BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

LUẬN VĂN THẠC SỸ KHOA HỌC

LẬP TRÌNH RÀNG BUỘC VỚI BÀI TOÁN NGƯỜI CHƠI GÔN

NGHÀNH: CÔNG NGHỆ THÔNG TIN
MÃ SỐ:

NGUYỄN VĂN HẬU

Người hướng dẫn khoa học: PGS. TS. NGUYỄN THANH THUỶ
TS. FRANCISCO AZEVEDO

HÀ NỘI 2006

MỤC LỤC

LỜI NÓI ĐẦU

Người đầu tiên mà tôi xin dành sự cảm ơn và kính trọng đặc biệt là *PGS. TS. Nguyễn Thanh Thủy*. Không những cuốn sách đầu tiên đã làm tôi say mê với "Trí tuệ Nhân tạo" là của Thầy mà Thầy còn là người trực tiếp hướng dẫn của tôi. Chính Thầy là người đã tin tưởng và tạo điều kiện tốt nhất cho tôi hoàn thành Luận văn tốt nghiệp này.

Chắc chắn sẽ không thể nói hết được những tình cảm mà tôi muốn nói, muốn cảm ơn tới *TS. Francisco Azevedo*. Thầy là người cùng tôi ngồi viết những chương trình đầu tiên và sửa lỗi cho tôi. Mọi thắc mắc của tôi đều được Thầy giải đáp và còn hơn thế nữa. Thầy coi tôi là một người bạn, với tôi, Thầy là một người bạn lớn.

Tôi cũng rất muốn dành lời cảm ơn tới *TS. Trần Đình Khang*, người đã có những giúp đỡ tôi, động viên tôi rất nhiều về mặt tinh thần.

Tôi xin cảm ơn tới tất cả những đồng nghiệp trong khoa CNTT, trường ĐHSPKT Hưng Yên, đặc biệt là *Th.S Ngô Hữu Tình*, *Th.S Nguyễn Minh Quý* và *Th.S Nguyễn Đình Hân*, họ là nguồn động viên rất lớn cho tôi.

Xin cảm ơn những người bạn tốt của tôi: *Việt*, *Lý*, *Chuẩn*, *Hiếu*, *Thế*, *Zhang Dong*, *Manoela*, họ đã cổ vũ và chia sẻ với tôi mọi điều trong cuộc sống.

Những người cuối cùng mà tôi xin dành lời cảm ơn, là gia đình tôi. Họ luôn là điểm tựa đầu tiên và mãi mãi của tôi. Mọi điều tôi làm, tôi đều nghĩ tới họ.

Lisbon, Ngày 26 tháng 10 năm 2006

ACKNOWLEDGEMENTS

The first person I would like to thank and respect specially is *Prof. Nguyen Thanh Thuy*. Not only the first book that I read made me interested in "Artificial Intelligence", but also he is my excellent supervisor. He believed in me, gave me a good change to do my thesis. If he had not taught and led me, probably I could have not got this thesis.

I am sure that there are not enough words to thank *Prof. Francisco Azevedo* for all things he have been doing for me since I came here. He helped me with the first steps from "Prolog" to "Constraint Programming". He read, try to understand and correct for my program. I have learnt lots of things from him. He invited me to go to his home, enjoin dinner with him and take me around Lisbon many times. He is so kind, thoughtful. He is a outstanding person. He consider me as a friend, for me, he is my great friend.

I also would like to thank *Dr. Tran Dinh Khang* for his help and support me during the time I have done the thesis.

My acknowledgements to all my colleagues, especially *M.Sc.Ngo Huu Tinh*, *M.Sc.Nguyen Minh Quy*, and *M.Sc.Nguyen Dinh Han* for encouraging me a lot.

Thank you to my best friend: *Viet*, *Ly*, *Chuan*, *Hieu*, *The*, *Zhang Dong*, and *Manoela*, they have been encouraging me in everything.

The last people I would like to thank are my family, all of them help, support, love me during whole my life. They are my the first fulcrum and forever. Everything I do, I do it for them.

Lisbon, 26 September, 2006

KÍ HIỆU VÀ Ý NGHĨA CÁC TỪ VIẾT TẮT

Viết tắt	Ý nghĩa
CSP, CSPs	Bài toán thỏa mãn ràng buộc
CLP	Lập trình Logic Ràng buộc
CP	Lập trình Ràng buộc
SGP	Bài toán người chơi gôn
SB	Loại bỏ đối xứng
SBDS	Loại bỏ đối xứng trong thời gian tìm kiếm
SBDD	Loại bỏ đối xứng dựa vào sự ưu thế
ND	Kỹ thuật hạn chế miền
F	Kỹ thuật cố định một số tay gôn
NDF	Kết quả tốt nhất giữa ND và F
DFS	Tìm kiếm theo chiều sâu
BT	Quay lui
NC	Thỏa mãn điều kiện cho ràng buộc một ngôi
AC	Thỏa mãn điều kiện cho ràng buộc hai ngôi
MOLS	Tập hình vuông Latin trực giao
MOLR	Tập hình chữ nhật Latin trực giao

Ký hiệu	Ý nghĩa
P	Chỉ một bài toán thỏa mãn ràng buộc
Z hoặc X	Chỉ tập các biến trong CSP
D	Chỉ miền cho toàn bộ các biến trong CSP
C	Lập trình Logic Ràng buộc
n	Số tay gôn trong bài toán "Người chơi gôn"
g	Số nhóm trong một tuần
s	Số phần tử trong mỗi nhóm
w	Số tuần đạt được
$G_{i,j}$	Chỉ tay gôn trong tuần thứ *i* ở nhóm thứ *j*
$G_{i,j}(n)$	Chỉ tay gôn trong tuần thứ *i* ở nhóm thứ *j* tại vị trí *n*
$\|S\|$	Số phần tử của tập *S*
φ_P	Đối xứng trong nhóm (các tay gôn thay đổi)
φ_G	Đối xứng trong tuần (các nhóm thay đổi)
φ_W	Đối xứng giữa các tuần (các tuần thay đổi)
φ_X	Đối xứng giữa các tay gôn (các tay gôn hoán vị)
$N(n)$	Số hình vuông lớn nhất có thể từ tập MOLS cấp *n*
$N(m \times n)$	Số hình chữ nhật lớn nhất có thể từ tập MOLR cấp $m \times n$
r MOLS(*n*)	Có *r* hình vuông Latin trực giao cấp *n*
r MOLR($m \times n$)	Có *r* hình chữ nhật Latin trực giao cấp $m \times n$

PHẦN I. GIỚI THIỆU VỀ LẬP TRÌNH RÀNG BUỘC

Lập trình ràng buộc (Constraint Programming - CP) là một trong những phát triển thú vị và mạnh mẽ nhất của ngôn ngữ lập trình trong thập kỷ gần đây[5, 7,10,11,24,28,36,37]. Được xây dựng trên cơ sở lý thuyết toán học vững chắc, nó đang phát triển và đặc biệt là nó cũng đang thu hút sự quan tâm mạnh mẽ trong việc áp dụng vào lĩnh vực thương mại, nó trở thành phương pháp mô hình hóa cho nhiều loại bài toán tối ưu, cụ thể là trong các ràng buộc có sự hỗn tạp và các bài toán tìm kiếm có tính tổ hợp.

Lý giải cho sự quan tâm trong CP thật đơn giản. Ngôn ngữ lập trình ra đời sớm là FORTRAN-66, rất gần với cấu trúc vật lý của máy tính. Vì vậy, xu hướng chính của ngôn ngữ lập trình là mang lại sự tự do cho người lập trình đối với việc định nghĩa các đối tượng và thủ tục tương ứng với các thực thể và thao tác trong miền ứng dụng.

Ngôn ngữ lập trình hướng đối tượng (Object Oriented Programming Language) cung cấp một kỹ thuật tốt cho việc khai báo các thành phần để kiểm soát hành vi của thực thể trong một miền bài toán cụ thể. Tuy nhiên, ngôn ngữ lập trình truyền thống, bao gồm ngôn ngữ lập trình hướng đối tượng, cung cấp rất ít sự hỗ trợ với các thực thể mà người lập trình muốn diễn tả những *ràng buộc* và những *quan hệ*. Người lập trình mong muốn vai trò của ngôn ngữ để duy trì những quan hệ và tìm ra những đối tượng thỏa mãn.

Ví dụ, xét định luật Ôm sau:

$$U=I \times R,$$

Công thức mô tả mối quan hệ giữa hiệu điện thế, cường độ dòng điện và điện trở. Trong ngôn ngữ lập trình truyền thống, người lập trình không thể dùng quan hệ này một cách trực tiếp, thay vào đó nó phải được mã hóa thành câu

lệnh mà từ đó việc tính toán giá trị của một thành phần dựa trên 2 thành tố còn lại. Vì vậy, I có thể được suy ra từ U và R bằng công thức sau:

I:= U/R,

Nhưng nếu giá trị của được tính từ hai thành phần còn lại, một công thức khác lại phát sinh:

R:= U/I,

Việc đòi hỏi người lập trình mô tả và duy trì các quan hệ giữa các đối tượng trong lập trình là hợp lý cho các ứng dụng có sử dụng. Tuy nhiên trong nhiều ứng dụng, vấn đề quan trọng là mô hình các quan hệ và tìm ra các đối tượng thỏa mãn. Vì lý do đó mà từ cuối những năm 60, đã có nhiều chuyên gia quan tâm đến các ngôn ngữ lập trình cho phép người lập trình đơn giản hóa các quan hệ giữa các trạng thái của đối tượng. Nó là vai trò thực thi cơ bản nhằm đảm bảo rằng những quan hệ đó hay những *ràng buộc* được duy trì. Những ngôn ngữ như vậy được coi là ngôn ngữ CP (Constraint Programming).

Ban đầu những ngôn ngữ CP chỉ thành công với một số phần. Chúng bổ trợ cho một ngôn ngữ truyền thống với việc giải quyết các ràng buộc bằng các kỹ thuật không định trước đơn giản. Những ngôn ngữ này phần lớn phụ thuộc vào phương pháp **lan truyền** cục bộ (local propagation). Phương pháp "lan truyền cục bộ" dùng một ràng buộc để gán một giá trị vào một biến chưa biết từ các giá trị đã biết cho các biến khác trong ràng buộc. Ví dụ, trong định luật Ôm có thể tính toán một giá trị R, I hoặc V từ hai giá trị đã biết. Bài toán với lan truyền cục bộ là phương pháp giải quyết ràng buộc giữa các quan hệ yếu. Ví dụ, nó không thể dùng để giải các phương trình xảy ra đồng thời như X= Y-Z và X= 2Y+Z. Như vậy việc dựa trên lan truyền cục bộ của những ngôn ngữ thời kỳ đầu có hai điểm yếu: Những thuận lợi giải quyết những ràng buộc

là không đủ mạnh và chính ngôn ngữ không đủ mạnh để diễn tả những ràng buộc.

Trong thập kỷ gần đây ngôn ngữ lập trình ràng buộc được quan tâm mạnh mẽ. Hơn nữa, các ngôn ngữ đã khắc phục được những khó khăn của những ngôn ngữ trước. Chúng hỗ trợ ràng buộc và tích hợp triệt để vào ngôn ngữ lập trình, nó cho phép người lập trình làm việc với bài toán ở mức độ cao hơn, trong khi các kỹ thuật thực thi ở mức dưới cũng sử dụng kỹ thuật thích hợp cho ràng buộc. Việc ra đời các ngôn ngữ lập trình ràng buộc thế hệ mới đáp ứng được những yêu cầu cho một lượng lớn các ứng dụng.

Một ví dụ đơn giản cho ứng dụng trong khi dùng ngôn ngữ lập trình ràng buộc, hãy tưởng tượng rằng bạn đang mua một ngôi nhà và muốn kiểm tra những lựa chọn khác nhau đối với việc trả lãi. Với mỗi khoảng trả lãi, tổng tiền lãi dồn lại là PxI, trong đó P là tổng số tiền vay và I là tỷ lệ lãi suất. Lãi suất dồn lại được cộng thêm với tiền vay để đạt được một số tiền vay mới NP. Nếu bạn đem trả R thì đó chính là số tiền bị trừ đi. Như vậy ta có phương trình ràng buộc:

$$NP= P+P\times I -R,$$

Sự cầm cố trong khoảng thời gian T có thể được mô tả bởi việc lặp lại việc tính toán này, ở mỗi thời điểm, cho đến khi hết thời gian. Tổng cuối cùng gọi là B cân bằng.

Bài toán này có thể tóm gọn trong chương trình sau:

```
mortgage(P, T, I, R, B):-
        T=0,
        B=P.
mortgage(P, T, I, R, B):-
        T>=1,
```

NT= T-1,

NP= P + P*I –R,

mortgage(NP, NT, I, R, B).

Ở đây, ràng buộc *mortgage* chỉ ra quan hệ giữa tiền vốn ban đầu P, thời gian vay T, tỷ lệ lãi suất I, tổng số phải là R và điểm cân bằng là B. Luật đầu tiên (3 dòng đầu) xử lý trường hợp khi kết thúc thời gian. Trong trường hợp này điểm cân bằng chính là số tiền vốn hiện tại. Luật thứ hai (5 dòng tiếp theo) xử lý trường hợp khi số khoảng thời gian lớn hơn hoặc bằng 1. Trong trường hợp này một thời điểm mới (*NT*) sẽ trừ đi 1. Khi đó việc thay đổi vốn cũng được tính lại. Phần còn lại của việc cho vay được xác định trên một lượng vay mới và tổng vốn mới khi mà thời gian giảm đi 1.

Chương trình trên dường như có thể dễ viết trên ngôn ngữ lập trình truyền thống, ví dụ Pascal hay C. Thuận lợi của chương trình là, bởi vì tất cả các câu lệnh được xem xét dưới góc độ ràng buộc, nó diễn tả bài toán rất linh hoạt. Thực thi chương trình được khởi tạo bằng cách đưa ra một đích. Ví dụ, nếu việc mượn $1000 trong 10 năm với lãi suất 10% cùng với việc phải trả $150 một năm. Chúng ta có thể viết như sau:

mortgage(1000, 10, 10/100, 150, B).

Khi ta đưa ra đích như trên, câu trả lời sẽ là B=203.129, chỉ ra thời điểm cân bằng là $203.129.

Một chương trình tương tự có thể được dùng trong nhiều cách khác nhau. Ví dụ việc mượn $150 và đến khi hết hạn việc trả lãi tại thời điểm cân bằng là 0, chúng ta có thể đạt câu hỏi “Cần phải vay bao nhiêu trong vòng 10 năm với lãi suất 10% với việc trả $150 một năm”. Câu hỏi này có thể được viết:

mortgage(P, 10, 10/100, 150, 0).

Câu trả lời là P=921.685.

Một câu hỏi phức tạp hơn giữa quan hệ vốn vây ban đầu, số tiền phải trả hàng năm trong 10 năm với lãi suất 10%. Chúng ta có thể đưa ra:

mortgage(P, 10, 10/100, R, B).

Câu trả lời là một ràng buộc P=0.3855*B +6.1446*R, một quan hệ giữa các biến P, B, và R.

Trong tất cả các trường hợp đều được cùng một chương trình giải. Điều này tương phản với lập trình truyền thống khi mà trong một chương trình khác nhau có thể yêu cầu trả lời một trong các câu hỏi đó. Thực vậy, việc viết chương trình trả lời câu hỏi cuối cùng quả thực là khó. Lập trình ràng buộc cho phép chúng ta chỉ ra bài toán một cách rất tự nhiên, và ở mức cao với việc dùng các ràng buộc số học. Nó là công việc của ngôn ngữ thực thi mức dưới để giải những ràng buộc này hơn là công việc của người lập trình.

Ví dụ này tuy đơn giản những cũng minh họa tại sao lập trình ràng buộc giải quyết được nhiều ứng dụng, bài toán trong đời sống thực với mô hình phức tạp một cách tự nhiên và hiệu quả.

Số công ty đầu tư nghiên cứu và ứng dụng công nghệ ràng buộc, hàng năm, tăng lên đáng kể. Xin kể ra một số công ty lớn như: *Oracle, British Airways, SAS, Swissair, French railway authority SNCF, Hong Kong International Terminals, Michelin, Dassault, Ericsson*, …Có rất nhiều công ty cung cấp các giải pháp dựa trên ràng buộc như *PeopleSoft, i2 Technologies, InSol, SAP, jdedwards, Vine Solutions, ...* cũng như có các công ty cung cấp các công cụ dựa trên ràng buộc như *PeopleSoft, i2 Technologies, InSol, Vine Solutions,...*

Xin nêu ra đây một vài hệ thống được đóng gói cho phép giải các bài toán thỏa mãn ràng buộc:

- Prolog: CHIP, ECL^{i}PS^e, SICStus Prolog, Prolog IV, GNU Prolog, IF/Prolog

- C/C++: CHIP++, ILOG Solver
- Java: JCK, JCL, Koalog
- Mozart

Cũng vì lý do này mà CP đã và đang được dùng với nhiều vùng khác nhau cho nhiều bài toán trong cuộc sống. Nhiều bài toán kỹ thuật đặc biệt phù hợp với CP vì chúng thường liên quan đến sự kết hợp có trật tự trong các hệ thống phức tạp:

- Các mô hình toán học hay Boolean, và đặc biệt là trong các trường hợp chuẩn đoán và thiết kế- lập luận dựa trên luật.
- Một vùng lớn khác là lập lịch tài chính và trong lĩnh vực thương mại, nơi mà các ứng dụng thường được các chuyên gia giúp đỡ cùng với mô hình toán học.
- Tính toán số, khi mà việc giải các ràng buộc đa thức cần sự có sự đảm bảo độ chính xác.
- Sinh học phân tử, chúng liên quan đến chuỗi DNS, và việc xây dựng mô hình 3D cho các Protein.
- Kỹ thuật điện tử, tìm ra vị trí rò trong mạch điện, tính toán sự sắp đặt mạch điện, kiểm tra và thẩm định sự thiết kế.
- Nó cũng được ứng dụng trong việc xử lý ngôn ngữ tự nhiên (tạo ra những bộ phân tích cú pháp hiệu quả).
- Hệ thống đồ họa tương tác, nhằm diễn tả tính chặt chẽ về mặt hình học trong trường hợp phân tích hoạt cảnh.
- Hơn nữa nó cũng được áp dụng liên quan đến di truyền và tạo ra các dữ liệu thử cho các giao thức trao đổi thông tin.
- Người ta cũng cho rằng, hầu hết các ứng dụng quan trọng của ngôn ngữ CP được áp dụng cho các bài toán *mang tính tổ hợp khó*, và nó

cũng là một mô hình đầy sức mạnh cho việc giải những bài toán tối ưu tổ hợp. Ví dụ như việc phải giải quyết liên quan đến *lập bảng thời gian* (timetabling), *lập lịch* (scheduling), *định tuyến* (routing). Những kiểu bài toán này rất khó để diễn tả và giải quyết trên các ngôn ngữ lập trình truyền thống. Điều này do chúng yêu cầu tìm kiếm trong một không gian nghiệm cỡ hàm mũ nhằm tìm ra được nghiệm tối ưu cho bài toán. Để đạt được hiệu quả, các ràng buộc phải dùng các kỹ thuật cắt không gian tìm kiếm.

Trong quá trình giải quyết các bài toán tổ hợp khó, có hai cách tiếp cận truyền thống: hoặc là thực hiện thủ công thuật toán sẽ giải chính xác trong lập trình truyền thống, hoặc là mô hình bài toán dùng thuật toán có sẵn (off the shelf) cho lớp các ràng buộc số học cụ thể. Hạn chế của phương pháp thứ nhất là việc thiết kế nó đòi hỏi chi phí lớn và không dễ gì biến đổi khi bài toán thay đổi. Hạn chế của phương pháp thứ hai là không dễ gì diễn tả hiệu quả các ràng buộc trong miền ứng dụng đủ mềm dẻo và hiệu quả cho phép các kinh nghiệm trong các miền được chỉ ra để giải quyết chúng. Ngôn ngữ CP hiện đại có thể khắc phục được những điểm yếu này bằng cách cung cấp một ngôn ngữ lập trình dựa trên việc giải ràng buộc tinh vi nhất. Điều này có nghĩa là người lập trình có thể viết chương trình trong khi kỹ thuật giải chung đã được cung cấp trong việc thực thi ngôn ngữ.

Chúng ta có thể xét một ví dụ đơn giản sau, một ví dụ rất quen thuộc [1, 24,28]:

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline MONEY \end{array}$$

Với mỗi ký tự là một số khác nhau trong phương trình số học.

Bài toán này có thể được giải trong ngôn ngữ ràng buộc như sau:

```
:- use_module(library(clpfd)).

mm([S,E,N,D,M,O,R,Y], Type) :-
     domain([S,E,N,D,M,O,R,Y], 0, 9),      % step 1
     S#>0, M#>0,
     all_different([S,E,N,D,M,O,R,Y]),     % step 2
     sum(S,E,N,D,M,O,R,Y),
     labeling(Type, [S,E,N,D,M,O,R,Y]).    % step 3

sum(S, E, N, D, M, O, R, Y) :-
                  1000*S + 100*E + 10*N + D
     +            1000*M + 100*O + 10*R + E
     #= 10000*M + 1000*O + 100*N + 10*E + Y.
```

Trong chương trình trên, các biến S, E, N, D, M, O, R và Y được khai báo trong miền giá trị khoảng [0,9] trong khi ràng buộc được người dùng định nghĩa trong **all_different()** và **sum()**. Việc giải các ràng buộc như vậy trong trường số nguyên là rất nhanh nhờ việc áp dụng các kỹ thuật lan truyền linh động. Cái giá của tốc độ trong cách giải này là việc giải không trọn vẹn, vì vậy chương trình có thể cho câu trả lời **không biết,** để chỉ rằng nó không biết có nghiệm hay không. Trong trường hợp này người lập trình có thể dùng hàm bổ trợ, **labeling**, dùng để tìm kiếm các giá trị khác nhau trong khoảng [0,9] cho các biến. Điều này có nghĩa rằng chương trình được đảm bảo tìm ra nghiệm khi nó tồn tại. Trong trường hợp này, **labeling** là một hàm thư viện đã được cung cấp cho hệ thống, nhưng sức mạnh của giải pháp CP là người lập trình có thể định nghĩa ra việc giải bài toán cụ thể theo cách của riêng họ. Trong ví dụ trên, nếu chúng ta gọi:

```
| ?- mm([S,E,N,D,M,O,R,Y], []).
D = 7,
E = 5,
M = 1,
N = 6,
O = 0,
R = 8,
S = 9,
Y = 2
```

Khái niệm các bài toán thỏa mãn điều kiện ràng buộc (Constraint Satisfaction Problems - CSPs) cũng được chính thức công nhận bởi cộng đồng trí tuệ nhân tạo (AI). Họ cũng chỉ ra những khái niệm cơ bản của tính nhất quán cục bộ (local consistency) và các thuật toán để giải chúng. Một cách độc lập, nhiều phương pháp khác nhau cũng đã được hình thành, Một trong số chúng, như quay lui (backtracking) được đưa ra từ thế kỷ 19, trong khi khái niệm nhánh-cận (branch and bound) được đưa ra trong tối ưu tổ hợp. Những đóng góp của CP là đã chỉ ra những dạng mới khác nhau trong tìm kiếm khi kết hợp những kỹ thuật đã biết với các thuật toán lan truyền ràng buộc khác nhau. Một số sự tổ hợp đặc trưng cũng đã được nghiên cứu trong vùng tối ưu tổ hợp.

Sự phát triển của CSP đã dẫn đến sự ra đời của ngôn ngữ lập trình ràng buộc. Trong thập niên 80, những ngôn ngữ CP đầu tiên đã ra đời. Việc quan trọng là những ngôn ngữ này đều dựa trên những nguyên lý Lập trình Logic. Chính điều này dẫn đến sự phát triển của Lập trình Logic Ràng buộc (Constraint Logic Programming-CLP) và được mở rộng từ ngôn ngữ lập trình logic như Prolog bằng cách thay thế phép hợp nhất (unification) bằng việc kiểm tra việc thỏa mãn ràng buộc dùng bộ giải đã định. Chúng được bắt đầu từ Châu Âu và Australia trong những năm cuối thập niên 1980. Cả hai ví dụ ở trên đều được thể hiện qua CLP. Các bộ giải khác nhau và miền ứng dụng khác nhau sẽ cần đến các ngôn ngữ khác nhau nhưng có một kỹ thuật đánh giá chung.

Bất chấp sự thành công của CLP, gần đây, một số ngôn ngữ lập trình ràng buộc khác đang được bàn thảo. Bao gồm: ngôn ngữ ràng buộc đồng thời, nó dùng sự kế thừa ràng buộc để mở rộng ngôn ngữ CLP bằng cách cung cấp các thông tin không đồng bộ giữa các tác tử (agents); ngôn ngữ truy vấn ràng buộc cho cơ sở dữ liệu, nó mở rộng cơ sở dữ liệu quan hệ bằng cách cho phép các bộ chứa các biến đã được ràng buộc; ngôn ngữ lập trình hàm ràng buộc,

ngôn ngữ lập trình mệnh lệnh ràng buộc và bộ công cụ giải ràng buộc hướng đối tượng.

Tuy nhiên, Ngôn ngữ CLP là ngôn ngữ lập trình ràng buộc nguyên mẫu. Theo cảm nhận, chúng là ngôn ngữ lập trình ràng buộc "tinh khiết" và "nhỏ nhất" do về bản chất chỉ có thao tác người lập trình có thể thực hiện là việc định nghĩa các ràng buộc mới của họ từ những ràng buộc cở sở đã được trang bị. Vì lý do này, việc hiểu CP là công việc liên quan đến bất kỳ ngôn ngữ lập trình ràng buộc nào.

Đặc tính nổi bật của lập trình ràng buộc là các ràng buộc được liên kết chặt chẽ một cách tự nhiên. Nó liên quan mật thiết với các khía cạnh của toán học, khoa học máy tính truyền thống và trí tuệ nhân tạo. Lập trình ràng buộc phác họa công việc trong thuật toán giải quyết ràng buộc từ việc tìm kiếm các thao tác, tính toán số và kỹ thuật giải quyết ràng buộc trong các bài toán thỏa mãn ràng buộc, một lĩnh vực quan trọng trong trí tuệ nhân tạo. Nó cũng phác họa những kỹ thuật từ việc thiết kế và thực thi ngôn ngữ lập trình, cũng như lập luận tự động, đến lý thuyết và việc thực thi cơ sở dữ liệu.

PHẦN II. NHỮNG CƠ SỞ VỀ BÀI TOÁN THỎA MÃN RÀNG BUỘC

Bài toán thỏa mãn ràng buộc (Constraint Satisfaction Problem – CSP) đang ngày càng trở nên phổ biến trong cộng đồng khoa học máy tính cũng như trí tuệ nhân tạo. Mục đích chính của phần này là giới thiệu những kiến thức cô đọng nhất cho CSPs: Những định nghĩa cùng với những khái niệm quan trọng cho CSPs; các kỹ thuật áp dụng nhằm biến đổi bài toán sao cho dễ giải hơn, đồng thời cũng nêu ra các cách tiếp cận và giải CSPs [7,24,28,36].

CHƯƠNG 1. GIỚI THIỆU NHỮNG KHÁI NIỆM CƠ BẢN

1.1. Những định nghĩa quan trọng trong CSP

Trong phần này, chúng ta sẽ nêu những định nghĩa quan trọng trong CSP. Trước khi làm điều đó, chúng ta sẽ phải định nghĩa **miền**, **nhãn** và khái niệm **sự thỏa mãn**.

1.1.1. Định nghĩa miền và nhãn

Định nghĩa 1.1:

> **Miền** của một biến là tập các giá trị có thể gán tới biến. Nếu x là một biến, ta ký hiệu D_x là miền của x.■
>
> Khi miền chỉ chứa các số, các biến được gọi là *biến số*. Miền của biến số có thể được hạn chế trong số **nguyên, hữu tỉ** hay **số thực**. Ví dụ, miền của biến nguyên là một tập vô hạn {1, 2, 3, …}. Trong Luận văn này chỉ tập trung vào CSP với miền hữu hạn.
>
> Khi miền chỉ chứa giá trị boolean, biến sẽ được gọi là **biến boolean**. Khi mà miền chứa kiểu liệt kê các đối tượng, biến sẽ được gọi là **biến**

biểu tượng. Ví dụ, một biến thể hiện ngày trong tuần là biến biểu tượng vì miền của nó là một tập hữu hạn {thứ hai, thứ ba, thứ tư, thứ năm, thứ sáu, thứ bảy, chủ nhật}.

Định nghĩa 1.2

Nhãn là một cặp *biến-giá trị* thể hiện rằng biến đó đã được gán giá trị. Chúng ta dùng <x, v> để chỉ rằng biến x được gán giá trị v. <x, v> chỉ có nghĩa nếu v là một giá trị thuộc miền của x. ■

Định nghĩa 1.3

Một phép gán **nhãn kết hợp** là một phép gán đồng thời các giá trị (có thể là rỗng) đến tập các biến. Chúng ta ký hiệu (<x_1, v_1>, < x_2, v_2>…< x_n, v_n>) để chỉ việc gán kết hợp v_1, v_2 , …, v_n tới x_1, x_2 , …, x_n tương ứng. ■

Định nghĩa 1.4

Một phép gán **nhãn k-kết hợp** là một phép gán nhãn kết hợp đồng thời của *k* giá trị tới *k* biến. ■

Deleted: k
Deleted: k

Định nghĩa 1.5

Nếu *m* và *n* là các số nguyên sao cho $m \leq n$, khi đó **phép chiếu** của một nhãn n-kết hợp $\mathcal{N}$ tới một nhãn m-kết hợp $\mathcal{M}$, được coi như phép chiếu *projection*($\mathcal{N}$, $\mathcal{M}$) nếu tất cả các nhãn của $\mathcal{M}$ đều có mặt trong $\mathcal{N}$

Deleted: n
Deleted: n

$$\forall(< x_1, v_1 > ... < x_m, v_m >), (< z_1, w_1 > ... < z_n, w_n >) : \{x_1, ..., x_m\} \subseteq \{z_1, ..., z_m\} :$$

$$projection((< z_1, w_1 >, ... < z_n, w_n >), (< x_1, v_1 > ... < x_m, v_m >)) \equiv$$

$$< x_1, v_1 >, ..., < x_m, v_m > \in \{< z_1, w_1 >, ... < z_n, w_n >\}$$

Ví dụ, (<a,1><c,3>) là một phép chiếu của (<a,1><b,2><c,3>), cũng có nghĩa là *projection*((<a,1><b,2><c,3>), (<a,1><c,3>)) là đúng.

Định nghĩa 1.6

Các biến trong gán **nhãn kết hợp** là tập các biến xuất hiện trong nhãn kết hợp đó.

$$\text{var}\,iables\_of((< x_1, v_1 >< x_2, v_2 > ... < x_k, v_k >)) \equiv \{x_1, x_2, ..., x_k\}$$

1.1.2. Định nghĩa ràng buộc

Một ràng buộc là tập các biến được hạn chế giá trị sao cho chúng có thể đạt được một cách đồng thời. Một cách khái niệm, một ràng buộc có thể được xem như một tập chứa tất cả các nhãn kết hợp cho các biến; trong thực tế, ràng buộc có thể được thể hiện nhiều cách khác, ví dụ như hàm, ma trận, bất phương trình…

Định nghĩa 1.7

Một **ràng buộc** trên một tập các biến được coi như là một tập các nhãn kết hợp tương ứng với biến đó. Để thuận tiện, chúng ta dùng C_s để ký hiệu cho ràng buộc trên tập biến của S. ■

Định nghĩa 1.8

Biến của ràng buộc là các biến của các thành viên trong ràng buộc

$$\text{var}\,iables\_of(C_{x_1, x_2, ..., x_k}) \equiv \{x_1, x_2, ..., x_k\}$$

Định nghĩa 1.9

Nếu m và n là các số nguyên sao cho $m \leq n$, khi một *m-ràng buộc* $\mathcal{M}$ là một **subsumed-by** của *n-ràng buộc* $\mathcal{N}$ (được ký hiệu *subsumed-by*($\mathcal{N}$; $\mathcal{M}$)) nếu mọi phần tử c trong $\mathcal{M}$ đều tồn tại một phần tử d trong $\mathcal{N}$ sao cho c là phép chiếu của d.

Deleted: n

Deleted: n

$$\forall C_M, C_N . |M| = m \wedge |N| = n \wedge m \le n :$$
$$subsumed - by(C_M, C_N) \equiv$$
$$(\forall(< x_1, v_1 > ... < x_m, v_m >) \in C_M :$$
$$(\exists(< z_1, w_1 > ... < z_n, w_n >) \in C_N :$$
$$projection((< z_1, w_1 > ... < z_n, w_n >), (< x_1, v_1 > ... < x_m, v_m >))))$$

Ở đây ký hiệu |M| và |N| là số biến trong M và N. Nếu chúng ta có :

$$C_M = \{(< a,1 >< c,3 >), (< a,4 >< c,6 >)\}$$
$$C_N = \{(< a,1 >< b,2 >< c,3 >), (< a,1 >< b,4 >< c,3 >), (< a,4 >< b,5 >< c,6 >)\}$$

thì subsumed-by(C_M, C_N) là đúng.

1.1.3. Định nghĩa sự thỏa mãn

Sự thỏa mãn (satisfies) là một quan hệ nhị phân giữa các nhãn hoặc nhãn kết hợp với ràng buộc.

Định nghĩa 1.10a

Nếu các biến trong nhãn kết hợp *X* cũng chính là biến trong nhãn kết hợp của ràng buộc *C*, khi đó *X* **satisfies** *C* nếu và chỉ nếu *X* là một phần tử trong *C*.

$$satisfies((< x_1, v_1 >< x_2, v_2 > ... < x_k, v_k >), C_{x_1, x_2, ..., x_k}) \equiv$$
$$(< x_1, v_1 >< x_2, v_2 > ... < x_k, v_k >) \in C_{x_1, x_2, ..., x_k}$$

Định nghĩa 1.10b

$$satisfies(< x, v >, C_x) \equiv (< x, v >) \in C_x$$

Định nghĩa 1.11

Cho một tập nhãn kết hợp *L* và một ràng buộc *C* sao cho biến trong *C* là tập con của tập biến trong *L*, nhãn kết hợp *L* **satisfies** ràng buộc *C*

nếu và chỉ nếu phép chiếu của L nên các biến trong C là một phần tử của C.

$$\forall x_1, x_2, ..., x_k : \forall v_1 \in D_{x_1, v_2} \in D_{x_2, ..., v_k} \in D_{x_k} :$$

$$(\forall S \subseteq \{x_1, x_2, ..., x_k\} :$$

$$satisfies((< x_1, v_1 >< x_2, v_2 > ... < x_k, v_k >), C_S) \equiv$$

$$(\exists cl \in C_S : projection((< x_1, v_1 > ... < x_k, v_k >), cl)))$$

Ví dụ (<a,1><b,2><c,3><d,4>) **satisfies** ràng buộc $C_{c,d}$ nếu và chỉ nếu (<c,3><d,4>) là một phần tử của $C_{c,d}$.

1.1.4. Định nghĩa bài toán thỏa mãn ràng buộc (CSP):

Định nghĩa 1.12

Một bài toán thỏa mãn ràng buộc là một bộ ba (*Z*, *D*, *C*), trong đó

Z = tập hữu hạn ***biến*** { $x_1, x_2, \ldots, x_n$};

D = một hàm ánh xạ mỗi biến trong *Z* tới tập các đối tượng của biến tương ứng:

D: Z→tập đối tượng hữu hạn

Chúng ta gọi D_{xi} là tập đối tượng ánh xạ từ x_i bởi D. Như vậy

D_{xi} là ***miền*** của x_i

C = tập (có thể rỗng) các ***ràng buộc*** trên một tập con tùy ý của các biến trong *Z*. Nói một cách khác, *C* là tập của tập các nhãn kết hợp.

Chúng ta dùng CSP($\mathcal{P}$) để ký hiệu rằng $\mathcal{P}$ là một bài toán thỏa mãn ràng buộc. ■

Chú ý sự khác nhau giữa C_x và D_x: C_x là tập các nhãn trong khi D_x là tập các giá trị. Giá trị của x không hẳn chỉ trong ràng buộc C_x, điều đó có nghĩa là <x, a> satisfies C_x , và tất cả các ràng buộc chứa x, bao gồm $C_{y,x}$, $C_{x,y,z}$, …

Chúng ta tập trung vào CSP có số biến hữu hạn và miền của chúng cũng hữu hạn.

1.1.5. Nhiệm vụ trong bài toán CSP

Nhiệm vụ của CSP là gán giá trị tới mỗi biến sao cho chúng thỏa mãn đồng thời tất cả các ràng buộc.

Định nghĩa 1.13

Một **bộ nghiệm** của một CSP là một nhãn kết hợp cho tất cả các biến trong tất cả các ràng buộc:

$$\forall csp((Z,D,C)) : \forall x_1, x_2, ..., x_n \in Z : (\forall v_1 \in D_{x_1,v_2} \in D_{x_2,...,v_n} \in D_{x_n} :$$

$$solution\_tuple((< x_1, v_1 >< x_2, v_2 > ... < x_n, v_n >), (Z,D,C)) \equiv$$

$$((Z = \{x_1, x_2, ..., x_n\}) \wedge$$

$$(\forall c \in C : satisfies((< x_1, v_1 >< x_2, v_2 > ... < x_n, v_n >), c)))$$

CSP được coi là thỏa mãn nếu tồn tại bộ nghiệm. Tùy thuộc vào yêu cầu ứng dụng, CSPs có thể phân loại thành:

1. CSPs chỉ ra **không tồn tại** nghiệm.
2. CSPs chỉ cần tìm ra **một** bộ nghiệm bất kỳ
3. CSPs cần tìm ra **toàn bộ** các bộ nghiệm.

4. CSPs cần tìm ra một nghiệm **tối ưu** (chúng thường gặp trong lập lịch)

1.2. CSP cho ràng buộc nhị phân

Định nghĩa 1.14

> Một **CSP nhị phân,** hay **bài toán ràng buộc nhị phân**, là một CSP chỉ có ràng buộc một ngôi (unary) hoặc hai ngôi (binary). Một CSP mà ràng buộc không bị giới hạn trong một hoặc hai ngôi được coi như là một CSP tổng quát. ■

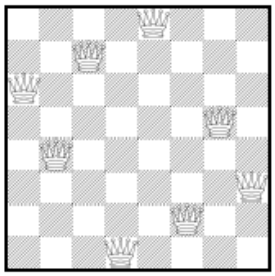
Có một điểm khá quan trọng là mọi CSPs đều có thể chuyển về được dưới dạng CSP nhị phân.

1.3. Một vài ví dụ

1.3.1. Bài toán N-quân hậu

Đây có thể coi là bài toán kinh điển nhất trong CSP. Bởi vì nó có nhiều đặc tính mang tính đặc thù trong CSPs. Từ đó các nhà nghiên cứu có thể vận dụng vào việc tìm hiểu các kỹ thuật chung cho CSP.

Chúng ta cần xếp n quân hậu vào bàn cờ vua n×n, n>2, sao cho chúng không tấn công lẫn nhau (Hình 1.1):



Hình 1.1: Minh họa một nghiệm cho bài toán 8-quân hậu

Ở đây, ta dễ dàng chuyển đổi sang CSP:

Biến : Z={Q1, Q2, …, Qn} chính là các cột

Miền : $D_{Q1}= D_{Q2}=…= D_{Qn}$ = {1, 2, …, n }, chính là các hàng

Ràng buộc :

- Thứ nhất, 2 quân không cùng cột

$$\forall i, j: Q_i \neq Q_j$$

- Thứ hai, 2 quân không cùng đường chéo

$$\forall i, j, \text{if } Q_i = \text{a and } Q_j = \text{b, then } i - j \neq \text{a} - \text{b, and } i - j \neq \text{b} - \text{a.}$$

Nói chung, một bài toán N-quân hậu sẽ có khoảng N^N khả năng khi tìm nghiệm cho bài toán.

1.3.2. Bài toán SEND+MORE=MONEY

Thay mỗi ký tự sau bằng những số khác nhau, sao cho phép tính tổng là đúng

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline MONEY \end{array}$$

Chuyển đổi sang CSP:

Biến : S, E, N, D, M, O, R, Y

Miền : $D_S= D_E=…= D_Y$ = {0, 1, 2, …, 9 }, chính là các hàng

Ràng buộc :

- Thứ nhất, phép tổng

$$\begin{aligned} & 1000 \cdot S + 100 \cdot E + 10 \cdot N + D \\ + & 1000 \cdot M + 100 \cdot O + 10 \cdot R + E \\ = 10000 \cdot M & + 1000 \cdot O + 100 \cdot N + 10 \cdot E + Y \end{aligned}$$

- Thứ hai, bất kỳ hai ký tự nào cũng phải khác nhau

$$x \neq y \text{ for } x, y \in \{S, E, N, D, M, O, R, Y\},$$
$$x \prec y.$$

Formatted: Heading 2, Left, Space Before: 3 pt, After: 3 pt, Line spacing: single

CHƯƠNG 2. GIẢI BÀI TOÁN THỎA MÃN RÀNG BUỘC

CSPs thực sự rất đáng quan tâm vì nó xuất hiện trong một số lớn các ứng dụng. Nó cũng có những đặc tính riêng cần được khám phá và phát triển bằng những thuật toán hiệu quả riêng. Chương này, chúng ta sẽ đi qua tổng quan các kỹ thuật giải CSP, chúng ta có thể phân thành 3 loại: *Rút gọn bài toán, tìm kiếm và sự tổng hợp nghiệm.*

2.1. Rút gọn bài toán (Problem redution)

2.1.1 Các định nghĩa

Định nghĩa 2.1

Chúng ta gọi 2 CSPs là **tương đương** nếu chúng có chung tập biến và bộ nghiệm:

$$\forall csp((Z,D,C)), csp((Z',D',C')):$$
$$equivalent((Z,D,C),(Z',D',C')) \equiv$$
$$Z = Z' \wedge \forall T : (solution\_tuple(T,(Z,D,C)) \Leftrightarrow$$
$$solution\_tuple(T,(Z',D',C')))$$

Định nghĩa 2.2

Bài toán $\mathcal{P}$ =(Z, D, C) được rút gọn (**reduced**) thành $\mathcal{P}$'=(Z', D', C') nếu:

a) $\mathcal{P}$ và $\mathcal{P}$' là tương đương

b) Mọi miền của biến trong D' là tập con của miền biến trong D

c) C' được hạn chế hơn hoặc bằng C (mọi nhãn kết hợp thỏa mãn C' sẽ thỏa mãn C)

Chúng ta quy ước quan hệ giữa $\mathcal{P}$ và $\mathcal{P}$' bởi công thức *reduced($\mathcal{P}$, $\mathcal{P}$'):*

$$\forall csp((Z,D,C)), csp((Z',D',C')):$$
$$reduced((Z,D,C),(Z',D',C')) \equiv$$
$$equivalent((Z,D,C),(Z',D',C')) \wedge$$
$$(\forall x \in Z : D_x^{'} \subseteq D_x) \wedge$$
$$(\forall S \in Z : (C_S \in C \Rightarrow C'_S \subseteq C_S))$$

Việc rút gọn bài toán chính là việc loại bỏ đi những phần tử trong ràng buộc mà không xuất hiện trong bộ nghiệm. Chúng ta cũng cần định nghĩa sự dư thừa giá trị và dư thừa nhãn kết hợp.

Định nghĩa 2.3

Một giá trị trong miền được gọi là dư thừa **(redundant)** nếu nó không có mặt trong bộ nghiệm:

$$\forall csp((Z,D,C)) : \forall x \in Z : \forall v \in D_x :$$
$$redundant(v,x,(Z,D,C) \equiv$$
$$\neg\exists T : (solution\_tuple(T,(Z,D,C)) \wedge projection(T,(< x,v >)))$$

Những giá trị như vậy được gọi là "redundant" bởi vì việc loại bỏ nó không làm ảnh hưởng tới tập nghiệm.

Định nghĩa 2.4

Một nhãn kết hợp trong ràng buộc được gọi là **redundant** nếu nó không có mặt trong phép chiếu của bất kỳ bộ nghiệm nào:

$$\forall csp((Z,D,C)) : \forall C_S \in C : \forall cl \in C_S :$$
$$redundant(cl,(Z,D,C) \equiv$$
$$\neg\exists T : (solution\_tuple(T,(Z,D,C)) \wedge projection(T,cl))$$

2.1.2 Việc rút gọn bài toán:

Kỹ thuật rút gọn bài toán để biến đổi CSPs thành một bài toán khác tương đương với hy vọng nó dễ giải hơn bằng cách giảm đi cỡ của miền và ràng

buộc trong bài toán. Điều này là hoàn toàn làm được trong khi giải CSPs vì miền và ràng buộc được định rõ.

Rút gọn bài toán liên quan đến 2 công việc chính:

(1) Loại bỏ những giá trị thừa từ các miền của biến

(2) Làm chặt những ràng buộc sao cho chỉ một vài nhãn kết hợp thỏa mãn chúng, nếu các ràng buộc được coi như là các tập thì điều này có nghĩa là loại bỏ các nhãn kết hợp dư thừa ra khỏi ràng buộc. Nếu miền của bất kỳ biến hoặc ràng buộc nào là rỗng, thì có thể kết luận rằng bài toán vô nghiệm.

Rút gọn bài toán yêu cầu cần có khả năng nhận ra những giá trị và nhãn kết hợp dư thừa (redundant). Những thông tin như vậy có thể được lấy từ các ràng buộc. Thuật toán rút gọn sẽ được thảo luận ở chương 3.

Cũng cần phải nói thêm rằng việc rút gọn bài toán thường liến quan đến việc bảo toàn sự nhất quán (consistency maintainance). Bảo toàn sự nhất quán cũng có nghĩa là rút gọn bài toán tới một bài toán khác có các tính chất đã được xác định.

2.1.3 Bài toán tối thiểu

Định nghĩa 2.5

Một graph của một CSP nhị phân được gọi là graph tối thiểu nếu không miền nào chứa giá trị dư thừa và không ràng buộc nào chứa nhãn kết hợp dư thừa. Nói một cách khác, mỗi nhãn kết hợp trong một ràng buộc nhị phân đều xuất hiện trong một vài bộ nghiệm:

$\forall$ csp((Z, D, C)):
  minimal_graph((Z, D, C)) $\equiv$
    ($\forall x \in Z$: ($\neg \exists v \in D_x$: redundant($v, x, (Z, D, C)$))) $\wedge$
      ($\forall C_{y,z} \in C$: ($\neg \exists cl \in C_{y,x}$: redundant($cl$, (Z, D, C)))) ■

2.2. Tìm kiếm bộ nghiệm

Phần lớn lỗ lực trong việc nghiên cứu CSPs tập trung vào kỹ thuật tìm kiếm. Trong phần này, chúng ta sẽ mô tả một cách cơ bản thuật toán tìm kiếm sau đó phân tích các tính chất CSPs. Các thuật toán có thể được thiết kế để giải CSPs một cách hiệu quả nhờ có được những tính chất này.

2.2.1 Thuật toán quay lui đơn giản (Simple Backtracking)

Thuật toán cơ bản để tìm kiếm nghiệm là thuật toán quay lui đơn giản, nó chính là một chiến lược tìm kiếm tổng quát và được dùng rộng rãi trong việc giải các bài toán (Prolog dùng nó để trả lời các câu hỏi). Trong một trường hợp cụ thể của CSPs, thao tác cơ bản là chọn một biến tại một thời điểm và xét một giá trị cho nó, đảm bảo rằng nhãn được chọn đó sẽ phù hợp với tất cả các nhãn trong tương lai. Việc gán một giá trị vào một biến gọi là **labelling**. Nếu **labelling** biến hiện tại với giá trị được chọn không phù hợp với một ràng buộc nào đó, thì một giá trị khác có sẵn sẽ được chọn. Nếu tất cả các biến được gán nhãn, khi đó bài toán được giải.

Bởi vì thuật toán quay lui (Backtracking- BT) luôn luôn quay lui tại điểm quyết định cuối cùng khi quá trình kết thúc, nên nó cũng được gọi là quay lui theo một trình tự (chronological backtracking). Chúng ta có đoạn mã sau:

```
PROCEDURE Chronological_Backtracking( Z, D, C );
BEGIN
      BT-1( Z, { }, D, C );
END

PROCEDURE BT-1( UNLABELLED, COMPOUND_LABEL, D, C );
/* UNLABELLED is a set of variables to be labelled; */
/* COMPOUND_LABEL is a set of labels already committed to */
BEGIN
   IF (UNLABELLED = { }) THEN return(COMPOUND_LABEL)
   ELSE BEGIN
      Pick one variable x from UNLABELLED;
      REPEAT
         Pick one value v from D_x;
         Delete v from D_x;
         IF COMPOUND_LABEL + {<x,v>} violates no constraints
         THEN BEGIN
            Result ←
               BT-1(UNLABELLED – {x}, COMPOUND_LABEL +
                  {<x,v>}, D, C);
            IF (Result ≠ NIL) THEN return(Result);
         END
      UNTIL (D_x = { });
      return(NIL); /* signifying no solution */
   END /* of ELSE */
END /* of BT-1*/
```

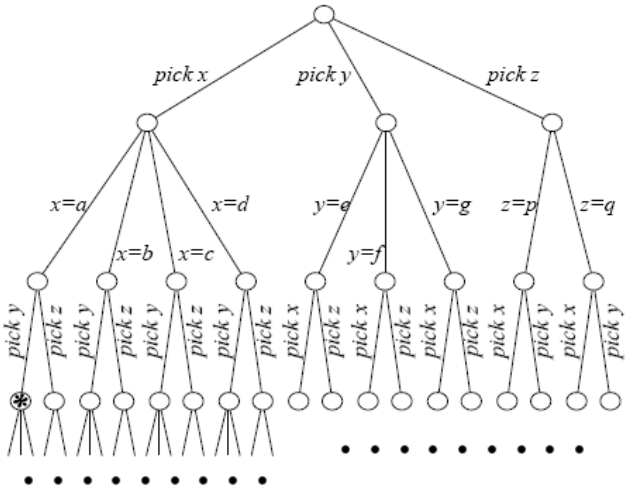
Nếu số biến là n, số ràng buộc là e, số miền là a cho mỗi biến trong CSP. Khi đó có thể có a^n khả năng cho bộ nghiệm, và độ phức tạp thời gian cho việc kiểm tra toàn bộ ràng buộc là $O(a^n e)$.

Độ phức tạp bộ nhớ của bài toán là $O(na)$. Thuật toán BT không đòi hỏi bộ nhớ tạm thời nhiều hơn $O(n)$ để lưu trữ nhãn kết hợp. Vì vậy, độ phức tạp không gian lưu trữ cho Chronological_Backtracking là $O(na)$.

Chú ý rằng độ phức tạp thời gian ở trên chỉ ra rằng thuật toán sẽ hiệu quả hơn nếu ta giảm được a. Điều này có thể đạt được bằng các kỹ thuật rút gọn bài toán. Chúng ta sẽ thảo luận ở phần sau.

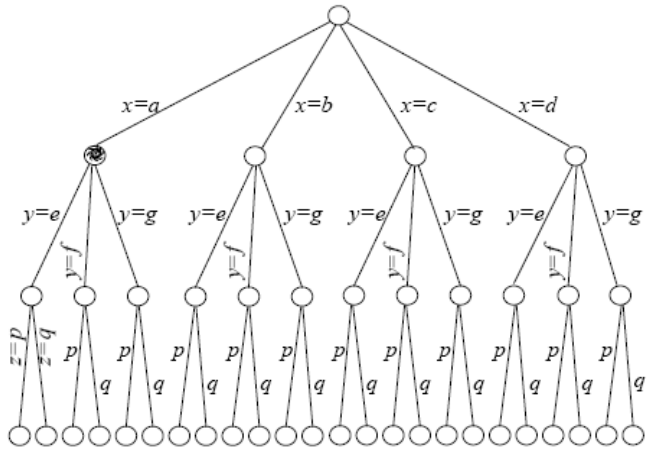
2.2.2 Không gian tìm kiếm của CSPs

Không gian tìm kiếm là không gian của tất cả các trạng thái mà việc tìm kiếm có thể đạt tới.



Hình 2.1:Không gian tìm kiếm của thuật toán BT trong CSP(Z, D, C) khi các biến không được sắp thứ tự, Z={x, y, z}, D_x={a, b, c, d}, D_y={e, f, g}, D_z={p, q}

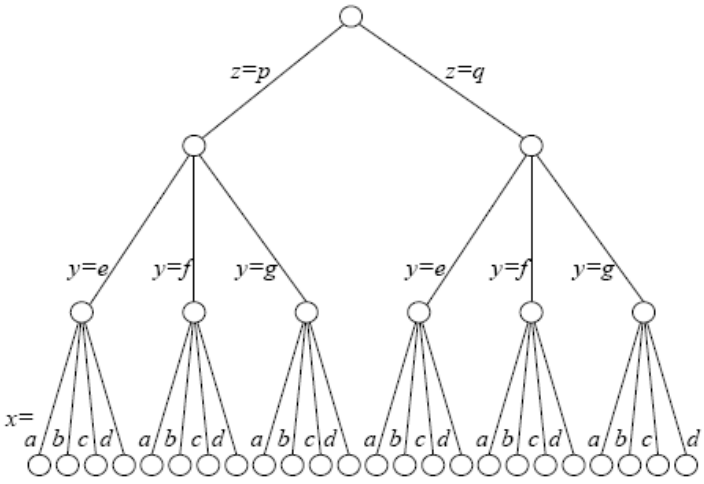
Chú ý node ⊛ trong hình 2.1 thể hiện trạng thái khi *x* được gán nhãn *a*, và *y* được chọn cho việc gán nhãn những vẫn được được gán giá trị. Chú ý rằng các ràng buộc không đóng vai trò trong định nghĩa không gian tìm kiếm, mặc dù nó sẽ trở nên rõ ràng hơn sau đó, nó sẽ ảnh hưởng lên không gian tìm kiếm bằng thuật toán.



Hình 2.2: Không gian tìm kiếm của thuật toán BT trong CSP(Z, D, C) khi các biến sắp thứ tự {x, y, z}, Z={x, y, z}, D_x={a, b, c, d}, D_y={e, f, g}, D_z={p, q}

Chú ý node ⊛ trong hình 2.2 thể hiện trạng thái khi *x* được gán nhãn *a*, *y* và *z* vẫn được được gán giá trị.

Cần phải chú ý rằng không gian tìm kiếm sẽ khác nhau nếu trật tự các biến được sắp thứ tự khác nhau.



Hình 2.3: Không gian tìm kiếm của thuật toán BT trong CSP(Z, D, C) khi các biến sắp thứ tự {z , y, x}, Z={x, y, z}, D_x={a, b, c, d}, D_y={e, f, g}, D_z={p, q}

2.2.3 Đặc tính tổng quát của không gian tìm kiếm trong CSPs

Chúng ta cần tìm hiểu một số thuộc tính CSPs vì nó khác so với bài toán tìm kiếm tổng quát để khi giải bài toán hiệu quả hơn.

(1) Cỡ của không tìm kiếm là hữu hạn

Số lá trong cây tìm kiếm là $L = |D_{x1}||D_{x2}|...|D_{xn}|$, trong đó D_{xi} là miền của biến x_i, và $|D_{xi}|$ là cỡ của miền đó. Chú ý rằng L không bị ảnh hưởng bởi trật tự khi chúng ta quyết định gán nhãn cho biến. Tuy nhiên, trật tự lại ảnh hưởng đến số nút trung gian trong không gian tìm kiếm. Ví dụ, trong tổng số nút trung gian trong Hình 2.2 là 16, trong khi Hình 2.3 là 8. Tổng quát hơn, nếu chúng ta giả sử các biến được sắp theo thứ tự x_1 , x_2 , …, x_n thí số nút trong cây tìm kiếm là:

$$1 + \sum_{i=1}^{n} |D_{x1}||D_{x2}|...|D_{xn}|$$

Với công thức trên, cùng với 2 ví dụ đã nêu, không khó khăn cho chúng ta nhận ra rằng nếu các biến được sắp theo trật tự khi các miền của nó giảm dần thì số nút trong không gian tìm kiếm sẽ đạt giá trị lớn nhất. Đó cũng chính là biên của cỡ trong không gian tìm kiếm. Ngược lại nếu các biến được sắp theo trật tự khi các miền của nó tăng dần thì số nút trong không gian tìm kiếm sẽ đạt giá trị nhỏ nhất.

Tuy nhiên, cỡ của bài toán lại bị chi phối bởi tích cuối cùng: $L = |D_{x1}||D_{x2}|...|D_{xn}|$, chính là số nút lá, nó không thay đổi khi trật tự các biến thay đổi.

(2) Độ sâu của cây được cố định

Khi các biến được cố định, độ sâu của cây tìm kiếm luôn luôn bằng số biến trong bài toán. Trong cả hai ví dụ Hình 2.2 và 2.3, độ sâu đều là 3. Khi trật tự của biến không cố định, độ sâu chính xác là 2*n*, với *n* là số biến.

(3) Các cây con tương tự nhau

Nếu chúng ta cố định biến, khi đó các cây con cùng mức sẽ tương tự nhau. Điều này rất có ý nghĩa trong khi tìm kiếm một cây con khi chúng ta đã tìm kiếm được anh em của nó (sẽ nói thêm trong phần loại bỏ đối xứng).

2.2.4 Kết hợp tìm kiếm và rút gọn bài toán

Hiệu quả của quay lui sẽ được cải thiện nếu một biến có thể bị cắt đi khi nó không có mặt trong nghiệm. Điều đó được giúp đỡ bởi quá trình rút gọn bài toán. Như phần trước chúng ta đã biết, việc rút gọn bài toán chính là quá trình làm giảm cỡ miền của biến và làm chặt ràng buộc. Việc giảm cỡ miền có hiệu quả tương tự như việc cắt bỏ một nhánh trong cây tìm kiếm. Việc làm chặt ràng buộc có tiềm năng giúp chúng ta giảm không gian tìm kiếm ở một trạng thái sau. Rút gọn bài toán có thể được thực hiện tại bất kỳ một trạng thái nào của tìm kiếm. Có rất nhiều chiến lược khác nhau nhằm kết hợp việc tìm kiếm và rút gọn bài toán (Chúng ta sẽ nói ở những phần tiếp theo).

2.2.5 Những điểm chọn trong tìm kiếm

(1) Biến nào sẽ được chọn tiếp theo?

(2) Giá trị nào sẽ được chọn tiếp theo?

(3) Ràng buộc nào sẽ được kiểm tra tiếp theo?

Hai lựa chọn đầu đã được xét đến. Sự khác nhau trong không gian tìm kiếm sẽ được khám phá dưới trật tự khác nhau của biến và giá trị. Vì ràng buộc có thể được lan truyền, trật tự khác nhau của biến và giá trị được xem xét có thể ảnh hưởng đến hiệu quả trong thuật toán tìm kiếm. Điều này đặc biệt có ý nghĩa khi tìm kiếm được kết hợp với vấn đề rút gọn bài toán.

Với bài toán chỉ cần tìm một nghiệm, hiệu quả tìm kiếm có thể được cải thiện bằng cách dùng heuristics- nó sẽ chỉ ra những nhánh trong không tìm kiếm có khả năng nhất để tìm tới nghiệm.

Trong một số bài toán, việc kiểm tra một ràng buộc có thỏa mãn hay không chi phí là khá lớn. Trong trường hợp đó, trật tự ràng buộc để kiểm tra có thể ảnh hưởng tới hiệu quả bài toán.

2.1.3 Tìm kiếm Backtrack-free

Trong chương 1, chúng ta đã định nghĩa khái niệm cơ bản của ràng buộc và sự thỏa mãn. Trong phần này, chúng ta sẽ mở rộng chúng.

Định nghĩa 2.6

> Một **thể hiện ràng buộc** trong một tập biến S, chúng ta ký hiệu là CE(S), là một tập hợp các ràng buộc trong S và tập các biến con của nó.

Định nghĩa 2.7

> Một **thể hiện ràng buộc** trong một tập con các biến S của CSP $\mathscr{P}$, chúng ta ký hiệu là CE(S, $\mathscr{P}$), là một tập hợp tất cả các ràng buộc liên quan trong $\mathscr{P}$ tại S và tập con của các biến.

$$\forall\, \mathrm{csp}((Z, D, C)): \forall\, S \subseteq Z: (\mathrm{CE}(S, (Z, D, C)) \equiv \{C_Y \mid Y \subseteq S \wedge C_Y \in C\}) \blacksquare$$

Như vậy không khó khăn khi chúng ta chuyển từ (Z, D, C) thành (Z, D, CE(Z, (Z, D, C))).

Định nghĩa 2.8

Một nhãn kết hợp $\mathscr{A}$ thỏa mãn một thể hiện ràng buộc $\mathscr{C}$ nếu $\mathscr{A}$ thỏa mãn mọi ràng buộc trong $\mathscr{C}$:

$$\forall\, csp((Z, D, C)): \forall\, x_1, x_2, ..., x_k \in Z: (\forall\, v_1 \in D_{x_1}, v_2 \in D_{x_2}, ..., v_k \in D_{x_k}:$$
$$\forall\, S \subseteq \{x_1, x_2, ..., x_k\}:$$
$$satisfies((<x_1,v_1><x_2,v_2>...<x_k,v_k>), CE(S, (Z, D, C))) \equiv$$
$$\forall\, C_R: (C_R \in CE(S, (Z, D, C)) \Rightarrow$$
$$satisfies((<x_1,v_1><x_2,v_2>...<x_k,v_k>), C_R))) \blacksquare$$

Định nghĩa 2.9

Một tìm kiếm trong CSP là một backtrack-free khi tìm kiếm theo chiều sâu khi trật tự các biến được sắp xếp nếu mỗi biến được gán nhãn, khi đó một biến luôn có thể tìm thấy giá trị phù hợp với tất cả các nhãn.

$$\forall\, csp((Z, D, C)): (\forall <: total\_ordering(Z, <):$$
$$backtrack\text{-}free((Z, D, C), <) \equiv$$
$$(\forall\, x_1, x_2, ..., x_m \in Z: (x_1 < x_2 < ... < x_m \Rightarrow$$
$$(\forall\, v_1 \in D_{x_1}, v_2 \in D_{x_2}, ..., v_m \in D_{x_m}:$$
$$(satisfies((<x_1,v_1>... <x_m,v_m>), CE(\{x_1,...,x_m\}, (Z, D, C))) \Rightarrow$$
$$(\forall\, y \in Z: (x_m < y \Rightarrow \exists\, a \in D_y:$$
$$satisfies((<x_1,v_1>...<x_m,v_m><y,a>),$$
$$CE(\{x_1,...,x_m,y\}, (Z, D, C)))))))) \blacksquare$$

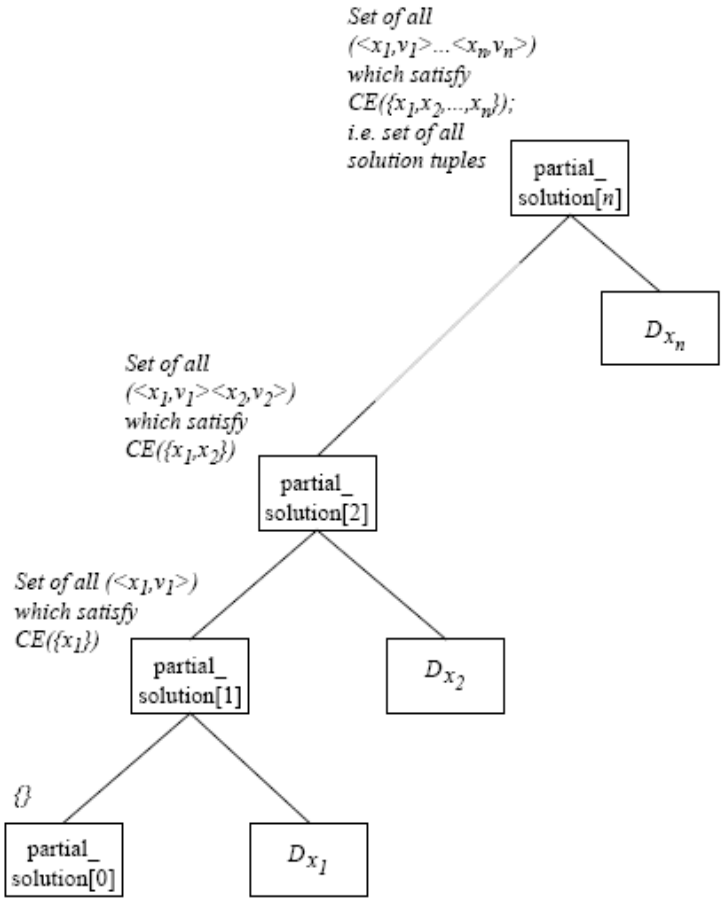
2.2 Tổng hợp nghiệm

Trong phần này, chúng ta sẽ đưa ra tổng quan về giải pháp tổng hợp nghiệm trong khi giải CSPs. Việc tổng hợp nghiệm giống như thuật toán tìm kiếm, chúng khám phá đồng thời một lúc nhiều nhánh. Nó cũng được xem như việc rút gọn bài toán khi mà ràng buộc đối với tập tất cả các biến (có nghĩa là *n*-

ràng buộc cho một bài toán với *n* biến) được tạo ra và rút gọn đến khi một tập chứa toàn bộ các bộ nghiệm và chỉ bộ nghiệm thôi.

Deleted: n

Trong quá trình tìm kiếm một nghiệm thành phần được xem xét tại một thời điểm. Một nhãn kết hợp được mở rộng bằng cách thêm một nhãn tại thời điểm đó cho đến khi một bộ nghiệm được tìm thấy hoặc toàn bộ nhãn kết hợp được xét. Ý tưởng cơ bản của tổng hợp nghiệm là tập hợp tập tất cả các nhãn hợp lệ cho các tập biến lớn hơn, cho đến khi tập toàn bộ các biến được làm. Để đảm bảo tính đúng đắn, thuật toán tổng hợp nghiệm phải đảm bảo chắc chắn rằng toàn bộ nhãn kết hợp không hợp lý sẽ được loại bỏ khỏi tập này. Để đảm bảo tính đầy đủ, thuật toán tổng hợp nghiệm phải đảm bảo chắc chắn rằng không nhãn kết hợp hợp lệ nào bị loại bỏ khỏi tập này. Chúng ta xem Hình 2.4 và đoạn mã.




```
PROCEDURE Naive_synthesis(Z, D, C)
BEGIN
   order the variables in Z as x1, x2, ..., xn;
   partial_solution[0] ← {()};
   FOR i = 1 to n DO
      BEGIN
         partial_solution[i] ← { cl + <xi, vi> | cl ∈ partial_solution[i–1]
            ∧ vi ∈ Dxi ∧ cl + <xi, vi> satisfies all the constraints on
            variables_of(cl) + xi };
      END
   return(partial_solution[n]);
END /* of Naive_synthesis */
```

Hình 2.4 Giải pháp tổng hợp nghiệm

CHƯƠNG 3. THUẬT TOÁN NHẰM RÚT GỌN VÀ TÌM KIẾM LỜI GIẢI CHO BÀI TOÁN

Do khuôn khổ của Luận văn không cho phép nêu hết được những khái niệm và đặc biệt là thuật toán quan trọng. Chương này sẽ nêu ngắn gọn hai kỹ thuật và thuật toán quan trọng nhất cho CSPs: Rút gọn và Tìm kiếm.

3.1. Một số thuật toán nhằm rút gọn thuật toán.

Như chúng ta đã giải thích ở chương 2, rút gọn bài toán là quá trình loại bỏ những giá trị và làm chặt ràng buộc trong CSP mà không loại bỏ nghiệm. Ý tưởng cơ bản là chúng ta loại bỏ những giá trị hay những nhãn kết hợp dư thừa, những thành phần không xuất hiện trong nghiệm. Sau đó, chúng ta tiếp tục xét đến ràng buộc trên tập biến S và tập các nhãn kết hợp hợp lệ trong S. Như vậy chúng ta đã rút gọn CSP ban đầu thành một bài toán tương đương- bài toán có chung bộ nghiệm với bài toán ban đầu- với hy vọng là dễ giải hơn.

Mặc dù việc rút gọn bài toán rất hiếm khi đạt được nghiệm, tuy nhiên nó giúp cho việc giải CSP dễ dàng hơn rất nhiều. Nó có thể dùng trong quá trình tiền xử lý, tức là nó có thể được dùng trước khi bất kỳ kỹ thuật nào khác được áp dụng. Nó cũng có thể được dùng trong thời gian tìm kiếm – bằng cách cắt một số không gian tìm kiếm sau khi mỗi nhãn đã được hoàn tất. Đôi khi chúng ta có thể giảm được một lượng đáng kể không gian tìm kiếm nhờ việc rút gọn bài toán.

Chúng ta có thể tổng quát khi áp dụng rút gọn với việc tìm kiếm sẽ đạt được những điều sau:

(1) Giảm không tìm kiếm

Vì cỡ của không gian tìm kiếm được đo bằng tích của toàn bộ cỡ của miền biến trong bài toán, rút gọn bài toán có thể giảm không gian tìm kiếm bằng cách giảm cỡ của miền biến.

(2) Tránh tìm kiếm lặp lại các nhánh cây thừa

Những giá trị và nhãn kết hợp thừa được thể hiện trên các nhánh và các *nhánh(path)* mà sẽ dẫn đến vô nghiệm. Điều này có thể được loại bỏ bằng việc rút gọn bài toán.

(3) Nhận ra các bài toán vô nghiệm

Nếu thuật toán rút gọn bài toán trả về một CSP mà có ít nhất một miền rỗng, khi đó có thể kết luận rằng bài toán đó là vô nghiệm mà không phải làm gì nữa.

Và từ đó có các thuật toán thực thi áp dụng để loại bỏ giá trị dư thừa từ các miền, và một số các nhãn kết hợp từ các ràng buộc như thuật toán đưa CSPs về chuẩn dạng: NC (Node-consistent), AC(arc-consistent), DAC (Directional arc-consistent), PC (Path-consistent), DPC (Directional path-consistent) là vô cùng quan trọng.

3.2. Một số thuật toán nhằm tìm kiếm lới giải cho bài toán

Như đã giới thiệu ở chương 2 rằng việc tìm kiếm là một trong những chiến lược được quan tâm nhất trong khi giải CSP. Chúng ta có phân ra thành các chiến lược tìm kiếm cơ bản sau:

(1) Các chiến lược tìm kiếm tổng quát

Những chiến lược này được phát triển trong những ứng dụng thông thường, và nó không dùng ràng buộc để đạt tính hiệu quả. Có hai chiến lược trong phần này:

- Tìm kiếm quay lui tuần tự (chronological backtracking)

- Tìm kiếm mở rộng lặp (iterative broadening-IB)

(2) Các chiến lược nhìn về phía trước (lookahead)

Chiến lược lookahead sẽ thực hiện việc rút gọn bài toán thông qua việc áp dụng các ràng buộc. Chiến lược này dựa trên thực tế rằng biến và ràng buộc là hữu hạn, và ràng buộc có thể được áp dụng kết hợp với nhau nhiều lần. Có ba chiến lược trong phần này:

- Kiểm tra phía trước (Forward Checking- FC)
- AC-kiểm tra phía trước có định hướng (Directional AC-L)
- AC-kiểm tra phía trước (AC-L)

(3) Các chiến lược thu thập thông tin trong khi tìm kiếm

Chiến lược sẽ ghi lại các tình huống dẫn đến lỗi bất cứ khi nào quay lui cần đến trong thời gian tìm kiếm. Điều này giúp chúng ta tránh được những nhánh lỗi đã biết. Chiến lược này khám phá ra rằng nhiều cây con tương tự với những nhánh khác đã xét. Có hai chiến lược trong phần này:

- Quay lui định hướng phụ thuộc (BackJumping - BJ)
- Thuật toán học từ phần không gian đã xét (Learning nogood compound labels- LNCL)

PHẦN III. BÀI TOÁN NGƯỜI CHƠI GÔN

Bài toán "Người chơi gôn" đã và đang được nhiều sự quan tâm từ cộng đồng ràng buộc[2,6,13,14,16,20,21,25,31,34,35,39], nó được dùng rất nhiều để so sánh giữa các kỹ thuật, các phương pháp khác nhau trong CSPs. Trong phần này, phần đóng góp chủ yếu của Luận văn, sẽ cố gắng trình bày hệ thống để phù hợp với các tiêu chí cho việc giải SGPs: các mô hình khác nhau, các kỹ thuật khác nhau, các phương pháp giải khác nhau, đồng thời sẽ cố gắng so sánh giữa chúng khi có thể.

Chính vì vậy, trình bày phần này là công việc khá khó khăn. Đầu tiên, *Chương 1*, Luận văn sẽ giới thiệu bài toán cùng với những kiến thức cần thiết cho những phần sau. *Chương 2* giới thiệu phương pháp loại bỏ đối xứng tĩnh trong bài toán vì hầu như các phương pháp khác đều sử dụng một phần (hoặc toàn bộ) các kỹ thuật trong phần này, đồng thời cũng đưa ra 2 ràng buộc dư thừa mới để áp dụng cho việc giải SGP. *Chương 3*, Luận văn bàn luận về các mô hình quan trọng và thường được dùng nhất trong cộng đồng ràng buộc, từ đó cũng đưa ra những ưu-nhược điểm của từng mô hình khi áp dụng giải SGP. *Chương 4* và *chương 5* nói về kỹ thuật loại bỏ đối xứng động khi giải SGP, nhưng do tính đặc thù của các phương pháp ta tách ra làm hai chương, ở đây nêu ra hầu như các phương pháp động thường được dùng để giải SGP. *Chương 6* là chương nói về loại bỏ đối xứng tĩnh khi giải SGP. Sở dĩ chương này được sắp xếp sau vì nó gồm phương pháp sẽ được so sánh với những phương pháp đã trình bày phần trước. *Chương 7* giới thiệu một trường hợp đặc biệt cho SGP; giải thích sự liên quan thú vị giữa SGP và hình vuông Latin trực giao (MOLS), đồng thời cũng đưa ra một thuật toán để giải cho trường hợp đặc biệt, từ đó rút ra một số kết luận và cũng khẳng định thuật toán có thể xây dựng nên tập MOLS(n) trong trường hợp n là số nguyên tố.

CHƯƠNG 1. GIỚI THIỆU BÀI TOÁN

1.1. Giới thiệu

Bài toán người chơi gôn (Social Golfer Problem- SGP, hãy xem bài số 10 trong [39]) đã được sự chú ý đặc biệt của cộng đồng lập trình ràng buộc kể từ khi nó được gửi bởi *bigwind777@aol.com* (Bigwind777) tại *sci.op-research* năm 1998:

> *Có 32 người chơi gôn muốn chia thành 8 nhóm (mỗi nhóm có 4 người) cho mỗi tuần, sao cho hai người bất kỳ sẽ chơi cùng nhóm với nhau nhiều nhất một lần. Hỏi có tối đa bao nhiêu tuần họ có thể chơi cùng nhau?*

Chúng ta có thể chỉ ra rằng họ không thể chơi quá 10 tuần, bởi vì mỗi người khi đó sẽ phải chơi với 30 người và chỉ còn lại 1 người chưa chơi. Bài toán này đã được cộng đồng ràng buộc tìm ra với lời giải 9 tuần. Tuy nhiên, thách thức còn lại là có tồn tại cho lời giải 10 tuần hay không?

Bài toán chưa dừng lại ở đó. Chúng ta có được bài toán tối ưu tổng quát hơn như sau:

> *Có **n** người chơi gôn muốn chia thành **g** nhóm, mỗi nhóm có **s** người (**n**=**g**×**s**) cho mỗi tuần, sao cho hai người bất kỳ sẽ chơi cùng nhóm với nhau nhiều nhất một lần. Hỏi có tối đa bao nhiêu **w** tuần họ có thể chơi cùng nhau?*

Bài toán có thể được thể hiện bằng bộ ba *g*-*s*-*w* , không khó khăn để suy ra $w \leq (g*s-1)/(s-1)$. Bài toán người chơi gôn ban đầu chính là dạng 8-4-*w,* chúng ta cần tìm $\max\{w\}$. Vì bộ ba đại diện cho một lớp bài dạng *g*-*s*-*w* xuất phát từ bài toán người chơi gôn nên các bài toán dạng *g*-*s*-*w* cũng được coi là bài toán người chơi gôn.

Khi (*s*-1)*w*=*gs*-1, bài toán sẽ phải giải quyết trường hợp mỗi tay gôn sẽ phải chơi với mỗi tay gôn khác đúng một lần. Điều này tương ứng với bài toán *resolvable Balanced Incomplete Block Design*, phần I.1 trong [9].

Có lẽ bài toán được nhiều người biết đến nhất là bài toán Kirkman's Schoolgirl, được đưa ra (và được giải) bởi Thomas Kirkman năm 1850[9,40]:

> " *Một cô giáo phải đưa 15 học sinh nữ đến trường hàng ngày. Cô ta muốn sắp thành 5 nhóm (mỗi nhóm 3 người). Yêu cầu của bài toán là sắp xếp chúng sao cho trong 7 ngày không có cặp nào đi với nhau quá một lần*".

Bài toán Kirkman's Schoolgirl tương đương với SGP 5-3-7. Một nghiệm của bài toán như sau:

Sun	ABC	DEF	GHI	JKL	MNO
Mon	ADH	BEK	CIO	FLN	GJM
Tue	AEM	BHN	CGK	DIL	FJO
Wed	AFI	BLO	CHJ	DKM	EGN
Thu	AGL	BDJ	CFM	EHO	IKN
Fri	AJN	BIM	CEL	DOG	FHK
Sat	AKO	BFG	CDN	EIJ	HLM

Bảng 1.1

SGP cũng là bài toán thực tế [20], một thành viên trong câu lạc bộ của Melbourne, Australia, khi muốn tổ chức một cuộc đi dạo và chơi gôn trong 5 ngày ở sông Murray, anh ta muốn sắp xếp các cặp đấu hàng ngày sao cho mỗi người sẽ đấu với người mới sau mỗi ngày, vấn đề này tương đương với bài toán *g-2-5*.

Chúng ta có thể tham khảo những trường hợp cụ thể cho SGP tại [41, 43], những kết quả này được giải nhờ nhiều những phương pháp khác nhau (cộng đồng lập trình ràng buộc, xây dựng và chứng minh bằng toán học, …).

1.2. Những vấn đề cần giải quyết trong bài toán

Mặc dù việc mô tả SGP là hoàn toàn dễ hiểu, tuy nhiên để giải quyết nó, hầu hết các giải pháp đều gặp khó khăn rất lớn ngay cả những trường hợp nhỏ. Có hai vấn đề chính mà khi giải bài toán ta phải đương đầu với độ phức tạp lớn đó:

- Bài toán người chơi gôn có tính đối xứng rất cao.
- Cấu trúc quay vòng ràng buộc nhằm đảm bảo bất kỳ hai người chơi không chơi cùng với nhau (trong cùng một nhóm) quá một lần, gây khó khăn cho việc tạo ra bộ gán thành phần hợp lý.

SGP có độ phức tạp bài toán là NP-hard. Chính vì vậy mà chi phí cho việc tính toán sẽ tăng rất nhanh khi cỡ của bài toán tăng lên.

Nguyên nhân chính là các bài toán trong lập trình ràng buộc mang tính đối xứng cao, do vậy việc xử lý tính đối xứng là phổ biến và vô cùng quan trọng trong quá trình tìm nghiệm cho bài toán.

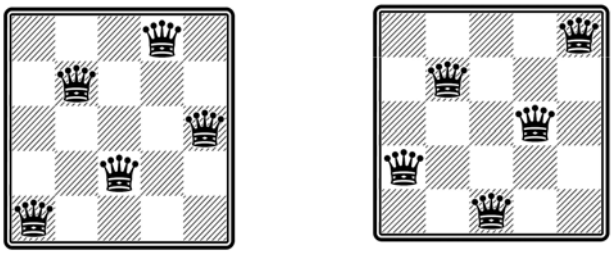
1.3. Sự đối xứng trong bài toán lập trình ràng buộc

1.3.1. Định nghĩa sự đối xứng trong CSPs

Nhiều bài toán thỏa mãn ràng buộc có tính đối xứng: bất kỳ một bộ gán nào cũng có thể được biến đổi thành tập các bộ gán khác tương đương (có tính đối xứng với bộ ban đầu). Hay nói một cách khác, đối xứng trong CSPs là ánh xạ từ nghiệm tới nghiệm, từ không phải là nghiệm tới không phải là nghiệm. Chúng là nhân tố hàng đầu làm giảm tính hiệu quả trong quá trình giải CSPs,

bởi vì quá trình tìm kiếm sẽ duyệt qua những vùng thông tin đối xứng nhiều lần (điều này giảm tính hiệu quả). Như vậy việc loại trừ đối xứng (symmetry breaking-SB) luôn được ưu tiên hàng đầu trong quá trình giải CSPs. SB làm giảm không gian tìm kiếm tăng tính hiệu quả cho quá trình tìm lời giải.

Chúng ta hãy cùng xem một ví dụ cho bài toán 5-quân hậu, Hình 1.2[8]:



Hình 1.2: Bài toán 5 quân hậu

Ký hiệu $x_i=j$ khi quân hậu ở hàng thứ i, cột thứ j. Khi đó chúng ta nhận thấy rằng:

- Nếu chúng ta đổi $x_i=j \rightarrow x_j = i$ thì bài toán có nghiệm mới.
- Nếu chúng ta đổi $x_i=j \rightarrow x_{6-i}=j$ thì bài toán có nghiệm mới.
- Nếu chúng ta đổi $x_i=j \rightarrow x_i = 6-j$ thì bài toán có nghiệm mới.
- …

Như vậy chỉ cần từ một nghiệm, chúng ta có thể suy ra các nghiệm khác (có tính đối xứng).

Chúng ta có thể định nghĩa như sau:

Định nghĩa 1.1

Với bất kỳ một thể hiện của một CSP $\mathscr{P}$ = <Z, D, C>, một ánh xạ g ánh xạ một phép gán này đến một phép gán khác sao cho:

- Với tất cả phép gán thành phần, $A=\{v_i= a_i\}$ thì $g(A)=\{g(v_i= a_i)\}$

- Với tất cả phép gán đầy đủ A, nếu A là nghiệm của $\mathscr{P}$ nếu và chỉ nếu $g(A)$ là nghiệm của $\mathscr{P}$

Khi đó g là hàm đối xứng trong $\mathscr{P}$. ■

Trong CSPs có hai loại đối xứng đó là đối xứng biến và đối xứng giá trị. Tuy nhiên ta sẽ quay lại phần này ở chương 6 trong một mục phù hợp để giải bài toán SGP.

1.3.2. Các phương pháp loại bỏ đối xứng

Có hai phương pháp chính để loại bỏ đối xứng trong CSPs:

- Phương pháp thứ nhất, còn gọi là phương pháp tĩnh (Symmetry Breaking Statically)[3,30]. Các ràng buộc được thêm vào CSP để tạo ra sự "kiểm định" nhằm ngăn chặn việc tạo ra nghiệm đối xứng, và mỗi một nghiệm được sinh ra bởi một bộ gán tương ứng cho lớp nghiệm đó, do vậy chỉ còn lại nghiệm không đối xứng. Phương pháp này làm một công việc tương tự như việc phát biểu lại CSPs trước khi tìm kiếm nhằm giảm không gian tìm kiếm ban đầu cho CSP. Một ví dụ của phương pháp khi bài toán liên quan đến mô hình ma trận là thêm ràng buộc theo một trật tự từ điển nhằm loại bỏ đối xứng trong hàng và cột [17,23,38]. Điều bất lợi cho phương pháp này là việc đòi hỏi người giải toán cần có nhiều kinh nghiệm, ngay cả khi đó việc loại bỏ hoàn toàn đối xứng cũng hết sức khó khăn. *Nhưng cũng cần nhấn mạnh rằng phương pháp này rất dễ dàng để kết hợp với bất kỳ một phương pháp nào khác*.
- Phương pháp thứ hai là phương pháp loại bỏ đối xứng động (Symmetry Breaking Dynamically)[12,31,32,38]. Nó được thực hiện bằng cách thêm ràng buộc trong thời gian tìm kiếm nhằm cắt bỏ những nhánh cây

thừa mà nó phát hiện ra nhờ việc cập nhật những ràng buộc được thêm vào. Trong đó, có hai cách tiếp cận là loại bỏ đối xứng trong thời gian tìm kiếm (Symmetry Breaking During Search - SBDS)[19] và loại bỏ đối xứng nhờ việc nhận ra sự ưu thế (Symmetry Breaking via Dominance Detection - SBDD) [6,14,16,19,20,34]. Trước khi quay lui, SBDS thêm ràng buộc loại bỏ đối xứng vào CSP nhằm loại bỏ mọi trạng thái đối xứng gây ra sự quay lui (sẽ không lặp lại với những trạng thái tương tự trong tương lai). Còn với SBDD, bất cứ khi nào thuật toán tìm kiếm tạo ra một trạng thái mới, nó sẽ kiểm tra xem nó có bị mất "lấn át" bởi một nút nào đó hay không (sẽ không lặp lại với những trạng thái tương tự trong quá khứ).

Như vậy, tất cả các hướng tiếp cận trên đây đều nhằm tránh những phần đối xứng, nhằm tìm ra một nghiệm trong tập nghiệm đối xứng, để từ đó chúng ta có thể phục hồi toàn bộ nghiệm nhờ tính đối xứng. Điều này làm tăng tính hiệu quả trong quá trình tìm kiếm nghiệm vì không gian tìm kiếm là rất lớn.

1.4. Sự đối xứng trong SGP

Như trên đã nói (phần 1.2) SGP có hai khó khăn chính, trong đó khó khăn thứ hai là do yêu cầu của bài toán, ta không thể thay đổi được yêu cầu đó. Trong Luận văn này sẽ tập trung giải quyết khó khăn thứ nhất, đó chính là tính đối xứng cao trong SGP.

SGP được đại diện bởi bộ ba *g-s-w*. Bài toán có $(s!)^{gw}(g!)^{w}w!(gs)!$ đối xứng (con số này sẽ tăng lên rất nhanh theo cỡ của bài toán). Khi giải CSPs có những mục đích cần chỉ ra:

- CSPs có nghiệm hay không
- Một nghiệm (bất kể nghiệm nào)

- Tất cả các nghiệm
- Nghiệm tối ưu

Tùy theo yêu cầu của bài toán, nhiều khi không cần phải loại bỏ toàn bộ đối xứng trong bài toán; mà có khi chỉ quan tâm đến việc phát hiện ra nghiệm càng nhanh càng tốt (như thách thức của SGP *8-4-w*, *w*=10?). Như vậy ở đây cần có sự thỏa hiệp giữa chi phí dùng để loại bỏ đối xứng và việc tiết kiệm những công việc phải làm. Ngoài ra, nói chung, loại bỏ được đối xứng càng sớm càng tốt.

Chúng ta có riêng định nghĩa đối xứng cho SGP[20]:

Định nghĩa 1.2

Hai nghiệm của SGP được coi là tương đương nếu một nghiệm đạt được từ nghiệm còn lại nếu áp dụng bất kỳ một chuỗi trong các thao tác sau đây:

1. Các tay gôn trong nhóm có thể bị thay đổi (φ_P)
2. Các nhóm trong tuần có thể bị thay đổi (φ_G)
3. Các tuần có thể bị thay đổi (φ_W)
4. Tên các tay gôn có thể bị thay đổi (có *n!* hoán vị) (φ_X). ■

CHƯƠNG 2. LOẠI BỎ ĐỐI XỨNG BẰNG PHƯƠNG PHÁP TĨNH TRONG BÀI TOÁN SGP

Khi giải bài toán SGP nói riêng và CSPs nói chung thường chúng ta gặp phải tính chất đối xứng trong nghiệm (trong cả ràng buộc [8]), khi đó có hai cách loại bỏ đối xứng: tĩnh và động. Tĩnh là phương pháp thêm ràng buộc trước khi tìm kiếm nghiệm, còn động là phương pháp thêm ràng buộc trước khi tìm kiếm nghiệm. Trong phần này sẽ giới thiệu phương pháp tĩnh trong việc loại bỏ đối xứng đối với riêng bài toán SGP. Phần này được giới thiệu trước vì hầu hết các phương pháp và các mô hình ít nhiều đều sử dụng chúng.

2.1 Loại bỏ đối xứng tĩnh cơ bản

Ta gọi đây là những đối xứng tĩnh cơ bản vì nó được sử dụng bởi hầu hết trong các mô hình ràng buộc [2,3,6,13,14,16,19,20,34].

SGP được thể hiện bởi bộ ba *g*-*s*-*w*:

Ta gọi $G_{i,j} \subseteq \{1,2,\ldots,n\}$ thể hiện *s* tay gôn ở nhóm thứ j ($1 \le j \le g$) trong tuần thứ *i* ($1 \le i \le w$). Từ đó ta có:

Do các nhóm trong tuần không có chung bất cứ cầu thủ nào, nên $G_{i,j} \cap G_{i,j'} = \varnothing$.

Do 2 tay gôn không thể gặp nhau quá một lần do vậy các nhóm cũng không chung quá một phần tử $1 \le \boldsymbol{i} < \boldsymbol{i}' \le \boldsymbol{w},\ 1 \le \boldsymbol{j} < \boldsymbol{j}' \le \boldsymbol{g} \quad | \boldsymbol{G}_{i,j} \cap \boldsymbol{G}_{i',j'} | \le 1$.

1	2	3		4	5	6		7	8	9
1	4	7		2	5	8		3	6	9
1	5	9		2	6	7		3	4	8
1	6	8		2	4	9		3	5	7

Bảng 2.1: Một lời giải bài toán cho trường hợp 3-3-4

Sau đây là những ràng buộc để loại bỏ đối xứng:

- Cố định tuần thứ nhất.

  Với mỗi $G_{1,i}$ có thể được cố định bởi { $i.g$+1, …, i.(g+1)}, ví dụ trong trường hợp bài toán là 3-3-w, nhóm $G_{1,2}$ chính là {4,5,6}.

  Bài toán chỉ còn $w.(s!)^g.(g!)$ đối xứng!

- *Đối xứng 1* (φ_P) có thể được loại bỏ bằng cách thiết lập trật tự bên trong mỗi nhóm (tăng dần).

  Đối xứng 2 (φ_G) có thể được loại bỏ bằng cách thiết lập trật tự giữa các nhóm trong tuần:

  Gọi $M1_{i,j}$ là phần tử nhỏ nhất trong Gi,j. Sau đó, với mỗi tuần i, chúng ta thêm ràng buộc $M1_{i,j} < M1_{i,j+1}$ (cho mọi $j < s$). Trong bảng 2.1 phần tử nhỏ nhất trong nhóm 1 là 1, nhóm 2 là 2 (trừ tuần đầu là 4) và nhóm 3 là 3 (trừ tuần đầu là 7)

- *Đối xứng 3* (φ_W) có thể được loại bỏ bằng một cách tương tự nhờ thiết lập ràng buộc giữa các tuần:

  Buộc $M1_{i,1}$ (tay gôn đầu tiên trong nhóm thứ nhất) là 1, và gọi phần tử nhỏ thứ hai trong $G_{i,1}$ là $M2_i$. Chúng ta lại thêm ràng buộc $M2_i < M2_{i+1}$ (cho mọi $i < w$). Trong bảng 2.1 phần tử nhỏ thứ hai trong nhóm 1 của tuần 1 là 2, phần tử nhỏ thứ hai trong nhóm 1 của tuần 2 là 4 và cho tuần 3 là 5, tuần 4 là 6.

- Nhóm đầu tiên của tuần thứ 2 có thể được cố định với những tay gôn nhỏ nhất có thể {1,s+1, …,(g-1)s+1}.

  Ví dụ như trong bảng 2.1, nhóm đầu tiên trong tuần thứ hai là {1, 4, 7}.

- Các tay gôn đầu tiên trong s nhóm đầu của mỗi tuần (từ tuần thứ 2) cũng có thể được cố định với các giá trị nhỏ nhất theo một trật tự (chúng là các tay gôn 1…s)

Trong bảng 2.1, tuần thứ hai trở đi, tay gôn 1 luôn ở nhóm số 1, tay gôn 2 luôn ở nhóm số 2, và tay gôn 3 luôn ở nhóm số 3.

Thật không may, khi kết hợp tất cả các kỹ thuật trên vẫn không loại hết được đối xứng giữa các tay gôn (đối xứng 4). Chúng ta hãy xem ví dụ sau [6], với trường hợp 5-2-2, cho 2 nghiệm:

1 2	3 4	5 6	7 8	9 10
1 3	2 4	5 7	6 9	8 10

1 2	3 4	5 6	7 8	9 10
1 3	2 5	4 6	7 9	8 10

Cả hai nghiệm trên đều thỏa mãn mọi ràng buộc loại bỏ đối xứng ở trên nhưng nghiệm thứ hai có thể được suy ra từ nghiệm thứ nhất thông qua các hàm sau:

$\varnothing_X = \{1\to7, 2\to8, 3\to9, 4\to10, 5\to1, 6\to2, 7\to3, 8\to4, 9\to5, 10\to6\}$

$\varnothing_G = \{1\to4, 2\to5, 3\to1, 4\to2, 5\to3\}$

$\varnothing_W = \{1\to1, 2\to2\}$

- Như vậy, *đối xứng thứ 4* (φ_X) khó xử lý hơn rất nhiều, chúng cần kỹ thuật loại bỏ đối xứng bằng phương pháp động: SBDS hay SBDD. Chúng ta sẽ thảo luận chúng ở phần sau.

Cần chú ý rằng khi áp dụng tất cả các kỹ thuật và các ràng buộc trên đều không làm mất tính bảo toàn nghiệm của bài toán (có nghĩa là nếu bài toán có nghiệm thì khi áp dụng các kỹ thuật trên sẽ vẫn có nghiệm. Và ngược lại, khi bài toán không có nghiệm khi áp dụng các kỹ thuật trên sẽ vẫn không có nghiệm).

2.2 Loại bỏ đối xứng tĩnh bằng kỹ thuật hạn chế miền (ND)

Ta có đưa ra một ràng buộc dư thừa mới cho bài toán này. Hãy cùng xem bài toán khi ở dạng *5-4-w*:

Tuần đầu tiên

{{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16},{17,18,19,20}}

và một tuần bất kỳ trong bài toán sẽ có dạng

{{1,?,?,?},{2,?,?,?},{3,?,?,?},{4,?,?,?},{?,?,?,?}}

Hãy coi $G_{i,j}(k)$ là phần tử thứ k trong nhóm $G_{i,j}$*(được sắp theo trật tự tăng)* trong mô hình CLP(FD) (Constraint Logic Programming over Finite Domains [29]). Từ nhận xét rằng $G_{i,2}(3)$ không thể nhận giá trị trong {17,18,19,20}; nếu không, miền của $G_{i,2}(4)$ sẽ rỗng (vì $G_{i,2}(3) < G_{i,2}(4)$). Vì vậy $G_{i,2}(3)$ sẽ phải nhỏ hơn 17. Tương tự như vậy, nếu $G_{i,2}(3)$ nhận giá trị trong {5,6,7,8} miền của $G_{i,2}(2)$ sẽ rỗng. Vì vậy ta rút ra kết luận:

$$8 < G_{i,2}(3) < 17$$

Bằng cách lập luận tương tự: $4 < G_{i,2}(2) < 13$

$$12 < G_{i,2}(4) < 21$$

Một cách tổng quát chúng ta có công thức sau:

$$\boldsymbol{s*(k-1) < G_{i,j}(k) < n-s*(s-k)+1},$$

$$1 \le k \le s,\ 1 \le j \le g,\ 1 \le i \le w$$

Chú ý khi bài toán có dạng *g-g-w*, mỗi biến có miền được giới hạn chỉ trong *g* giá trị (tương ứng với nhóm ở trong tuần thứ nhất). Ví dụ, trong bảng 2.2, bài toán dạng *4-4-w*, $G_{3,3}(2)$ có miền tương ứng là {5,6,7,8}, $G_{3,3}(3)$ miền tương ứng là {9,10,11,12}, và $G_{3,3}(4)$ miền tương ứng là {13,14,15,16}.

1	2	3	4		5	6	7	8		9	10	11	12		13	14	15	16
1					2					3					4			
1					2					3	?	?	?		4			

Bảng 2.2: SGP cho trường hợp 4-4-w

Cần nhấn mạnh lại rằng, kỹ thuật này là thêm ràng buộc dư thừa. Điều đó có nghĩa là kỹ thuật này bảo toàn nghiệm (không làm mất nghiệm). Cũng từ kỹ

thuật này, ta có ý tưởng cho việc giải SGP trong trường hợp đặc biệt *p-p-(p+1)* khi p là nguyên tố trong chương 7.

2.3 Loại bỏ đối xứng tĩnh bằng kỹ thuật cố định một số tay gôn (Fixing Players-F)

Kích cỡ của cây tìm kiếm chính là không gian tìm kiếm. Do vậy càng hạn chế được nhiều nhánh cây (bằng cách thêm các ràng buộc) thì không gian tìm kiếm sẽ càng nhỏ đi. Điều đặc biệt quan trọng là càng loại bỏ nhánh cây ở mức càng thấp (gần gốc) thì hiệu quả càng cao (không gian tìm kiếm càng nhỏ đi). Điều này lý giải tại sao một số lớn các ràng buộc tập trung vào tuần thứ 2 (phần 2.1).

Trong phần này, cũng đưa ra một ràng buộc nữa cho tuần thứ hai. Do các tay gôn ở nhóm cuối trong tuần thứ nhất sẽ luôn luôn là phần tử cuối cùng của các nhóm kể từ tuần thứ hai trở đi (vì các nhóm được sắp theo thứ tự tăng dần). Chính vì vậy mà ta có đề xuất thêm ràng buộc cho tuần thứ hai như sau: Các tay gôn trong nhóm cuối cùng ở tuần thứ nhất được gán lần luợt (theo trật tự tăng dần) vào các nhóm cuối của tuần thứ hai (xem bảng 2.3, các phần tử 10, 11, 12 được cố định trong tuần thứ 2)

1	2	3	4	5	6	7	8	9	10	11	12
1	4	7	2	?	**10**	3	?	**11**	?	?	**12**

Bảng 2.3: Kỹ thuật cố định phần tử cho trường hợp 4-3-w

Hẳn nhiên, nó là kỹ thuật không bảo toàn nghiệm cho trường hợp $g < s$ (Bảo toàn cho trường hợp $g=s$). Nhưng cũng thật ngạc nhiên, thực tế nó lại rất hiệu quả trong nhiều trường hợp. Tất nhiên ở đây có sự thỏa thuận giữa tính bảo toàn nghiệm và tính hiệu quả.

CHƯƠNG 3. CÁC MÔ HÌNH CÙNG PHƯƠNG PHÁP GIẢI SGP

Sở dĩ phần này được giới thiệu sau Chương 2 là vì hầu hết các mô hình đều sử dụng những kỹ thuật loại bỏ đối xứng tĩnh (phần 2.1). SGP có nhiều mô hình. Đây cũng là một lý do tại sao SGP tạo được sự thú vị. Trong phần này ta cần quan tâm đến hai vấn đề, đó là mô hình hóa bài toán và phương pháp giải (kỹ thuật, thuật toán). Mô hình hóa bài toán liên quan đến hai vấn đề: *chọn biến* và *chọn ràng buộc*. Trong thực tế, thì hai vấn đề này gần như nhau vì việc chọn biến phụ thuộc rất lớn vào việc chọn ràng buộc. Ta sẽ thấy, SGP có nhiều cách mô hình khác nhau, điều quan trọng là mô hình nào có ràng buộc dễ thực thi trong môi trường lập trình ràng buộc.

3.1 Mô hình dùng biến tập

Một ví dụ viết bằng ECL^{i}PS^e trong [39] có thể tìm tại [42] dùng biến tập (giá trị của biến là một tập) để thể hiện cho mỗi nhóm trong mỗi tuần. Thực tế, đây là một ý tưởng tốt, nhằm loại bỏ đối xứng trong nhóm (φ_P), các nhóm khi đó được coi như là một danh sách hay một mảng. Như vậy chúng ta có thể thể hiện mô hình bài toán:

- **Biến**: Nhóm được thể hiện như một mảng của các mảng các biến tập: Group[k][i] là nhóm thứ *i* của tuần thứ *k*.
- **Ràng buộc** :
  - Số phần tử trong mỗi biến là *s*: |Group[k][i]|=*s*
  - Các tập trong mỗi tuần không giao nhau:

    Group[k][i] $\cap$ Group[k][i']= $\varnothing$ sao cho $1 \le i \le g$ với mọi $1 \le k \le w$
  - Bất kỳ hai tập nào cũng chỉ giao nhau nhiều nhất 1 phần tử:

|Group[k][i] ∩ Group[k'][i']| ≤ 1 sao cho 1 ≤ *i, i'* ≤ *g* , 1 ≤ *k, k'* ≤ *w*

Với mô hình như vậy, và sử dụng những kỹ thuật loại bỏ đối xứng (phần 2.1), được thực thi trên ILOG Solver, máy PC Pentium/166MHz. Kết quả như sau:

- Trường hợp 8-4-4 có thể giải được
- Trường hợp 8-4-5 không tìm thấy lời giải trong nhiều giờ.
- Chỉ ra được trường hợp 4-3-5 không có nghiệm trong 5800 giây.

Có thể rút ra nhận xét rằng điểm bất lợi của mô hình này là việc gặp khó khăn khi thêm các ràng buộc để loại bỏ đối xứng.

3.2 Mô hình dùng biến nguyên

Một mô hình khác là mô hình dùng biến nguyên. Trong mô hình này, chúng ta cũng cần xác định:

- **Biến**: là mảng số nguyên **playInWeek**. Nó dùng để chứa tất cả các khả năng của các cặp khi chơi với nhau, ví dụ, cặp số 1 là giữa tay gôn *1* và tay gôn *2*, cặp số 2 là giữa tay gôn *1* và tay gôn *3*, … và cặp cuối cùng giữa tay gôn *n-1* và tay gôn *n*. Có (n-1)*n/2 biến như vậy. Giá trị được gán cho biến **playInWeek[pair(i,j)],** nó trả về số tuần mà **pair(i,j)** gặp nhau.
- **Ràng buộc**: chúng ta phải đối mặt với việc biểu diễn ràng buộc trong mô hình này.
  - Chúng ta cần một lượng lớn các ràng buộc để đảm bảo tính bắc cầu: nếu *i*, *j* chơi cùng với nhau trong tuần *k* và *j*, *l* chơi cùng với nhau trong tuần k, thì khi đó *i*, *l* cũng chơi cùng với nhau.
  - Chúng ta cũng cần ràng buộc để đảm bảo rằng mỗi tay gôn sẽ chơi với *s*-1 tay gôn khác trong mỗi tuần. Chúng ta cần những

Formatted: Font color: Black
Formatted: Font: 14 pt, Not Italic, Font color: Black
Formatted: Font color: Black
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic

ràng buộc như vậy cho mọi tay gôn trong mọi tuần, hơn là cho các nhóm như mô hình trước. Hay nói một cách khác, mô hình không cần ràng buộc để đảm bảo mỗi cặp tay gôn được gán chung một nhóm nhiều nhất một lần.

- Chúng ta cũng có thể dùng ràng buộc để loại bỏ đối xứng giữa các tuần. Chúng ta gọi *j* là tay gôn nhỏ nhất chơi với tay gôn 1 ở tuần thứ k, **playInWeek[pair(1,j)]**=*k*, và *l* là tay gôn nhỏ nhất chơi với tay gôn 1 ở tuần thứ *k*+1, **playInWeek[pair(1,j)]**=*k*+1, khi đó *j*<*l*.

Ràng buộc cho tuần thứ nhất cũng được mô hình này sử dụng, tuy nhiên sẽ không có khái nhiệm nhóm thứ nhất, nhóm thứ 2, …

Mô hình này ít tính đối xứng hơn mô hình tập. Chúng không có đối xứng giữa các thành viên trong nhóm, và cũng mất sự đối xứng giữa các nhóm trong tuần vì thực tế nhóm không được thể hiện trong mô hình này.

Với mô hình này, kết quả như sau:

- Chỉ ra được trường hợp 4-3-5 không có nghiệm trong 570 giây (nhanh hơn 10 lần so với mô hình trước)
- Tuy nhiên số ràng buộc đảm bảo tính bắc cầu sẽ tăng lên rất nhanh và đòi hỏi nhiều không gian bộ nhớ. Chính điều này mà mô hình không thể SGP cho trường hợp 8-4-9.

3.3 Mô hình kết hợp giữa biến tập và biến nguyên

Khó khăn chính của mô hình biến nguyên là việc mô tả ràng buộc bắc cầu. Tuy nhiên nó lại không xuất hiện ở mô hình tập, bởi vì các tay gôn chơi với nhau trong một nhóm được sắp tự động trong cùng một tập, và trong mô hình

biến nguyên thì lại không thể hiện nhóm. Vì vậy chúng ta có thể kết hợp hai mô hình để giải quyết bài toán.

- **Biến**: **PlayWith[i][k]** để chỉ tập các tay gôn chơi cùng tay gôn *i* trong tuần thứ *k* (bao gồm chính tay gôn *i*). Chú ý rằng biến tập này khác với mô hình tập ban đầu (một tay gôn cho mỗi tuần thay cho mỗi nhóm cho mỗi tuần). Do đó, chúng ta tránh được đối xứng trong nhóm.
- **Ràng buộc**:
  - Để đảm bảo cỡ của mỗi nhóm là s, chúng ta có thể dùng biến **PlayWith** hoặc **playInWith**. Mô hình không cần bất kỳ ràng buộc bắc cầu nào
  - Do chúng ta dùng biến **playInWith**, nên không phải dùng ràng buộc để mô tả mỗi cặp gặp nhau nhiều nhất một lần.

  Chúng ta cần ràng buộc để nối 2 tập biến:
  - Cặp (*i*, *j*) chơi cùng với nhau trong tuần *k* nếu và chỉ nếu **PlayWith [i][k] = PlayWith[j][k]**
  - Cặp (*i*, *j*) không gặp nhau trong tuần *k* nếu và chỉ nếu **PlayWith [i][k]** ∩ **PlayWith[j][k]** = ∅

Kết quả cho 3 mô hình, đều được chạy trên cùng máy PC Pentium/166MHz, chúng ta có thể xem Bảng 3.1, ở đó cột "QL" để chỉ khi tìm kiếm nghiệm chương trình phải thực hiện số lần quay lui (lỗi).

Số tuần (w)	Mô hình tập		Mô hình nguyên		Mô hình kết hợp	
	QL	Giây	QL	Giây	QL	Giây
2	22	0.03	16	0.11	16	0.11
3	36	0.04	107	0.27	95	0.26
4	91	0.10	91	0.57	89	0.51
5	2744282	5800	72544	570	51208	350

Bảng 3.1: Kết quả của 3 mô hình cho bài toán 4-3-w

3.4 Mô hình AMPL

Mô hình này có ở phần "Model" trong [38]. Ta diễn tả lại để mô hình của bài toán dễ nhìn hơn.

Mô tả bài toán bằng cách dùng lập trình tuyến tính với biến chỉ nhận giá trị 0-1.

AMPL model of `Maximum socializing on the 41 course'

June 8, 1998, J.P. Walser, Programming Systems Lab

- **Biến**:
  - set Players ordered := { 1..32 } ;
  - set Foursomes ordered := { 1..card(Players)/4 };
  - set Weeks ordered := { 1..8 } ;
  - P[*i*,*f*,*t*]=1 nếu và chỉ nếu tay gôn *i* trong nhóm *f* ở tuần thứ *t*.
  - M[*i*,*j*,*t*]=1 nếu và chỉ nếu tay gôn *i* gặp tay gôn *j* ở tuần thứ *t, i<j* (M[*i*,*j*,*t*] được coi như biến bổ trợ)
- **Ràng buộc**:
  - Mỗi tay gôn phải tham gia một nhóm trong mỗi tuần, có nghĩa là:

    $$\Sigma_{f \text{ in Foursomes}} P[i,f,t] = 1;$$

  - Mỗi nhóm có chính xác 4 tay gôn, có nghĩa là:

    $$\Sigma_{f \text{ in Players}} P[i,f,t] = 4;$$

  - Kết nối hai biến M và P, biểu diễn nó như sau:

    $$P[i,f,t]+P[j,f,t] - M[i,j,t] \leq 1;$$

    i in Players, j in Players, f in Foursomes, t in Weeks: i<j
  - Để đảm bảo 2 tay gôn bất kỳ gặp nhau nhiều nhất 1 lần, điều này có nghĩa là:

    $$\Sigma_{t \text{ in Weeks}} M[i,j,t] \leq 1;$$

i in Players, j in Players: i<j

- Cố định tuần thứ nhất

  fix { p in Players } P[p,int((p-1)/4)+1,1] := 1;

- Thêm ràng buộc loại bỏ đối xứng

  fix { p in Players, t in Weeks: t >= 2 and p <= 4 } P[p,p,t] := 1;

Chúng ta có thể tính được 5152 biến nhị phân và 22652 ràng buộc.

Với mô hình này dùng Local Search cho ta kết quả như sau:

- Tìm ra lời giải cho 8-4-7 trong vòng 30 giây
- Tìm ra lời giải cho 8-4-8 trong thời gian vài giờ. Nghiệm được thể hiện trong [39]

CHƯƠNG 4. LOẠI BỎ ĐỐI XỨNG BẰNG PHƯƠNG PHÁP THÊM RÀNG BUỘC TRONG THỜI GIAN TÌM KIẾM CHO SGP

Như phần trước đã giới thiệu (Chương 1 và 2), phương pháp động là phương pháp loại bỏ đối xứng trong quá trình tìm kiếm nghiệm. Trong phương pháp này có hai cách tiếp cận chính là: loại bỏ đối xứng trong thời gian tìm kiếm (Symmetry Breaking During Search - SBDS)[19] và loại bỏ đối xứng nhờ việc nhận ra sự ưu thế (Symmetry Breaking via Dominance Detection - SBDD) [14,16]. Hai phương pháp này đều có một nguyên lý chung: *đừng tìm kiếm nghiệm ở những phần không gian tìm kiếm mà nó có tính chất đối xứng với những phần đã được xét*. Vì vậy sẽ giới thiệu việc áp dụng hai phương pháp này cho bài toán SGP. Xin phép được nhắc lại là các phương pháp loại bỏ đối xứng được áp dụng chung cho CSPs, nên nó cũng được áp dụng cho SGP, vì SGP thuộc lớp CSPs.

4.1 Phương pháp SBDS

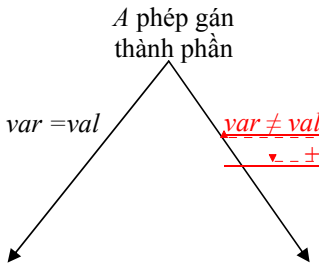
4.1.1 Giới thiệu SBDS

Trong bài [19], Smith B. đã giới thiệu phương pháp SBDS và chứng minh tính đúng đắn và đảm bảo trả về một nghiệm duy nhất từ tập các nghiệm đối xứng với nó. Bà đã áp dụng phương pháp này vào bài toán n-quân hậu và một số bài khác trong lập trình ràng buộc. Đây là một bài rất tuyệt vời để tham khảo. Trong phần này chỉ nêu những ý chính để áp dụng vào SGP, và bài toán cũng chính được Smith giải [35].

Chúng ta nhận thấy rằng một tính chất vô cùng quan trọng của đối xứng là nó bảo tồn nghiệm, có nghĩa là: với một phép gán đầy đủ cho A và bất kỳ một đối xứng g nào, thì $g(A)$ là nghiệm nếu và chỉ nếu A là nghiệm. Thông thường ta

mở rộng cho một phép gán thành phần từ A tới $A+(var=val)$, khi đó $g(A+(var=val)) = g(A)+g(var=val)$.

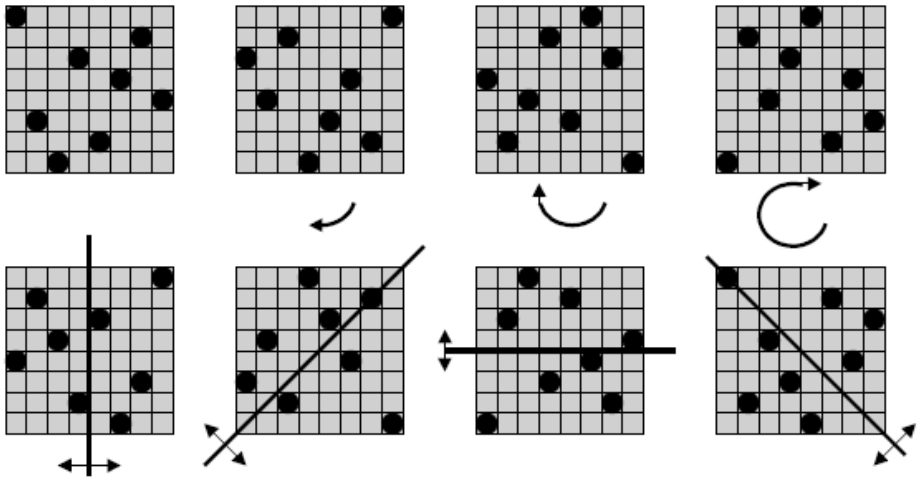
Nếu chúng ta cố gắng mở rộng cho từ A tới $A+(var=val)$ và phép gán thành phần này lỗi, việc tìm kiếm sẽ phải chuyển sang nhánh khác, nơi mà $var \neq val$. Chúng ta cũng thêm vào nhánh này ràng buộc $g(A) \rightarrow g(var \neq val)$ cho mỗi đối xứng g. Điều này có thể được phát biểu lại như sau: nếu đối xứng tương đương của phép gán A là đúng, thì $g(var \neq val)$ cũng sẽ đúng trong tất cả nhánh này. Nếu chúng ta chú ý rằng việc mở rộng từ A tới $A^+ = A +$ **vars[i]=j**, thì $g(A^+) = g(A) + g$**(vars[i]=j)**. Chúng ta xây dựng một biến boolean mới cho mỗi đối xứng g xem $g(A)$ có thỏa mãn hay không. Giá trị cho $g(A^+)$ là sự kết hợp của $g(A)$ và g**(vars[i]=j)**. Vì vậy chúng ta có thể tính $g(A)$ từng bước một.



Hình 4.1: Phương pháp SBDS trong khi tìm kiếm nghiệm

Chúng ta hãy xem hoạt động phương pháp thông qua bài toán 8-quân hậu. Giả sử phép gán đầu tiên của chúng ta là v_1=2, có nghĩa là quân hậu tại hàng đầu tiên ở cột số 2. Trong bài toán 8-quân hậu có 7 đối xứng: quay 90^0, 180^0, 270^0, sự phản xạ theo chiều ngang, chiều dọc và qua hai đường chéo chính. Ví dụ khi v_1=2, ta có nghiệm đối xứng tương ứng với 7 đối xứng trên một cách tương ứng là v_2=8, v_8=7, v_7=1, v_8=2, v_1=7, v_2=1 và v_7=8. Trong đó 4 giá trị cuối tương thích với v_1=2, nên chúng không được xét trong nhánh này (cắt nhánh dư thừa). Như vậy chúng ta biết ngay là đối xứng bằng phép quay

không bị loại bỏ bởi phép gán này. Hình 4.2 minh họa nghiệm đối xứng trong bài toán bằng phép lấy đối xứng từ nghiệm ban đầu.



Hình 4.2: Ứng với mỗi nghiệm của bài toán 8-quân hậu sẽ có 7 nghiệm đối xứng.

Sau phép gán v_1=2, giả sử chúng ta có phép gán thứ hai v_2=4 và nhánh này (nhánh trái) lỗi, như vậy ràng buộc $v_2 \neq 4$ được tạo ra. Chúng ta thêm ràng buộc $g(A) \rightarrow g(var \neq val)$ cho các đối xứng còn lại. Ví dụ, cho đối xứng quay 90^0, chúng ta có $g(v_1=2) \rightarrow g(v_2 \neq 4)$ điều này tương đương với $v_2=8 \rightarrow v_4 \neq 7$.

Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Not Italic

Chúng ta có thể tham khảo một bài báo phân tích rất cô đọng về loại bỏ đối xứng được áp dụng cho nhiều bài CSPs, trong đó có cả SGP [31].

Có hai thuận lợi trong việc dùng SBDS so với việc thêm ràng buộc vào mô hình:

- Nó có tính toàn diện: nếu chúng ta thêm một hàm mô tả đối xứng, chúng ta có thể loại bỏ đối xứng đó, trong khi chúng ta khó có thể đạt được hiệu quả tương tự nhờ việc thêm ràng buộc vào mô hình
- SBDS không tranh chấp, mâu thuẫn với các chiến lược tìm kiếm (trật tự các biến và các giá trị gán), nó chỉ làm nhiệm vụ ngăn những giá trị đối

xứng được gán khi tiến hành quay lui. Ngược lại, việc thêm ràng buộc vào mô hình có thể gây mâu thuẫn với chiến lược tìm kiếm.

4.1.2 SBDS cho SGP

Với mô hình được mô tả ở phần trước (phần 3.3), đối xứng là sự hoán đổi các nhãn của tay gôn và giữa các tuần, hoặc cả hai. Vì vậy, nếu có $n=g\times s$ tay gôn chơi trong w tuần, bất kỳ một nghiệm nào cũng cần phải loại bỏ $w!\times(gs)!$ nghiệm đối xứng. Chúng ta sẽ thấy rõ qua một trường hợp của SGP 8-4-10, số nghiệm đối xứng là 9.5×10^{41}. Con số này là quá lớn. Chính vì vậy người ta không thể giải quyết quá nhiều được chỉ bằng SBDS, việc thêm ràng buộc cho mô hình để loại bỏ đối xứng cũng vẫn là một ý tưởng tốt.

Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic

Chúng ta có thể loại bỏ đối xứng bằng một số kỹ thuật như đã nói ở trên (phần 2.1) sau đó áp dụng SBDS. Một đặc tính hữu ích của loại bỏ đối xứng là không cần thiết phải loại bỏ chúng hoàn toàn. Bởi vì chúng ta không thể đảm bảo rằng chỉ có nghiệm không đối xứng được tìm ra trừ trường hợp chúng ta loại bỏ hết được được, mà công việc chỉ ra được hết đối xứng trực tiếp là việc làm không dễ trong SGP. Bảng 4.3 sau chỉ rõ hiệu quả của SBDS hơn là chỉ thêm ràng buộc đối xứng vào mô hình.

g	*s*	*w*	Ràng buộc		SBDS	
			OL	Giây	OL	Giây
4	3	2	16	0.11	7	0.1
		3	95	0.87	39	0.2
		4	89	0.61	69	0.42
		5	51208	350	821	5.62
5	3	3	430	1.74	160	0.87
		4	821	4.66	391	2.68
		5	963	6.58	898	6.66
		6	48141	278	24507	125

Bảng 4.3: So sánh giữa hai phương pháp thêm ràng buộc vào mô hình và SBDS.

Đối xứng trong SGP là nguyên nhân chính gây khó khăn cho thuật toán tìm kiếm. Với các cách tiếp cận trên vẫn không thể giải được bài toán Kirkman's schoolgirls trong một thời gian chấp nhận được. Phương pháp SBDS, trong mỗi điểm chọn, SBDS mở rộng mô hình động bằng cách thêm các ràng buộc nhằm loại bỏ đối xứng. Đây là một phương pháp mới cho phép giải SGP nói riêng và bài toán tổ hợp nói chung, tuy nhiên như thế vẫn là chưa đủ để giải quyết tốt bài toán. Sau đây chúng ta sẽ có những cách tiếp cận mới cho SGP.

4.2 Phương pháp SBDD

4.2.1 Giới thiệu SBDD

SBDD là phương pháp có thể nhận ra đối xứng trong quá trình tìm kiếm. Tại mỗi thời điểm thuật toán tìm kiếm tạo ra một nút mới, nó sẽ kiểm tra xem nút đó có phải là một đối xứng với những nút đã được xét hay không. Nếu đúng, nhánh đó có thể bị cắt. Nếu không quá trình tìm kiếm diễn ra bình thường.

Mục đích của loại bỏ đối xứng là tránh khám phá phần không gian Δ mà có thể được ánh xạ từ phần đã được xét bằng một hàm đối xứng. Bởi vì nếu không chứa bất kỳ một nghiệm nào, thì Δ cũng không chứa nghiệm. Ngược lại, tất cả các nghiệm trong Δ có thể được suy ra từ phần đã xét. Trước hết, chúng ta cần nêu ra một số định nghĩa.

Định nghĩa 4.1

Gọi $X=\{x_1,\ldots, x_n\}$ là tập biến của mô hình, D(x) là miền của biến $x \in X$.

Bộ $\mathscr{P}^c=(D^c(x_1),\ldots, D^c(x_n))$ là trạng thái được chọn hiện tại của điểm *c*.■

Định nghĩa 4.2

Gọi $\mathscr{P}^c = (D^c(x_1),\ldots, D^c(x_n))$, $\mathscr{P}^{c'} = (D^{c'}(x_1),\ldots, D^{c'}(x_n))$ là hai trạng thái được xét.

- Nếu $\mathscr{P}^{c'}$ bao gồm $\mathscr{P}^{c}$ và ta ký hiệu $\mathscr{P}^{c} \subseteq \mathscr{P}^{c'}$ nếu và chỉ nếu $\forall x \in X: D^{c} \subseteq D^{c'}$
- Chúng ta đặt $MD^{c} = D^{c}(x_1) \times \ldots \times D^{c}(x_n)$.
- Một ánh xạ đối xứng $\varphi: MD^{c} \rightarrow MD^{c}$, chúng ta nói rằng $\mathscr{P}^{c'}$ ***"Lấn át"*** *(dominate)* $\mathscr{P}^{c}$ (với ánh xạ đối xứng φ) nếu và chỉ nếu $\varphi(\mathscr{P}^{c}) \subseteq \mathscr{P}^{c'}$. Khi đó chúng ta ký hiệu $\mathscr{P}^{c} \angle \mathscr{P}^{c'}$. ■

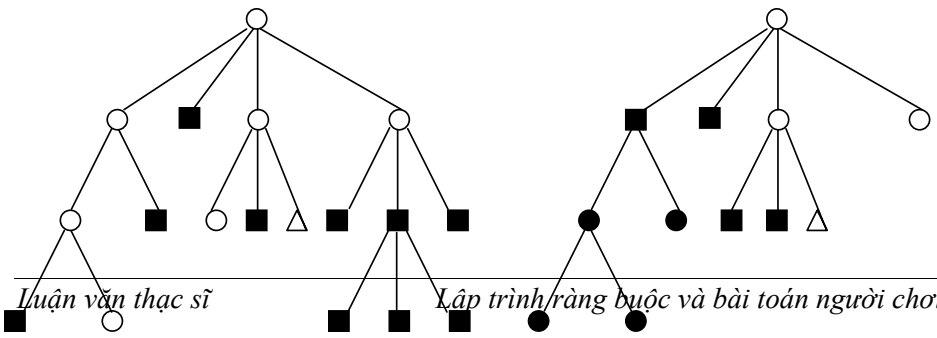
Hệ quả 1

Cho hai điểm chọn *c* và *c'*, trong đó *c'* là thế hệ sau của *c* trong cây tìm kiếm. Khi đó chúng ta có: $\mathscr{P}^{c'} \subseteq \mathscr{P}^{c}$.

Giải pháp mà SBDD dùng để cắt bớt phần đối xứng trong không gian tìm kiếm được dựa trên sự tích hợp sau:

- Một cơ sở dữ liệu $\mathscr{T}$ dùng để chứa toàn bộ thông tin của không gian tìm kiếm đã được duyệt.
- Một hàm chỉ định: Φ: $(P^{\Delta}, P^{\square}) \rightarrow$ {*false*, *true*} trả giá trị *true* nếu và chỉ nếu P^{Δ} bị *"Lấn át"* bởi $P^{\square}$ với một số hàm đối xứng φ
- Các đối xứng sẽ sử dụng thuật toán lan truyền, với mọi biến *x*, việc loại bỏ mọi giá trị *b* từ miền *x* sao cho $\Phi(P^{\Delta}[x=b], P^{\square}) = true$.

Ở mọi điểm chọn, chúng ta kiểm tra xem trạng thái P^{Δ} có bị *"Lấn át"* bởi một số trạng thái trong $\mathscr{T}$ không. Nếu như vậy, trạng thái hiện tại sẽ được bỏ qua, ngược lại chúng ta có thể dùng hàm Φ áp dụng thuật toán lan truyền. Chúng ta hãy xem Hình 4.4, minh họa cho SBDD:



Formatted: Not Superscript/ Subscript

Formatted: Font: .VnCommercial Script

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Times New Roman, 14 pt

(a) (b)

Hình 4.4: Minh họa cách thức hoạt động của SBDD: Nút trắng là nút đang đối được xét, nút đen là nút đã được xét hoàn toàn. Hình vuông là trạng thái trong $\mathcal{T}$, hình tròn thể hiện nó không ở trong $\mathcal{T}$ hoặc đã ở trong $\mathcal{T}$. Hình tam giác chỉ nút hiện tại đang xét. (a) Ban đầu, trạng thái Δ phải được kiểm tra thông qua toàn bộ các nút đã được xét. (b) Dùng DFS, nút hiện tại chỉ cần so sánh với các nút kề-trái (từ nút gốc tới Δ).

Chúng ta có hai cách tìm kiếm khi áp dụng phương pháp này, đó là: tìm kiếm theo chiều sâu (DFS-Depth First Search) và tìm kiếm tùy ý (theo một cách thức khác). Tuy nhiên khi áp dụng tìm kiếm tùy ý, số trạng thái cần lưu trữ trong $\mathcal{T}$ sẽ tăng lên rất nhanh và rất lớn. Do vậy SBDD sẽ chỉ phù hợp nhất với DFS. Phần kế tiếp sẽ giới thiệu cách thức thực hiện SBDD.

4.2.2 SBDD với DFS

Với DFS, chúng ta không cần lưu trữ toàn bộ trạng thái phía trước. Thay vào đó chúng ta chỉ cần lưu trữ các nút các nút anh em bên trái trong $\mathcal{T}$ để có thể quay lui. Chúng ta xét bổ đề sau:

Bổ đề 4.3:

Cho c là điểm chọn với trạng thái $\mathcal{P}^c = (D^c(x_1),\ldots, D^c(x_i),\ldots, D^c(x_n))$, trong đó i là chỉ số biến nhánh trong c, và $D^c(x_i) = \{v_1, \ldots, v_l\} \subseteq D(x_i)$.

Formatted: Font: Italic

Hơn nữa, ta ký hiệu $\mathscr{P}^{ck} = (D^c(x_1),\ldots, \{v_k\},\ldots, D^{ck}(x_n))$, $\forall 1 \le k \le l$ là trạng thái của con $c_1, \ldots, c_n$ của c. Cuối cùng, coi $\mathscr{P}^{c'}$ là trạng thái trong điểm chọn c' với $\mathscr{P}^{c'} \angle \mathscr{P}^{ck}$ ứng với một số $\forall 1 \le k \le n$.

Khi đó: $\mathscr{P}^{c'} \angle \mathscr{P}^{c}$.

Dùng bổ đề 4.3 khi kết hợp với DFS, ta có thể thực hiện một cách hiệu quả như sau: Chúng ta khởi đầu với $\mathscr{T} = \varnothing$ và tiếp tục quá trình cho mỗi điểm chọn như sau:

1. Kiểm tra trạng thái Phép chiếu của mỗi điểm chọn hiện tại c với tất cả các trạng thái trong $\mathscr{T}$. Nếu $\exists \mathscr{P} \in \mathscr{T}$ sao cho $\Phi(\mathscr{P}^c, \mathscr{P}) = true$ thì sinh lỗi.
2. Quá trình diễn ra bình thường mà không có điểm chọn
3. Khi quay lui: nếu có quá nhiều nút anh em được xét, thì thêm trạng thái hiện tại vào $\mathscr{T}$, nếu không xóa toàn bộ trạng thái của các nút anh em khác từ $\mathscr{T}$.

Hiệu quả của phương pháp này phụ thuộc vào số trạng thái được kiểm tra. Số trạng thái nhiều nhất chính là độ sâu của cây tìm kiếm và số phần tử lớn nhất trong miền.

Formatted: Font: (Default) Times New Roman, 14 pt

Formatted: Font: (Default) Times New Roman, 14 pt

4.2.3 SBDD áp dụng vào SGP

Trong phần này sử dụng mô hình biến tập (phần 3.1) cho mỗi nhóm, mô hình không chưa đối xứng φ_P. Để có thể nhận ra sự "lấn át"của các trạng thái với các đối xứng khác, chúng ta mô tả ba hàm nhận diện đối xứng Φ_G , $\Phi_{W,G}$ và $\Phi_{W,G,X}$ dùng trong khi tìm kiếm. Hàm $\Phi_{W,G}$ bao hàm việc kiểm tra được tiến hành bởi Φ_G, và $\Phi_{X,W,G}$ bao hàm $\Phi_{W,G}$.

- **Φ_G** Với chỉ số cho 2 tuần $1 \le i, j \le w$, Φ_G được dùng để kiểm tra xem tuần thứ j với trạng thái P^Δ có "lấn át"trạng thái $P^\square$ của tuần thứ j không với hàm đối xứng φ_G. Điều này được thực thi bằng cách kiểm tra xem tất cả các tay gôn trong tuần j với trạng thái $P^\square$ có thể được ánh xạ tới tuần j với trạng thái P^Δ không. Ví dụ trong Hình 4.5, trạng thái $P^\square$của tuần 1 không thể được ánh xạ tới trạng thái P^Δ của tuần 2, bởi vì tay gôn 1 và 3 ở cùng nhóm trong P, nhưng lại khác nhóm trong P^Δ. Tương tự như vậy, tay gôn 2 và 3 của tuần 3 là nguyên nhân làm cho trạng thái P không thể ánh xạ tới trạng thái P^Δ trong tuần 2.

△

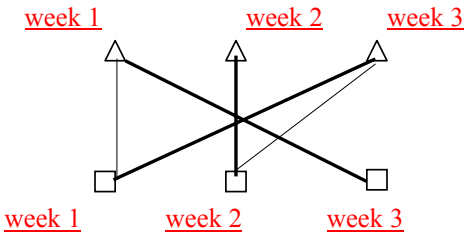
1 2 3	4 5 6	7 8 9
1 4 7	2	3
	1	2 8

□

1 2 3	4 5 6	7 8 9
1 4 7	2	3
	1	2 8

Hình 4.5:Hai trạng thái P^Δ và P.Mỗi trạng thái gồm 3 tuần trong SGP 3-3-3

- **$\Phi_{W,G}$** Dùng loại bỏ đối xứng φ_W và φ_G, hàm $\Phi_{W,G}$ được tạo như đồ thị G hai phía chứa các nút cho mỗi tuần trong P^Δ và P. Mỗi cạnh được chèn vào nếu và chỉ nếu một tuần của P "lấn át"một tuần trong P^Δ, khi dùng φ_G. Nếu G chứa . Hình 4.5 minh họa điều đó



Hình 4.5: P^Δ bị "lấn át"bởi $P^\square$

- **$\Phi_{X,W,G}$** Kết hợp φ_X với $\Phi_{W,G}$ (dùng để loại bỏ đối xứng được áp dụng cho (g.s)! hoán vị khác nhau). Để giảm chi phí cho việc kiểm tra, chúng ta cố định tuần thứ nhất. Chi phí cho việc gọi $\Phi_{X,W,G}$ là khá lớn, do vậy cần một tham số *q* để hạn chế mức kiểm tra đối xứng trong cây tìm kiếm. Nếu chúng ta thiết lập *q* là độ sâu của cây tìm kiếm thì nó sẽ kiểm tra tới tận nút lá.

Formatted: Font: Not Bold
Formatted: Font: Not Bold, Italic
Formatted: Font: Not Bold
Formatted: Font: Not Bold, Italic
Formatted: Font: Not Bold
Formatted: Not Highlight

4.2.4 Kết quả khi áp dụng SBDD cho SGP

Mô hình trên, đã được [14] thực thi trên ILOG Solver 5.0 và 400MHz Ultrasparc-II. Bảng 4.6 và 4.7 chỉ ra kết quả khi thực thi, thời gian tính bằng giây, cho nghiệm đầu tiên (t_1) và tất cả các nghiệm (t_{all}), số lần gọi hàm nhận dạng đối xứng $\Phi_{X,W,G}$ và $\Phi_{W,G}$. Trong phần *sym*, $\Phi_{W,G}$ được áp dụng để kiểm tra đối xứng cho φ_W và φ_G trong mỗi nút cây tìm kiếm. Vì đối xứng φ_X không được loại bỏ, do vậy có rất nhiều nghiệm đối xứng được nêu ra. Trong phần *nosym*, $\Phi_{W,G}$ cũng được áp dụng cho mỗi nút, ngoài ra chúng ta thêm hàm $\Phi_{X,W,G}$. Trong bảng có phần symmetries (số đối xứng được chỉ ra) , cp (choice points-số điểm chọn) và fails (chỉ số lỗi phát sinh).

Formatted: Font: Italic
Formatted: Font: Italic

problem	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	Φ	*symmetries*	*cp*	*fails*
sym								
2-4-3	48	0.00	0.03	226	0	0	195	148
3-4-3	2688	0.03	6.09	99454	0	0	28299	25612
4-4-3	1968	0.05	26.70	382120	0	2808	94845	92878
5-4-3	0	0.00	36.34	412456	0	3120	100389	200390
nosym								
2-4-3	1	0.00	0.04	226	47	47	195	194
3-4-3	4	0.01	10.00	99454	2687	2684	28299	28296
4-4-3	3	0.04	29.18	382120	1967	4773	94845	94843

5-4-3	0	0.00	36.28	412456	0	3120	100389	200390

Bảng 4.6: Kết quả cho SGP g-4-3

problem	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	Φ	*symmetries*	*cp*	*fails*
sym								
3-4-4	5184	0.01	8.71	74175	0	0	43755	38572
4-4-4	1296	0.01	20.53	140595	0	1296	82635	81340
5-4-4	432	0.01	25.90	132531	0	2160	75723	75292
6-4-4	0	0.00	30.76	114027	0	0	72267	72268
nosym								
3-4-4	2	0.01	136.31	74175	5183	5182	43755	43754
4-4-4	1	0.01	22.09	140595	1295	2591	82635	82634
5-4-4	1	0.02	26.51	132531	431	2591	75723	75723
6-4-4	0	0.00	30.71	114027	0	0	72267	72268

Bảng 4.7: Kết quả cho SGP g-4-4

Vì chi phí cho việc gọi hàm $\Phi_{X,W,G}$ khá lớn nên nếu áp dụng chúng cho mọi nút nhiều khi gây tốn rất nhiều thời gian, mặc dù số điểm chọn được giảm đi. Do vậy, có sự thỏa hiệp giữa việc giảm số điểm chọn và thời gian cho việc tìm và nhận ra đối xứng trong bài toán. Rất nhiều trường hợp chúng ta không nhất thiết phải áp dụng chúng tới tận nút lá, mà nhiều khi chúng tối ưu (về thời gian) ở một mức nào đó. Trong bài toán 4-4-4, mức 8 là mức tốt nhất, nếu áp dụng ở mức lá, sẽ tiêu tốn thời gian nhiều hơn. Xem Bảng 4.8 và 4.9.

problem	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	Φ	*symmetries*	*cp*	*fails*
nosym								
1	1	0.01	698.51	0	26	18	82	82
4	1	0.02	101.26	156	79	79	339	339
8	1	0.01	14.51	5292	1296	1296	4730	4730

leafs	1	0.01	22.09	1405959	1295	2591	82635	82634

Bảng 4.8: Kết quả cho SGP 4-4-4 ở các mức q khác nhau

problem	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	$\Phi_{X,W,G}$	*symmetries*	*cp*	*fails*
Nosym, level of $\Phi_{X,W,G}$ *=8*								
3-4-4	2	0.01	134.10	5283	1298	1297	6492	2891
4-4-4	1	0.01	14.51	5292	1296	1296	4730	4730
5-4-4	1	0.02	15.68	5291	1295	1296	4722	4722
6-4-4	0	0.00	17.16	5290	1295	1295	4714	4715

Bảng 4.9: Kết quả cho SGP g-4-4 ở mức q =8.

4.2.5 So sánh SBDS và SBDD

Trong phần trước (phần 4.2), để loại bỏ đối xứng, SBDS thêm ràng buộc vào mô hình trong thời gian tìm kiếm. Và một điểm quan trọng nữa là SBDS đòi hỏi phải liệt kê ra toàn bộ danh sách đối xứng. Mà số lượng đối xứng trong SGP quá lớn, do vậy không thể loại bỏ hoàn toàn bằng cách thêm tất cả các ràng buộc cần thiết. SBDS cũng không thể tìm ra được nghiệm cho trường hợp lớn hơn như 5-3-7 (Kirkman's School Problem).

Dùng SBDD cho SGP, chúng ta có thể phát hiện ra nghiệm duy nhất cho bài toán. Ngoài ra việc kết hợp chúng với mô hình đơn giản là điều rất đáng quan tâm.

SBDD dễ được thực thi vì tất cả nó đòi hỏi là định nghĩa cấu trúc trạng thái và hàm để kiểm tra trạng thái có bị "lấn át" hay không. Vì vậy, người dùng có thể tập trung vào thuật toán đối xứng mà không cần quá quan tâm đến ràng

buộc. SBDD đã giải bài toán Kirkman's School Problem trong vòng 13254 giây (khoảng 4 giờ).

CHƯƠNG 5. MỘT SỐ PHƯƠNG PHÁP LOẠI BỎ ĐỐI XỨNG ĐỘNG KHÁC CHO SGP

Deleted: hương

Deleted: ¶

Trong phần này sẽ giới thiệu 2 phương pháp khác để giải SGP bằng cách loại bỏ đối xứng động: Phương pháp Heuristic và Local Search. Chương 4 đã nói về SBDD và SBDS là hai phương pháp loại bỏ đối xứng bằng cách thêm ràng buộc trong thời gian tìm kiếm và chúng có nhiều điểm khá tương đồng nhau. Trong chương 5 hai phương pháp đều có nét rất riêng so với những phương pháp khác.

5.1 Loại bỏ đối xứng với Intelligent-Backtracking (IB)

5.1.1 Ý tưởng thuật toán

Chúng ta hãy bắt đầu bằng một ví dụ của SGP cho trường hợp 4-3-*w*. Nó cũng áp dụng những kỹ thuật cơ bản: cố định tuần thứ nhất, nhóm thứ nhất của tuần thứ hai, giữa các tuần cũng được sắp xếp (phần 2.1). Do vậy chúng ta có được sự khởi đầu như bảng 5.1:

1	2	3	4	5	6	7	8	9	10	11	12
1	4	7	2	?	?	3	?	?	?	?	?
1	?	?	2	?	?	3	?	?	?	?	?

Bảng 5.1

Chúng ta đạt được nhóm thứ nhất trong tuần thứ hai vì giữa các nhóm {4,5,6}, {7,8,9}, {10,11,12} được chọn vào nhóm thứ nhất thì tất cả chúng có vai trò như nhau, trong khi chúng ta muốn có một trật tự trong tuần thứ hai, do vậy chúng ta chọn 4 và 7.

Nếu chúng ta cứ tiếp tục gán như vậy có nghĩa là trong tuần 2 có {2,5,8}, và {3,6,9} thì sẽ dẫn đến 10,11, và 12 sẽ được gán vào nhóm cuối cùng. Điều

này là không thể được, vì lúc này giữa các phần tử còn lại đã có sự phân biệt. Chúng ta dễ nhận ra rằng phần tử 5 và 6 là không có sự phân biệt. Do vậy nếu 5 mà không đạt được nghiệm thì chúng ta cũng không thử với 6, vì nó sẽ dẫn đến một lỗi tương tự. Điều này cũng tương tự với các giá trị 10, 11 và 12. Còn giá trị 8 và 9 thì sao? Chúng thực ra cũng không có sự phân biệt với nhau và ngay cả với 5, và 6, vì chúng đều có đặc điểm chung: có một phần tử (đối tác) trong nhóm đã được gán trong tuần này. Như vậy chúng ta có thể chia ra làm 2 lớp có đặc diểm chung {5,6,8,9} và {10,11,12}. Như vậy chúng ta chỉ cần kiểm tra 2 giá trị trong 2 nhóm này để biết biểu thức có tồn tại nghiệm hay không (thay cho việc thử toàn bộ).

Vậy ý tưởng sẽ được tổng quát ra sao? Chúng ta hãy mô hình biểu thức *g-s-w* với các chuỗi nhóm S_{ij} như $<S_{1,1},S_{1,2},...S_{1,g},S_{2,1}, S_{2,2},...S_{2,g},... S_{w,1},S_{w,2},...S_{w,g}>$, ở đó *i* là chỉ số tuần, và *j* là chỉ số nhóm. Mỗi S_{ij} gồm *s* phần tử, phần tử thứ n trong S_{ij} được ký hiệu là $S_{ij}(n)$. Chúng ta cũng định nghĩa *Dom(i,j,n)* là miền có thể gán cho $S_{ij}(n)$ ở vị trí *i,j* và *n* theo: giá trị đã được gán trong tuần, *Assigned(i,j,n)*; các đối tác hiện tại trong nhóm, *GPartners(i,j,n)*; và tất cả các đối tác trước đó, *Ps(i,j,n)*:

$$Assigned\ (i,j,n) = \{x : \exists_{j1,\,n1}((j_1 \times s + n_1 < j \times s + n) \wedge S_{i,\,j1}(n_1) = x)\}\ . \quad (1)$$

$$GPartners\ (i,j,n) = \{x : \exists_{n1\,<\,n} S_{i,\,j}(n_1) = x\}\ . \quad (2)$$

$$Partners\ (v,i) = \{x : \exists_{i1\,<\,i} \exists_j (v \in S_{i1,\,j} \wedge x \in S_{i1,\,j} \wedge x \neq v)\}\ . \quad (3)$$

$$Ps(i,j,n) = \bigcup_{v \in GPartners\,(i,j,n)} Partners\,(v,i)\ . \quad (4)$$

$$Dom(i,j,n) = Golfers \setminus (Assigned(i,j,n) \cup Ps(i,j,n))\ . \quad (5)$$

Trong đó *Go1fers* dùng để chỉ các số nguyên từ 1 tới *g*s*, và *Partners(v,i)* là một tập các đối tác của tay gôn *v* trong tất cả các tuần trước tuần *i*.

Để phân biệt các tay gôn được chia ra các lớp như thế nào, chúng ta cần hai điều kiện:

1. Số các đối tác trong miền.
2. Số các đối tác đã được gán trong trong tuần

Hai giá trị, v_1 và v_2, với miền *Dom(i,j,n)*, được xem như là không phân biệt khi gán phần tử thứ *n* vào nhóm $S_{i,j}$ khi áp dụng cả (**6**) và (**7**) sau đây:

$$\#(Partners(v_1, i) \cap Dom(i, j, n)) = \#(Partners(v_2, i) \cap Dom(i, j, n)) . \quad (\mathbf{6})$$

$$\#(\boldsymbol{Partners}\ (\boldsymbol{v}_1, \boldsymbol{i}) \cap \boldsymbol{Assigned}\ (\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{n})) = \#(\boldsymbol{Partners}\ (\boldsymbol{v}_2, \boldsymbol{i}) \cap \boldsymbol{Assigned}\ (\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{n})) \quad (\mathbf{7})$$

Điều kiện (**6**) để phân biệt lớp {5,6,8,9} và lớp {10,11,12} trong ví dụ trên. Chúng ta sẽ giải thích điều kiện (**7**) với vị dụ sau đây: xét SGP cho 3-3-2:

{ {1,2}, {3,4}, {5,6} }

{ {1,3}, {2,5}, {4,6} }

{ {1,4}, {2,?}, {?,?} }

Khi đó phần tử $S_{3,2}(2)$ chỉ có thể được gán bởi 3 và 6, nhưng chỉ có 6 là dẫn tới nghiệm. Điều kiện (**6**) không thể phân biệt giữa chúng vì các đối tác tương ứng của chúng là {1,4} và {4,5}. Lý do chúng không thể giống nhau vì giá trị 3 có hai đối tác được gán trong tuần đang xét (1 và 4) trong khi giá trị 6 chỉ có một đối tác được gán (là 4). Do vậy khả năng dẫn tới nghiệm là lơn hơn. Đó chính là điều kiện (**7**) đã nêu.

5.1.2 Thuật toán

Thuật toán giải quyết SGP với *g-s-w* dùng thuật toán quay lui thông minh (Intelligent backtracking -IB) với 2 thủ tục dưới dạng mã giả.

Procedure *GolfersIB* (**In**: *w,g,s*, **Out**: *S*)
  for *i* **from** 1 to *w* do
    for *j* **from** 1 to *g* do
      $S_{i,j}(1) \leftarrow$ **first_value**(Dom($S_{i,j}(1)$))
      for *n* **from** 2 to *s* do
        AssignIB**(***S,i,j,n,Dom(i,j,n),* □**)**
      end for
    end for
  end for
end Procedure

Thủ tục *GolfersIB(w,g,s,S)* gọi thủ tục *AssignIB* sau:

Procedure AssignIB (**In/Out**:*S*, **In**: *i,j,n,Vals,Tried*)
  remove_first *v* **from** *Vals* **yielding** *RestVals*
  N1 $\leftarrow$ #(Dom(i,j,n) $\cap$ Partners(v,i))
  N2 $\leftarrow$ #(Assigned(i,j,n) $\cap$ Partners(v,i))
  ChoicePoint
    ($\langle N1,N2\rangle \notin$ *Tried*
    $S_{i,j}(n) \leftarrow v$
) or
    AssignIB**(***S,i,j,n, RestVals, Tried* $\cup$ {$\langle N1,N2\rangle$}**)**
  end ChoicePoint
end Procedure

Chú ý rằng thuật toán này có một nhược điểm rằng nó không bảo toàn tính nghiệm (có nghĩa là nó có thể làm mất nghiệm). Kết quả của thuật toán này sẽ được nêu ra và so sánh với phương pháp được đề xuất ở phần sau (phần 6.3.4)

5.2 Local Search cho SGP

5.2.1 Mô hình

Phương pháp này dùng biến quyết định để mô hình hóa bài toán $x[w, g, p]$ ký hiệu cho tay gôn vị trí p trong nhóm g của tuần w. Một lịch cho tay gôn σ là một phép gán các giá trị cho biến, giá trị $\sigma(x[w, g, p])$ ký hiệu lịch cho tay gôn đó. Ngoài ra, mô hình cần ràng buộc $m[a,b]$ để chỉ hay tay gôn a và b không gặp nhau quá một lần. Chính xác hơn có thể dung $\#_\alpha(a,b)$ để chỉ số lần tay gôn a và b gặp nhau:

$$\#_\alpha(a,b) = \#\{(w, g) \mid \exists p, p' : \sigma(x[w, g, p]) = a \ \&\ \sigma(x[w, g, p'])=b\},$$

Như vậy ràng buộc $m[a,b]$ đúng nếu $\#_\alpha(a,b) \leq 1$.

Dùng thêm ràng buộc

$$v_\alpha(m[a,b]) = max(0, \#_\alpha(a,b) - 1).$$

Như vậy trong Local Search, việc giải bài toán SGP tương đương với việc tìm giá trị nhỏ nhất của biến sau:

$$f(\sigma) = \sum_{a,b \in G} (v_\alpha(m[a,b])).$$

Trong đó G là tập $g \times p$ tay gôn. Một lịch σ sẽ là nghiệm của bài toán nếu $f(\sigma) = 0$.

5.2.2 Lân cận (Neighborhood) và thành phần Tabu

Lân cận trong Local Search bao gồm việc hoán đổi 2 tay gôn từ các nhóm khác nhau trong cùng một tuần. Tậpp hoán đổi có thể được định nghĩa như sau:

$$S = \{(\langle w, g_1, p_1\rangle, \langle w, g_2, p_2\rangle) \mid g_1 \neq g_2\}$$

Thành phần của Tabu có 3 ý tưởng chính

- Thứ nhất, danh sách tabu được thể hiện trong nhiều tuần khác nhau. Thành phần tabu gồm một mảng *tabu* trong đó *tabu*[w] thể hiện một danh sách tabu cho tuần w.

- Thứ hai, trong tuần *w*, danh sách tabu duy trì bộ ba ⟨ *a*, *b*, *i*⟩, với *a* và *b* là hai tay gôn và *i* thể hiện lần lặp đầu tiên mà hay tay gôn *a* và *b* hoán đổi trong tuần *w*. Các danh sách tabu chỉ lưu trữ các tay gôn, không lưu trữ vị trí ⟨ *w*, *g*, *p*⟩.
- Thứ ba, thời gian cho cặp tay gôn (*a*, *b*) trong danh sách tabu là động: Nó được tạo ngẫu nhiên trong khoảng [4,100]. Trong vòng lặp *k*, hai tay gôn *a* và *b*trong tabu được ký hiệu là:

$$tabu[w](\ a,b,k)$$

5.2.3 Thuật toán

Ở đây chỉ nêu thuật toán, phần giải thích được có thể tìm thấy trong [13]

```
SGLS(W, G, P)
    forall w ∈ W
        tabu[w] ← {};
    σ ← random configuration;
    σ* ← σ;
    k ← 0;
    s ← 0;
    while k ≤ maxIt  &  f(σ) > 0 do
        select (t1, t2) ∈ S^t(σ, k) ∪ S*(σ, σ*) minimizing f(σ[x[t1] ↔ x[t2]]);
        τ ← RANDOM([4,100]);
        tabu[week(t1)] ← tabu[week(t1)] ∪ {⟨σ(x[t1]), σ(x[t2]), k + τ⟩};
        σ ← σ[x[t1] ↔ x[t2]];
        if f(σ) < f(σ*) then
            σ* ← σ;
            s ← 0;
        else if s > maxStable then
            σ ←random configuration;
            s ← 0;
            forall w ∈ W do
                tabu[w] = {};
        else
            s++;
        k++;
```

Kết quả của phần này được so sánh với phương pháp được đề xuất ở phần sau (trong 6.3.2).

CHƯƠNG 6. LOẠI BỎ ĐỐI XỨNG BẰNG PHƯƠNG PHÁP TĨNH VÀ THÊM RÀNG BUỘC DƯ THỪA ĐỂ GIẢI SGP

Trong phần này sẽ tiến hành đối sánh những kết quả thử nghiệm và đánh giá. Phần này sẽ có hai phần: Phần thứ nhất với cách tiếp cận khá lạ cho việc giải SGP, đó là cách giải từ nhiều "*điểm nhìn*" (Multiple viewpoints). Chúng ta sẽ thấy được sự khéo léo trong cách biến đổi bài toán nhằm loại bỏ đối xứng. Phần thứ hai thực thi trên mô hình được nghiên cứu. Điều đáng nói ở đây là đã kết hợp với một số các ràng buộc được đề xuất (Phần 2.2 và 2.3), sau đó sẽ so sánh kết quả với một số phần đã trình bày.

Formatted: Font: Italic

6.1 Loại bỏ đối xứng trong SGP bằng nhiều điểm nhìn

Có hai kiểu đối xứng trong CSPs, đối xứng biến và đối xứng giá trị. Nói chung trong CSPs, đối xứng biến dễ diễn tả với ràng buộc loại bỏ đối xứng hơn đối xứng giá trị. Do vậy khi gặp phải đối xứng giá trị, phương pháp này muốn biến đổi bài toán để biến đối xứng giá trị đó thành đối xứng biến. Một ý tưởng rất linh hoạt. Chúng ta hãy cùng xem cách tiếp cận phương pháp để giải SGP bằng cách loại bỏ đối xứng tĩnh [17,25,26,27].

6.1.1 Một số khái niệm quan trọng

Thông thường một bài toán $\mathcal{P}$ trong CSPs có nhiều cách tiếp cận, tức là $\mathcal{P}$ có thể được công thức hóa bằng nhiều cách khác nhau. Vấn đề mấu chốt của việc công thức hóa là xác định biến và miền của biến. Việc lựa chọn biến và miền khác nhau là kết quả của việc xem xét bài toán $\mathcal{P}$ dưới góc độ khác nhau.

- Chúng ta định nghĩa "điểm nhìn" (*viewpoint*) là một cặp $(X, D_X)=V$, trong đó $X=\{x_1, \ldots, x_n\}$ là tập biến, và D_X là tập miền, $\forall x \in X$, ta có miền $D_X(x)$ tương ứng.
- Một phép gán $x \rightarrow a$ trong V có nghĩa là $a \in D_X(x)$, cũng như vậy $[x_{i1}, \ldots, x_{ik}] \rightarrow [v_1, \ldots, v_k]$ có nghĩa là $\{x_{ij} \rightarrow v_j | \ 1 \leq j \leq k\}$.

Khi công thức hòa bài toán $\mathcal{P}$ trong CSP, việc lựa chọn "điểm nhìn" không phải là tùy ý. Giả sử chúng ta gọi $sol(\mathcal{P})$ là tập nghiệm của $\mathcal{P}$, khi đó "điểm nhìn" V được gọi là phù hợp với $\mathcal{P}$ nếu và chỉ nếu tồn tại một tập con S trong tất cả các phép gán đầy đủ trong V sao cho tồn tại một ánh xạ 1-1 giữa S và $sol(\mathcal{P})$. Nói một cách khác, mỗi nghiệm trong $\mathcal{P}$ sẽ tương ứng với một phép gán đầy đủ trong V.

Một mô hình $\mathcal{M}$ của bài toán $\mathcal{P}$ là một cặp (V,C), trong đó V là một mô hình phù hợp của $\mathcal{P}$ và C là tập các ràng buộc trong V. Một ràng buộc có thể được xem như một tập con các biến trong V ánh xạ tới *true* hoặc *false*. Một nghiệm của $\mathcal{M}=(V,C)$ là tập các phép gán của tất cả các biến trong V sao cho ánh xạ các phép gán vào tất cả các ràng buộc là *true* (có nghĩa là tất cả các ràng buộc đều thỏa mãn).

6.1.2 Loại bỏ đối xứng bằng phương pháp nhiều "điểm nhìn"

6.1.2. 1 Định nghĩa kiểu đối xứng

Định nghĩa 6.1

Một **đối xứng biến** của CSP là một ánh xạ giữa tập biến X và chính nó, $\sigma: X \rightarrow X$, sao cho ánh xạ từ tập nghiệm tới tập nghiệm và từ tập không nghiệm tới tập không nghiệm. ■

Chúng ta hãy nhắc lại đối xứng trong bài toán SGP:

1. Các tay gôn trong nhóm có thể bị thay đổi (φ_P)
2. Các nhóm trong tuần có thể bị thay đổi (φ_G)
3. Các tuần có thể bị thay đổi (φ_W)
4. Tên các tay gôn có thể bị thay đổi (có *n!* hoán vị) (φ_X).

Để mô hình cho bài toán SGP, xét một "điểm nhìn" $V_1=(X, D_X)$, trong đó nó chứa biến $p_{i,k}$ cho tay gôn i ở tuần thứ k với $0 \le i \le n-1$ và $0 \le k \le w-1$ (Mô hình này đã được nói đến ở phần 3.1). Miền của biến $D_X(p_{i,k}) = \{0,\ldots, g-1\}$ chứa số thứ tự nhóm mà tay gôn chơi. Dùng "điểm nhìn" V_1, ta xem một nghiệm cho trường hợp 3-3-3 tại Bảng 6.1

Golfer / week	0	1	2	3	4	5	6	7	8
0	0	0	0	1	1	1	2	2	2
1	0	1	2	0	1	2	0	1	2
2	0	1	2	1	2	0	2	0	1

Bảng 6.1: Một nghiệm cho bài toán SGP trường hợp 3-3-3

Đối xứng 1(φ_P) có thể bị loại bỏ khi dùng V_1 vì khi đó mô hình không phân biệt vị trí các tay chơi trong nhóm. Tuy nhiên mô hình vẫn còn đối xứng 4 (φ_X). Do với một nghiệm bất kỳ của bài toán ta có thể thay đổi giá trị của 2 tay gôn bất kỳ, ta sẽ được nghiệm mới. Mà các tay gôn được coi như là biến trong V_1, vì vậy ta xác định được đây là đối xứng biến. Ví dụ chúng ta xem Bảng 6.1, với hay tay gôn 0 và 1, chúng ta có $[p_{0,0}, p_{0,1}, p_{0,2}] \rightarrow [0,0,0]$ và $[p_{1,0}, p_{1,1}, p_{1,2}] \rightarrow [0,1,1]$, chúng ta có thể đổi giá trị của hai tay gôn 0 và 1 để đạt được nghiệm mới $[p_{0,0}, p_{0,1}, p_{0,2}] \rightarrow [0,1,1]$, và $[p_{1,0}, p_{1,1}, p_{1,2}] \rightarrow [0,0,0]$. Trong ví dụ này, chúng ta dùng hàm đối xứng σ_1 như là ánh xạ tập nghiệm vào chính nó ngoại trừ $\sigma_1(p_{0,k}) = p_{1,k}$ và $\sigma_1(p_{1,k}) = p_{0,k}$ với k =0,1,2.

Đối xứng 3 (φ_W) cũng là đối xứng biến trong V_1 . Ví dụ như trong Bảng 6.1, các tuần xuất hiện như các biến, chúng ta có nghiệm $[p_{0,0},..., p_{8,0}] \rightarrow [0,0,0,1,1,1,2,2,2]$ và $[p_{0,1},..., p_{8,1}] \rightarrow [0,1,2, 0,1,2, 0,1,2]$. Bằng cách thay đổi biến gán giá trị ta được nghiệm mới $[p_{0,0},..., p_{8,0}] \rightarrow [0,1,2, 0,1,2, 0,1,2]$ và $[p_{0,1},..., p_{8,1}] \rightarrow [0,0,0,1,1,1,2,2,2]$. Trong ví dụ này, chúng ta dùng hàm đối xứng σ_2 như là ánh xạ tập nghiệm vào chính nó ngoại trừ $\sigma_2(p_{i,0}) = p_{i,1}$ và $\sigma_2(p_{i,1}) = p_{i,0}$ với $k = 0,1,\ldots 8$.

Đối xứng biến trong CSPs nói chung có thể được loại bỏ nhờ việc thêm vào ràng buộc nhằm tạo ra trật tự giữa các biến không được phân biệt. Trong ví dụ trên, chúng ta có thể loại bỏ đối xứng 4 (φ_X) giữa tay gôn 0 và tay gôn 1 bằng cách tạo ra thứ tự từ điển (*lexicographically*)[15,17,23] giữa hay tay gôn: $[p_{0,0}, p_{0,1}, p_{0,2}] <_{lex} [p_{1,0}, p_{1,1}, p_{1,2}]$. Tổng quát hơn, ta có thể đưa ra ràng buộc loại bỏ đối xứng là $[p_{i,0},..., p_{0,w-1}] <_{lex} [p_{i+1,0},..., p_{i+1,w-1}]$ cho mọi $0 \le i \le n-2$. Tương tự ta có thể loại bỏ đối xứng 3 (φ_W) bằng ràng buộc: $[p_{0,k},..., p_{n-1,k}] <_{lex} [p_{0,k+1},..., p_{n-1,k+1}]$ cho mọi $0 \le k \le w-2$.

Hai loại ràng buộc loại bỏ đối xứng trên tương ứng với việc loại bỏ đối xứng giữa hàng và cột trong ma trận. Trong nghiệm ở Bảng 6.1, các hàng và cột đã được sắp xếp thứ tự. Nếu thay đổi bất cứ một hàng hay cột nào cũng làm mất thứ tự.

Định nghĩa 6.2

Một đối xứng giá trị là khi một tập con $X' \subseteq X$ của các biến trong "điểm nhìn" (X,D_X) với $D_X(x) = D_X(x')$ với mọi x, x'$\in X'$. Nó là một ánh xạ giữa các tập giá trị miền, σ: $D_X(x) \rightarrow D_X(x)$ với x$\in X'$ sao cho ánh xạ từ tập nghiệm tới tập nghiệm và từ tập không nghiệm tới tập không nghiệm. Có nghĩa là có nhiều giá trị không được phân biệt trong miền giá trị của biến. ■

Trong bài toán SGP đối xứng 2 (φ_G) là đối xứng giá trị trong V_1. Ví dụ khi ta xét tập các biến $X' = \{ p_{0,0},..., p_{n-1,\,0} \} \subseteq X$ gồm toàn bộ các tay gôn trong tuần 0. Chúng ta có thể hoán vị toàn bộ giá trị các biến: 0→1, 1→2, 2→0, chúng ta đạt được nghiệm mới.

Nói chung, đối xứng giá trị khó được xử lý bằng ràng buộc loại bỏ đối xứng, vì chúng ta không biết trước được giá trị nào sẽ được gán cho biến cụ thể. Điều đó gây khó khăn cho việc thiết lập trật tự. Trong thực tế, nhiều khi nó cũng được khắc phục bằng cách gán trước giá trị cho một vài biến (càng nhiều càng tốt) mà không mất tính tổng quát. Từ đó quá trình tìm nghiệm sẽ được mở rộng từ những phép gán này. Tuy nhiên bất cứ trong trường hợp nào chúng ta không gán trước được giá trị, sẽ rất dễ sinh ra bỏ sót cơ hội loại bỏ đối xứng giá trị một phần và tốn thời gian tìm kiếm. Ví dụ, trong bài toán trên, không mất tính tổng quát, ta có thể gán:

$[p_{0,0},..., p_{n-1,\,0}] \rightarrow [0,\ldots,0,\ 1,\ldots,1,\ldots,g\text{-}1,\ldots,g\text{-}1]$ (mỗi giá trị được lặp lại s lần) và $[p_{0,k},..., p_{s-1,\,k}] \rightarrow [0,\ldots,s\text{-}1]$ với mọi $k \geq 1$.

Phép gán thứ nhất là để loại bỏ đối xứng giá trị trong tuần 0 và phép gán thứ hai là loại bỏ đối xứng giá trị một phần trong tuần 1, 2…Tuy nhiên sẽ rất ít giá trị được loại bỏ đối xứng khi s rất nhỏ so với g.

6.1.2. 2 Chuyển đổi "điểm nhìn" nhằm loại bỏ đối xứng giá trị

Trong phần này, ta sẽ chuyển đối xứng *giá trị* sang đối xứng *biến*, khi đó ràng buộc loại bỏ đối xứng sẽ diễn ra dễ dàng hơn. Điều này tương đương với việc chuyển giá trị miền của "điểm nhìn" thứ nhất thành biến của "điểm nhìn" thứ hai. Và như vậy hai loại đối xứng trong cùng một bài toán được loại bỏ đồng thời thông qua hai mô hình.

Formatted: Font: Italic

Formatted: Font: Not Italic

Formatted: Font: Italic

Quay trở lại bài toán. Với "điểm nhìn" V_1, chúng ta gán nhóm cho tay gôn. Như vậy nhóm xuất hiện như giá trị, còn tay gôn xuất hiện như biến. Chúng ta cũng có thể có "điểm nhìn" V_2 khác mà khi đó tay gôn được gán cho nhóm. Trong trường hợp đó, nhóm xuất hiện như biến, còn tay gôn xuất hiện như giá trị trong V_2. Trong "điểm nhìn" V_2, chúng ta dùng biến $G_{j,k}$ cho nhóm j trong tuần k là biến tập với miền của nó là tất cả các tay gôn $\{0,1,\ldots,n\text{-}1\}$. Chúng ta hãy xem nghiệm của V_2 trong Bảng 6.2

week / group	0	1	2
0	{0,1,2}	{0,3,6}	{0,5,7}
1	{3,4,5}	{1,4,7}	{1,3,8}
2	{6,7,8}	{2,5,8}	{2,4,6}

Bảng 6.2: Một nghiệm cho bài toán SGP trường hợp 3-3-3 dưới "điểm nhìn" V_2

Chúng ta hãy xem đối xứng 2 (φ_G), nó là đối xứng giá trị trong V_1 do các nhóm trong tuần không có sự phân biệt. nhưng trong V_2 nó là đối xứng biến do chúng ta dùng biến tập $G_{j,k}$. Và chúng ta loại bỏ đối xứng bằng cách thiết lập trật tự giữa các nhóm trong tuần: $\min G_{j,k} < \min G_{j+1,k}$ với $0 \le j \le g\text{-}2$, $0 \le k \le w\text{-}1$.

Điều đáng chú ý ở đây là việc liên kết giữa hai "điểm nhìn" là việc cần tồn tại một ràng buộc nối (*channeling constraints*). Trong ví dụ này là $p_{i,k}=j \Leftrightarrow i \in G_{j,k}$, $0 \le i \le n\text{-}1$, $0 \le j \le g\text{-}1$ và $0 \le k \le w\text{-}1$.

Để minh họa thêm cho ý tưởng, chúng ta xét thêm một "điểm nhìn" V_3 khác. Trong đó chúng ta dùng biến $z_{i,k,j}$ cho mỗi tay gôn i chơi trong nhóm j tại tuần thứ k. Miền của biến $z_{i,k,j}$ là $\{0,1\}$, biến $z_{i,k,j}=1$ nếu tay gôn i chơi trong nhóm j tại tuần thứ k và ngược lại nó bằng 0. Ràng buộc nối ở đây là $p_{i,k}=j \Leftrightarrow z_{i,k,j}=1$, $0 \le i \le n\text{-}1$, $0 \le j \le g\text{-}1$ và $0 \le k \le w\text{-}1$. Khi đó đối xứng 2 (φ_G) trở thành đối

Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Deleted: golfer
Deleted: 0¶ 1¶ 2
Deleted: 3¶ 4¶ 5
Deleted: 6¶ 7¶ 8
Deleted: week
Deleted: 0¶ 0¶ 0
Deleted: 1¶ 1¶ 1
Deleted: 2¶ 2¶ 2
Deleted: 0¶ 1¶ 2
Deleted: 0¶ 1¶ 2
Deleted: 0¶ 1¶ 2
Deleted: 0¶ 1¶ 2
Deleted: 1¶ 2¶ 0
Deleted: 2¶ 0¶ 1
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Italic
Formatted: Font: Not Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic

xứng giá trị trong V_3 . Dùng ràng buộc loại bỏ đối xứng $[z_{0,k,j},..., z_{n-1,k,j}] >_{lex} [z_{0,k,j+1},..., z_{n-1,k,j+1}]$ cho mọi $0 \le j \le g$-1 và $0 \le k \le w$-1. Ví dụ trong Bảng 6.1, chúng ta có $z_0 \rightarrow$ [1,0,0, 1,0,0, 1,0,0] và $z_1 \rightarrow$ [1,0,0, 0,1,0, 0,1,0], trong đó $z_0 \rightarrow [z_{0,0,0}, z_{0,0,1}, z_{0,0,2}, z_{0,1,0}, z_{0,1,1}, z_{0,1,2}, z_{0,2,0}, z_{0,2,1}, z_{0,2,2}]$ và $z_1 \rightarrow [z_{1,0,0}, z_{1,0,1}, z_{1,0,2}, z_{1,1,0}, z_{1,1,1}, z_{1,1,2}, z_{1,2,0}, z_{1,2,1}, z_{1,2,2}]$, khi đó $z_0 >_{lex} z_1$. Ở đây cần chú ý trật tự giữa z_l và $p_{i,k}$ để tránh bị mâu thuẫn.

Formatted: Font: Italic
Formatted: Font: Italic

6.1.2. 3 Kết quả của phương pháp nhiều "điểm nhìn" cho SGP

Kết quả trong phần này được thực thi trên ILOG Solver 4.4 với máy Sun Ultra 5/400, 256 MB RAM. Dấu "-" để chỉ nó chạy vượt quá 2 giờ.

g	*s*	*w*	V_1			V_3			$V_1 V_2$			$V_1 V_3$		
			fails			fails	choices		fails	choices		fails	choices	
6	2	11	142	180	0.57	166	425	0.32	142	180	0.2	166	259	**0.1**
7	2	13	1371	1428	9.63	1525	1966	2.97	1371	1482	1.94	1469	1604	**1.4**
6	3	5	-	-	-	47957	47959	332	37590	37591	306	37637	37638	**206**
7	3	4	687	729	3.33	853	1084	0.81	687	729	0.72	694	760	**0.4**
8	3	5	-	-	-	43326	43329	545	17005	17005	206	17067	17068	**133**
5	4	4	-	-	-	-	-	-	34727	34727	327	34780	34781	**229**
6	4	3	-	-	-	-	-	-	58802	58803	586	59035	59035	**370**
7	4	3	-	-	-	-	-	-	20705	20705	272	20947	20947	**169**
8	4	9	22	142	7.36	27	668	1.52	22	142	0.44	24	207	**0.3**
7	5	2	4879	4883	112	-	-	-	48794	48838	127.	50257	50313	**78.**
8	5	2	7146	7151	261	-	-	-	71463	71515	213.	74679	74745	**127**
8	8	9	19	182	41.7	19	821	7.59	19	182	1.75	19	245	**0.9**
5	4	3	1350	1350	352	49638	49648	435.	13503	13506	134.	15345	15349	**80.**
5	3	5	1350	1350	215	19643	19654	111.	13503	13506	94.6	16621	16626	**64.**
5	3	7	1350	1350	46	53824	53954	57.1	13503	13506	19.5	18616	18673	**13.**

Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold
Formatted: Font: 14 pt, Bold

Bảng 6.3: Kết quả khi dùng phương pháp nhiều "điểm nhìn" cho nghiệm đầu tiên của SGP

Trong bảng 6.3 ký hiệu V_1 V_2 là sự kết hợp của hai "điểm nhìn" V_1 và V_2. Tương tự V_1 V_3 là sự kết hợp của hai "điểm nhìn" V_1 và V_3. Lần lượt *fails*, *choices*, *time* chỉ số lần lỗi, số lần chọn khi quay lui và thời gian (giây) cho mỗi "điểm nhìn" tương ứng. Phần được bôi đậm để chỉ mô hình V_1 V_3 là tốt nhất.

Formatted: Font: 14 pt

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

Ta sẽ so sánh phương pháp này với phương pháp được đề xuất ở phần sau.

6.2 Loại bỏ đối xứng bằng hạn chế miền (ND) và cố định một số tay gôn (F)

Phần lý thuyết đã trình bày chi tiết trong phần 2.2 và 2.3. Ở đây chỉ đưa ra các kết quả cũng như để so sánh với một số phương pháp khác, qua đó để thấy được điểm mạnh cũng như điểm yếu của phương pháp. Kết quả được đưa ra trong Bảng 6.4 sẽ được so sánh với các phương pháp khác ở phần sau:

g	s	w	Basic	ND	ND+	g	s	w	Basi	ND	ND+	g	s	w	Basi	ND	ND+
4	2	2	0.02	0.01	0.01	6	2	6	0.22	0.18	1.20	8	2	11	1.29	1.23	1.11
		3	0.02	0.01	0.01			7	0.29	0.25	40.1			12	1.99	1.48	1.33
		4	0.06	0.03	0.03			8	0.39	0.32	313			13	2.39	1.79	1.59
		5	0.07	0.05	0.05			9	0.52	0.41	0.41			14	2.53	2.17	1.85
		6	0.1	0.08	0.08			10	0.63	0.52	0.52			**15**	**2.74**	**2.46**	**2.12**
		7	**0.12**	**0.11**	**0.1**			**11**	**0.77**	**0.63**	**0.63**		3	7	42.5	0.99	-
	3	2	0.02	0.01	0.01		3	6	-	-	0.35			8	-	1.30	-
		3	0.04	0.03	0.03			7	-	9.16	1.48			9	-	3.70	-
		4	**0.08**	**0.06**	**0.07**			**8**	-	-	-		4	4	1.25	0.45	0.40
	4	2	0.02	0.02	0.02		4	5	47.7	0.57	0.67			5	2.47	0.78	0.69
		3	0.07	0.04	0.05			6	-	-	-			6	53.7	1.61	1.06
		4	0.14	0.09	0.07		5	4	39.3	0.39	83.72			7	32.7	61.3	1.48
		5	**0.2**	**0.14**	**0.12**			5	-	-	-			8	-	-	2.16
5	2	5	0.1	0.09	0.09			**6**	-	-	-			9	-	-	2.49
		6	0.14	0.13	0.19		6	**3**	**0.38**	**0.20**	**0.22**		5	6	-	138.7	138.0
		7	0.19	0.17	0.63	7	2	7	0.36	0.33	0.33		6	5	-	231.2	231.0
		8	0.26	0.24	0.64			8	0.49	0.43	0.47		7	4	-	10.04	10.00
		9	**0,29**	**0.27**	**0.3**			9	0.63	0.56	0.60		8	**9**	**-**	**-**	**6.95**
	3	2	0.02	0.01	0.01			10	0.81	0.72	0.79	9	3	11	-	28.6	-
		3	0.03	0.05	0.04			11	1.01	0.89	0.92		4	8	-	18.9	-
		4	0.05	0.09	0.09			12	1.20	1.06	1.12		5	6	-	5.13	-
		5	0.21	0.16	0.2			**13**	**1.27**	**1.30**	**7.85**		6	5	-	6.06	-
		6	9.46	3.51	3.46		3	8	12.8	6.45			7	4	-	3.58	-
		7	**170.1**	**180.1**	**0.95**			9	-	-	-		8	3	-	0.84	0.81
	5	2	0.05	0.04	0.03		4	6	11.2	13.2	-		9	3	2.63	0.96	0.93
		3	0.15	0.1	0.09		5	5	-	1.91	-	10	2	**19**	**-**	**7.02**	**346.0**
		4	0.66	0.20	1.35		6	4	-	0.62	-		4	9	-	205.0	-
		5	1.22	0.34	1.88			5	-	-	-		9	3	-	1.30	1.24
		6	**1.79**	**0.49**	**0.78**		7	**8**	**-**	**-**	**287.3**		10	3	-	1.45	1.40

Bảng 6.4: Kết quả cho Basic, F và F+ND

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

Trong Bảng 6.4 chỉ ra kết quả được đề xuất (tính bằng giây) khi tìm được nghiệm đầu tiên. Ta dùng SICStus Prolog [29] chạy trên nền Windows XP, máy Laptop Pentium III/850MHz, 256MB. Dấu "-" chỉ thời gian vượt quá 5 phút. Với sự kết hợp khác nhau của 3 kỹ thuật đã mô tả ở trước: **Basic** (được mô tả trong phần 2.1), **ND** (được mô tả trong 2.2) và **F** (được mô tả trong phần 2.3), để có được kết quả khi dùng kỹ thuật **Basic**, **F** và **F+ND**. Trong đó những trường hợp được tô đậm để chỉ đó là trường hợp với số tuần lớn nhất có thể đạt được hoặc nhờ cộng đồng ràng buộc hoặc nhờ những mô hình toán học khác [9,41].

Như vậy dựa vào kết quả thực thi, ta rút ra một số kết luận như sau:

- Việc thêm kỹ thuật **ND** vào **Basic** cho kết quả hầu như bài toán được giải nhanh hơn trong tất cả các trường hợp. Nó cũng giúp giải được nhiều trường hợp hơn, trong đó có cả những trường hợp được coi là khó của cộng đồng ràng buộc (5-3-7, 7-5-5, 7-6-4, 7-7-8, 8-5-6, 8-5-6, 8-7-4, 9-4-8, 10-4-9, …).
- Khi áp dụng kỹ thuật **F** nhiều khi chúng ta sẽ mất nghiệm (do nó không bảo toàn tính nghiệm), tuy nhiên nó cho phép chúng ta giải bài toán với nhiều trường hợp nhanh hơn rất nhiều (giải bài toán Kirkman's School Problem trong vòng 1 giây) và cũng giải được trường hợp khó trong một thời gian nhỏ (bài toán SGP 8-4-9 trong vòng 2.5 giây, và giải bài toán với nghiệm tối ưu 8-8-9 trong 7 giây).

Formatted: Font: 14 pt, Font color: Black
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Not Bold
Formatted: Font: Not Bold
Formatted: Bullets and Numbering
Formatted: Font: Bold
Formatted: Font: Bold

6.3 So sánh với một số kỹ thuật khác

Trong phần này, sẽ so sánh với 3 phương pháp khác thường được sử dụng trong SGP: SBDD (được coi là mạnh nhất cho loại bỏ đối xứng bằng cách thêm ràng buộc trong thời gian tìm kiếm), Local Search (là phương pháp có

thời gian thực thi khá nhanh, đặc biệt là với một số trường hợp khó đối với lập trình ràng buộc trong bài toán SGP), phương pháp nhiều "điểm nhìn" (một phương pháp rất linh hoạt trong việc loại bỏ đối xứng tĩnh như chúng ta thấy ở phần trên), và phương pháp loại bỏ đối xứng bằng thuật toán quay lui thông minh.

6.3.1 So sánh với phương pháp SBDD

Đầu tiên, ta so sánh với phương pháp SBDD (nói chung, SBDD vẫn được đánh giá là tốt hơn SBDS, ít nhất cũng đúng cho SGP [20, 22]) trong [20]. [20] đã sử dụng mô hình biến tập (phần 3.1), thực thi chương trình viết bằng ECL^{i}PS^e trên máy Pentium II/450 MHz, môi trường Linux.

Trong Bảng 6.5: **NDF** cho kết quả tốt nhất khi kết hợp giữa **ND** và **F**, trong khi đó **SBDD mc** là kỹ thuật SBDD không cần sự kiểm tra kết hợp (without merged checks).

Thông qua Bảng 6.5, ta có thể nhận thấy là hầu hết những trường hợp mà SBDD giải được cũng được giải bằng **NDF** trong thời gian nhỏ hơn 1 giây! (bao gồm cả bài toán Kirkman's School). Hơn nữa ở đây cũng giải được nhiều trường hợp hơn (ví dụ, trong dạng g-s=6-2 đạt được 11 tuần trong thời gian 0.63s trong khi đó SBDD cần 1 giờ mà chỉ đạt được 6 tuần).

Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold

g	s	w	SBDD	SBDD	NDF	g	s	w	SBDD	SBDD	NDF
3	3	2	0.0	0.0	0.0	5	5	2	0.1	0.1	0.04
		3	0.0	0.0	0.0			3	0.2	0.2	0.1
		4	0.1	0.1	0.0			4	2277.	999.9	0.20
4	2	2	0.0	0.0	0.0			5	-	3639.	0.34
		3	0.1	0.1	0.0			6	-	-	0.49
		4	0.1	0.1	0.0	6	2	6	6494.	3607.	0.18
		5	0.2	0.2	0.0			7	-	-	0.25
		6	0.3	0.3	0.0			8	-	-	0.32
		7	0.4	0.3	0.1			9	-	-	0.41
	3	2	0.0	0.0	0.0			10	-	-	0.52
		3	0.1	0.1	0.0			11	-	-	0.63
		4	9.9	7.6	0.0		3	2	0.1	0.1	0.02
	4	2	0.0	0.0	0.0			3	0.2	0.2	0.06
		3	0.1	0.1	0.0			4	0.4	0.4	0.15
		4	0.2	0.2	0.0			5	1683	374.1	0.24
		5	0.3	0.3	0.1			6	-	-	0.35
5	2	5	0.3	0.3	0.0			7	-	-	1.48
		6	27.2	16.5	0.1			8	-	-	-
		7	504.4	216.1	0.1		4	2	0.2	0.1	0.04
		8	2842.4	1140.8	0.2			3	0.4	0.4	0.12
		9	5468.5	2410.9	0.2			4	0.5	0.6	0.22
	3	2	0.1	0.1	0.0			5	1461	537.2	0.57
		3	0.2	0.1	0.0			6	-	-	-
		4	29.9	17.8	0.0		5	2	0.2	0.2	0.05
		5	347.8	109.3	0.2			3	0.4	0.4	0.15
		6	-	-	3.4			4	1929	602.7	0.39
		7	-	-	0.9			5	-	-	-
	4	2	0.1	0.1	0.0			6	-	-	-
		3	0.2	0.2	0.0		6	2	0.2	0.1	0.07
		4	118.9	11	0.1			3	0.4	0.3	0.2
		5	1374.6	653.0	0.8						

Bảng 6.5: Kết quả SBDD cho SGP với kỹ thuật NDF

6.3.2 So sánh với phương pháp Local Search

Ở đây sẽ so sánh kết quả đạt được với việc giải SGP dùng Local Search (SGLS) [13], được thực thi bằng C, trên 3.06GHz-processor, 512MB RAM (vì vậy cấu hình có thể nhanh hơn cấu hình được thử nghiệm trong Luận văn tốt nghiệp xấp xỉ 50 lần). Cũng cần nói thêm rằng Local Search thường được cho là một phương pháp nhanh trong nhiều trường hợp mà CP rất khó khăn.

g	s	w	SGLS	NDF	g	s	w	SGLS	NDF
6	6	3	0.01	0.21	9	3	11	0.08	28.60
7	5	5	0.07	1.92		4	8	0.09	18.90
	6	4	0.09	0.64		5	6	0.09	5.13
	7	3	0.03	0.25		6	5	0.13	6.06
		8	-	287.21		7	4	0.14	3.58
8	3	10	0.23	-		8	3	0.09	0.84
	4	8	207.77	1.82		9	3	0.19	0.96
		9	-	7.36	10	4	9	0.16	205.05
	5	6	0.37	138.23		7	5	0.41	489.30
	6	5	1.21	231.41		9	3	0.21	1.34
	7	4	0.39	10.06		10	3	0.39	1.40
	8	4	360.00	501.46					
		9	-	7.36					

Bảng 6.6: Kết quả của SGLS với kỹ thuật NDF

Trong Bảng 6.6, dấu "-" chỉ những trường hợp thời gian vượt quá 10 phút.

Thông qua Bảng 6.6, trong một số trường hợp không tìm được nghiệm tối ưu so với SGLS trong thời gian 10 phút. Tuy nhiên, Luận văn tốt nghiệp cũng tìm ra nghiệm nhiều hơn SGLS 3 trường hợp khó (7-7-8, 8-4-9, 8-8-9).

Formatted: Font: 14 pt, Font color: Black

Formatted: Font: 14 pt, Italic, Font color: Black

Formatted: Font: 14 pt, Italic, Font color: Black

Formatted: Font: 14 pt, Font color: Black

6.3.3 So sánh với phương pháp nhiều "điểm nhìn"

Ở đây sẽ lấy kết quả tốt nhất trong Bảng 6.3 ("điểm nhìn" V_1V_3)và kết quả tốt nhất trong kỹ thuật đã nghiên cứu để so sánh (**NDF**). Thời gian đều được tính bằng giây.

g	*s*	*w*	V_1V_3 time	*NDF* time	*+ w*
6	2	11	0.13	0.63	0
7	2	13	1.47	1.30	0
6	3	5	2066.14	0.32	2
7	3	4	0.41	0.20	4
8	3	5	1330.29	0.49	4
5	4	4	2297.76	1.35	1
6	4	3	3703.93	0.13	2
7	4	3	1699.63	0.16	3
8	4	9	0.3	2.94	0
7	5	2	78.34	0.08	3
8	5	2	127.84	0.11	4
8	8	9	0.93	6.95	0
5	4	3	80.83	0.08	2
5	3	5	64.27	0.2	0
5	3	7	13.96	0.95	0

Bảng 6.7: So sánh kết quả trong việc tìm ra nghiệm đầu tiên giữa phương pháp nhiều "điểm nhìn" và NDF.

Trong Bảng 6.7 thời gian được tính bằng giây. Tại cột ký hiệu là "+*w*" để chỉ ra rằng phương pháp được đề xuất (**NDF**) có thể đạt được hơn "+*w*" số tuần

Formatted ... [1]
Formatted ... [2]
Formatted: Font: 14 pt
Formatted: Font: 14 pt, Italic
Formatted ... [3]
Formatted: Font: 14 pt
Formatted ... [4]
Formatted: Font: 14 pt
Formatted ... [5]
Formatted: Font: Not Bold
Formatted: Font: 14 pt
Formatted ... [6]
Formatted: Font: Not Bold
Formatted: Font: 14 pt
Formatted ... [7]
Formatted: Font: Not Bold
Formatted: Font: 14 pt
Formatted ... [8]
Formatted: Font: Not Bold
Formatted: Font: 14 pt
Formatted ... [9]
Formatted: Font: Not Bold
Formatted: Font: 14 pt
Formatted ... [10]
Formatted: Font: Not Bold
Formatted: Font: 14 pt
Formatted ... [11]
Formatted: Font: 14 pt
Formatted ... [12]
Formatted: Font: 14 pt
Formatted ... [13]
Formatted: Font: Not Bold
Formatted: Font: 14 pt
Formatted: Font: 14 pt, Not Bold
Formatted ... [14]
Formatted: Font: 14 pt, Not Bold
Formatted: Font: 14 pt, Not Bold
Formatted ... [15]

của phương pháp nhiều "điểm nhìn" trong vòng nhiều nhất là 5 phút (tham khảo ở Bảng 6.4).

Thông qua kết quả so sánh ta có kết luận sau đây:

- Hầu hết các trường hợp trong phương pháp nhiều "điểm nhìn" giải được thì đều được giải bằng phương pháp được đề xuất trong thời gian 7 giây!
- Có nhiều trường hợp phương pháp nhiều "điểm nhìn" đã cần đến 20, 30 thâm chí 60 phút, trong khi đó ở đây chỉ giải trong vòng xấp xỉ 1 giây!
- Có thể giải nhiều trường hợp trong phương pháp nhiều "điểm nhìn" giải trong thời gian lớn (thậm chí một giờ): 6-4-5, 8-3-5, 5-4-4, 6-4-3, 7-4-3, 8-5-2. Trong khi đó, ta có thể giải được với nghiệm tối ưu hơn (đạt được nhiều tuần hơn) trong thời gian 2 phút.

6.3.4 So sánh với phương pháp IB

Trong Bảng 6.8 trình bày kết quả so sánh giữa hai phương pháp IB (phần 5.1) và phương pháp NDF. Phương pháp IB [2] được thực thi bằng C++, trên Pentium IV/ 2,4GHz-processor. Thời gian được tính bằng giây.

Formatted: Font: 14 pt, Font color: Black

Formatted: Font: 14 pt, Font color: Black

Trong cột w_2 có những hàng có ký hiệu $a \pm b$ có nghĩa là: a là số tuần phương pháp **IB** đạt được trong 5 phút, còn b là số tuần mà phương pháp được đề xuất tìm được kém hơn (-) hay nhiều hơn (+) so với phương pháp **IB**.

Như vậy, thông qua Bảng 6.8 có thể đưa ra nhận xét như sau:

- Trong 5 phút hầu hết những trường hợp mà phương pháp **IB** giải được thì phương pháp **NDF** cũng giải được trong thời gian chấp nhận được.
- Có hai trường hợp **NDF** không tối ưu bằng IB (6-4-6 và 6-4-5) cụ thể hơn là số tuần của **NDF** đạt được kém 1 trong cả hai trường hợp

- Có 5 trường hợp mà **NDF** giải được với số nghiệm tối ưu hơn (hơn 1,2,3,4 tuần) so với phương pháp **IB**.

Cần chú ý là thời gian cho hai phương pháp đều trong giới hạn 5 phút.

g	*s*	IB		NDF	
		w_1	Time	w_2	Time
4	3	4	0.01	4	0.06
	4	5	0.00	5	0.12
5	3	7	1.10	7	0.95
	4	5	0.10	5	0.86
	5	6	0.03	6	0.49
6	3	7	1.36	7	1.48
	4	6	123.01	6-1	0.66
	5	5	6.56	5-1	0.39
7	3	6	0.08	6+2	6.45
	4	6	0.05	6	13.21
	5	5	0.11	5	1.9
8	3	8	0.08	8+1	3.70
	4	5	0.02	5+4	2.49
	5	6	15.63	6	138.77
9	3	9	4.29	9+2	28.6
	4	5	0.51	5+3	18.9

Bảng 6.8: So sánh kết quả trong việc tìm ra nghiệm đầu tiên giữa phương pháp IB và NDF.

CHƯƠNG 7. GIẢI SGP TRONG MỘT SỐ TRƯỜNG HỢP ĐẶC BIỆT VÀ MỐI LIÊN QUAN VỚI CÁC HÌNH VUÔNG LATIN TRỰC GIAO

Ở đây sẽ giải SGP trong trường hợp đặc biệt *p*-*p*-(*p*+1), khi *p* là số nguyên tố, ngoài ra cũng đưa ra một vài kết luận với trường hợp *s*-*s*-*w* cho cả trường hợp *s* chẵn và *s* lẻ. Phần này cũng nêu ra một quan hệ rất thú vị giữa SGP với hình vuông latin trực giao.

7.1 Giới thiệu thuật toán

Thuật toán này dùng để giải SGP cho trường hợp *p*-*p*-(*p*+1), khi *p* là số nguyên tố. Điều đặc biệt là nó không mất thời gian cho việc tìm kiếm.

Ta sẽ minh họa ý tưởng của thuật toán thông qua một ví dụ. Bảng 7.1 là một trường hợp của SGP cho 3-3-4

week	group1			group 2			group 3		
1	1	2	3	4	5	6	7	8	9
2	**1**	**4**	**7**	**2**	5	8	**3**	6	9
3	**1**	*	**	**2**	?	?	**3**	?	?
4	**1**			**2**			**3**		

Bảng 7.1: Một trường hợp của SGP cho 3-3-4

Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold

Như trong phần 2.1 đã nói, các tay gôn 1, 2, 3 được cố định tại các nhóm 1, 2, 3 kể từ tuần thứ hai trở đi. Tuần thứ hai cũng được cố định ở nhóm thứ nhất. Không khó khăn gì để chúng ta có được nghiệm cho tuần thứ 2. Từ đây thuật toán được bắt đầu. Ta sử dụng ý tưởng xuất phát từ kỹ thuật hạn chế miền (ND trong phần 2.2) rằng các phần tử ở vị trí thứ nhất chỉ quan tâm đến nhóm thứ nhất (trong tuần 1), những phần tử ở vị trí thứ hai chỉ quan tâm đến nhóm thứ hai (trong tuần 2), …

Như vậy phần tử * trong Bảng 7.1 chỉ quan tâm đến những phần tử {4,5,6}, nhưng tay gôn 4 đã gặp tay gôn 1 trong tuần thứ hai, do vậy tay gôn * chỉ có thể gặp tay gôn {5,6}. Một cách rất tự nhiên ta chọn tay gôn 5. Và khi chọn tay gôn 5 cho phần tử * thì phần tử ** chỉ còn một lựa chọn duy nhất là tay gôn 9. Làm tương tự chúng ta sẽ được nhóm thứ hai, và nhóm thứ ba cho tuần thứ ba. Tiếp tục làm như vậy và chúng ta sẽ đạt được tuần thứ 4 nhờ tuần thứ 3. Cuối cùng chúng ta được nghiệm hoàn chỉnh (tối ưu cho số tuần) trong Bảng 7.2 .

week	group1			group 2			group 3		
1	1	2	3	4	5	6	7	8	9
2	1	4	7	2	5	8	3	6	9
3	1	5	9	2	6	7	3	4	8
4	1	6	8	2	4	9	3	5	7

Bảng 7.2: Một nghiệm hoàn chỉnh cho SGP 3-3-4

Và từ đây có thể phát hiện ra một quy luật cho thuật toán. Mô tả dưới dạng mã giả như sau:

Procedure SGP_PrimeNumber(**In**: s, **Out**: S)

n ← $s \times s$

W_1 ← {<1, 2, …,s>, <s+1, …,2*s>, …<(s-1)*s+1, …, s*s>}

W_2 ← {G1, …, Gs}={<1, s+1, …, (s-1)*s+1>, …<s, 2*s, …s*s>}

const ← s+1

for *i* **from** 3 to s+1 do {generate i^{th} week}

begin

W_i←Ø

for j from 1 to s do {generate j^{th} group}

```
            Gi ←{Pj}
            pos←j
            for k from 2 to s do        {generate k^th player}
                pos← (pos+const) mod n
                Pk ← W[i-1, pos]
                Gj ←Gj U{Pk}
            end for;
        end for;
        Wi ← Wi U{ Gj };
    end for;
end Procedure
```

Chú ý W[i,j]: dùng để chỉ tay gôn ở vị trí thứ j trong tuần W[i].(W[i] được coi là một danh sách với độ dài n). G_i chỉ nhóm thứ i.

7.2 Một số thảo luận cùng kết quả xung quanh thuật toán

Dotu và Van Hentenryck [12] cũng có một thuật toán giải trong SGP *p-p-(p+1)* khi p là nguyên tố (khá tương tự với thuật toán đã được đề xuất). Ở đó các tác giả đã kết luận về thuật toán như sau:

- Khi *odd* chia hết cho 3, thuật toán đạt được nghiệm dạng *odd-odd*-4.
- Khi *odd* chia hết cho 5, thuật toán đạt được nghiệm dạng *odd-odd*-6.
- Khi *odd* chia hết cho 7, thuật toán đạt được nghiệm dạng *odd-odd*-8.

Thông qua thuật toán được đề xuất, ta cũng có một số kết luận như sau:

- Thuật toán giải SGP cho trường hợp *p-p-(p+1)*, khi p là số nguyên tố (Đã kiểm tra cho mọi số p < 73).
- Thuật toán đúng cho mọi trường hợp *odd-odd*-4, khi *odd* là số lẻ.

- Thuật toán đúng cho mọi trường hợp *number-number*-3. Nó được chứng minh bằng những lập luận sau:
  - Theo như thuật toán tuần thứ hai được gán với mỗi nhóm đạt được các tay gôn từ các mỗi nhóm khác nhau (do số tay gôn trong mỗi nhóm chính là số nhóm trong tuần), do vậy tuần thứ hai thỏa mãn tuần thứ nhất.
  - Mỗi nhóm trong tuần thứ ba cũng có được các tay gôn từ các nhóm khác nhau trong tuần thứ hai và tuần thứ nhất.

Chuyện gì xảy ra khi tiếp tục áp dụng cho tuần thứ 4 khi *s=number* là chẵn? Chúng ta hãy cùng xem xét trường hợp này.

- Trong tuần thứ hai, tay gôn 1 và tay gôn (s*s/2+1) trong cùng nhóm thứ nhất, bởi vì tay gôn (s*s/2+1) là tay gôn đầu tiên trong trong nhóm (s/2+1).
- Nó cũng không khó hình dung ra tay gôn (s*s/2+1) sẽ ở nhóm (s/2+1) trong tuần thứ ba.
- Tiếp đến tuần thứ tư, tay gôn (s*s/2+1) sẽ ở nhóm đầu tiên. Và như vậy là nó gặp lại tay gôn 1. Như vậy thuật toán không còn đúng nữa.

Chúng ta hãy xem minh họa cho lập luận trên khi *number* =4 trong Bảng 7.3

Formatted: Font: Not Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font color: Black
Formatted: Font color: Black
Formatted: Font: Italic

week	group1				group 2				group 3				group 4			
1	1	2	3	4	5	6	7	8	**9**	10	11	12	13	14	15	16
2	**1**	5	**9**	13	2	6	11	14	3	7	11	15	4	8	12	16
3	1	6	11	16	2	7	12	13	3	8	**9**	14	4	5	11	15
4	**1**	7	**9**		2				3				4			

*Bảng 7.3: Một minh họa cho thuật toán với trường hợp 4-4-w, khi này tay gôn (s*s/2+1) chính là tay gôn số 9.*

Formatted: Font: Italic

Như vậy, có thể kết luận thuật toán đúng cho mọi trường hợp *number-number*-3 với *number* là bất kỳ (chẵn hay lẻ) mà không mất thời gian tìm kiếm (trường hợp tối ưu cho SGP 6-6-3 cũng được giải bởi thuật toán).

Formatted: Font: Italic

7.3 Liên hệ SGP với hình vuông Latin trực giao

7.3.1 Giới thiệu hình vuông Latin trực giao

Trước hết, ta muốn giới thiệu về hình vuông (Latin Square) và hình chữ nhật Latin (Latin Rectangle).

Định nghĩa 7.1

Một ma trận n×n là hình vuông Latin cấp *n*, nếu mỗi số 0, 1, …, *n*-1 xuất hiện đúng một lần trên mỗi hàng và mỗi cột. ■

Ví dụ về một hình vuông Latin như trong Bảng 7.4 là hình vuông Latin cấp 4. Tổng quát hơn, chúng ta có thể thay các số bằng các đối tượng khác nhau.

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

Bảng 7.4: Một ví dụ về hình vuông Latin cấp 4

Định nghĩa 7.2

Một ma trận $r \times n$ được gọi là hình chữ nhật Latin, nếu mỗi số 0, 1, …, n-1 xuất hiện nhiều nhất một lần trên mỗi hàng và mỗi cột. ■

Ví dụ về một hình chữ nhật Latin như trong Bảng 7.5.

0	1	2	3
2	3	0	1
3	2	1	0

Bảng 7.5: Một ví dụ về hình chữ nhật Latin 3×4

Như vậy với mỗi hình vuông Latin ta có thể đạt được hình chữ nhật Latin bằng cách loại bỏ một số hàng trong hình vuông Latin. Còn từ hình chữ nhật Latin, liệu chúng ta có đạt được hình vuông Latin không? Câu hỏi đã có trong định lý sau:

Định lý 7.3

Nếu $r < n$, thì với bất kỳ một hình chữ nhật Latin $r \times n$ có thể được mở rộng thành hình chữ nhật Latin $(r+1) \times n$.

Việc chứng minh có thể tham khảo tại [4].

Bây giờ đã tới lúc chúng ta thảo luận tới các hình vuông Latin trực giao (Mutually orthogonal Latin squares - MOLS).

Định nghĩa 7.4

Một tập MOLS là một tập hình vuông Latin cỡ n sao cho bất kỳ một cặp hình vuông L_α và L_β nào từ tập đó, thì cặp $(L_\alpha(i,j), L_\beta(i,j))$ phải khác nhau với mọi $1 \le i, j \le n$. ■

Từ đây suy ra một tính chất sau:

Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic

Tính chất 7.5

Nếu tập MOLS có k hình vuông Latin cỡ $n\{L_i | 1 \le i \le k\}$ thì khi đó bộ k - giá trị $(L_{i1}(i,j),\ldots, L_{ik}(i,j))$ phải khác nhau với mọi i và j, $1 \le i, j \le n$.

Ví dụ trong Bảng 7.6 là một tập MOLS gồm 3 hình vuông Latin

0	1	2	3
3	2	1	0
1	0	3	2
2	3	0	1

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

Bảng 7.6: Hai hình vuông Latin trực giao

Từ định nghĩa 7.4 chúng ta cũng suy ra được tập hình chữ nhật trực giao (Mutually orthogonal Latin rectangles - MOLR).

Định nghĩa 7.6

Một tập MOLR là một tập hình chữ nhật Latin sao cho bất kỳ một cặp hình chữ nhật $L_\alpha$và L_β nào từ tập đó, thì cặp $(L_\alpha(i,j), L_\beta(i,j))$ phải khác nhau với mọi i và j. ■

Và như vậy thì MOLR cỡ $n \times n$ chính là tập MOLS cỡ n. Chú ý rằng tính chất 7.5 cũng đúng cho MOLR.

Chúng ta gọi $N(n)$ là số hình vuông lớn nhất có thể từ tập MOLS cấp n; gọi $N(m \times n)$ là số hình chữ nhật lớn nhất có thể từ tập MOLR cấp $m \times n$. Khi đó ta có 2 tính chất sau:

- $N(n) \le N(m \times n)$, cho mọi $m \le n$
- $N(m \times n) \le n$-1, cho mọi $1 < m \le n$

7.3.2 Mối liên hệ giữa MOLS và SGP

Thực ra cũng không phải tự nhiên mà có sự liên hệ giữa MOLS và SGP. Sự liên hệ này thực chất là cả hai đều liên quan đến các bài toán mang tính tổ hợp.

Tại [II.2.3] trong [9], có giới thiệu một thuật toán xây dựng tập MOLS như sau:

Nếu $n=p^e$, khi p là số nguyên tố, thì ta có thể xây được dựng tập MOLS như sau:

- Đặt GF(n) là một trường hữu hạn cấp n (gồm các số 0,1,…, n-1)
- Với mỗi $\alpha \in GF(n)\backslash\{0\}$, đặt $L_\alpha(i,j)= \alpha i+j$, với $i, j \in GF(n)$, các phép toán thực hiện trong GF(n).

Khi đó ta có tập $\{ L_\alpha | \alpha \in GF(n)\backslash\{0\}\}$ chính là tập (n-1) MOLS cấp n.

Và từ chính tập MOLS này chúng ta có thể xây dựng được nghiệm cho bài toán SGP (trong trường hợp n là lũy thừa của số nguyên tố, từ đây trở đi ta chỉ xét trường hợp n là lũy thừa của số nguyên tố, nếu xét trường hợp khác chúng tôi sẽ nêu cụ thể).

Khi ta đặt cạnh nhau các (n-1) MOLS cấp n ta nhận thấy rằng chính chúng thỏa mãn tính của SGP cho dạng g- g- (g+1):

- Có g nhóm và mỗi nhóm có g tay gôn chính được thể hiện bằng (g-1) hình vuông cỡ g (sẽ giải thích sau vì sao chỉ cần g-1 hình vuông)
- Trong trường hợp này mọi tay gôn đều gặp nhau đúng một lần.
- Các tay gôn không gặp lại nhau chính là tính chất của MOLS được thể hiện thông qua tính chất 7.5

Ta sẽ lấy một ví dụ để làm rõ kết luận trên.

Chúng ta hãy xem bảng sau:

week	group1	group 2	group 3
1	1 2 3	4 5 6	7 8 9
2	1 4 7	2 5 8	3 6 9
3	1 5 9	2 6 7	3 4 8
4	1 6 8	2 4 9	3 5 7

Bảng 7.7: Một nghiệm cho trường hợp 3-3-4

Khi chúng ta chuyển mô hình sang dạng khác (chính là V_1 trong phần 6.1) chúng ta được bảng sau:

Golfer / week	1 2 3	4 5 6	7 8 9
1	1 1 1	2 2 2	3 3 3
2	1 2 3	**1 2 3**	**1 2 3**
3	1 2 3	**3 1 2**	**2 3 1**
4	1 2 3	**2 3 1**	**3 1 2**

Bảng 7.8: Nghiệm cho trường hợp 3-3-4 ở mô hình lấy nhóm làm giá trị

Từ này trở đi ta ký hiệu r hình vuông Latin trong tập MOLS(n) bằng r MOLS(n).

Như vậy, với tập 2MOLS(3) được tô đậm ở trên chúng ta xây dựng được nghiệm SGP cho trường hợp 3-3-4.

Tóm lại với n là lũy thừa của số nguyên tố $n=p^e$, chúng ta sẽ có được (n-1) MOLS(n) và chúng ta sẽ xây dựng được nghiệm cho SGP với trường hợp n-n-

Formatted: Font: Italic
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic

(n+1). Như vậy đây là sự tổng quát so với thuật toán được đề xuất và thuật toán của [13]. Từ đó suy ra trong trường hợp muốn tìm nghiệm SGP cho trường hợp g-g-n chính là tương đương với việc tìm ta (n-2) MOLS(g). Đây quả thực là một sự liên hệ rất thú vị.

Hơn nữa, ta cũng có thể khẳng định rằng thuật toán được đề xuất có thể dễ dàng cải tiến (phần 7.1) để có thể tạo ra tập (n-1) MOLS(n) khi n là nguyên tố (tức là e =1, trong [9]).

7.3.3 Mối liên hệ giữa SGP và MOLR

Trong [21] cũng tóm tắt 2 phương pháp xây dựng nghiệm cho SGP từ tập các MOLR(m×n). Ở đây xin đưa ra 1 cách tạo dựng thú vị sau:

1. Sharma và Das's

   r MOLR(m×n) tương ứng với g- s - w trong SGP

   và khi đó (g =n)- (s=m)- (w=r+1) (nếu g không chia hết cho s)

   (w >r+1) (nếu g chia hết cho s)

2. Mathtalk-ga'

   r MOLR(m×n) tương ứng với g- s - w trong SGP

   và khi đó (g =n)- (s=r+1)- (w=m) (nếu g không chia hết cho s)

   (w >m+1) (nếu g chia hết cho s)

Đây quả thực là những sự liên hệ rất thú vị, nó đã tìm ra được nhiều nghiệm cho SGP một cách nhanh chóng (so với lập trình ràng buộc), đồng thời chúng cũng đưa ra được những nghiệm mới, góp phần bổ sung đáng kể vào tập nghiệm cho SGP.

Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: 14 pt, Not Bold, Font color: Black
Formatted: Font: 14 pt, Not Bold, Font color: Black
Formatted: Font: 14 pt, Italic, Font color: Black
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: 14 pt, Italic, Font color: Black
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: 14 pt, Italic, Font color: Black
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Not Italic
Formatted: Font: Italic
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: Italic
Formatted: Font: 14 pt, Italic, Font color: Black
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: 14 pt, Font color: Black
Formatted: Font: 14 pt, Not Bold, Font color: Black

PHẦN IV. KẾT LUẬN

Theo cách giải quyết truyền thống với các bài toán tổ hợp khó, có hai cách tiếp cận truyền thống: Hoặc là thực hiện thủ công thuật toán sẽ giải chính xác trong lập trình truyền thống, hoặc là mô hình bài toán dùng thuật toán có sẵn cho lớp các ràng buộc số học cụ thể. Hạn chế của phương pháp thứ nhất là việc thiết kế nó đòi hỏi chi phí lớn và không dễ gì biến đổi khi bài toán thay đổi. Hạn chế của phương pháp thứ hai là không dễ gì diễn tả hiệu quả các ràng buộc trong miền ứng dụng đủ mềm dẻo và hiệu quả cho phép các kinh nghiệm trong các miền được chỉ ra để giải quyết chúng. Ngôn ngữ CP hiện đại có thể khắc phục được những điểm yếu này bằng cách cung cấp một ngôn ngữ lập trình dựa trên việc giải ràng buộc tinh vi nhất. Điều này có nghĩa là người lập trình có thể viết chương trình trong khi kỹ thuật giải chung đã được cung cấp trong việc thực thi ngôn ngữ. Chúng ta đều biết rằng mọi bài toán đều có ràng buộc, công việc của chúng ta là tìm ra giá trị của các biến thỏa mãn các ràng buộc đó. Lập trình ràng buộc càng tỏ rõ hiệu quả trong những bài toán mà việc yêu cầu mô tả ràng buộc mang tính mềm dẻo. Và càng ngày lập trình ràng buộc càng thể hiện rõ vai trò mạnh mẽ của mình trong việc giải những bài toán liên quan đến những thực trong cuộc sống, càng ngày lĩnh vực ứng dụng của nó càng trở nên phong phú (toán học, y học, kinh tế, vật lý, tin học,…).

Phần II, Luận văn những kiến thức cô đọng nhất cho CSPs: những định nghĩa cùng với những khái niệm quan trong cho CSPs; Luận văn cũng nêu và thảo luận một cách khái lược khi giải CSPs, các kỹ thuật áp dụng nhằm biến đổi bài toán sao cho bài toán sau khi biến đổi gọn hơn và dễ giải hơn. Luận văn tốt nghiệp cũng nêu tổng quan về các thuật toán tìm kiếm vì nó là một nhân tố quyết định chính trong quá trình tìm nghiệm của bài toán.

Trong phần III, *phần đóng góp chủ yếu của Luận văn*, đã giới thiệu và trình bày hệ thống sao cho phù hợp với các tiêu chí cho việc giải SGP: các mô hình khác nhau, các kỹ thuật khác nhau, các phương pháp giải khác nhau, và sự kết hợp giữa chúng. Đồng thời Luận văn cũng so sánh giữa các mô hình, các phương pháp khác nhau.

Deleted: PHẦN III: BÀI TOÁN NGƯỜI CHƠI GÔN¶
¶
Phần này chúng tôi sẽ giới thiệu bài toán đồng thời giới thiệu một số kiến thức nền tảng. Từ đó chúng tôi sẽ tiếp cận bài toán với những mô hình và phương pháp khác nhau để so sánh và thảo luận. Ở phần cuối chúng tôi có giải quyết một trường hợp đặc biệt của bài toán (khi s=g và là số nguyên tố) và có những kết luận xung quanh đó. Phần này chính là phần đóng góp trọng tâm của đồ án.

Luận văn tốt nghiệp đã giới thiệu bài toán cùng với những kiến thức cần thiết cho bài toán "Người chơi gôn". Từ đó chúng được ứng dụng cho những phần sau. Tiếp đến đã giới thiệu một số kỹ thuật, ràng buộc nhằm loại bỏ đối xứng tĩnh trong bài toán, vì hầu như các phương pháp khác đều sử dụng một phần (hoặc hầu hết) các kỹ thuật trong phần này, đồng thời cũng đưa ra 2 kỹ thuật mới để áp dụng cho việc giải SGP. Hai kỹ thuật đó là

- Kỹ thuật **ND**: Đây là một ràng buộc dư thừa, nó không làm mất nghiệm bài toán. Làm bớt khả năng xét trong quá trình tìm kiếm. Nó cũng rất dễ dàng áp dụng cho các phương pháp khác.
- Kỹ thuật **F**: Tuy nó không bảo toàn tính nghiệm cho bài toán, nhưng trong rất nhiều trường hợp nó giúp bài toán giải trong thời gian rất nhanh (Bài toán Kirkman, hay bài toán SGP cho trường hợp 8-4-9).

Sau đó, Luận văn cũng giới thiệu bốn mô hình thường được áp dụng cho việc giải SGP (có thể tham khảo thêm mô hình SAT cho SGP[18]). Từ đó nêu ra những đặc trưng của từng mô hình khi áp dụng giải SGP.

Vì tính đặc thù của CSPs là tính đối xứng trong bài toán (đối xứng nghiệm, đối xứng ràng buộc) gây tốn thời gian cũng như chi phí cho việc tìm kiếm nghiệm (chúng ta phải tìm kiếm lại những không gian nghiệm mà đáng ra chúng ta có thể suy ra từ những phần khác đã xét). Do vậy việc loại bỏ đối

xứng là một yêu cầu tự nhiên và thiết yếu. Một kỹ thuật được áp dụng rất phổ biến trong lĩnh vực CSPs đó là kỹ thuật loại bỏ đối xứng động khi giải SGP. Nhưng do tính đặc thù của các phương pháp, ở đây tách ra làm hai phần.

- Phần thứ nhất nói về việc áp dụng các kỹ thuật loại bỏ đối xứng trong thời gian tìm kiếm SBDS và SBDD cho SGP và cũng nêu ra một vài điểm nhằm so sánh giữa chúng.
- Phần hai giới thiệu hai phương pháp loại bỏ đối xứng động cho SGP đó là Intelligent Backtracking (thực thi bằng quay lui thông minh) và Local Search.

Phần tiếp theo, Luận văn tập trung vào việc loại bỏ đối xứng tĩnh khi giải SGP. Chương cuối đưa ra phương pháp được đề xuất cùng với các so sánh kết quả với những phương pháp đã trình bày ở những phần trước.

Ta có so sánh kết quả thực nghiệm với các phương pháp:

- Phương pháp SBDD (*thực thi trên ILOG Solver 5.0 và 400MHz Ultrasparc-II*) . Phương pháp này được đánh giá là tốt hơn phương pháp SBDS . Ta có kết luận như sau:
  - Hầu hết những trường hợp mà SBDD giải được cũng được giải bằng phương pháp được đề xuất trong thời gian nhỏ hơn 1 giây! (bao gồm cả bài toán Kirkman's School).
  - Đã tìm được nghiệm tối ưu hơn (số tuần nhiều hơn) trong nhiều trường hợp.
- Phương pháp Local Search (*thực thi bằng C, trên 3.06GHz-processor, 512MB RAM*). Trong [13, 21] đều nhận xét rằng có rất nhiều trường hợp mà giải bằng lập trình ràng buộc rất khó khăn thì Local Search có thể giải nó một cách dễ dàng và ngược lại. Tuy nhiên trong một số

Formatted: Font: 14 pt, Italic, Font color: Black

trường hợp không tìm được nghiệm tối ưu so với SGLS trong thời gian 10 phút. Tuy nhiên ở đây cũng tìm ra nghiệm nhiều hơn SGLS 3 trường hợp khó (7-7-8, 8-4-9, 8-8-9).

- Phương pháp nhiều "điểm nhìn" (*ILOG Solver 4.4 với máy Sun Ultra 5/400, 256 MB RAM*). Sau khi so sánh ta đưa ra được khẳng định như sau:
  - Hầu hết các trường hợp trong phương pháp nhiều "điểm nhìn" giải được thì đều được giải bằng phương pháp được đề xuất trong thời gian 7 giây.
  - Có nhiều trường hợp phương pháp nhiều "điểm nhìn" đã cần đến 20, 30 thâm chí 60 phút, trong khi đó chỉ cần giải trong vòng xấp xỉ 1 giây!
  - Luận văn tốt nghiệp cũng giải nhiều trường hợp trong phương pháp nhiều "điểm nhìn" giải trong thời gian lớn (thậm chí một giờ): 6-4-5, 8-3-5, 5-4-4, 6-4-3, 7-4-3, 8-5-2. Trong khi đó, có thể giải được với nghiệm tối ưu hơn (đạt được nhiều tuần hơn) trong thời gian 2 phút
- Phương pháp IB (*được thực thi bằng C++, trên Pentium 4/ 2,4GHz-processor*). Ta có những kết luận như sau:
  - Trong 5 phút hầu hết những trường hợp mà phương pháp Intelligent Backtracking giải được thì phương pháp NDF cũng giải được trong thời gian chấp nhận được.
  - Có hai trường hợp NDF không tối ưu bằng Intelligent Backtracking (6-4-6 và 6-4-5) cụ thể hơn là số tuần của NDF đạt được kém 1 trong cả hai trường hợp

Formatted: Font: 14 pt, Italic, Font color: Black

Formatted: Font: 14 pt, Italic, Font color: Black

- Có 5 trường hợp mà NDF giải được với số nghiệm tối ưu hơn (hơn 1,2,3,4 tuần) so với phương pháp Intelligent Backtracking.

Trong chương 7, Luận văn có nêu ra một thuật toán để giải SGP cho trường hợp *p*-*p*-(*p*+1), khi p là nguyên tố, mà không tốn thời gian tìm kiếm. Từ thuật toán này, đã cũng rút ra được một kết luận là thuật toán có thể dùng để giải cho trường hợp tổng quát *number*- *number* – 3 với *number* là số bất kỳ.

Cũng từ thuật toán này, đã tạo ra được một cách xây dựng tập MOLS. Trên cơ sở đó, Luận văn cũng giải thích được mối liên hệ thú vị SGP và hình vuông Latin trực giao (MOLS).

Như vậy là từ bài toán Kirkman' schoolgirls, tưởng như là một bài toán đố vui, cũng như bài toán "Người chơi gôn" xuất phát từ một câu hỏi thực tế; Luận văn đã tiếp cận và giới thiệu những kỹ thuật trong CSPs để giải quyết cho SGP. Việc giải SGP cũng liên quan rất nhiều đến những bài toán tổ hợp (Trong SGP khi (s-1)w=gs-1, nó tương đương với bài toán "Balanced Incomplete Block Design" tại I.1 trong [9], "Steiner Triples", MOLS, …). Chính chúng có những sự tương đồng, sự bổ trợ; giải quyết một vấn đề sẽ tác động đến các vấn đề còn lại, thúc đẩy và tạo ra sự phát triển chung.

Có nhiều phương pháp, mô hình để giải SGP. Tuy nhiên tất cả các mô hình đều muốn loại bỏ đối xứng càng nhiều càng tốt (vì SGP cũng như các bài toán trong CSPs có tính đối xứng rất lớn) bằng tĩnh, động hoặc kết hợp cả hai. Việc khó khăn khi giải SGP cũng là những khó khăn chung cho cộng đồng ràng buộc khi phải đối mặt với việc giải CSPs. Mong muốn rằng trong tương lai có thể *kết hợp những kỹ thuật được đề xuất với những kỹ thuật khác như SBDD hay SBDS* để nâng hiệu quả trong việc tìm kiếm nghiệm cho SGP.

Ngoài ra cũng có thể thấy rằng có nhiều hướng để phát triển Luận văn như: Phát triển hơn nữa phương pháp (SBDS, SBDD, Intelligent Backtracking, Multiple-viewpoints, Local Search, thêm các ràng buộc dư thừa…). Mỗi phương pháp, mô hình đều đó ưu và nhược trong từng trường hợp riêng. Nếu có thể, sẽ nghiên cứu sự kết hợp giữa các phương pháp (Local Search + Ràng buộc dưa thừa, SBDD + ràng buộc dư thừa, SBDD + IB, …), trong đó có cả *sự kết hợp giữa các mô hình*, …

Việc giải SGP là công việc khó, đòi hỏi phải *có sự kết hợp những phương pháp* và thậm chí là *cần tìm thêm những phương pháp mới*, *cách tiếp cận mới*. Đây cũng tạo tiền đề và mong muốn cho những nghiên cứu trong tương lai.

Bất chấp sự nỗ lực của cộng đồng ràng buộc, trường hợp 8-4-10 cũng như một số trường hợp khác trong SGP vẫn còn mở. Nhưng chính thách thức này đã thúc đẩy cộng đồng ngày càng phát triển công nghệ ràng buộc để ngày càng phù hợp với những bài toán khó trong thực tế.

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Nguyễn Thanh Thủy, (1996), *Trí tuệ nhân tạo: Các phương pháp giải quyết vấn đề*, NXBGD. Việt Nam.

Tiếng Anh

2. Azevedo F. (2006), "An Attempt to Dynamically Break Symmetries in the Social Golfers Problem", Azevedo et al. (eds): *Proceedings of the 11th Annual ERCIM Workshop on Constraint Programming (CSCLP'2006),* pp. 101–115.
3. Azevedo F., Van Hau N. (2006), "Extra Constraint for the Social Golfers Problem", *The 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Phnom Penh, Cambodia, 13th - 17th November* (Short paper)
4. Anderson I., Honkala I. (1997), "A Short Course in Combinatorial Designs", *http://www.utu.fi/~honkala/cover.html*.
5. Association for Constraint Programming, *http://slash.math.unipd.it/acp/.*
6. Barnier N., and Brisset P. (2002), "Solving the Kirkman's Schoolgirl Problem in a Few Seconds", *Proceeding of Principles and Practice of Constraint Programming - CP 2002,* Springer Verlag,, LNCS 2470, pp. 477–491.
7. Barták R., *http://ktiml.mff.cuni.cs/~bartak/constraints/*
8. Cohen D., Jeavons P., Jefferson C., Karen E. P. and Smith B. (2005) "Symmetry Definitions for Constraint Programming", *Proceeding of Principles and Practice of Constraint Programming - CP 2005*, Springer Verlag, LNCS 3709, pp. 17-31.

9. Colbourn C. J., and Dinitz J. H.(1996), "The CRC Handbook of Combinatorial Design", CRC Press, Boca Raton, FL.

10. Constraint Programming online, *http://www.cp-online.org/*

11. Constraints Archive, *http://4c.ucc.ie/web/archive/index.jsp.*

12. Crawford J., Luks G., Ginsberg M., and Roy A. (1996), "Symmetry breaking predicates for search problems", *Proceeding of the 5th Int. Conf. on Knowledge Representation and Reasoning, (KR '96)*, pp. 148–159.

13. Dotu I., and Van Hentenryck P. (2005), "Scheduling Social Golfers Locally", *Proceedings of the Second International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, Springer Verlag, LNCS 3524, pp. 155–167.

14. Fahle S., Torsten, Stefan, and Sellmann M. "Symmetry breaking", *Proceeding of Principles and Practice of Constraint Programming - CP 2001,* Springer Verlag, LNCS 2239, pp. 93–107.

15. Flener P., Frisch A., Hnich B., Kiziltan Z., Miguel I., Walsh T. (2002), "Breaking row and column symmetry in matrix models", *Proceeding of Principles and Practice of Constraint Programming - CP 2002*, Springer Verlag, LNCS 2470, pp. 462-476.

16. Focacci F., and Milano M. "Global cut framework for removing symmetries", *Proceeding of Principles and Practice of Constraint Programming - CP 2001,* Springer Verlag, LNCS 2239, pp. 77–92.

17. Frisch A., Hnich B., Kiziltan Z., Miguel I., Walsh T. (2002), "Global constraints for lexicographic orderings", *Proceeding of Principles and Practice of Constraint Programming - CP 2002*, Springer Verlag, LNCS 2470, pp. 93-108.

18.Gent I.P. and Lynce I. (2005), "A SAT Encoding for the Social Golfer Problem", *Proceeding of IJCAI'05 workshop on Modelling and Solving Problems with Constraints*.

19.Gent I.P., and Smith B. (2000), "Symmetry breaking during search in contraint programming", *W. Horn, editor, EACI'2000*, pp. 599–603.

20.Harvey W. (2001), "Symmetry Breaking and the Social Golfer Problem", *Proceedings of SymCon-01: Symmetry in Constraints*, pp. 9–16.

21. Harvey W. and Winterer T. (2005) "Solving the MOLR and Social Golfers Problems", *Proceedings of the 11th International Conference on Constraint Programming (CP-2005)*.

22.Karen E. P. (2003), "Comparison of Symmetry Breaking Method", *Proceeding of Principles and Practice of Constraint Programming - CP 2003*, Springer Verlag, LNCS 2833, pp. 990.

23.Kiziltan Z. (2004), "Symmetry Breaking Ordering Constraints", *PhD thesis*, Department of Information Science, Uppsala University.

24.Kzrysztof R.Apt (2003), *Princeples of Constraint Programming*, Cambridge Press.

25.Law Y.C., and Lee J.H.M. (2003), "Expressing symmetry breaking constraints using Multiple Viewpoints and channeling constraints", *Processing of the third International Workshop on Symmetry in Constraint Satisfaction Problem (SymCon'03)*, pp. 127–141.

26.Law Y.C., and Lee J.H.M. (2005), "Breaking value symmetries in matrix models using channeling constraints", *Processing of the 20th Annual ACM Symposium on Applied Computing (SAC-2005)*, pp. 375–380.

27.Law Y.C., and Lee J.H.M. (2006), "Symmetry Breaking Constraints for Value Symmetries in Constraint Satisfaction". *Constraints (2006) to*

appear 17th ECAI, European Conference on Artificial Intelligence, IOS Press.

28.Marriott K., and Stuckey P.J. (1998), *Programming with Constraints: An Introduction*, MIT Press.

29. Programming Systems Group of the Swedish Institute of Computer Science (2005), *SICStus Prolog User's Manual*.

30.Puget J.-F. (1993), "On the Satisfiability of Symmetrical Constraint Satisfaction Problems" *Proceedings of ISMIS'93 (1993)*, pp. 350–361.

31.Puget J.-F. (2002), "Symmetry breaking revisited", *Proceeding of Principles and Practice of Constraint Programming - CP 2002*, Springer Verlag, LNCS 2470, pp. 446–461.

32.Puget J.F. (2005), "Breaking symmetries in all different problems", *Proceeding of 19th IJCAI (2005)*, pp. 272-277.

33.Sellmann M., and Van Hentenryck P. (2005), "Structural symmetry breaking", *Proceeding of IJCAI'05 workshop on Modelling and Solving Problems with Constraints*.

34.Sellmann M., and Harvey, W. (2002), "Heuristic constraint propagation – Using local search for incomplete pruning and domain filtering of redundant constraints for the social golfer problem", *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pp. 191–204.

35.Smith B. (2001), "Reducing Symmetry in a Combinatorial Design Problem". *Proceedings of the International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'01)*, pp. 351–359.

36.Tsang E. (1993), *Foundations of Constraint Satisfaction*, Academic Press.

37. Van Hentenryck P. (1989), *Constraint Satisfaction in Logic Programming*, MIT Press.

38. Walsh T. (2006) "General Symmetry Breaking Constraints", *Proceeding of Principles and Practice of Constraint Programming - CP 2006*, LNCS 4204, pp. 650–664.

39. *http://csplib.org/*

40. *http://mathworld.wolfram.com/KirkmansSchoolgirlProblem.html*

41. *http://www.icparc.ic.ac.uk/~wh/golf/*

42. *http://www.icparc.ic.ac.uk/eclipse/examples/golf.pl.txt*

43. *http://www.cs.brown.edu/people/sello/solutions.html*

Page 94: [1] Formatted	**Hau**	**9/22/2006 6:34:00 PM**

Font: Bold

Page 94: [1] Formatted	**Hau**	**9/22/2006 6:34:00 PM**

Font: Bold

Page 94: [2] Formatted	**Hau**	**9/22/2006 6:14:00 PM**

Font: 14 pt

Page 94: [2] Formatted	**Hau**	**9/22/2006 6:14:00 PM**

Font: 14 pt

Page 94: [3] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [3] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [3] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [4] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [4] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [4] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [5] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [5] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [5] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [6] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [6] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [6] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [7] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [7] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [7] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [8] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [8] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [8] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [9] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [9] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [9] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [10] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [10] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [10] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [11] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [11] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [11] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [12] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [12] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [12] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [13] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [13] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [13] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt

Page 94: [14] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: 14 pt, Not Bold

Page 94: [14] Formatted	**Hau**	**9/22/2006 6:32:00 PM**

Font: Not Bold

Page 94: [15] Formatted	**Hau**	**9/22/2006 6:37:00 PM**

Font: Italic

Page 94: [15] Formatted	**Hau**	**9/22/2006 6:37:00 PM**

Font: Bold