BÀI BÁO CÁO LAB 2
CƠ SỞ TRÍ TUỆ NHÂN TẠO

Lê Hoàng Vũ - 22120461

Ngày 19 tháng 11 năm 2024

Mục lục

I

TÓM TẮT NỘI DUNG

1 Lab 2: Logic

§1.1 Tóm tắt bài toán

Cho một cơ sở tri thức (`Knowledge Base`, viết tắt là `KB`) và một câu α, cả hai được biểu diễn bằng logic mệnh đề và chuẩn hóa dưới dạng hội chuẩn `CNF` (`Conjunctive Normal Form`). Nhiệm vụ là xác định liệu `KB` $\models \alpha$ (tức là `KB` suy dẫn α) bằng cách sử dụng giải thuật `Resolution` (hợp giải).

(1) Dữ liệu đầu vào: `KB` và α theo dạng chuẩn `CNF` được lưu trong tập tin `input.txt`. Tập tin có định dạng quy ước như sau:

- Dòng đầu tiên chứa câu α.
- Dòng thứ hai chứa số nguyên N – số mệnh đề có trong `KB`.
- N dòng tiếp theo biểu diễn các mệnh đề trong `KB`, mỗi mệnh đề trên một dòng.

(2) Dữ liệu đầu ra: Tập hợp mệnh đề được phát sinh trong quá trình hợp giải và câu kết luận được lưu trong tập tin `output.txt`. Tập tin có định dạng quy ước như sau:

- Dòng đầu tiên chứa số nguyên M_1 – số mệnh đề được phát sinh trong vòng lặp đầu tiên. M_1 dòng tiếp theo biểu diễn các mệnh đề được phát sinh trong vòng lặp đầu tiên (kể cả mệnh đề rỗng), mỗi mệnh đề trên một dòng. Mệnh đề rỗng được biểu diễn bằng chuỗi `{}`.
- Các vòng lặp tiếp theo (lần lượt có M_2, M_3, ..., M_n mệnh đề) được biểu diễn tương tự như trên.
- Dòng cuối cùng trình bày câu kết luận, tức là trả lời câu hỏi "`KB` entails α?". In `YES` nếu `KB` entails α. Ngược lại, in `NO`.
- Bỏ qua các mệnh đề trùng (xuất hiện trong cùng vòng lặp, `KB` ban đầu hoặc những vòng lặp trước đó).

(3) Lưu ý:

- `Literal` dương được biểu diễn bằng ký tự đơn viết hoa (A-Z). `Literal` âm là `literal` dương có dấu trừ ('-') ngay trước ký tự.
- Từ khóa `OR` nối các `Literal` nối với nhau. Có thể có một hay nhiều khoảng trắng giữa các `Literal` và từ khóa

§1.2 Bảng đánh giá mức độ hoàn thành

Các yêu cầu chính	Nội dung	Tự đánh giá mức độ hoàn thành
Dữ liệu đầu vào và đầu ra		
`Test case`	Tạo ra 10 `test case` với độ phức tạp khác nhau để kiểm tra chương trình. Các `test case` này tuân thủ định dạng đề cho	Tốt (100%)
Xử lý dữ liệu	Dữ liệu đầu vào sau khi được đọc thì sẽ được lưu trong một cơ sở dữ liệu phù hợp	Tốt (100%)
Output	Các kịch bản (`test case`) cho ra những kết quả khác nhau phản ánh ưu điểm và khuyết điểm của thuật giải	Tốt (100%)
Source code		
Github	`PrORain-HCMUS, Lab2-AI-Fundamental`	Tốt (100%)
Giải thuật `Robinson`	Cài đặt giải thuật hợp giải trên logic mệnh đề	Tốt (100%)
	Các bước suy diễn phát sinh đủ mệnh đề và kết luận bài toán	Tốt (100%)
Những tính năng thêm vào		
Biểu diễn kết quả bài toán	In ra các bước hợp giải từ đầu đến khi kết thúc bài toán	Tốt (100%)
`GUI`	Một giao diện người dùng cho phép người dùng lấy file đầu vào từ máy và phần mềm sẽ mô phỏng bằng `graph` các bước cũng như các mệnh đề được tạo mới, cho đến kết quả	Tốt (100%)

II

CHI TIẾT NỘI DUNG

2 Giải thích thuật toán

§2.1 Khâu chuẩn bị

✎ Cây thư mục:

HW02
- *input*
  - *input_01.txt*
  - *input_02.txt*
  - ...
  - *input_09.txt*
- *output*
  - *output_01.txt*
  - *output_02.txt*
  - ...
  - *output_09.txt*
- *source*
  - *KnowledgeBase.py*
  - *PL_GUI.py*
  - *PL_Resolution.py*

✎ Ý tưởng:

(1) Đầu tiên, để đảm bảo dữ liệu đầu vào được lưu trữ trong một cơ sở dữ liệu phù hợp, ta sẽ triển khai một lớp quản lý cơ sở tri thức, gọi là `KnowledgeBase`. Về căn bản, cơ sở tri thức này sẽ ung cấp các phương thức để:

- Thêm mệnh đề mới vào cơ sở tri thức.
- Chuẩn hóa mệnh đề về dạng `CNF (Conjunctive Normal Form)`.
- Thực hiện phép hợp giải giữa các mệnh đề.
- Chứng minh một mệnh đề đích bằng phương pháp hợp giải.

(2) Sau khi đã có cơ sở dữ liệu để lưu trữ đầu vào cũng như xử lý các bước hợp giải, ta cần xây dựng một `script python` xử lý `input`. Đây sẽ là `script` chính, gọi là `PL_Resolution`, cung cấp các chức năng như:

- Đọc các mệnh đề từ `input` theo định dạng chuẩn.
- Tạo cơ sở tri thức từ các mệnh đề đọc được.
- Thực hiện chứng minh và ghi kết quả ra `output`.
- Có thể xử lý nhiều `input` cùng lúc.

(3) Sau cùng, nếu `output` có gây ra khó hiểu khi quan sát, thì sẽ có một `GUI` (`Graphical User Interface`), gọi là `PL_GUI`, cung cấp giao diện đồ họa để trực quan hóa quá trình hợp giải. `Script` này cho phép:

- `Browse input` từ trong máy.
- Hiển thị các bước hợp giải dưới dạng đồ thị có hướng.
- Tương tác với đồ thị (kéo thả các node).
- Chọn các kiểu layout khác nhau để hiển thị đồ thị.
- Theo dõi từng bước hợp giải qua `text log`.

§2.2 Khâu thực hiện

✎ Cài đặt các lớp và thuật toán:

(1) Lớp `KnowledgeBase`

Thứ tự	Phương thức	Chức năng
1	`insert`	Thêm mệnh đề mới vào cơ sở tri thức, đảm bảo không trùng lặp và không mâu thuẫn nội tại.
2	`invert_literal`	Đảo ngược một literal (thêm hoặc bỏ dấu -).
3	`invert_expression`	Phủ định toàn bộ một biểu thức logic, chuẩn hóa về dạng `CNF`.
4	`is_subset_exists`	Kiểm tra xem một mệnh đề có là tập con của một mệnh đề khác trong danh sách không.
5	`eliminate_redundant`	Loại bỏ các mệnh đề dư thừa khỏi danh sách các mệnh đề.
6	`has_contradiction`	Kiểm tra xem một mệnh đề có chứa các literal đối ngẫu không.
7	`standardize_statement`	Chuẩn hóa một mệnh đề bằng cách sắp xếp và loại bỏ các `literal` trùng lặp.
8	`apply_resolution`	Áp dụng hợp giải trên hai mệnh đề để tạo ra mệnh đề mới.
9	`prove_by_resolution`	Phương pháp chứng minh bằng hợp giải, tạo ra các bước suy diễn và kiểm tra kết quả cuối cùng.

- Bước 1: `KnowledgeBase` nhận mệnh đề `logic` cần chứng minh dưới dạng `CNF`.
- Bước 2: Chuyển biểu thức mục tiêu sang dạng phủ định qua phương thức `invert_expression`.
- Bước 3: Thêm lần lượt các mệnh đề (kể cả mệnh đề vừa phủ định) vào cơ sở tri thứ thông qua phương thức `insert`.
- Bước 4: Thông qua các phương thức như `is_subset_exists`, `eliminate_redundant` và `has_contradiction` để kiểm tra mâu thuẫn trong các mệnh đề, loại bỏ các `literal` trùng lặp hay dư thừa. Nếu mệnh đề hợp lệ thì được thêm vào cơ sở tri thức.

- Bước 5: Suy diễn và kiểm tra bằng hợp giải thông qua phương thức `prove_by_resolution`. Lặp lại hợp giải đến khi tìm được mệnh đề rỗng (`{}`) hoặc không thể tạo ra mệnh đề mới.

Với hai phương thức quan trọng nhất của `KnowledgeBase` là `apply_resolution` và `prove_by_resolution`, ta có mã giả như sau:

Algorithm 2.2.1 Knowledge Base Resolution

function ProveByResolution(KB, target) **returns** true **or** false
  inputs: KB ▷ cơ sở tri thức chứa các mệnh đề
    target ▷ mệnh đề cần chứng minh
  tempKB ← copy(KB)
  negated ← các mệnh đề phủ định của target ở dạng CNF
  clauses ← tempKB ∪ negated
  new ← { }
  loop
    for all cặp mệnh đề (C_i, C_j) trong clauses **do**
      resolvents ← ApplyResolution(C_i, C_j)
      if {} ∈ resolvents **then** ▷ Tìm thấy mệnh đề rỗng **return** true
      new ← new cup resolvents
    if new ⊆ clauses **then return** false
    clauses ← clauses ∪ new
function ApplyResolution(C_1, C_2) **returns** tập các resolvent
  resolvents ← { }
  for all literal ∈ C_1 **do**
    if ¬literal ∈ C_2 **then**
      C_{new} ← (C_1 - {literal}) ∪ (C_2 - {¬literal})
      resolvents ← resolvents ∪ $\{C_{new}\}$
  return resolvents

(2) Lớp `LogicResolver` của `script` `PL_Resolution`

Thứ tự	Phương thức	Chức năng
1	`_parse_input_file`	Phân tích nội dung file input: - Lấy mệnh đề cần query (dòng đầu). - Chuyển đổi các dòng còn lại thành các mệnh đề chuẩn CNF và thêm vào `KnowledgeBase`.
2	`_read_file`	Đọc nội dung file input từ đường dẫn, trả về `KnowledgeBase` cùng với danh sách query.
3	`_write_result`	Ghi kết quả ra file output: - Ghi từng bước hợp giải vào file. - Ghi kết quả cuối cùng (YES hoặc NO).
4	`_get_sorted_input_files`	Lấy danh sách các file input từ thư mục `input_dir`, sắp xếp theo số thứ tự trong tên file (`input_1.txt`, `input_2.txt`,...).
5	`_generate_output_filename`	Sinh tên file output tương ứng với file input (`input_1.txt` → `output_1.txt`).
6	`process_single_file`	Xử lý một file input đơn lẻ: - Đọc nội dung file. - Thực hiện hợp giải các mệnh đề của file bằng phương thức `prove_by_resolution` của `KnowledgeBase`. - Ghi kết quả ra file output.
7	`process_all_files`	Xử lý tất cả các file trong thư mục `input_dir`: - Lấy danh sách các file. - Gọi `process_single_file` cho từng file.

- Bước 1: `LogicResolver` nhận dữ liệu đầu vào trong một danh sách các `input`.
- Bước 2: Với mỗi file trong danh sách, gọi phương thức `process_single_file` để tách mệnh đề cần `query` từ dòng đầu tiên và chuyển các mệnh đề còn lại thành các mệnh đề chuẩn `CNF` và lưu vào cơ sở tri thức.
- Bước 3: Thực hiện hợp giải sử dụng phương thức `prove_by_resolution` của `KnowledgeBase`. Kết quả hợp giải (các bước suy luận và kết quả cuối) sẽ được ghi ra file `output` tương ứng bằng phương thức `_write_result`.
- Bước 4: Cuối cùng, phương thức `process_all_files` lặp qua toàn bộ danh sách file `input` và thực hiện xử lý tuần tự từng file.

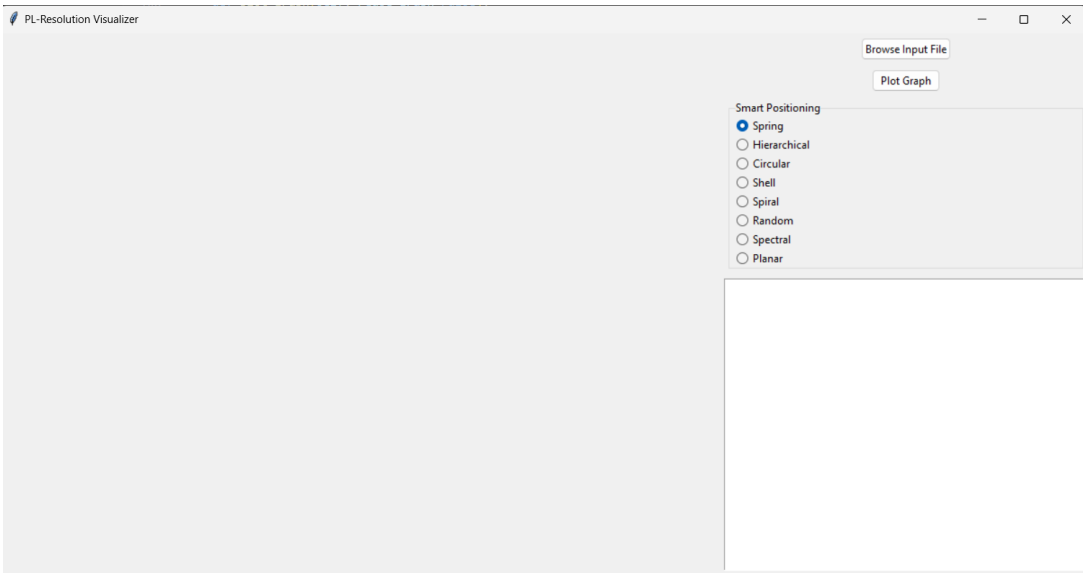
✏ Trực quan hóa kết quả bằng GUI:

Đây là một tính năng được thêm vào trong quá trình thực hiện bài tập. Ở phần này chỉ bàn về các tính năng và mô tả cách `GUI` này hoạt động.

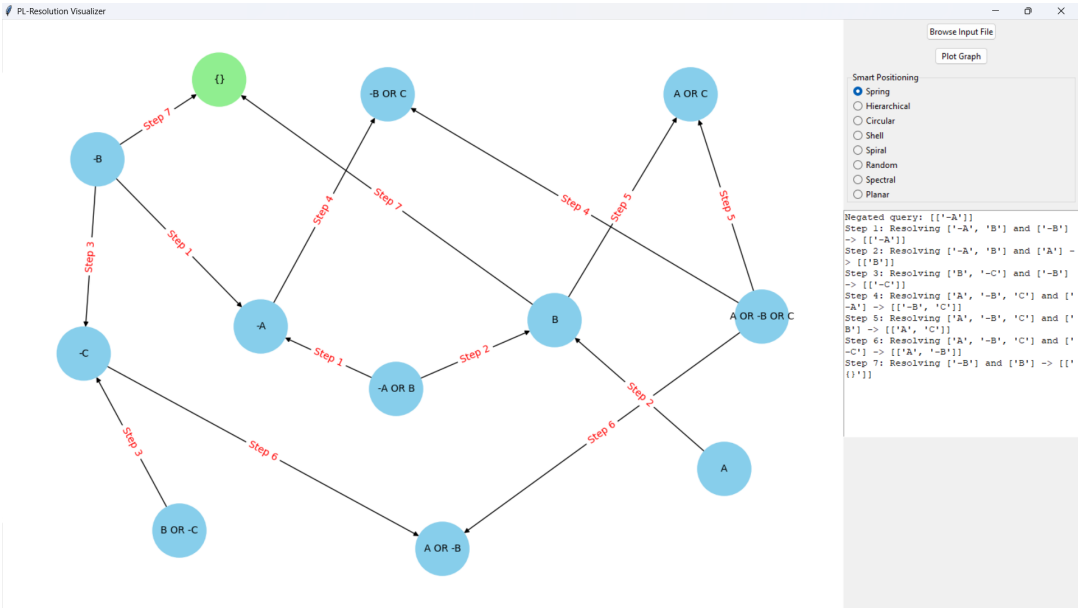
(1) Bảng tóm tắt tính năng:

Thứ tự	Tính năng	Mô tả
1	Chọn file đầu vào	Cho phép người dùng chọn file văn bản từ hệ thống để tải cơ sở tri thức và truy vấn. Tính năng được thể hiện qua phương thức: `browse_file`.
2	Đọc cơ sở tri thức	Đọc và phân tích các mệnh đề trong file được chọn, khởi tạo đối tượng `KnowledgeBase`. Tính năng được thể hiện qua phương thức: `read_knowledge_base`.
3	Vẽ đồ thị hợp giải	Vẽ đồ thị biểu diễn các bước hợp giải, sử dụng thư viện `networkx` và `matplotlib`. Tính năng được thể hiện qua phương thức: `plot_graph`.
4	Cập nhật layout đồ thị	Cập nhật lại đồ thị khi thay đổi layout, cho phép người dùng chọn kiểu hiển thị đồ thị khác nhau. Tính năng được thể hiện qua phương thức: `update_layout`.
5	Xác định và di chuyển vị trí các node	Trả về vị trí các node trong đồ thị theo layout đã chọn. Tính năng được thể hiện qua phương thức: `get_layout_positions`.
		Xử lý sự kiện click chuột để chọn node trong đồ thị. Tính năng được thể hiện qua phương thức: `on_click`.
		Xử lý sự kiện di chuyển chuột để kéo các node trong đồ thị. Tính năng được thể hiện qua phương thức: `on_motion`.
		Xử lý sự kiện khi thả chuột, hoàn tất thao tác kéo thả node. Tính năng được thể hiện qua phương thức: `on_release`.
6	Hiển thị các bước hợp giải	Hiển thị các bước hợp giải trong widget `Text`, cung cấp thông tin chi tiết cho người dùng. Tính năng được thể hiện qua phương thức: `steps_text`.
7	Vẽ đồ thị lên canvas	Vẽ đồ thị lên canvas chính trong cửa sổ ứng dụng, cho phép tương tác với đồ thị. Tính năng được thể hiện qua phương thức: `canvas`.

(2) Giao diện người dùng:



Hình 2.1: Giao diện mở đầu của phần mềm



Hình 2.2: Plot graph sau khi chọn một input từ máy

Phần mềm `PL_Resolution Visualizer` chính là tính năng được em thêm vào nhằm trực quan hóa quá trình hợp giải các mệnh đề logic. Trong cửa sổ làm việc có thể quan sát thấy 3 thành phần chính: Phần hiển thị đồ thị (`Graph visualization`), phần hiển thị các bước hợp giải (`Text log`) và phần công cụ (`Tools`). Trong đó phần công cụ bao gồm 3 thành phần nhỏ hơn, cụ thể đó là

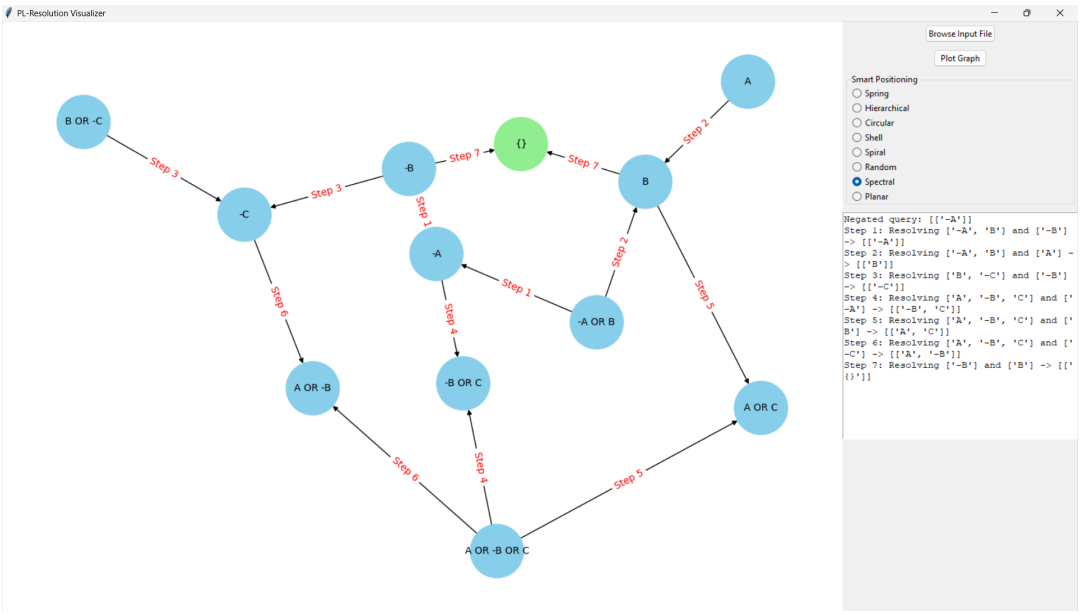
- Duyệt qua file `input` hay `Browse Input File`
- Dựng đồ thị để trực quan hóa (`Plot graph`)
- Vẽ đồ thị thông minh (`Smart Positioning`)

❒ Ngoài việc có thể tùy chỉnh các `node` của đồ thị theo ý muốn thông qua thao tác kéo thả bằng chuột, còn có một điểm nổi bật được thêm vào để tiện nghi hóa quá trình theo dõi của người dùng. Đó chính là tính năng `Smart Positioning`, cho phép tạo ra tự động đồ thị theo các `layout` (cách bố trí) khác nhau. Mô tả của những `layout` này như sau:

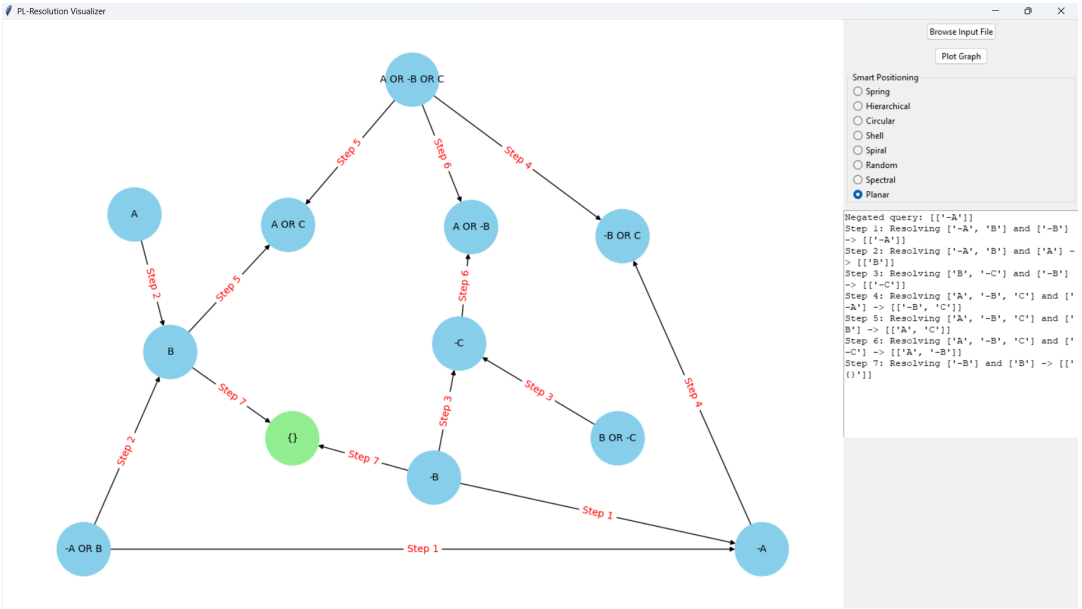
- Spring: `Layout` lò xo, các node đẩy/kéo nhau như có lò xo
- Hierarchical: `Layout` phân cấp
- Circular: `Layout` hình tròn, các node xếp thành vòng tròn
- Shell: `Layout` vỏ sò, các node xếp theo các lớp đồng tâm
- Spiral: `Layout` xoắn ốc, các node xếp theo đường xoắn ốc
- Random: `Layout` ngẫu nhiên, các node được đặt ngẫu nhiên
- Spectral: `Layout` phổ, sử dụng `eigenvectors` của ma trận `Laplacian`
- Planar: `Layout` phẳng, cố gắng vẽ đồ thị không có các cạnh cắt nhau nếu có thể

Ban đầu sau khi duyệt qua một `input` từ máy và `plot graph`, đồ thị sẽ được tự động dựng theo `layout spring`. Tuy nhiên không phải lúc nào cũng dựng ra được một đồ thị dễ quan sát, các `node`, các cạnh và chữ không bị đè lên nhau. Đó là lý do ta cần thêm những kiểu `layout` khác để phục vụ cho quá trình trực quan hóa diễn ra tốt hơn.

Sau đây là 2 hình ảnh minh họa cho việc trực quan hóa sẽ dễ nhìn hơn khi bạn chọn những `layout` để các cạnh không cắt nhau (tất nhiên là vẫn có thể cần phải điều chỉnh bằng chuột để khoảng cách các `node` giãn nhau ra hơn).



Hình 2.3: Plot graph theo layout spectral



Hình 2.4: Plot graph theo layout planar

3 Kết quả kiểm thử

§3.1 Test case

Ở Lab lần này em có chuẩn bị bộ `input` theo các loại sau:

- `Input` mẫu: Được cung cấp trong `notebook` của thầy.
- `Input` cơ bản: Những mệnh đề `logic` ở mức độ dễ hiểu và số bước hợp giải dưới `50`
- `Input` nâng cao: Được chuyển thể từ những bài toán `logic` nổi tiếng, kinh điển về dạng `CNF`.

Một ví dụ cho `input` nâng cao là bài toán Sói-Dê-Bắp cải qua sông. Đây là một bài toán kinh điển của thế kỷ IX, được phát biểu như sau:

> **Ví dụ 1 —** Một nông dân muốn vượt qua sông với một con sói, một con dê và một cái bắp cải. Người nông dân chỉ có một chiếc thuyền nhỏ và nó chỉ có thể chở theo anh ta cùng một trong ba thứ: con sói, con dê hoặc cái bắp cải. Vấn đề nằm ở chỗ nếu để sói và dê ở lại bờ, sói sẽ ăn thịt dê. Nếu để dê với bắp cải ở lại bờ, dê sẽ ăn bắp cải.
>
> Hãy tìm ra cách vận chuyển giúp người nông dân có thể mang được cả sói, dê và bắp cải qua sông.

Tất nhiên với khả năng hiện tại của các lớp hợp giải `logic` thì không tìm ra được tuần tự cách mà người nông dân di chuyển từng đối tượng còn lại qua sông như thế nào. Nhưng nếu câu hỏi là có thể thực hiện được không, thì những `script python` đã cài đặt vẫn có thể giúp ta trả lời. Cụ thể, ta sẽ chuyển bài toán trên thành các mệnh đề dạng `CNF` để áp dụng `PL Resolution` như sau:

Thứ tự	Mệnh đề	Giải thích
1	`T`	Biến T đại diện cho tình trạng "người nông dân và các vật phẩm đã qua sông" (True)
2	`-W OR -G OR F`	Sói chưa qua sông (-W), Dê chưa qua sông (-G), và Nông dân ở bờ ban đầu (F)
3	`-G OR -C OR F`	Dê chưa qua sông (-G), Bắp cải chưa qua sông (-C), và Nông dân ở bờ ban đầu (F)
4	`F OR -W`	Nông dân ở bờ ban đầu (F) hoặc Sói chưa qua sông (-W)
5	`F OR -G`	Nông dân ở bờ ban đầu (F) hoặc Dê chưa qua sông (-G)
6	`F OR -C`	Nông dân ở bờ ban đầu (F) hoặc Bắp cải chưa qua sông (-C)
7	`W OR G OR C`	Sói đã qua sông (W), Dê đã qua sông (G), hoặc Bắp cải đã qua sông (C)
8	`-F OR -W OR -G OR -C OR T`	Nông dân ở bờ bên kia (-F), Sói chưa qua sông (-W), Dê chưa qua sông (-G),
		Bắp cải chưa qua sông (-C), hoặc Biến T (tình trạng thành công)
9	`T OR F`	Biến T (thành công) hoặc Nông dân ở bờ ban đầu (F)
10	`T OR W`	Biến T (thành công) hoặc Sói đã qua sông (W)
11	`T OR G`	Biến T (thành công) hoặc Dê đã qua sông (G)
12	`T OR C`	Biến T (thành công) hoặc Bắp cải đã qua sông (C)

Bảng. Các mệnh đề và giải thích cho bài toán qua sông với nông dân, sói, dê và bắp cải.

§3.2 Output và đánh giá

✎ Output:

Với 10 file đầu vào được đặt tên từ `input_00.txt` cho đến `input_09.txt`, em đã chọn ra 5 file và chạy thử mỗi file 10 lần và lấy kết quả thời gian trung bình để làm bảng thống kê và so sánh quá trình hợp giải những file này. Trong đó thời gian thực thi sẽ được làm tròn đến chữ số thập phân thứ 4.

	`input_00.txt`	`input_03.txt`	`input_06.txt`	`input_07.txt`	`input_08.txt`
Số mệnh đề	4	5	6	9	7
Kết quả	YES	YES	YES	YES	NO
Thời gian (s)	0.0047	0.0096	0.0069	0.0683	0.0081
Số bước	7	32	17	128	24

Bảng. So sánh kết quả, thời gian thực thi và số bước với 5 file đầu vào.

✎ Nhận xét output:

(1) Về số bước:

- Dù số mệnh đề trong các file đầu vào là không lớn (tất cả đều dưới 10, nhưng số bước thực hiện cho ra trong các file đầu ra lại có sự khác biệt lớn. Chẳng hạn với `input_00.txt` yêu cầu ít bước nhất (7 bước) trong khi `input_03.txt` yêu cầu đến 128 bước.
- Tệp `input_03.txt` có số bước rất lớn mặc dù kết quả là **YES**, điều này cho thấy bài toán có thể dễ dàng tìm ra một giải pháp nhưng phải thực hiện nhiều bước để đạt được kết quả cuối cùng.
- Trong khi đó tệp `input_08.txt` mặc dù có số bước tương đối thấp (17 bước), lại không tìm được giải pháp (kết quả **NO**). Điều này chỉ ra rằng số bước không phải là yếu tố duy nhất quyết định đến kết quả, mà còn có thể liên quan đến cấu trúc hay bản chất của bài toán.

(2) Mối quan hệ giữa thời gian và số bước:

- Nhìn vào bảng thống kê trên, tệp có số bước lớn nhất là `input_07.txt` (128 bước), đồng thời cũng có thời gian thực thi dài nhất (0.0683 giây). Có thể thấy dường như với số bước càng nhiều thì thời gian xử lý cho ra kết quả cũng càng lâu.
- Tuy nhiên số bước cũng không hẳn là yếu tố duy nhất ảnh hưởng đến thời gian. Chúng ta cũng nên cân nhắc về số mệnh đề mà cơ sở tri thức cần phải xử lý. Với việc tệp `input_07.txt` có số mệnh đề lớn nhất (9 mệnh đề), cũng khiến số bước và thời gian tăng đáng kể.

✎ Đánh giá ưu-nhược điểm:

(1) Ưu điểm:

- Hiệu quả: Nếu mệnh đề cần chứng minh là đúng, thuật toán sẽ tìm ra cách chứng minh chính xác.
- Tiêu chuẩn: Chỉ cần đầu vào ở dạng mệnh đề chuẩn tắc (`CNF`), thuật toán có thể áp dụng ngay mà không cần nhiều điều chỉnh phức tạp. Điều này giúp thuật toán này trở thành nền tảng cho các công cụ kiểm tra tính thỏa mãn (`SAT solvers`) và các hệ thống suy diễn tự động trong `AI`, như trong chứng minh định lý tự động.
- Dễ triển khai: Quy trình áp dụng luật hợp giải trực tiếp, rõ ràng, dễ cài đặt và kiểm tra.

(2) Nhược điểm:

- Độ phức tạp tính toán: Độ phức tạp thuật toán cao, đặc biệt với các mệnh đề lớN, Số lượng phép hợp giải tăng theo cấp số nhân khi số lượng mệnh đề tăng dẫn đến chi phí tính toán rất lớn (tính `NP-Complete`). Điều này có thể gây ra nổ tổ hợp `Combinatorial Explosion`.
- Khó áp dụng cho các bài toán lớn: Với cơ sở tri thức chứa nhiều mệnh đề hoặc bài toán phức tạp, thuật toán trở nên kém hiệu quả và đòi hỏi tài nguyên bộ nhớ đáng kể.
- Hạn chế về đầu vào: Hiện tại thuật toán chỉ áp dụng cho những đầu vào đã chuẩn hóa `CNF`,có thể làm tăng kích thước và độ phức tạp của biểu thức. Điều này làm tăng chi phí xử lý trước. Tuy có thể cải tiến để đọc được những định dạng `input` chưa chuẩn hóa nhưng điều này sẽ đi kèm với độ phức tạp trong quá trình cài đặt.
- Hạn chế về tính trực quan: Dù tìm được kết quả đúng, thuật toán thường không trực quan trong việc giải thích vì sao mệnh đề là đúng hoặc sai. Điều này có thể làm giảm tính hữu ích trong một số ứng dụng.
- Không tối ưu hóa: Hợp giải không có chiến lược tối ưu hóa, dẫn đến nhiều bước hợp giải không cần thiết trong quá trình đưa đến kết quả.
- Hạn chế trong xử lý `Logic`: Thuật toán chỉ đang giải quyết được trong phạm vi `Logic` mệnh đề mà không thể trực tiếp áp dụng cho các hệ thống `Logic` bậc cao hoặc `Logic` vị từ mà không mở rộng.

§3.3 Đề xuất giải pháp

❒ Ở đây những giải pháp được đề xuất chủ yếu để cải thiện tính tối ưu của thuật giải trong phạm vi `Logic` mệnh đề.

✎ Forward Chaining (Dây chuyền tiến)

(1) Ý tưởng chính:

- `Forward Chaining` là một phương pháp suy luận theo kiểu dữ kiện (`data-driven`), trong đó các mệnh đề mới được suy luận từ các mệnh đề có sẵn.
- Bắt đầu từ các mệnh đề đã biết (hoặc mệnh đề cơ sở), tiến hành tìm kiếm các mệnh đề có thể suy ra từ chúng và cập nhật danh sách các mệnh đề đã suy luận (inferred). Quá trình này tiếp tục cho đến khi tìm được kết quả mong muốn hoặc không còn mệnh đề nào có thể suy luận thêm.

(2) Lý do tối ưu:

- Tiết kiệm thời gian tìm kiếm: Phương pháp này tự động tiếp cận các mệnh đề có thể suy luận từ các mệnh đề hiện có mà không phải quay lại kiểm tra lại các kết quả trước đó.
- Thích hợp cho bài toán với tập dữ liệu lớn: Khi có nhiều mệnh đề cơ sở, `Forward Chaining` sẽ giúp nhanh chóng loại bỏ các mệnh đề không có khả năng suy ra, giúp giảm thiểu không gian tìm kiếm.
- Linh hoạt và dễ triển khai: Đặc biệt hữu ích khi muốn xây dựng hệ thống khuyến nghị hoặc hệ thống ra quyết định tự động.

✎ Backward Chaining (Dây chuyền lùi)

(1) Ý tưởng chính:

- `Backward Chaining` là một phương pháp suy luận theo kiểu mục tiêu (`goal-driven`), trong đó bắt đầu từ một mục tiêu hoặc câu hỏi cần chứng minh và tìm kiếm các mệnh đề trong cơ sở dữ liệu có thể chứng minh mục tiêu đó.
- Phương pháp này sẽ kiểm tra khả năng chứng minh mục tiêu bằng cách lùi lại và kiểm tra các mệnh đề trung gian có liên quan, lặp lại quá trình cho đến khi đạt được kết quả mong muốn hoặc không thể chứng minh được.

(2) Lý do tối ưu:

- Giảm không gian tìm kiếm: `Backward Chaining` chỉ tập trung vào các mệnh đề có liên quan trực tiếp đến mục tiêu cần chứng minh, giúp tiết kiệm thời gian và tài nguyên tính toán.
- Hiệu quả với các bài toán chứng minh: Phương pháp này rất phù hợp trong các bài toán kiểm tra chứng minh, chẳng hạn như kiểm tra tính đúng đắn của một giả thuyết hoặc giải quyết bài toán có cấu trúc logic.
- Phù hợp với các hệ thống chứng minh tự động: Được sử dụng rộng rãi trong các hệ thống chứng minh lý thuyết hoặc AI khi cần tìm kiếm chứng cứ logic cho các câu hỏi.

✎ Heuristic Resolution (Giải quyết theo cách tiếp cận heuristic)

(1) Ý tưởng chính:

- `Heuristic Resolution` sử dụng các kỹ thuật heuristic để ưu tiên các mệnh đề hoặc hướng suy luận có khả năng mang lại kết quả nhanh chóng, thay vì kiểm tra tất cả các khả năng một cách ngẫu nhiên hoặc tuần tự.
- Phương pháp này lựa chọn các mệnh đề có tính khả thi cao hoặc đơn giản hơn, dựa vào một số tiêu chí đánh giá như độ dài mệnh đề, mức độ liên quan, giúp tăng tốc quá trình suy luận.

(2) Lý do tối ưu:

- Giảm độ phức tạp tính toán: `Heuristic Resolution` giúp hệ thống tìm kiếm giải pháp theo thứ tự hợp lý, chỉ xử lý các mệnh đề có khả năng mang lại kết quả sớm nhất, từ đó tiết kiệm tài nguyên tính toán.
- Tăng tốc quá trình suy luận: Phương pháp này đặc biệt hiệu quả trong các bài toán tối ưu hoặc bài toán có không gian tìm kiếm lớn, giúp rút ngắn thời gian tìm kiếm và tính toán.
- Ứng dụng trong bài toán tối ưu: Thường được sử dụng trong các tình huống cần tối ưu hóa, như trong các hệ thống tìm kiếm thông minh hoặc khi giải quyết các vấn đề có quy mô lớn.

4 Tài liệu tham khảo

[1] Giáo trình Cơ sở Trí tuệ Nhân tạo, Thầy Lê Hoài Bắc, Thầy Tô Hoài Việt - Nhà xuất bản Khoa học và Kỹ thuật - Chính sửa và bổ sung năm 2014.
[2] Artificial Intelligence A Modern Approach - Third Edition, Stuart Russell, Peter Norvig
[3] A novel approach of solving the CNF-SAT problem
[4] Artificial Intelligence Methods –WS 2005/2006 –Marc Erich Latoschik
[5] A brief history of reasoning - Massachusetts Institute of Technology