nhận được một tập trả lời kích thước lớn nhất là 3 như sau:

{clique(f), clique(g), clique(h)}

Ngoài tập trả lời tìm được này, một tập trả lời khác của bài toán với kích thước 3 là:

{clique(f), clique(h), clique(e)}

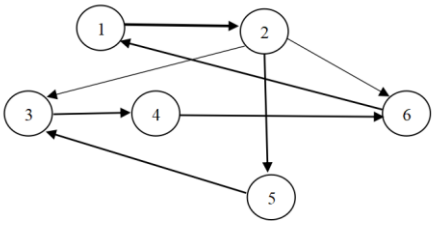
3.3. Bài toán tìm chu trình Hamilton

Trong đồ thị *G*, đường đi qua tất cả các đỉnh của đồ thị và mỗi đỉnh đúng một lần được gọi là đường đi Hamilton. Chu trình bắt đầu từ một đỉnh *v* nào đó qua tất cả các đỉnh còn lại, mỗi đỉnh đúng một lần rồi quay trở về *v* được gọi là chu trình Hamilton. Đồ thị *G* được gọi là đồ thị Hamilton nếu nó chứa chu trình Hamilton và gọi là đồ thị nửa Hamilton nếu nó có đường đi Hamilton.

Cho đến nay việc tìm một tiêu chuẩn nhận biết đồ thị Hamilton vẫn còn là mở, mặc dù đây là một vấn đề trung tâm của lý thuyết đồ thị. Hơn thế nữa, hiện nay cũng chưa có thuật toán hiệu quả để kiểm tra một đồ thị có là Hamilton hay không. Phần này sẽ trình bày việc biểu diễn bài toán tìm chu trình Hamilton qua bài toán minh họa như sau:

Bài toán tham quan du lịch: Một cơ sở du lịch muốn tổ chức một chuyến tham quan cho khách hàng. Họ cần tổ chức tham quan các địa điểm sao cho mỗi địa điểm chỉ đến một lần, với hành trình bắt đầu từ một địa điểm và kết thúc tại địa điểm đó. Đây chính là bài toán tìm chu trình Hamilton trong đồ thị vô hướng.

Giả sử *G* là một đồ thị có hướng như hình sau:



Hình 4. Đồ thị *G* minh họa cho bài toán tìm chu trình Hamilton

Bài toán này được biểu diễn bởi chương trình logic như sau:

vertex(1..6)

edge(1,2) *edge*(2,5) *edge*(2,6) *edge*(2,3)

edge(5,3) *edge*(3,4) *edge*(4,6) *edge*(6,1)

start (1) (1)

{ *cycle*(*X*,*Y*) } ← *edge*(*X*,*Y*) (2)

← *cycle*(*X*,*Y*), *cycle*(*X*,*Z*), *edge*(*X*,*Y*), *edge*(*X*,*Z*), *Y*!=*Z* (3)

← *cycle*(*Y*,*X*), *cycle*(Z,*X*), *edge*(*Y*,*X*), *edge*(Z,*X*), *Y*!=*Z* (4)

← *vertex*(*X*), not *reachable*(*X*) (5)

reachable(*Y*) ← *cycle*(*X*,*Y*),*edge*(*X*,*Y*), *start* (*X*) (6)

reachable(*Y*) ← *cycle*(*X*,*Y*), *edge*(*X*,*Y*), *reachable*(*X*), *not* start(*X*) (7)

Trong chương trình logic này, nguyên tố *cycle*(*X*,*Y*) để chỉ một cạnh từ địa điểm *X* đến địa điểm *Y* trong chu trình Hamilton. Nguyên tố *start*(1) để chỉ địa điểm xuất phát là địa điểm 1. Ý nghĩa của các quy tắc của chương trình:

Quy tắc (2) chỉ ra rằng tập các cạnh trong chu trình Hamilton là tập con các cạnh của đồ thị *G*.

Ràng buộc (3), (4) chỉ ra có nhiều nhất một đỉnh được chọn trong cạnh đến và cạnh đi và chỉ có các tập con các cạnh *cycle*(a,b) thỏa ràng buộc sẽ tạo nên chu trình.

Ràng buộc (5) và các quy tắc (6), (7) có ý nghĩa: mỗi đỉnh tìm được từ đỉnh khởi đầu và thông qua việc chọn các cạnh *cycle*(a,b).

Tiến hành thực thi chương trình này bằng Hệ thống Clingo ta nhận được chỉ một tập trả lời, tương ứng 1 chu trình Hamilton xuất phát từ đỉnh 1 của đồ thị *G* như sau:

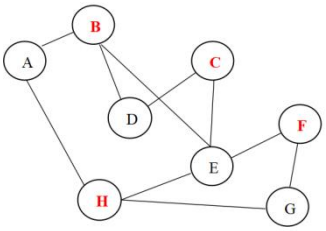
{cycle(1,2), cycle(2,5), cycle(5,3), cycle(6,1), cycle(3,4), cycle(4,6)}

3.4. Bài toán tìm phủ đỉnh

Trong lý thuyết đồ thị, phủ đỉnh của đồ thị là một tập hợp các đỉnh chứa ít nhất một điểm cuối của mỗi cạnh của đồ thị [7]. Bài toán đặt ra là tìm phủ đỉnh có kích thước nhỏ nhất, bài toán này có nhiều ứng dụng trong thực tế, ta minh họa qua bài toán sau:

Bài toán lắp đặt camera: Giả sử bạn có một phòng trưng bày nghệ thuật với nhiều hành lang và lối rẽ. Phòng trưng bày của bạn đang trưng bày những bức tranh có giá trị và bạn muốn giữ chúng an toàn. Bạn đang có ý định lắp đặt camera an ninh ở mỗi hành lang để camera có thể quan sát mọi bức tranh. Nếu có camera an ninh ở hành lang thì có thể nhìn thấy mọi bức tranh ở hành lang. Nếu có camera ở góc nơi hai hành lang gặp nhau thì có thể xem tranh ở cả hai hành lang. Chúng ta có thể mô hình hóa hệ thống này dưới dạng đồ thị *G* trong đó các đỉnh biểu thị nơi các hành lang gặp nhau hoặc khi hành lang trở thành ngõ cụt và các cạnh là hành lang. Trong đồ thị này, hãy chỉ ra nơi bạn sẽ đặt các camera sao cho tất cả các bức tranh đều được giám sát - đây là bài toán tìm phủ đỉnh của một đồ thị và có thể có nhiều phương án trả lời.

Đối với bài toán này, ta cần tìm một tập *V'* gồm ít nhất các đỉnh khác nhau trong đồ thị *G* sao cho ít nhất một đỉnh của mỗi cạnh của *G* là thuộc *V'*, đây chính là bài toán tìm phủ đỉnh cực tiểu của đồ thị *G*. Để minh họa cho bài toán này, ta giả sử đồ thị *G* được cho như hình sau:



Hình 5. Đồ thị minh họa bài toán lắp đặt camera

Bài toán này được biểu diễn bởi chương trình logic như sau:

vertex(*a*) *vertex*(*b*) *vertex*(*c*) *vertex*(*d*) *vertex*(*e*) *vertex*(*f*) *vertex*(*g*) *vertex*(*h*)

edge(*a*,*b*) *edge*(*a*,*h*) *edge*(*b*,*d*) *edge*(*b*,*e*) *edge*(*c*,*d*)

edge(*c*,*e*) *edge*(*e*,*f*) *edge*(*e*,*h*) *edge*(*f*,*g*) *edge*(*g*,*h*)

{ *cover*(*X*)} :- *vertex*(*X*) ← (1)

← *edge*(*X*,*Y*), *not cover*(*X*), *not cover*(*Y*) (2)

#*minimize*{1,*X* : *cover*(*X*)} (3)

Trong chương trình logic này, quy tắc (1) yêu cầu tập trả lời là tập con của của tập đỉnh. (2), (3) là ràng buộc phải thỏa mãn theo yêu cầu bài toán.

Tiến hành thực thi chương trình này bằng Hệ thống Clingo ta nhận được một tập trả lời tương ứng với một phủ đỉnh của *G* có kích thước nhỏ nhất là 4 sau đây:

{ cover(b), cover(h), cover(c), cover(f)}

4. KẾT LUẬN

Hiện nay lập trình tập trả lời có nhiều ứng dụng trong rất nhiều lĩnh vực khác nhau như Trí tuệ nhân tạo, Xử lý ngôn ngữ tự nhiên, Hệ chuyên gia, web ngữ nghĩa,…Trong khuôn khổ bài báo, chúng tôi chỉ giới thiệu một số ứng dụng tiêu biểu của lập trình tập trả lời để biểu diễn và thực thi một số bài toán tối ưu kinh điển của lý thuyết đồ thị và áp dụng qua thực tế của những bài toán này. Các bài toán minh họa đều thuộc lớp bài toán NP và việc giải quyết là không đơn giản, tuy nhiên có thể thấy rằng việc sử dụng lập trình tập trả lời để biểu diễn các bài toán là khá tự nhiên, điều này nhằm thể hiện các ưu điểm của việc sử dụng logic trong việc biểu diễn tri thức.

TÀI LIỆU THAM KHẢO

[1]. Apt K. R. (1990). *Logic Programming*, Elsevier Science Publishers.

[2] Arias, J.; Carro, M.; Salazar, E.; Marple, K.; Gupta, G. (2018). "Constraint Answer Set Programming without Grounding". Theory and Practice of Logic Programming. 18 (3–4): 337–354. arXiv:1804.11162.

[3] Falkner, A.A., Friedrich, G., Schekotihin, K., Taupe, R., Teppan, (2018) E.C.: Industrial applications of answer set programming. KI 32(2–3), 165–176 (2018).

[4] Gelfond M., Lifschitz V., Van Hermelen F., Porter B., (2008). *Answer sets*, Handbook of Knowledge Representation, chapter 7, pages 285-316. Elsevier Science.

[5] Gelfond M. and Leone N. (2002). *Logic programming and knowledge representation - the A-Prolog perspective*, Artificial Intelligence, 138(1-2):3-38.

[6] Guohua L. Jia H. Y. (2011). Relating weight constraint and aggregate programs: Semantics and representation, *Theory and Pratice of Logic Programming,* Vol 13, Issue 1, pages 1-31.

[7] Jonathan L. Gross Jay Yellen Mark Anderson, (2019), Graph Theory and Its Applications, CRC Press, Third Edition.

[8] Jori B., Martin G., Tomi J. (2014). Improving the Normalization of Weight Rules in Answer Set Programs, *Logics in Artificial Intelligence*, pages 166-180.

[9] Marek Victor W., Miroslaw T. (1999), Stable models and an alternative logic programming paradigm, Book of The Logic Programming Paradigm, pages 375–398.

[10] Martin G., Max O., Torsten S. (2009). Constraint answer set solving. In Proceedings of 25th International Conference on Logic Programming (ICLP), pages 235–249. Springer.

[11] Mirosław T. (2007). Logic Programming for Knowledge Representation, International Conference on Logic Programming, ICLP 2007, pp 76–88.

[12] Simons P., Niemela. I., Soininen T., (2002). *Extending and implementing the stable model semantics*, Artificial Intelligence , Vol. 138, Iss.1-2, 181–234.

[13] Vladimir Lifschitz (2019). Answer Set Programming, Springer Link.

[14] Wolfgang F. (2020). An Introduction to Answer Set Programming and Some of Its Extensions, Springer Link, Lecture Notes in Computer Science ((LNISA,volume 12258)).

[15] Association for the Advancement of Artificial Intelligence, AI Magazine, Vol. 37 No. 3, Fall 2016.

APPLICATION OF ANSWER SET PROGRAMMING TO REPRESENT AND IMPLEMENT SOME PROBLEMS OF GRAPH THEORY

Truong Cong Tuan

Faculty of Information Technology, Hue University of Sciences

Email: tctuan@husc.edu.vn

ABSTRACT

In the past decades, research on logic programming has achieved significant advancements in both theory and application. Logic programming has been widely applied for knowledge representation and information processing in various fields. Nowadays, logic programs with answer set semantics have been proposed as a new paradigm for applying declarative programming techniques to represent knowledge. In this paper, we present the application of answer set programming to represent and implement some problems of graph theory.

Keyword: Logic Programming, answer set, answer set semantics.

Trương Công Tuấn sinh ngày 28/4/1960 tại Hội An. Ông tốt nghiệp đại học Tổng hợp Huế năm 1982, chuyên ngành toán học tối ưu; Tốt nghiệp Tiến sỹ tại Viện Công nghệ thông tin Hà Nội năm 2003 chuyên ngành Khoa học máy tính; Được phong học hàm PGS năm 2012. Hiện ông đang công tác tại Khoa Công nghệ thông tin, trường Đại học Khoa học, Đại học Huế.

Lĩnh vực nghiên cứu: Hệ cơ sở tri thức, Lập trình logic và Cơ sở dữ liệu suy diễn.