Explanation of Survey_Property

Kengo Hayashi

Here is how to handle the system we have developed, Survey_Property.

First, overall, this system repeatedly performs the following steps: "randomly generating AFs (graphs) and the rankings for them, and then checking if they satisfy the conditions of each property expressed as functions." Through this process, the system investigates whether there are dependencies between the properties based on the results.
Furthermore, to briefly outline the overall structure, it begins with definitions of several variables, followed by definitions of the functions used, then the definitions of each property, and finally the definition of the main function to be repeatedly executed.

This folder contains include the following two files:

- **config_and_execute.py**
  This file allows you to input parameters such as the range of the number of nodes in the graph to be surveyed and the number of iterations. By executing this file, the actual survey will commence.
- **main.py**
  This file contains the source code of the system's program. There is no need to manipulate this file to operate the system.

To run the system, you should operate the **config_and_execute.py** file. You can modify the parameters provided within this file. Detailed explanations of the parameters that can be set in this file are as follows.

config_and_execute.py

- **num_to_one**:
Sets the number of properties for which you want to investigate dependencies against one. In other words, it determines how many properties will be on the left side of the dependency (Properties → property). For example, if you want to investigate a 2-to-1 relationship such as (property1 and property2 → property3), set this variable to 2.

・**num_node**:

Sets the range of the number of nodes for the generated graph. If set to [3,6], graphs with nodes ranging from 3 to 6 will be generated.

・**per_edge**:
Sets the range of edge generation probabilities for the generated graph.

・**num_check**:
Sets how many steps to output intermediate results.

・**num_finish**:
Sets at which step the system should stop.

・**checkout_c_2**:
This is a list to set combinations for filtering dependencies from the system results. It is useful to include known incompatibility relationships in this list.

The remaining variables are for mechanisms that can be used to efficiently execute the investigation:

・**check_property_sign, check_property_num**:
If you want to view only AFs that satisfy a specific property, you can set these variables to achieve that. Setting check_property_sign to True enables this feature, and check_property_num specifies which property to check.

・**fix_sign, fix_situation, fix_arguments, fix_attacks**:
If you want to fix a particular graph for investigation, you can set these variables to achieve that. Setting fix_sign to True enables this feature, fix_situation determines whether to add a defense branch, add an attack branch, or not perform any addition. fix_arguments and fix_attacks set the actual shape of the fixed graph.

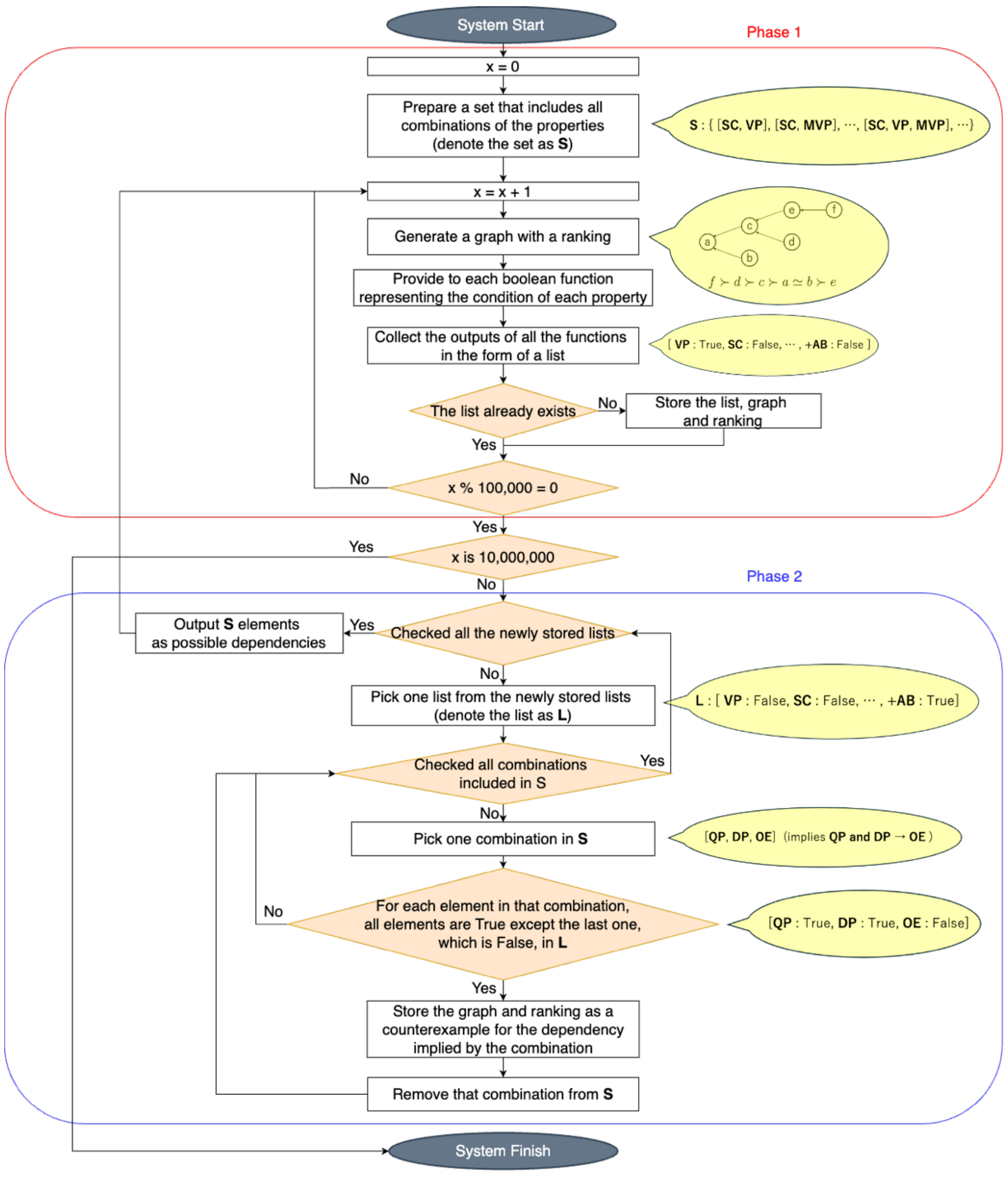
・**fix_rank_sign, fix_rank**:
If you want to fix not only a specific graph but also the rank (i.e., if you have a specific graph and ranking in mind for the investigation), you can set these variables to achieve that. Setting fix_rank_sign to True enables this feature, and fix_rank sets the actual ranking to be given.

Next, explain the structure and algorithm of **main.py** as follows.

main.py

The overall structure is as follows: first, definitions of several variables, followed by definitions of functions to be used, then definitions of each property, and finally, the definition of the main function to repeatedly execute the steps.

The algorithm is as follows:


System Start
Phase 1
x = 0
Prepare a set that includes all combinations of the properties (denote the set as S)
S : { [SC, VP], [SC, MVP], ⋯, [SC, VP, MVP], ⋯}
x = x + 1
Generate a graph with a ranking
a
b
c
d
e
f
$f \succ d \succ c \succ a \simeq b \succ e$
Provide to each boolean function representing the condition of each property
Collect the outputs of all the functions in the form of a list
[VP : True, SC : False, ⋯ , +AB : False]
The list already exists
No
Store the list, graph and ranking
Yes
No
x % 100,000 = 0
Yes
Yes
x is 10,000,000
No
Phase 2
Output S elements as possible dependencies
Yes
Checked all the newly stored lists
No
Pick one list from the newly stored lists (denote the list as L)
L : [VP : False, SC : False, ⋯ , +AB : True]
Checked all combinations included in S
Yes
No
Pick one combination in S
[QP, DP, OE] (implies QP and DP → OE)
No
For each element in that combination, all elements are True except the last one, which is False, in L
[QP : True, DP : True, OE : False]
Yes
Store the graph and ranking as a counterexample for the dependency implied by the combination
Remove that combination from S
System Finish

Important variables are follows :

- **Matrix_A**:
This variable stores the results of the dependency investigations. Matrix_A[0] contains the results of 1-to-1 dependency investigation, Matrix_A[1] contains the results of 2-to-1 dependency investigation, and so on. Each investigation result is in the form of a nested list (denoted as L), as shown in Readme, where L[property1's number][property2's number]⋯[propertyN's number] corresponds to the result of (property1 and ⋯ and propertyN-1 → propertyN). The value of each element can be 0 or 1, indicating whether a counterexample has been found (1) or not (0). Matrix_A is periodically downloaded to the PC as **Matrix_A.pickle**(updating if it already exists). Even after terminating the program and running it again, the investigation resumes by reading from that pickle file. In other words, the investigation results are accumulated in this pickle file and continue to be updated.

- **Graph_Matrix_A**:
Similar to Matrix_A, but each element contains the actual counterexamples. Counterexamples are provided in the form [set of arguments, set of attack relations, ranking], as explained in the Readme. For cases where counterexamples haven't been found, they will contain a 0. The investigation results of counterexamples will be periodically downloaded to the PC under the name **Graph_Matrix_A.pickle** (updating if it already exists). The findings of counterexamples will be accumulated and continually updated in this pickle file.

- **Table_A**, **UntilT**, **Previous_num, Dep**:
These variables are also periodically downloaded as pickle files on your PC to facilitate smooth execution of the program. They are essential components for ensuring the program runs correctly, even if parameters like num_to_one are changed midway.