Báo cáo giải quyết bài toán Weapon – Target Assignment

**1. Đặt vấn đề**

Đối với báo cáo trước, ta đã chỉ ra được 2 vấn đề còn tồn tại của bài toán như sau

* Vấn đề 1:

Trong quá trình mô phỏng, sẽ có nhiều lúc gặp phải trường hợp các biến trạng thái x không hội tụ về hai giá trị 0 hoặc 1 có thể do việc lựa chọn p và u ngẫu nhiên sẽ làm xuất hiện các cặp nghiệm cho ra kết quả giống nhau.

* Vấn đề 2:

Đối với số lượng lớn, khối lượng tính toán tăng lên rất nhiều lần dẫn tới việc thời gian thu được kết quả lớn. Hiện tượng này có thể được giải thích do trong quá trình tính đạo hàm của hàm số f ta phải đạo hàm theo từng biến trạng thái của ma trận x. Các phép tính này là độc lập với nhau tuy nhiên hiện nay chương trình đang sử dụng tính toán một cách tuần tự nên việc tốn thời gian tính toán là không thể tránh khỏi.

Sau đây là phương án mà em đã làm để giải quyết 2 vấn đề trên.

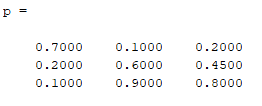
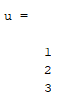
**2. Nguyên nhân và phương án giải quyết**

**2.1 Vấn đề 1**

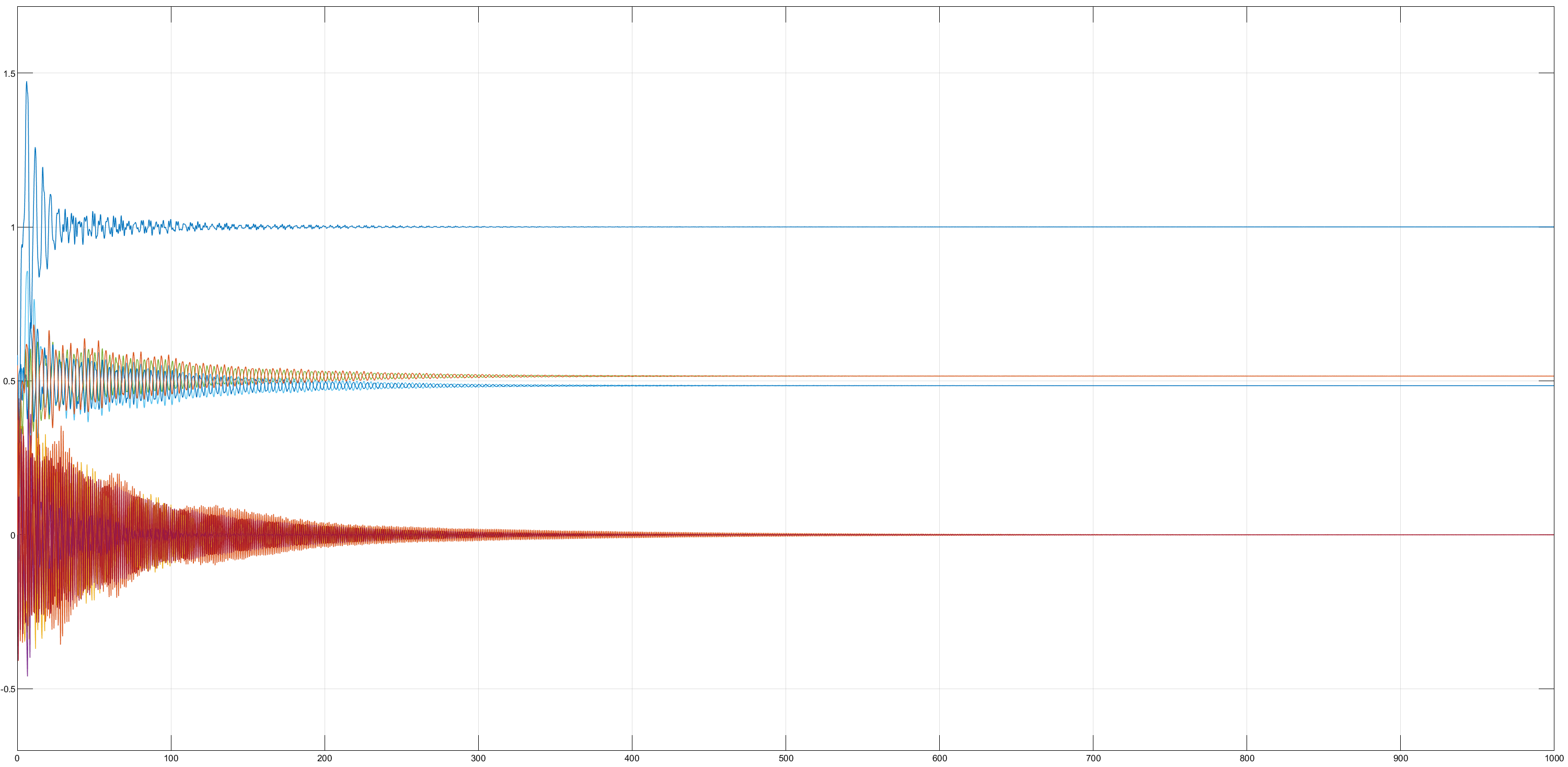
A. Nguyên nhân

Sau khi thử nghiệm, khẳng định được nguyên nhân làm cho các biến trạng thái không hội tụ về 0 là do khi chọn ngẫu nhiên u và p sẽ xuất hiện các khả năng tạo ra các tình huống khác nhau nhưng lại có kết quả giống nhau ( giá trị hàm F là giống nhau). Để mô tả hiện tượng này, ta lấy một ví dụ đơn giản với số mục tiêu và số vũ khí bằng 3.

Giá trị u và p được chọn như sau

Sau khi chạy thuật toán ra được kết quả như hình



Ta thấy x không hội tụ về 1 hay 0 mà có giá trị hội tụ về 0.5. Tức là sẽ có 2 trường hợp cho ta giá trị hàm F là tối ưu

Kiểm chứng với bảng sau

|  |  |
| --- | --- |
| x | F |
| [1 0 0; 0 1 0; 0 0 1] | 1.7 |
| [1 0 0; 0 0 1; 0 1 0] | 1.7 |
| [0 1 0; 1 0 0; 0 0 1] | 3.1 |
| [0 1 0; 0 0 1; 1 0 0] | 4.7 |
| [0 0 1; 1 0 0; 0 1 0] | 2.7 |
| [0 0 1; 0 1 0; 1 0 0] | 4.3 |

Ta thấy giả thiết đã đưa ra phù hợp với thực tế.

B. Cách khắc phục

Ta sẽ xử lý như sau

* Sau khi đã có được giá trị hội tụ của mảng x, trước tiên với các giá trị hội tụ các xác xuất lớn, gần với giá trị 1 ta sẽ cho giá trị của nó bằng 1 và giá trị của các x khác với cùng giá trị hàng hoặc cùng giá trị cột bằng 0 để giảm khối lượng cần phải xử lý.
* Còn lại sẽ là các giá trị hội tụ trong khoảng mà ta cần phải xử lý, tiến hành như sau
* Phần tử x tại hàng i cột j sẽ được so sánh với toàn bộ x cùng hang và x cùng cột. Nếu thấy giá trị của x[i][j] là lớn nhất trong các giá trị đó, ta gán cho x[i][j] giá trị bằng 1
* Quét lần cuối trong mảng đã xử lý, nếu giá trị của x[i][j] bất kỳ khác 1 thì cho nó về 0.

**2.2. Vấn đề 2**

A. Nguyên nhân

Có 2 nguyên nhân chính gây nên việc tốn thời gian trong tính toán

* Vẽ đồ thị của x sau mỗi vòng lặp tính toán
* Sử dụng quá nhiều vòng lặp for

Trong đó nguyên nhân thứ 2 là chủ yếu gây nên hiện tượng chậm trễ trong tính toán, việc vẽ đồ thị sẽ chỉ có ý nghĩa đối với bài toán có số lượng vũ khí – mục tiêu nhỏ, giúp ta có cái nhìn trực quan hơn về sự hội tụ. Tuy nhiên khi số lượng tăng lên ta đồ thị sẽ trở nên vô nghĩa. Mặt khác, ta cũng chỉ cần giá trị cuối cùng nên cũng không nhất thiết phải quan tâm đến toàn bộ giá trị của x theo từng vòng lặp tính toán.

B. Cách khắc phục

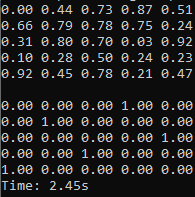
* Để xử lý nguyên nhân thứ nhất, trong tuần qua, em đã thử thuật toán đã bỏ qua phần vẽ đồ thị và nhận thấy kết quả đạt được khả quan hơn khá nhiều. Sau khi chạy thuật toán trên hai ngôn ngữ khác đó là C++ và python thì thấy C++ cho thời gian tính toán nhanh nhất.
* Đối với nguyên nhân thứ hai, em cũng đã thử sử dụng công cụ trong python để tính toán các vòng lặp for theo cách tính song song, kết quả đạt được giảm được 20% thời gian tính toán. Tuy nhiên, cách này mới chỉ sử dụng đa luồng của CPU để tính toán. Hy vọng sử dụng GPU sẽ có kết quả tốt hơn. Đối với chương trình viết bằng C++ em chưa có thời gian để thử thuật toán song song.

**3. Kết quả thu được**

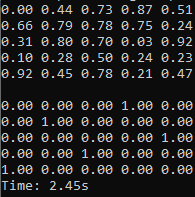
Sau đây sẽ là kết quả tốt nhất đạt được, sử dụng ngôn ngữ lập trình C++ với phần mềm Visual Stuio.

* Đối với tình huống được cho trong bài báo
* Đối với tình huống tạo một cách bất kỳ
* Với n = 5

Ma trận p được sinh ngẫu nhiên, u có giá trị tăng dần

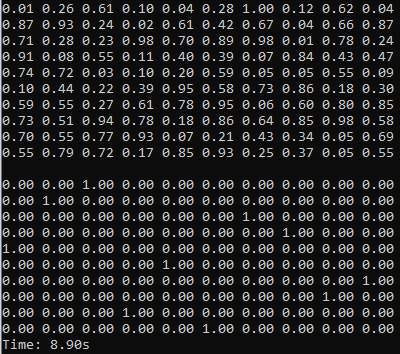


Ta có kết quả

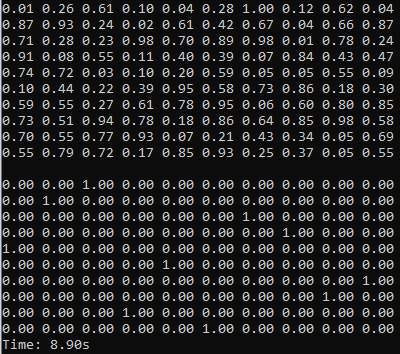


* Với n = 10

Giá trị p được sinh ngẫu nhiên, u có giá trị tăng dần

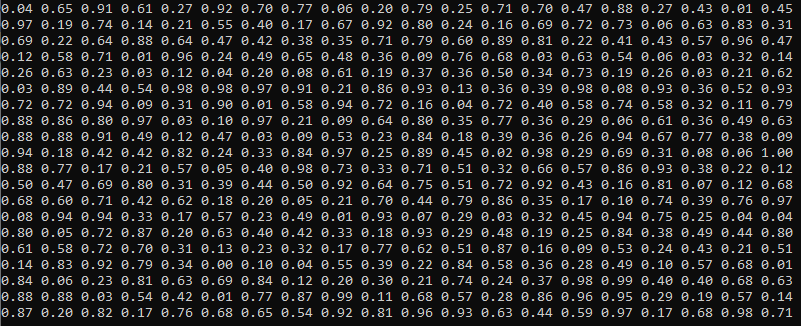


Ta thu được kết quả

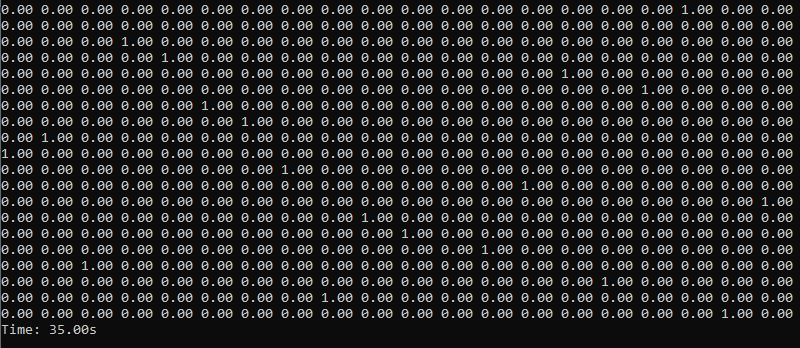


* Với n = 20

Ma trận p được sinh ngẫu nhiên, u có giá trị tăng dần



Thu được kết quả như sau

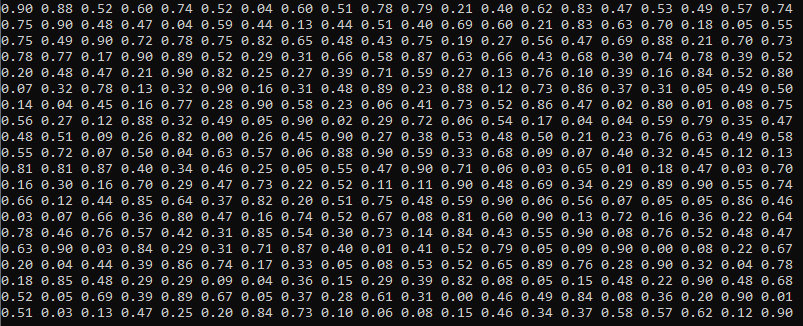


* Tổng kết lại ta có bảng sau

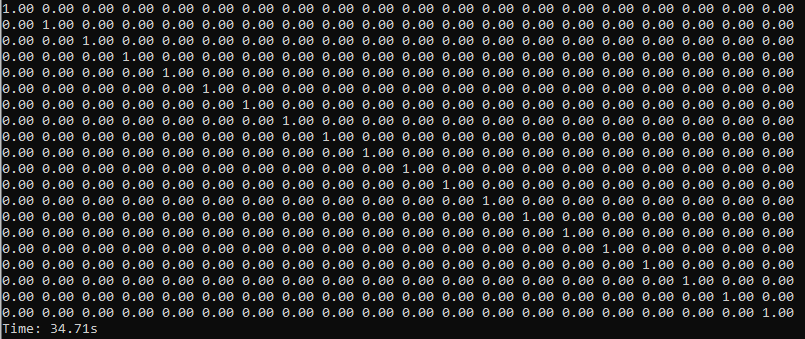
|  |  |
| --- | --- |
| N | Time |
| 5 | 2.45 s |
| 10 | 8.90 s |
| 20 | 35.00 s |
| 50 | 206.47 s |
| 80 | 522.06 s |
| 100 | 820.59 s |
| 200 | > 30 phút |

* Để có thể kiểm chứng kết quả trên, ta sẽ chạy thuật toán với tình huống biết trước kết quả như sau
* Số lượng N=T=W=20
* p sẽ là ma trận sao cho đường chéo có xác suất là 0.9, còn lại các giá trị khác sẽ là ngẫu nhiên và nhỏ hơn 0.9.
* u là mảng với các giá trị tăng dần từ 1 đến 20

Giá trị của p



Kết quả thu được



Nhận thấy ta thu được kết quả x là ma trận đơn vị. Đúng với giả thiết đặt ra

Nhận xét và định hướng phát triển

* Hiện tại vấn đề 1 coi như đã giải quyết xong.
* Vấn đề 2 tuy đã tối ưu được một phần tuy nhiên nhận thấy thời gian chạy thuật toán đối với số lượng mục tiêu – vũ khí lớn vẫn đang dài. Do vậy vẫn cần phải tối ưu thêm về mặt thời gian. Dự định sẽ sử dụng công cụ CUDA cho C++ để có thể sử dụng GPU cho việc tính toán song song.
* Hiện tại mới đang sử dụng Console App, cần phải làm thêm giao diện để giao tiếp với người dùng một cách trực quan hơn.