# ***Mục Lục***

[Mục Lục 1](#_Toc471591346)

[DANH MỤC CÁC CHỮ VIẾT TẮT 3](#_Toc471591347)

[DANH MỤC CÁC BẢNG 3](#_Toc471591348)

[DANH MỤC CÁC HÌNH ẢNH 4](#_Toc471591349)

[Lời mở đầu 5](#_Toc471591350)

[Chương 1](#_Toc471591351). [GIỚI THIỆU VỀ GIẢI THUẬT](#_Toc471591352) [DI TRUYỀN 7](#_Toc471591353)

[1.1. Tổng quan về Giải Thuật Di Truyền: 7](#_Toc471591354)

[*1.1.1.* *Khái niệm* 7](#_Toc471591355)

[*1.1.2.* *Mục tiêu nghiên cứu Giải Thuật Di Truyền:* 8](#_Toc471591356)

[*1.1.3.* *Tính chất quan trọng của Giải Thuật Di Truyền:* 8](#_Toc471591357)

[*1.1.4.* *Sự khác nhau giữa Giải Thuật Di Truyền và những giải thuật khác:* 8](#_Toc471591358)

[1.2. Cấu trúc của giải thuật di truyền: 9](#_Toc471591359)

[1.3. Các thao tác cơ bản: 11](#_Toc471591360)

[*1.3.1.* *Mã hoá* 11](#_Toc471591361)

[*1.3.2.* *Phát sinh quần thể ban đầu* 13](#_Toc471591362)

[*1.3.3.* *Hàm đánh giá và hàm thích nghi* 14](#_Toc471591363)

[*1.3.4.* *Các toán tử di truyền* 14](#_Toc471591364)

[1.4. Các bước quan trọng trong việc áp dụng giải thuật di truyền: 20](#_Toc471591366)

[Chương 2](#_Toc471591367).[BÀI TOÁN TÌM KIẾM VĂN BẢN](#_Toc471591368) [VỚI GIẢI THUẬT DI TRUYỀN 22](#_Toc471591369)

[2.1. Yêu cầu đặt ra bài toán tìm kiếm văn bản 22](#_Toc471591370)

[2.2. Xây dựng hàm tìm kiếm 23](#_Toc471591371)

[2.3. Phát biểu bài toán tìm kiếm văn bản theo hướng tiếp cận di truyền 24](#_Toc471591372)

[2.4. Tìm độ dài xâu con chung lớn nhất bằng quy hoạch động 25](#_Toc471591373)

[2.5. Áp dụng giải thuật di truyền 27](#_Toc471591374)

[*2.5.1.* *Biểu diễn nhiễm sắc thể* 27](#_Toc471591375)

[*2.5.2.* *Khởi tạo quần thể* 28](#_Toc471591376)

[*2.5.3.* *Hàm mục tiêu* 28](#_Toc471591377)

[*2.5.4.* *Các toán tử di truyền* 29](#_Toc471591378)

[*2.5.5.* *Các tham số* 29](#_Toc471591379)

[*2.5.6.* *Chi phí thời gian* 29](#_Toc471591380)

[Chương 3](#_Toc471591381). [ỨNG DỰNG GIẢI THUẬT DI TRUYỀN](#_Toc471591382) [CHO BÀI TOÁN TÌM KIẾM VĂN BẢN 31](#_Toc471591383)

[3.1. Giới thiệu chung 31](#_Toc471591384)

[*3.1.1.* *Giới thiệu về ứng dụng* 31](#_Toc471591385)

[*3.1.2.* *Giao diện chính* 31](#_Toc471591386)

[*3.1.3.* *Các tham số đầu vào và đầu ra* 32](#_Toc471591387)

[*3.1.4.* *Chức năng của các trường và nút lệnh* 33](#_Toc471591388)

[3.2. Một số ví dụ thử nghiệm 36](#_Toc471591389)

[*3.2.1.* *Bài toán 1* 36](#_Toc471591390)

[*3.2.2.* *Bài toán 2* 40](#_Toc471591391)

[*3.2.3.* *Bài toán 3:* 43](#_Toc471591392)

[Chương 4](#_Toc471591393). [TỔNG KẾT 49](#_Toc471591394)

[4.1. Nhận xét về giải thuật di truyền trong bài toán tìm kiếm văn bản 49](#_Toc471591395)

[*4.1.1.* *Ưu điểm* 49](#_Toc471591396)

[*4.1.2.* *Khuyết điểm* 49](#_Toc471591397)

[4.2. Đánh giá kết quả nghiên cứu 49](#_Toc471591398)

[4.3. Kiến nghị hướng phát triển 50](#_Toc471591399)

[4.4. Kết quả đạt được 50](#_Toc471591400)

[4.5. Hạn chế 51](#_Toc471591401)

[TÀI LIỆU THAM KHẢO 52](#_Toc471591402)

[PHỤ LỤC 53](#_Toc471591403)

# **DANH MỤC CÁC CHỮ VIẾT TẮT**

**GA** : Genetic Algorithm

**NST** : Nhiễm sắc thể

**QHĐ** : Quy hoạch động

# **DANH MỤC CÁC BẢNG**

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên bảng** | **Trang** |
| 1 | Bảng 1.1. Một số cách biểu diễn NST | 12 |
| 2 | Bảng 1.2. Một số dạng đột biến | 19 |
| 3 | Bảng 3.1. Tham số đầu vào của bài toán 1 | 36 |
| 4 | Bảng 3.2. Kết quả chạy thử nghiệm bài toán 1 | 39 |
| 5 | Bảng 3.3. Tham số đầu vào của bài toán 2 | 40 |
| 6 | Bảng 3.4. Kết quả chạy thử nghiệm bài toán 2 | 42 |
| 7 | Bảng 3.5. Tham số đầu vào của bài toán 3 | 43 |
| 8 | Bảng 3.6. Kết quả chạy thử nghiệm bài toán 3 | 46 |

# **DANH MỤC CÁC HÌNH ẢNH**

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên hình** | **Trang** |
| 1 | Hình 1.1. Sơ đồ tổng quan của giải thuật di truyền | 10 |
| 2 | Hình 1.2. Lai ghép đa điểm | 17 |
| 3 | Hình 1.3. Lai ghép đồng nhất | 17 |
| 4 | Hình 1.4. Lai ghép thứ tự | 17 |
| 5 | Hình 3.1. Giao diện chính của ứng dụng dành cho người dùng | 32 |
| 6 | Hình 3.2. Ba phần chính của giao diện | 33 |
| 7 | Hình 3.3. Phần chức năng thứ nhất của ứng dụng | 34 |
| 8 | Hình 3.4. Phần chức năng thứ hai của ứng dụng | 35 |
| 9 | Hình 3.5. Phần chức năng thứ ba của ứng dụng | 36 |
| 10 | Hình 3.6. Kết quả chạy thử nghiệm bài toán 1 lần thứ 1 | 37 |
| 11 | Hình 3.7. Kết quả chạy thử nghiệm bài toán 1 lần thứ 2 | 38 |
| 12 | Hình 3.8. Kết quả chạy thử nghiệm bài toán 1 lần thứ 3 | 39 |
| 13 | Hình 3.9. Kết quả chạy thử nghiệm bài toán 2 lần thứ 1 | 40 |
| 14 | Hình 3.10. Kết quả chạy thử nghiệm bài toán 2 lần thứ 2 | 41 |
| 15 | Hình 3.11. Kết quả chạy thử nghiệm bài toán 2 lần thứ 3 | 42 |
| 16 | Hình 3.12. Kết quả chạy thử nghiệm bài toán 3 lần thứ 1 | 44 |
| 17 | Hình 3.13. Kết quả chạy thử nghiệm bài toán 3 lần thứ 2 | 45 |
| 18 | Hình 3.14. Kết quả chạy thử nghiệm bài toán 3 lần thứ 3 | 46 |

# **Lời mở đầu**

Thuật giải di truyền, cũng như các thuật toán tiến hóa nói chung, hình thành dựa trên quan niệm cho rằng, quá trình tiến hóa tự nhiên là hoàn hảo nhất, hợp lý nhất, và tự nó đã mang tính tối ưu. Quan niệm này có thể được xem là một tiên đề đúng, không chứng minh được, nhưng phù hợp với thực tế khách quan. Quá trình tiến hóa thể hiện tính tối ưu ở chỗ, thế hệ sau bao giờ cũng tốt hơn, phát triển hơn và hoàn thiện hơn thế hệ trước.

Hiện nay, thuật toán di truyền cùng với logic mờ được ứng dụng rất rộng rãi trong các lĩnh vực tương đối phức tạp. Thuật toán di truyền kết hợp logic mờ đã chứng tỏ được hiệu quả của nó trong các vấn đề khó giải quyết bằng các phương pháp thông thường hay các phương pháp cố điển, nhất là trong các bài toán cần có sự lượng giá, đánh giá sự tối ưu của kết quả thu được. Chính vì vậy, thuật giải di truyền (Genetic Algorithm) đã trở thành đề tài nghiên cứu thú vị và mang đến nhiều ứng dụng thực tiễn ngày nay.

Thuật giải di truyền giải bài toán bằng cách mô phỏng lại và áp dụng các quy luật trong quá trình tiến hóa của tự nhiên. Phương pháp này không đảm bảo đưa ra lời giải tối ưu mà chỉ cố gắng tìm kiếm các lời giải tương đối tối ưu.

Các thuật toán tìm kiếm không có thông tin cổ điển, vét cạn sử dụng phương pháp tìm kiếm đơn giản và trực quan. Hay những phương pháp tìm kiếm có thông tin, sử dụng cấu trúc của không gian tìm kiếm giúp giảm thời gian tìm kiếm. Tuy nhiên những phương pháp này chỉ có tác dụng tốt khi áp dụng đối với những không gian tìm kiếm nhỏ. Trong thực tiễn, có rất nhiều bài toán tìm kiếm với không gian tìm kiếm lớn. Vì vậy, đòi hỏi cần có những thuật toán chất lượng hơn, và sử dụng trí tuệ nhân tạo là rất cần thiết trong giải quyết những bài toán có không gian tìm kiếm lớn. Thuật giải di truyền (genetic algorithm) là một trong những thuật giải đáp ứng được yêu cầu đó.

Hiện nay, nhiều hệ điều hành đã cung cấp công cụ hỗ trợ tìm kiếm các file văn bản chủ yếu dựa trên 2 cách là: tìm theo từ khoá tên file và tìm theo từ khoá nội dung trong file. Mặc dù đã được hỗ trợ tốt như vậy nhưng những phương pháp này vẫn còn nhiều hạn chế, chẳng hạn như chỉ tìm được các file văn bản có chính xác từ khoá chứa trong nội dung văn bản, như vậy sẽ rất khó khăn cho người dùng nếu họ không nhớ chính xác từ khoá, hơn nữa phương pháp này chưa chỉ ra được vị trí cụ thể của từ khoá trong đoạn văn bản và tần suất xuất hiện của chúng.

Vì lẽ đó bài toán tìm kiếm văn bản là bài toán rất thiết thực đang được nhiều người quan tâm, vấn đề cấp thiết đặt ra là giải quyết bài toán tìm kiếm văn bản sao cho hiệu quả, đáp ứng được nhu cầu của người sử dụng. Khoá luận này định hướng nghiên cứu sử dụng giải thuật di truyền tìm trong file văn bản các đoạn văn bản giống hoặc gần giống với mẫu (từ khoá) cần tìm kiếm.

Với mục tiêu đó, nhóm chúng em lựa chọn đề tài nghiên cứu của khoá luận là “Bài toán tìm kiếm văn bản sử dụng giải thuật di truyền”. Đây là hướng tiếp cận khá mới đối với bài toán này, hy vọng rằng kết quả đạt được sẽ có hiệu quả đáng kể so với các phương pháp tìm kiếm khác.

# **Chương 1**

# *GIỚI THIỆU VỀ GIẢI THUẬT*

# *DI TRUYỀN*

* 1. **Tổng quan về Giải Thuật Di Truyền:**
     1. *Khái niệm*

Giải thuật di truyền (GA-Genetic Algorithm) là kỹ thuật phỏng theo quá trình thích nghi tiến hóa của các quần thể sinh học dựa trên học thuyết Darwin. GA là phương pháp tìm kiếm tối ưu ngẫu nhiên bằng cách mô phỏng theo sự tiến hóa của con người hay của sinh vật. Tư tưởng của thuật toán di truyền là mô phỏng các hiện tượng tự nhiên, là kế thừa và đấu tranh sinh tồn.

Các giả thuyết trong GA thường được mô tả bằng các chuỗi bit, việc hiểu các chuỗi bit này tùy thuộc vào ứng dụng, ý tưởng các giả thuyết cũng có thể được mô tả bằng các biểu thức kí hiệu hoặc ngay cả các chương trình máy tính.

GA thuộc lớp các giải thuật xuất sắc nhưng lại rất khác các giải thuật ngẫu nhiên vì chúng kết hợp các phần tử tìm kiếm trực tiếp và ngẫu nhiên. Khác biệt quan trọng giữa tìm kiếm của GA và các phương pháp tìm kiếm khác là GA duy trì và xử lý một tập các lời giải, gọi là một quần thể (population). Trong GA, việc tìm kiếm giả thuyết thích hợp được bắt đầu với một quần thể, hay một tập hợp có chọn lọc ban đầu của các giả thuyết. Các cá thể của quần thể hiện tại khởi nguồn cho quần thể thế hệ kế tiếp bằng các hoạt động lai ghép và đột biến ngẫu nhiên – được lấy mẫu sau các quá trình tiến hóa sinh học. Ở mỗi bước, các giả thuyết trong quần thể hiện tại được ước lượng liên hệ với đại lượng thích nghi, với các giả thuyết phù hợp nhất được chọn theo xác suất là các hạt giống cho việc sản sinh thế hệ kế tiếp, gọi là cá thể (individual). Cá thể nào phát triển hơn, thích ứng hơn với môi trường sẽ tồn tại và ngược lại sẽ bị đào thải. GA có thể dò tìm thế hệ mới có độ thích nghi tốt hơn. GA giải quyết các bài toán quy hoạch toán học thông qua các quá trình cơ bản: *lai tạo*(crossover), *đột biến*(mutation) và *chọn lọc* (selection) cho các cá thể trong quần thể. Dùng GA đòi hỏi phải xác định được: khởi tạo quần thể ban đầu, hàm đánh giá các lời giải theo mức độ thích nghi – hàm mục tiêu, các toán tử di truyền tạo hàm sinh sản.

* + 1. *Mục tiêu nghiên cứu Giải Thuật Di Truyền:*

- Trừu tượng hóa và diễn đạt chính xác về các quá trình thích nghi trong hệ thống tự nhiên.

- Thiết kế những phần mềm về hệ thống nhân tạo nhằm duy trì các cơ chế quan trọng của hệ thống tự nhiên.

Những mục tiêu này đã dẫn đến những khám phá quan trọng trong hệ thống khoa học tự nhiên lẫn nhân tạo.

GA ra đời và phát triển dựa trên quá trình tiến hóa trong tự nhiên và đã được ứng dụng thành công trong nhiều lĩnh vực nhất là tối ưu hóa và máy học.

* + 1. *Tính chất quan trọng của Giải Thuật Di Truyền:*

GA lập luận có tính chất ngẫu nhiên để tìm giải pháp tối ưu cho những vấn đề phức tạp. Tuy nhiên đây là hình thức ngẫ u nhiên có hướng dẫn bởi giá trị hàm thích nghi. Chính hàm thích nghi là vật chỉ đường cho GA tìm ra lời giải tối ưu trong muôn ngàn lời giải có thể.

Vấn đề thích hợp nhất cho GA là tìm điều kiện tối ưu. Tối ưu đây không nhất thiết phải là tuyệt đối, mà có thể chỉ là tương đối trong hoàn cảnh và thời gian cho phép.

Một trong những bước quan trọng và khó khăn nhất là tìm hàm số thích nghi.

Hàm số thích nghi phải có liên hệ trực tiếp đến vấn đề cần giải quyết. GA và mạng nơron nhân tạo đều thuộc vào nhóm khoa học trí tuệ nhân tạo, tuy nhiên GA lập luận dựa theo sự tiến hóa và xét vấn đề ở tầm mức của gen và NST, khác với mạng nơron nhân tạo dựa trên kinh nghiệm và cách giải quyết vấn đề mà bộ óc con người thường dùng.

* + 1. *Sự khác nhau giữa Giải Thuật Di Truyền và những giải thuật khác:*

GA khác với những sự tối ưu hóa thông thường và những giải thuật tìm kiếm khác bởi 4 điểm sau:

* GA làm việc với sự mã hóa một bộ các thông số, chứ không phải bản thân các thông số.
* GA tìm kiếm từ một số điểm quần thể, chứ không phải từ một điểm.
* GA sử dụng các thông tin về hàm mục tiêu chứ không phải đạo hàm (derivatives) hay những tri thức phụ khác.
* GA sử dụng các luật chuyển đổi theo xác suất, chứ không phải các luật chuyển đổi tiền định.
* GA đòi hỏi một tập hợp các thông số tự nhiên của bài toán tối ưu để mã hóa thành các chuỗi có chiều dài hữu hạn, dựa trên một số hữu hạn các ký tự.
  1. **Cấu trúc của giải thuật di truyền:**

Trong GA các cá thể (hay còn gọi là các NST) được mã hóa bởi các chuỗi nhị phân, mỗi vị trí trên chuỗi nhị phân chỉ nhận một trong hai giá trị “0” hoặc “1”. Một NST trong GA có dạng như sau:

**1001101011**

GA cổ điển được J. H Holland giới thiệu để giải bài toán tối ưu: max {f (x) /x∈A}, trong đó A là một miền trong không gian n-chiều, f (x) >0 với mọi x∈A. Cấu trúc của GA cổ điển như sau:

procedure GeneticAlgorithms

begin

T = 0; // Thế hệ thứ T

Khởi tạo quần thể P(T);

Đánh giá độ thích nghi cho các cá thể trong P(T);

while (not Điều\_Kiện\_Kết\_Thúc) do

begin

T = T + 1;

Chọn lọc P(T) từ P(T-1);

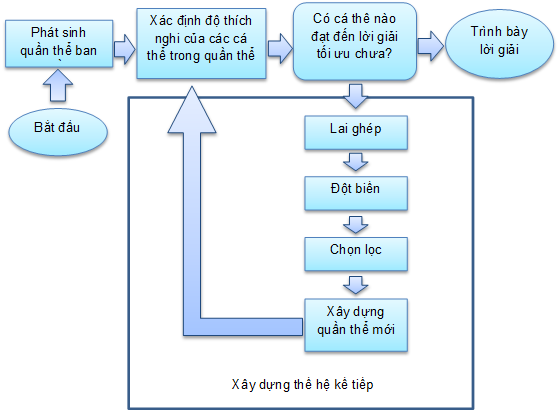
Kết hợp các cá thể (bằng lai ghép & đột biến) trong P(T);

Đánh giá độ thích nghi cho các cá thể trong P(T);

end

end

Quá trình tiến hóa được diễn ra trong vòng lặp while, tại thế hệ thứ t, giải thuật duy trì một tập lời giải P (t) ={xt1, …, xtn}. Mỗi lời giải xti được đánh giá “*độ thích nghi*”. Một tập lời giải mới được xây dựng bằng cách “*chọn lọc*” các cá thể có độ thích nghi cao hơn, ta được một tập lời giải trung gian. Tiếp theo, một số cá thể trong tập lời giải này được biến đổi bằng phương pháp “*lai ghép"  và “đột biến”* để tạo thành các lời giải mới cho thế hệ t+1. Sơ đồ sau minh họa hoạt động của giải thuật di truyền.



***Hình 1.1.*** *Sơ đồ tổng quan của giải thuật di truyền*

* 1. **Các thao tác cơ bản:**

Để giải một bài toán tối ưu bằng cách áp dụng thuật giải di truyền, ta thường phải giải quyết 5 vấn đề:

* Biểu diễn di truyền hay chọn cách mã hóa các lời giải của bài toán
* Phát sinh quần thể ban đầu P(0).
* Chọn hàm đánh giá để xác định mức độ thích nghi của lời giải
* Áp dụng các toán tử di truyền (lai ghép, đột biến) để tạo ra các cá thể mới
* Xác định các tham số đầu vào của thuật giải. Chẳng hạn như kích thước quần thể, xác suất lai ghép, xác suất đột biến,…
  + 1. *Mã hoá*

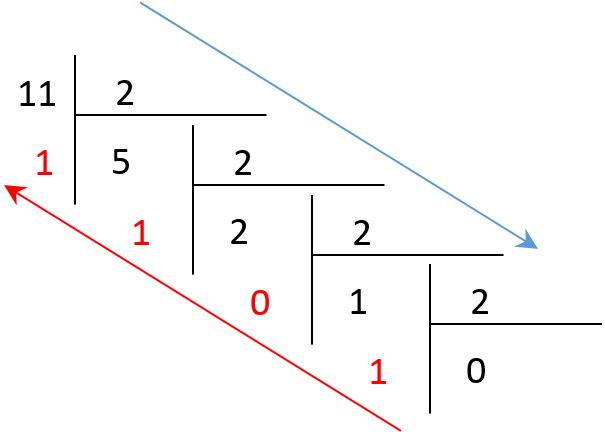
Mã hóa là việc xác định cách thức biểu diễn di truyền đối với lời giải của bài toán. Việc mã hóa phụ thuộc nhiều vào từng bài toán cụ thể. Một số cách biểu diễn thông dụng:

* Biểu diễn nhị phân: mỗi nhiễm sắc thể là một dãy số nhị phân. Mỗi gen có thể được mã hóa nhờ một số lượng bit nào đó.
* Biểu diễn theo số tự nhiên: mỗi nhiễm sắc thể là một dãy các số tự nhiên. Phương pháp này được dùng nhiều trong các bài toán tối ưu tổ hợp hay hoán vị.
* Biểu diễn số dấu chấm động: Mỗi gen trong nhiễm sắc thể được mã hóa bởi một số ở dạng dấu chấm động. Cách biểu diễn này giải quyết được một số nhược điểm của biểu diễn nhị phân và biểu diễn được miền giá trị lớn.
* Một số cách khác: Biểu diễn số ở hệ Octal (hay Hexa), biểu diễn bởi chuỗi ký tự, hay kết hợp các dạng trên.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cách biểu diễn | Nhị phân | Số tự nhiên | Số thực | Cách khác |
| Nhiễm sắc thể | 100011011  010101010 | 123459876  679218345 | 5.5 2.3 7.1 4.0  3.1 1.9 2.5 5.6 | 2314506 (Octal)  ABCDE (Hexa / chuỗi) |

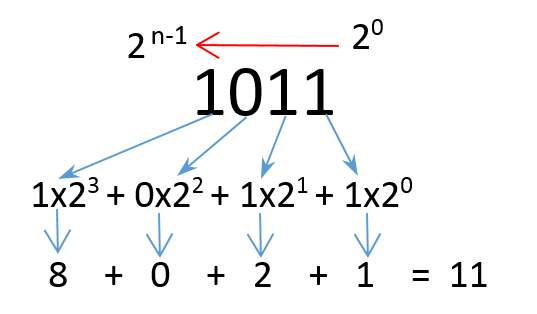
***Bảng 1.1.*** *Một số cách biểu diễn nhiễm sắc thể*

* *Đối với khóa luận này, áp dụng việc chuyển đổi từ số thập phân sang nhị phân và ngược lại.*
* Để đổi một số thập phân sang nhị phân chúng ta lấy số cần đổi chia cho 2 và sau đó lấy kết quá chia tiếp cho 2 cho đến khi ta nhận được kết quả là 0 (từ trên xuống theo mũi tên trên). Ở phép chia này ta lấy dư là 0 và 1, sau khi chia đến kết quả bằng 0 ta sẽ lấy các con số dư ghi lại từ dưới lên (theo chiều mũi tên dưới) ta được dãy số gồm 0 và 1 đây chính là giá trị ta cần tìm (Các số dư chỉ là 0 và 1, không được chia kết quả ra phần lẻ, ví dụ như 2,5).

***Ví dụ:***

Trong phép chia trên ta muốn tìm giá trị nhị phân của số 11 ta lấy số 11 chia cho 2 và sau đó chia liên tục kết quả với 2 cho đến khi nào kết quả bằng 0. Sau đó ta lầy số dư ghi lại và ta được kết quả Nhị phân của số 11 là 1011.

* Để đổi giá trị nhị phân ra thành thập phân, ta lấy dãy số nhị phân cần chuyển, nhân lần lượt các phần tử của chúng bắt dầu từ phần từ phần tử cuối (theo chiều mũi tên màu đỏ) với 20 cho đến 2n-1  (với n là số phần tử của dãy số), sau đó chúng ta tiến hành cộng các giá trị tìm được từ phép nhân ta sẽ được kết quả một con số dưới dạng thập phân.

***Ví dụ:***

Ở đây, chúng ta cần tìm giá trị của chuổi 1011 vậy ta nhân lần lượt các phần tử trong chuỗi số bắt đầu từ phần tử cuối cùng nhân với 20 đến 23 (vì ở đây dãy này có n = 4 số vậy n-1 =3). Sau đó chúng ta tiến hành cộng các giá trị tìm được từ phép nhân ta sẽ được kết quả Thập phân của dãy Nhị phân 1011 là 11.

* + 1. *Phát sinh quần thể ban đầu*

Thông thường, việc khởi tạo các cá thể trong quần thể ở thế hệ đầu tiên là phát sinh ngẫu nhiên và/hoặc có thêm một số ràng buộc (chẳng hạn sinh ra một hoán vị). Do đó, hầu hết chúng có độ thích nghi thấp hay không phải là lời giải tối ưu. Trong một số trường hợp, việc phát sinh ngẫu nhiên có thể làm quá trình tiến hóa chậm (chẳng hạn như với bài toán người du lịch).

Một cách khởi tạo khác cũng hay được áp dụng, đó là sử dụng phương pháp tham lam (greedy). Tùy vào đặc trưng của từng bài toán cụ thể, ta có thể phát sinh ra các cá thể sao cho có khả năng gần với lời giải nhất có thể. Ví dụ, với bài toán người du lịch, tại gen thứ k, ta chọn đỉnh gần nhất với đỉnh trong gen thứ k-1 và chưa được chọn.

* + 1. *Hàm đánh giá và hàm thích nghi*

Trong hầu hết các bài toán áp dụng thuật giải di truyền, ta quy về việc tối ưu cực đại (hoặc cực tiểu) một hàm một hoặc nhiều biến. Khi đó, *độ tốt của cá thể* là giá trị hàm tương ứng của cá thể đó. Như vậy, nếu chọn một cá thể là nghiệm của bài toán thì cá thể càng tốt khi giá trị hàm càng gần với giá trị tối ưu (cực đại hoặc cực tiểu) càng tốt. Hàm để *đánh giá độ tốt* của cá thể hay lời giải gọi là *hàm mục tiêu*.

Tuy nhiên, để chọn lọc các cá thể di truyền cho thế hệ tiếp theo, ta cần phải biến đổi hàm mục tiêu sang *hàm thích nghi*. Điều này cũng tương tự như trong tự nhiên, các cá thể thích nghi tốt với môi trường sống sẽ được bảo tồn cho thế hệ kế tiếp.

*Độ thích nghi của cá thể* được định nghĩa là khả năng cá thể đó được chọn lọc vào thế hệ sau hoặc được chọn cho việc lai ghép, tạo ra cá thể mới.

Hàm mục tiêu là cơ sở để đánh giá độ thích nghi của các cá thể. Việc ánh xạ từ hàm mục tiêu sang hàm thích nghi tùy thuộc vào mục đích của bài toán thực tế.

* + 1. *Các toán tử di truyền*

Trong thuật giải di truyền, các cá thể mới liên tục được sinh ra trong quá trình tiến hoá nhờ sự lai ghép ở thế hệ cha -mẹ. Một cá thể mới có thể mang những tính trạng của cha-mẹ (di truyền), cũng có thể mang những tính trạng hoàn toàn mới (đột biến). Di truyền và đột biến là hai cơ chế có vai trò quan trọng như nhau trong tiến trình tiến hoá, dù rằng đột biến xảy ra với xác xuất nhỏ hơn rất nhiều so với hiện tượng di truyền. Các thuật toán tiến hoá, tuy có những điểm khác biệt, nhưng đều mô phỏng ba toán tử cơ bản: Chọn lọc, lai ghép, đột biến.

* + - 1. *Chọn lọc*

*Chọn lọc* là quá trình chọn một cách ngẫu nhiên các cặp cá thể trong quần thể để thực hiện việc lai tạo ra cá thể con cho thế hệ sau. Mục đích của chọn lọc là chú trọng vào các cá thể có độ thích nghi cao với hi vọng rằng, các con của chúng sẽ có độ thích nghi cao hơn nữa. Có các toán tử chọn lọc sau:

\* **Toán tử chọn lọc tỷ lệ:** Được sử dụng thường xuyên nhất trong GA. Xác suất lựa chọn của mỗi cá thể tỷ lệ thuận với giá trị độ thích hợp của nó, được tính theo công thức: Pi = f (vi) /F (i = 1..pop-size – kích cỡ của quần thể) gọi là xác suất chọn cho mỗi NST vi.

Trong đó: f (vi) là hàm thích nghi của mỗi cá thể vi.

F là tổng của các giá trị thích nghi của quần thể.

Việc chọn lọc cá thể nào phụ thuộc vào vị trí xác suất qi của mỗi NST vi được tính như sau:

Tiến trình chọn lọc được thực hiện bằng cách quay bánh xe Roulette pop- size lần; mỗi lần chọn một NST từ quần thể hiện hành vào quần thể mới theo cách sau:

* Phát sinh ngẫu nhiên một số r trong khoảng [0..1]
* Nếu r < qi thì chọn NST đầu tiên (v1); ngược lại thì chọn NST thứ i, vi (2 ≤ *i* ≤ *pop* - *size* ) sao cho *qi* -1 < *r* ≤ *qi* .

Hiển nhiên, có thể sẽ có một só nhiễm sắc thể được chọn nhiều lần. Điều này phù hợp với lý thuyết sơ đồ (Nguyễn Đình Thúc, [3]): các nhiễm sắc thể tốt nhất có nhiều bản sao hơn, các nhiễm sắc thể trung bình không thay đổi, các nhiễm sắc thể kém nhất thì chết đi.

\* **Toán tử chọn lọc cạnh tranh**: Mỗi lần chọn lọc ta tiến hành chọn ngẫu nhiên t cá thể từ quần thể hiện tại. Bản sao của cá thể tốt nhất trong t cá thể kể trên được sao chép vào quần thể bố mẹ.Tiến hành N lần chọn như vậy ta thu được quần thể bố mẹ. Giá trị t được gọi là kích cỡ cạnh tranh.

\* **Toán tử chọn lọc xếp hạng**: Các cá thể của quần thể hiện tại được sắp xếp theo thứ tự giảm dần của giá trị độ thích nghi. Cá thể tốt nhất được xếp thứ nhất và cá thể tồi nhất xếp cuối cùng.

* + - 1. *Lai ghép*

*Lai ghép* là quá trình lấy hai giải pháp bố - mẹ và sinh ra con của chúng. Sau quá trình chọn lọc (sinh sản), quần thể được làm giàu bởi các cá thể tốt hơn. Lai ghép tạo ra bản sao của các cá thể tốt chứ không tạo ra các cá thể mới vì lai ghép chỉ sao chép giá trị gen giữa cặp cá thể bố - mẹ để tạo ra cá thể con. Toán tử *lai ghép* được áp dụng vào tổ hợp sinh sản với hi vọng tạo ra được thế hệ con tốt hơn.

*Quá trình được mô tả như sau:*

* + - * + Chọn ngẫu nhiên một cặp NST (để làm cha mẹ) trong quần thể. Giả sử, NST cha mẹ có cùng độ dài m.
        + Tạo một số ngẫu nhiên trong khoảng từ 1 đến m-1 (gọi là điểm lai ghép).
        + Điểm lai ghép chia NST cha mẹ thành hai chuỗi con có độ dài m1, m2.

***Ví dụ 1:***

Cha: **100100100**

Mẹ: 000011100

Thì việc trao đổi chéo các NST sau tại gen thứ 5 sẽ tạo ra hai con:

Con 1: **1001**11100

Con 2: 0000**00100**

***Ví dụ 2:***

Cha: **10110011**

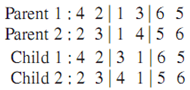
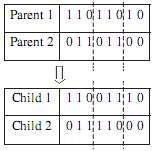
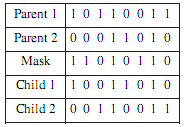
Mẹ: 10101010

Thì việc trao đổi chéo các NST sau tại gen thứ 7 sẽ tạo ra hai con:

Con 1: **101100**10

Con 2: 101010**11**

*Có một số dạng toán tử lai ghép như:*



***Hình 1.2.*** *Lai ghép đa điểm* ***Hình 1.3.*** *Lai ghép đồng nhất* ***Hình 1.4.*** *Lai ghép thứ tự*

Các kỹ thuật lai ghép thường được áp dụng:

* Lai ghép một điểm (Single-point Crossover)

Lai ghép một điểm là loại lai ghép đơn giản nhất, được sử dụng cả trong GA mã hoá nhị phân lẫn GA mã hoá số thực. Với cặp cha mẹ X, Y là các vectơ m chiều như ký hiệu trên, toán tử lai ghép 1 điểm chọn ngẫu nhiên một vị trí k (1 ≤ k ≤ m) rồi sinh ra 2 cá thể con theo công thức:

X‟ = (x1,..., xk, yk+1,..., ym )

Y‟ = (y1,..., yk, xk+1,..., xm )

***Ví dụ:***

Cha: **11010111**

Mẹ: 11111111

Thì việc lai ghép một điểm tại gen thứ 3 sẽ tạo ra hai con:

Con 1: **10**111111

Con 2: 10**010111**

* Lai ghép đa điểm (Multi-point Crossover)

Toán tử lai ghép đa điểm được mô tả như sau:

Chọn ngẫu nhiên k điểm j1,..., jk (1 <= j1 < j2 <... < jk < m), lai ghép đa điểm tạo ra cặp con (X', Y') bằng cách đánh số các đoạn [jt, jt+1] từ 0 trở đi, sau đó

* x'i lấy bằng xi tại những đoạn có số hiệu chẵn và bằng yi tại những đoạn có số hiệu lẻ.
* y'i lấy bằng xi tại những đoạn có số hiệu lẻ và bằng yi tại những đoạn có số hiệu chẵn.

***Ví dụ 1:***

Cha: **10011110**

Mẹ: 11110000

Thì việc lai ghép đa điểm với k=4 từ gen thứ 2 sẽ tạo ra hai con:

Con 1: **10**1**1**0**110**

Con 2: 11**0**1**1**000

|  |  |  |  |
| --- | --- | --- | --- |
| Con 1: | **1** | **0** 1 **1** 0 | **1 1 0** |
| Con 2: | 1 | 1 **0**  1 **1** | 0 0 0 |

***Ví dụ 2:***

Cha: **10101010**

Mẹ: 01010101

Thì việc lai ghép đa điểm với k=5 từ gen thứ 3 sẽ tạo ra hai con:

Con 1: **101**1**1**1**10**

Con 2: 010**0**0**0**01

|  |  |  |  |
| --- | --- | --- | --- |
| Con 1: | **1 0** | **1**  1 **1** 1 **1** | **0** |
| Con 2: | 0 1 | 0 **0** 0 **0** 0 | 1 |

Ngoài ra còn có một số kiểu lai ghép khác như:

* Lai ghép đồng nhất (Uniform Crossover)
* Lai ghép chu trình (CX – Cycle Crossover)
* Lai ghép thứ tự (OX – Ordered Crossover)
* Lai ghép so khớp từng phần (PMX – Partially Matched Crossover)
  + - 1. *Đột biến*

Đột biến là hiện tượng NST con mang một số đặc tính không có trong mã di truyền của cha- mẹ. Toán tử đột biến được gán xác suất pm (nhỏ hơn nhiều so với xuất suất lai ghép pc). Điều này được suy diễn bởi trong tự nhiên, đột biến gen thường rất ít xảy ra.

*Đột biến* là thay đổi một cách ngẫu nhiên giá trị của một hoặc vài gen trong NST. Nếu lai ghép được xem là khám phá các giải pháp hiện thời để tìm ra các giải pháp tốt hơn thì *đột biến* được cho là để giúp thăm dò toàn bộ không gian tìm kiếm. *Đột biến* được xem như một toán tử nền tảng để duy trì tính đa dạng di truyền trong quần thể. *Đột biến* giúp thoát khỏi cái bẫy của cực tiểu cục bộ và duy trì tính đa dạng trong quần thể.

Có nhiều dạng đột biến khác nhau tùy thuộc vào cách mã hóa hay biểu diễn của NST. Sau đây là một số kỹ thuật đột biến thường dùng:

* Đột biến đảo bit (Flipping): Thay thế giá trị của gen bằng một giá trị khác
* Đột biến đổi chổ (Interchanging): Hoán vị giá trị của hai gen
* Đột biến đảo ngược (Reversing): Đảo ngược một đoạn gen trong NST

|  |  |  |  |
| --- | --- | --- | --- |
|  | Flipping | Interchanging | Reversing |
| Parent | 1 **0** 0 1 0 **1** 0 0 **0** | 1 0 **0** 1 0 **1** 0 0 0 | 1 0 0 **1 0 1 0** **0** 0 |
| Child | 1 **1** 0 1 0 **0** 0 0 **1** | 1 0 **1** 1 0 **0** 0 0 0 | 1 0 0 **0 0 1 0 1** 0 |

***Bảng 1.2.*** *Một số dạng đột biến*

Phép đột biến được mô tả như sau:

* + - * + Chọn ngẫu nhiên một NST trong quần thể.
        + Tạo một số ngẫu nhiên k trong khoảng từ 1 tới m, 1 ≤ k ≤ m.
        + Thay đổi bít thứ k. Đưa nhiễm sắc thể này vào quần thể để tham gia quá trình tiến hóa ở thế hệ tiếp theo.

## *Ví dụ:*

v1: 1011**0**1010

v2: 1011**1**1010

NST v1 được chọn để đột biến tại vị trí gen thứ 5, gen này hiện tại là 0, sau khi đột biến sẽ trở thành 1. Khi đó NST v1 trở thành v2.

* + 1. *Các tham số của thuật giải*

*Kích thước quần thể* (PopSize) là số lượng cá thể trong mỗi thế hệ tiến hóa. Giá trị của nó phụ thuộc vào sự phức tạp của từng bài toán cụ thể. Kích thước quần thể lớn có thể giúp khám phá nhanh toàn bộ không gian tìm kiếm. Tuy nhiên, nó yêu cầu chi phí cho việc tính toán, bộ nhớ và cả thời gian thực thi. Trong thực tế, giá trị này thường được chọn là 100.

*Xác suất lai ghép* (pc) là tham số mô tả mức độ thường xuyên mà phép lai ghép được thực hiện. Nếu pc = 100% thì mọi cá thể con đều được tạo ra từ việc lai ghép. Điều này đôi khi là không tốt vì chúng ta muốn lưu giữ một phần của quần thể trước nhằm duy trì tính đa dạng của quần thể. Nếu pc = 0% thì thế hệ sau chỉ là bản sao y nguyên của thế hệ trước, có nghĩa là không có sự tiến hóa. Số lượng cá thể tham gia vào quá trình lai ghép là: pc \* PopSize.

*Xác suất đột biến* (pm) là tham số quyết định mức độ thường xuyên mà các phần tử của nhiễm sắc thể bị đột biến. Đột biến ngăn thuật giải di truyền rơi vào cực trị cục bộ. Giá trị này càng lớn thì càng nhiều cá thể bị biến đổi làm mất tính ổn định của quần thể. Số lượng gen bị đột biến trong toàn bộ quần thể sẽ là:

pm \* PopSize \* kích thước NST.

* 1. **Các bước quan trọng trong việc áp dụng giải thuật di truyền:**

Để giải quyết vấn đề bài toán bằng giải thuật di truyền, chúng ta cần thực hiện 7 bước quan trọng sau:

***Bước 1*:** Chọn mô hình cho giải pháp của vấn đề, chọn một số đặc trưng cho toàn bộ các giải pháp (quần thể) có thể có cho vấn đề.

***Bước 2*:** Chỉ định cho mỗi giải pháp (cá thể) một ký hiệu. Ký hiệu có thể là một dãy các số 0, 1 thuộc hệ nhị phân, hay dãy các số thập phân, dãy các chữ hay hỗn hợp của số và chữ. Ký hiệu đơn giản nhất và thường dùng nhất là số nhị phân.

***Bước 3*:** Tìm hàm số thích nghi cho vấn đề và tính hệ số thích nghi cho từng giải pháp (lời giải).

***Bước 4*:** Dựa trên hệ số thích nghi của các giải pháp để thực hiện sự tạo sinh (reproduction) và biến hóa các giải pháp. Các phương thức biến hóa bao gồm: lai ghép (crossover), đột biến (mutation).

***Bước 5*:** Tính các hệ số thích nghi cho các giải pháp mới và loại bỏ những giải pháp kém nhất để chỉ còn giữ lại một số nhất định của giải pháp.

***Bước 6*:** Nếu chưa tìm được giải pháp tối ưu hay tương đối khá nhất hay chưa hết kỳ hạn ấn định, trở lại bước 4 để tìm giải pháp mới.

***Bước 7*:** Tìm được giải pháp tối ưu hoặc nếu thời gian cho phép đã chấm dứt thì kết thúc giải thuật và báo cáo kết quả tìm được.

# **Chương 2**

# *BÀI TOÁN TÌM KIẾM VĂN BẢN*

# *VỚI GIẢI THUẬT DI TRUYỀN*

Trong phần này sẽ trình bày các nội dung nghiên cứu chính của khoá luận, từ yêu cầu đặt ra cho bài toán tìm kiếm văn bản ta đi xây dựng hàm mục tiêu tìm kiếm. Trên cơ sở đó phát biểu bài toán dưới dạng tối ưu hàm một biến và dùng phương pháp giải thuật di truyền để giải quyết bài toán.

* 1. **Yêu cầu đặt ra bài toán tìm kiếm văn bản**

Các thuật toán tìm kiếm tất cả vị trí xuất hiện của mẫu trên một văn bản đều dựa theo phương pháp tìm kiếm tuyến tính (tìm tuần tự từ đầu đến cuối văn bản). Theo tư tưởng đó, sẽ tìm được chính xác tất cả các vị trí xuất hiện của mẫu trong văn bản. Trong thực tế, đôi khi chúng ta không cần quan tâm đến mẫu tìm kiếm có chính xác hay không mà ta chỉ quan tâm đến nội dung liên quan đến mẫu (hoặc có chứa một phần trong mẫu). Đơn cử ví dụ như công ty Google dựa trên ý tưởng đó đã phát triển ứng dụng tìm kiếm trên Web rất hiệu quả.

Vậy vấn đề cần đặt ra là ***tìm trong văn bản S vị trí xuất hiện đoạn văn bản gần giống với đoạn văn bản mẫu Sm nhất.*** Yêu cầu tìm kiếm ở đây không đòi hỏi vị trí xuất hiện chính xác của mẫu mà là tìm vị trí xuất hiện gần đúng của xâu mẫu, tìm kiếm có thể đạt kết quả tốt nhất khi vị trí xuất hiện đó chính là mẫu cần tìm. Với mục tiêu này, tại một vị trí i trong văn bản, thay vì việc đi so sánh đoạn văn bản M kí tự (từ vị trí i đến vị trí i+M) đang xét với mẫu thì ta đi tìm số kí tự trùng khớp (cả về giá trị và vị trí) lớn nhất giữa 2 văn bản này. Hiển nhiên trong trường hợp xuất hiện mẫu thì số kí tự trùng khớp lớn nhất sẽ bằng M. Trên cơ sở đó ta hoàn toàn có thể đưa ra các vị trí gần đúng với mẫu nhất trong trường hợp không có đoạn văn bản mẫu trong đoạn văn bản tìm kiếm.

Tìm kiếm với yêu cầu như trên có thể đáp ứng được các nhu cầu của người sử dụng để tìm kiếm văn bản. Với các thuật toán tìm kiếm tuyến tính ta chỉ cần cải tiến một chút là cũng có thể tìm được đúng với yêu cầu đặt ra. Tuy nhiên với những văn bản có số kí tự rất lớn thì tìm kiếm tuyến tính như đã nói ở trên lại không hiệu quả về mặt thời gian (với độ phức tạp là O(MN)). Đã có một số giải pháp để giải quyết vấn đề này là các thuật toán so sánh mẫu theo thứ tự bất kỳ. Theo đó người ta tiến hành so sánh mẫu với cửa sổ theo một thứ tự ngẫu nhiên, nhưng sẽ khó có thể biết trước được khả năng đưa ra lời giải vì ở đây chỉ là việc so sánh với các vị trí ngẫu nhiên mà không có cơ sở toán học rõ ràng để hướng đến một vị trí xuất hiện mẫu trong văn bản.

Cũng trên cơ sở so sánh ngẫu nhiên, ta đi nghiên cứu một hướng tiếp cận giải quyết bài toán theo hướng khác, đó là ***hướng tiếp cận giải thuật di truyền*** để giải quyết các yêu cầu đặt ra với bài toán tìm kiếm văn bản.

* 1. **Xây dựng hàm tìm kiếm**

Để xác định tiêu chí tính toán chobài toán tìm kiếm văn bản bằng giải thuật di truyền ta sẽ xây dựng hàm tìm kiếm như sau:

Hàm tìm kiếm có tiêu chí đánh giá bằng tổng của hai đại lượng:

1. Độ dài của xâu con chung dài nhất giữa đoạn văn bản đó và mẫu (đều có độ dài M ký tự)
2. Độ dài trùng khớp về giá trị và vị trí của đoạn văn bản đó với mẫu.

Xâu con chung dài nhất ở đây là dãy ký tự dài nhất theo thứ tự giống nhau giữa hai xâu (không nhất thiết phải liền nhau), trường hợp tốt nhất xảy ra là xâu con chung dài nhất có độ dài M (dài bằng văn bản mẫu) - tức là hai xâu so sánh là giống nhau – đó chính là vị trí xuất hiện của cả mẫu. Để tìm xâu con chung dài nhất thuật toán hiệu quả là dùng quy hoạch động có độ phức tạp O(M2). Trong thực tế khi tìm kiếm số M thường không lớn nên hoàn toàn chấp nhận được.

Hàm tìm kiếm được xây dựng là: **F(x) = a\*G(x) + b\*H(x)**

Trong đó x: là vị trí trong văn bản (x ∈ [1,…n])

G(x) là tần suất xuất hiện Sm trong đoạn S[x…x+M] của S (kể từ vị trí x cho đến vị trí x+M trong văn bản S. G(x) được tính bằng hàm quy hoạch động tìm độ dài xâu con chung lớn nhất.

H(x) là độ đo thứ tự, phản ánh thứ tự xuất hiện các kí tự trong S[x…x+m] trùng với Sm. Ta có thể viết là G(Sx,Sm) thay cho G(x).

H(x) được tính bằng cách so khớp lần lượt từng kí tự, giá trị trả về chính là số kí tự trùng khớp (cả vì giá trị và vị trí) của hai văn bản Sm và S[x…x+M]. Ta có thể viết H(Sx,Sm) thay cho H(x).

a và b là các tham số đóng vai trò trọng số của G(x) và H(x), để thuận cho việc đánh giá hàm F ta có thể quy định ràng buộc cho a và b là: a = 1 – b.

Như vậy dễ thấy G(x) và H(x) có giá trị trong khoảng [0…M], và do đó hàm F cũng có miền giá trị trong khoảng [0…M], tức là Fmax(x)=M. Tuỳ thuộc vào mục tiêu của bài toán và căn cứ vào giá trị của hàm tìm kiếm F ta có thể giải quyết được mọi yêu cầu đặt ra cho bài toán tìm kiếm văn bản.

* 1. **Phát biểu bài toán tìm kiếm văn bản theo hướng tiếp cận di truyền**

Dựa vào hàm tìm kiếm, ta phát biểu bài toán tìm kiếm văn bản dưới dạng bài toán tối ưu hàm một biến như sau:

Xét bài toán:

***“Cho trước một văn bản S có độ dài N và một văn bản mẫu Sm có độ dài M (M ≤ N). Tìm các giá trị của x ∈ [1…N] sao cho F(x) = a\*G(x) + b\*H(x) ≥ k”.***

Trong đó k là giá trị ngưỡng cho trước (0≤k≤Fmax(x)), k đóng vai trò tham số xác định độ chính xác của hàm mục tiêu.

Bài toán đặt ra là tìm các giá trị x sao cho F(x) đạt giá trị lớn hơn hoặc bằng ngưỡng k. Nếu tìm được các giá trị xmax để F(xmax) = M thì xmax chính là vị trí xuất hiện chuỗi Sm cần tìm trong văn bản S. Trường hợp bài toán chỉ cho kết quả tương đối tốt thì x là các vị trí mà trong đoạn [x, x+M] có xuất hiện một phần trong xâu mẫu (gần giống với xâu mẫu). Trong trường hợp này ta có giữ lại kết quả hay không phụ thuộc vào ngưỡng k.

Để đạt được mục tiêu tìm kiếm ta đưa ra một ngưỡng tìm kiếm k và xem xét bài toán tìm đoạn văn bản trong S gần đúng với mẫu Sm, hoặc có độ dài đoạn trùng khớp lớn hơn một ngưỡng k cho trước. Thực chất trong trường hợp tìm giá trị Max thì chỉ cần hàm G(x) hoặc H(x) là đã đủ để đánh giá hàm F(x) nhận cực đại. Nhưng khi đi tìm giá trị hàm F đạt một ngưỡng k cho trước, nếu đoạn mẫu văn bản là ngắn thì việc dùng hàm H(x) lại có ý nghĩa, trường hợp đoạn mẫu văn bản dài thì H(x) lại đóng vai trò quan trọng vì lẽ nếu chỉ căn cứ vào hàm G(x) để đánh giá thì giả sử trong S có đoạn văn bản M kí tự mà một nửa số kí tự đầu tiên xuất hiện trong một nửa sau của chuỗi S thì sự giống nhau là không đáng kể so với tại vị trí mà hai nửa đầu của đoạn văn bản trong S và đoạn văn bản mẫu trùng khớp với nhau.

Phương pháp tiếp cận di truyền có thể không tìm được hết tất cả các vị trí xuất hiện mẫu trong văn bản, nhưng nó sẽ rất hữu hiệu trong việc giải quyết bài toán tìm kiếm với yêu cầu đặt ra là có xuất hiện (chính xác) hay không hoặc tìm xuất hiện gần đúng nhất. Đặc biệt khi ta phải tìm trong toàn ổ đĩa máy tính các file văn bản có chứa một nội dung nào đó, thì mục tiêu trở thành tìm thấy file có chứa nội dung gần giống với văn bản đó. Khi đó ta chỉ cần sự xuất hiện gần đúng nhất của nội dung tìm kiếm trong file và đưa ra vị trí xuất hiện đó (chứ không nhất thiết phải đưa ra tất cả các vị trí xuất hiện trong file).

* 1. **Tìm độ dài xâu con chung lớn nhất bằng quy hoạch động**
* Định nghĩa xâu con chung: xâu s1 được gọi là con của xâu s2 nếu mọi s1[i] thuộc s1 đều xuất hiện trong s2 theo thứ tự.
* Bài toán tìm độ dài xâu con chung lớn nhất: Cho 2 xâu X, Y. Hãy tìm xâu con của X và cùa Y có độ dài lớn nhất.

*Công thức QHĐ:*

Gọi L(i, j) là độ dài xâu con chung dài nhất của xâu X(i) gồm i kí tự phần đầu của X (X(i) = X[1…i]) và xâu Y(j) gồm j kí tự phần đầu của Y (Y(j) = Y[1…j]).

Ta có công thức sau:

L(0,j)=L(i,0)=0.

L(i,j)=L(i-1,j-1)+1 nếu X[i]=Y[j]

L(i,j)=maxL(i-1,j), L(i,j-1) nếu X[i] ≠ Y[j]

* Cài đặt:

Bằng phương án làm một mảng 2 chiều L[0…m, 0…n] để lưu các giá trị của hàm QHĐ L(i,j). Đoạn chương trình cài đặt công thức QHĐ trên như sau:

for i:=0 to m do L[i,0]:=0;

for j:=0 to nd o L[0,j]:=0;

for i:=1 to m do

for j:=1 to n do

if X[i]=Y[j] then L[i,j]:=L[i-1,j-1]+1

else L[i,j]:=max(L[i-1,j],L[i,j-1]);

Như vậy chi phí không gian của bài toán là O(n2), chi phí thời gian là O(n2). Có một phương pháp cài đặt tốt hơn , chỉ với chi phí không gian O(n) dựa trên nhận xét sau: để tính ô L[i,j] của bảng phương án, ta chỉ cần 3 ô L[i-1,j-1], L[i-1,j], L[i,j-1]. Tức là để tính dòng L[i] thì chỉ cần dòng L[i-1]. Do đó ta chỉ cần 2 mảng 1 chiều để lưu dòng vừa tính (P) và dòng đang tính (L) mà thôi. Cách cài đặt mới như sau:

for j:=0 to n do P[j]:=0

for i:=1 to m do

begin

L[0]:=0;

for j:=1 to n do

if X[i]=Y[j] then L[j]:=P[j-1]+1;

else L[i,j]:=max(P[j],L[j-1]);

P:=L;

end

Kết quả trả về độ dài xâu con chung lớn nhất là P(n)

Cần lưu ý rằng với bài toán tìm kiếm văn bản là ta đi tìm xâu con chung dài nhất của hai chuỗi văn bản có cùng độ dài là M (cùng độ dài với chuỗi văn bản mẫu). Khi đó nếu xâu con dài nhất có độ dài bằng M có nghĩa là 2 xâu giống nhau. Độ dài của xâu con chung càng tịnh tiến đến M có nghĩa là 2 xâu so sánh càng giống nhau. Trên cơ sở đó ta có thể đưa ra một vị trí xuất hiện đoạn văn bản gần giống với văn bản mẫu theo yêu cầu đặt ra cho bài toán tìm kiếm văn bản ở trên.

* 1. **Áp dụng giải thuật di truyền**

Với bài toán tìm kiếm văn bản được phát biểu như sau:

***Tìm x ∈ [1…N] | F(x) = a\*G(x) + b\*H(x) ≥ k***

Có nghĩa là tìm x trong khoảng [1…N] để hàm F(x) đạt giá trị vượt ngưỡng k cho trước, x là các giá trị nguyên dương ứng với các vị trí trong văn bản tìm kiếm có độ dài N kí tự. Hai tham số a và b là các tham số xác định độ ưu tiên đánh giá theo G(x) và H(x), giả sử ta để a + b = 1.

Hàm F có thể đạt giá trị vượt ngưỡng k tại nhiều vị trí, giá trị lớn nhất của hàm F là M. Để thuận tiện cho việc đánh giá ta định lại giá trị F:=F/M. Khi đó hàm F sẽ đạt giá trị lớn nhất bằng 1 và F có miền giá trị thuộc [0,1].

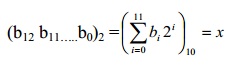
Dùng giải thuật di truyền giải bài toán trên ta có hàm F là hàm mục tiêu (hàm lượng giá), x chính là các nhiễm sắc thể, các thành phần chính của giải thuật như sau:

* + 1. *Biểu diễn nhiễm sắc thể*

Ta sử dụng một vector nhị phân v làm nhiễm sắc thể để biểu diễn các giá trị nguyên của biến x. Chiều dài của vector cũng chính là số bit trong dãy bit nhị phân biểu diễn được số nguyên lớn nhất trong miền giá trị của x tức là chiều dài vector nhị phân l =log2n. Như vậy vector nhị phân có chiều dài l sẽ biểu diễn được số nguyên bằng 21. Ví dụ văn bản có chiều dài tối đa (số kí tự) là n = 4000 thì cần có 12 bit cho vector nhị phân (nhiễm sắc thể):

*2048 = 211 < 4000 ≤ 212 = 4096*

Ánh xạ biến chuỗi nhị phân (b12b11…b0) thành số nguyên x trong khoảng [1…4000] được thực hiện như sau:



Ví dụ NST v1 = (110001100010) biểu diễn số 3170 cũng là vị trí kí tự thứ x=3170 trong văn bản. Nhiễm sắc thể v2 = (000000001100) biểu diễn tại x = 12.

* + 1. *Khởi tạo quần thể*

Khởi tạo quần thể đơn giản như sau: Tạo ra một quần thể các nhiễm sắc thể, trong đó mỗi nhiễm sắc thể là một vector nhị phân 12 bit, tất cả 12 bit của mỗi nhiểm sắc thể đều được khởi tạo ngẫu nhiên.

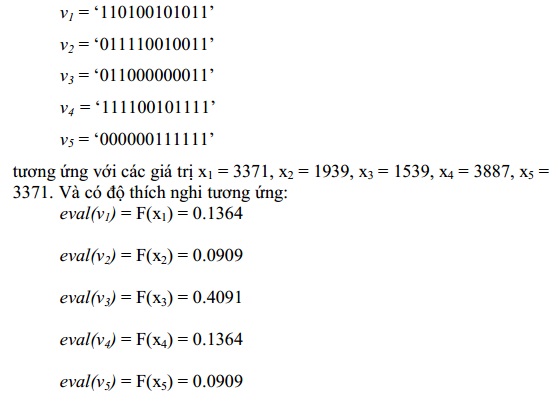
* + 1. *Hàm mục tiêu*

Hàm mục tiêu eval của cá cvector nhị phân v chính là hàm F:

*eval(v)=F(x)*

trong đó, nhiễm sắc thể v biểu diễn giá trị nguyên x như đã nói ở trên, hàm mục tiêu đóng vai trò môi trường, đánh giá từng lời giải theo độ thích nghi của chúng. F(x) được đánh giá qua hai hàm G(x) và H(x) đã trình bày như trên.

Ví dụ:

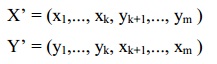


Dễ thấy, nhiễm sắc thể v3 là tốt nhất trong 5 nhiễm sắc thể này, vì hàm mục tiêu của nó trả về giá trị cao nhất.

* + 1. *Các toán tử di truyền*

Trong nghiên cứu này ta sử dụng 3 phép toán di truyền cơ bản là chọn lọc, lai ghép và đột biến, cụ thể:

* **Toán tử chọn lọc**: sử dụng toán tử ***chọn lọc tỉ lệ***, ta thực hiện tiến hành chọc lọc bằng cách quay bánh xe ru lét pop-size lần, mỗi lần chọn một nhiểm sắc thể từ quần thể hiện hành vào quần thể mới.
* **Toán tử lai ghép:** sử dụng toán tử lai ghép một điểm (One-point Crossover). Với cặp cha mẹ X, Y là các vector m chiều như kí hiệu trên, toán tử lai ghép một điểm chọn ngẫu nhiên 1 vị trí k (1 ≤ k ≤ m) rồi sinh ra 2 cá thể con theo công thức:

****

Nếu cá thể con X’ thích nghi tốt hơn cá thể cha mẹ X thì ta thay thế cá thể mẹ X bởi cá thể con X’, tương tự Y’ cũng được thay thế Y nếu Y’ thích nghi tốt hơn.

* **Toán tử đột biến:** sử dụng toán tử đột biến như sau:
  + - * + Chọn ngẫu nhiên một nhiễm sắc thể trong quần thể.
        + Tạo mốt số ngẫu nhiên k trong khoảng từ 1 tới m, 1 ≤ k ≤ m
        + Thay đổi bit thứ k. Nếu nhiễm sắc thể này không xấu hơn nhiễm sắc thể ban đầu thì đưa nhiễm sắc thể này vào quần thể để tham gia vào quá trình tiến hoá ở thế hệ tiếp theo.
    1. *Các tham số*

Đối với bài này ta sử dụng các tham số như sau: số thế hệ = 100, xác suất lai tạo pc = 0.8, xác suất đột biến pm = 0.05 (nhỏ hơn nhiều so với xác suất lai). Xác suất lai pc = 0.8 nghĩa là cá thể v trong quần thể có 80% cơ hội được chọn để thực hiện phép lai, còn xác suất đột biến pm = 0.05 lại là 5% 1 bit bất kì của một cá thể bất kì trong quần thể bị đột biến

* + 1. *Chi phí thời gian*

Thời gian tính toán (độ phức tạp) của giải thuật di truyển tìm kiếm văn bản được trình bày ở trên là O(i\*Size\*SoBit\*M2). Trong đó i là số thế hệ tiến hoá, độ lớn của i phụ thuộc vào từng bài toán cụ thể, thường là i có thể lớn đến hàng nghìn, size là kích thước của quần thể - số cá thể trong quần thể, M là chiều dài văn bản mẫu, M2 là thời gian thực hiện QHĐ, SoBit là chiều dài nhiễm sắc thể (số bit của vector lời giải) được tính bằng log2N (N độ dài văn bản), con số này cũng chỉ lên đến vài chục bit. SoBit và Size thường rất nhỏ (coi như hằng số), do đó độ phức tạp của giải thuật chỉ là O(i\*M2) cho một lần tìm kiếm, chỉ tương đương hoặc nhỏ hơn độ phức tạp O(N\*M) của các thuật toán tìm kiếm tuyến tính trên các văn bản dài – số N là rất lớn. Trong nghiên cứu này ta dùng giải thuật di truyền để giải bài toán tìm kiếm văn bản sẽ đáp ứng được tốt yêu cầu về thời gian.

# **Chương 3**

# *ỨNG DỰNG GIẢI THUẬT DI TRUYỀN*

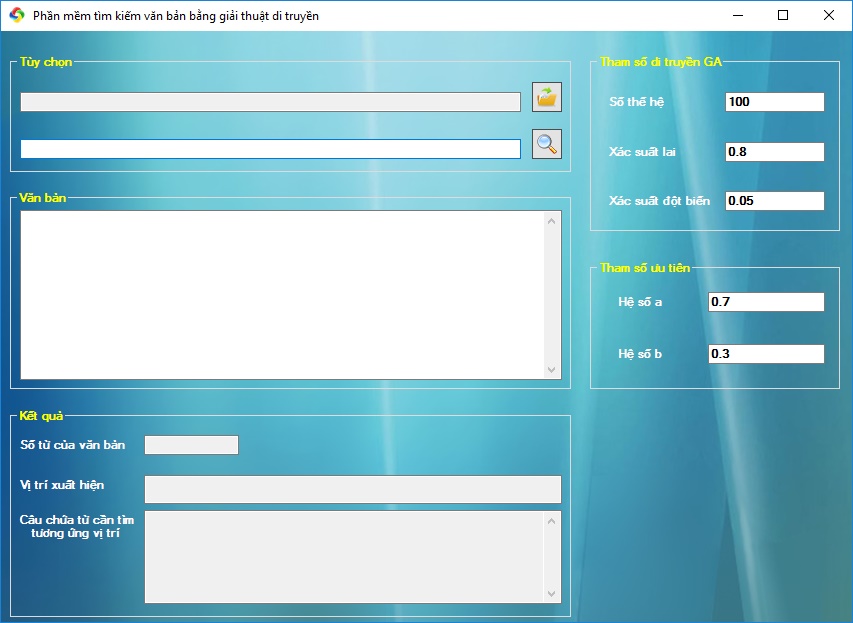
# *CHO BÀI TOÁN TÌM KIẾM VĂN BẢN*

Như đã trình bày khá đầy đủ và rõ ràng ở chương 1 và chương 2 về Giải thuật di truyền nói chung và việc ứng dụng giải thuật di truyền vào bài toán tìm kiếm văn bản nói riêng. Trong chương 3 này, chúng em sẽ vận dụng những kiến thức đã được trình bày ở hai chương trước để thiết kế và vận hành một ứng dụng đơn giản áp dụng giải thuật di truyền vào bài toán tìm kiếm văn bản.

* 1. **Giới thiệu chung**
     1. *Giới thiệu về ứng dụng*

Ứng dụng được xây dựng dựa trên ngôn ngữ lập trình C# của phần mềm Microsoft Visual Studio và do chính các thành viên xây dựng thông qua những nghiên cứu đút kết được.

* + 1. *Giao diện chính*



***Hình 3.1.*** *Giao diện chính của ứng dụng dành cho người dùng*

* + 1. *Các tham số đầu vào và đầu ra*
       1. *Tham số đầu vào*
          - Tham số di truyền GA

+ Số thế hệ: số nguyên dương các thế hệ

+ Xác suất lai ghép: số thực với xác suất 0 ≤ pc ≤ 1

+ Xác suất đột biến tính trên số gen: số thực với 0 ≤ pm ≤ 1

* + - * + Tham số ưu tiên

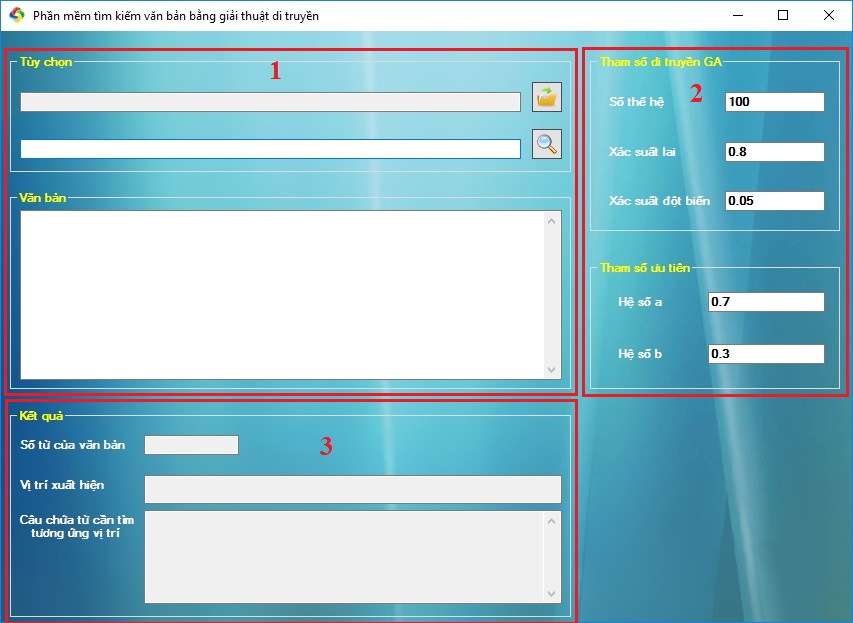
+ Hệ số a: số thực

+ Hệ số b: số thực

*(với a = 1- b)*

* + - * + Đoạn văn bản (dưới định dạng \*.pdf) cần tìm kiếm từ khoá.
        + Từ hoặc cụm từ cần tìm kiếm.
      1. *Tham số đầu ra*
         * Các vị trí xuất hiện từ hoặc cụm từ cần tìm trong văn bản.
         * Tổng các số từ có trong văn bản được chọn.
         * Câu chứa từ hoặc cụm từ cần tìm tại vị trí tương ứng.
    1. *Chức năng của các trường và nút lệnh*

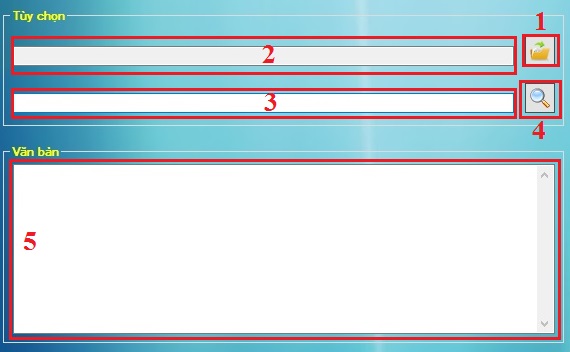
Giao diện chính được chia thành 3 phần khác nhau với 3 chức năng khác nhau như hình bên dưới:



***Hình 3.2.*** *Ba phần chính của giao diện*

Các chức năng chính cụ thể của 3 phần như sau:

* **Phần chức năng thứ 1**



***Hình 3.3.*** *Phần chức năng thứ nhất của ứng dụng*

* + - * + Nút lệnh (**1**): nút này nhấp chọn sẽ mở ra khung chọn đường dẫn cho phép người dùng mở văn bản cần tìm kiếm với định dạng \*.pdf có sẵn trên máy.
        + Trường số (**2**): nơi hiển thị chi tiết đường dẫn mà người dùng đã chọn.
        + Trường số (**3**): cho phép người dùng nhập từ hoặc cụm từ cần tìm kiếm trong đoạn văn bản đã chọn.
        + Nút lệnh (**4**): nhấp chọn ứng dụng sẽ tiến hành tìm kiếm những từ hoặc cụm từ đã nhập ở (**3**).
        + Vùng văn bản số (**5**): nơi hiển thị toàn bộ văn bản mà người dùng đã chọn.
* **Phần chức năng thứ 2**

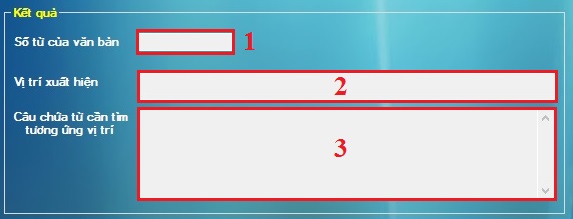


*Hình* ***3.4****. Phần chức năng thứ hai của ứng dụng*

* + - * + Trường số (**1**): vùng hiển thị số các thế hệ.
        + Trường số (**2**): vùng hiển thị tỉ lệ xác suất lai ghép.
        + Truồng số (**3**): vùng hiển thị tỉ lệ xác suất đột biến.
        + Trường số (**4**): vùng hiển thị tham số a (a = 1 – b).
        + Trường số (**5**): vùng hiển thị tham số b.

*(các tham số trong các trường đặc gán trước mặc định)*

* **Phần chức năng thứ 3**



***Hình 3.5****. Phần chức năng thứ ba của ứng dụng*

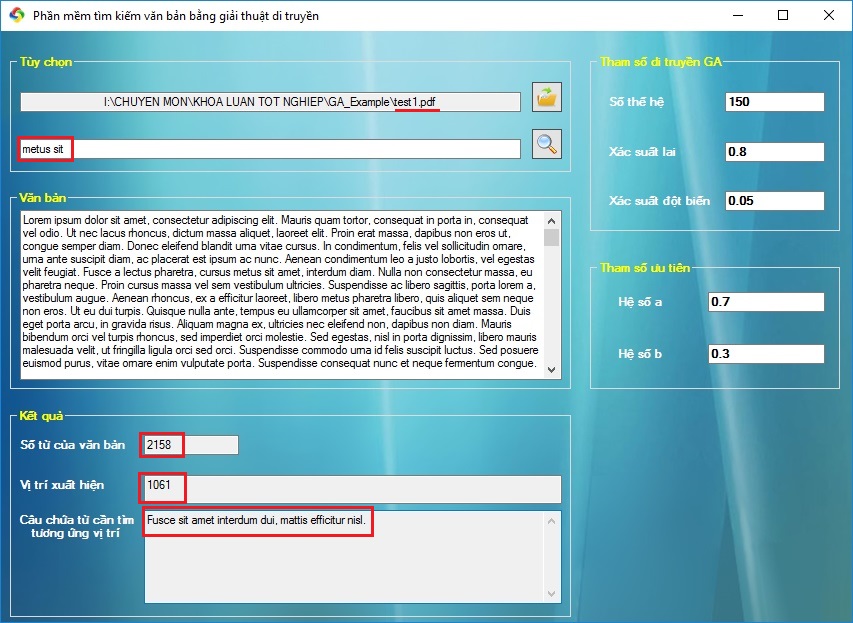
* + - * + Trường số (**1**): hiển thị tổng số lượng các từ có trong văn bản đã chọn.
        + Trường số (**2**): hiển thị các vị trí của từ khoá cần tìm xuất hiện trong văn bản.
        + Trường số (**3**): hiển thị câu có chứa từ khoá cần tìm tương ứng với từng vị trí xuất hiện của từ khoá.
  1. **Một số ví dụ thử nghiệm**
     1. *Bài toán 1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Số thế hệ*** | ***Xác suất lai*** | ***Xác suất***  ***đột biến*** | ***Hệ số a*** | ***Hệ số b*** | ***Văn bản***  ***thử*** | ***Từ khoá*** |
| *150* | *0.8* | *0.05* | *0.7* | *0.3* | *test1.pdf* | *metus sit* |

***Bảng 3.1.*** *Tham số đầu vào của bài toán 1*

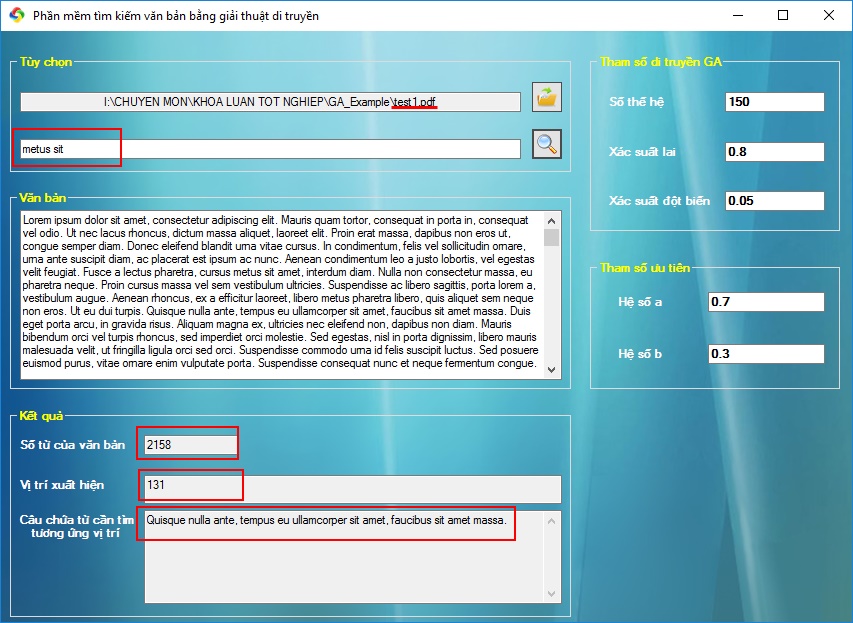
*Hình ảnh kết quả thu được sau khi chạy thử nghiệm bài toán 1:*

* *Lần thứ 1:*

**

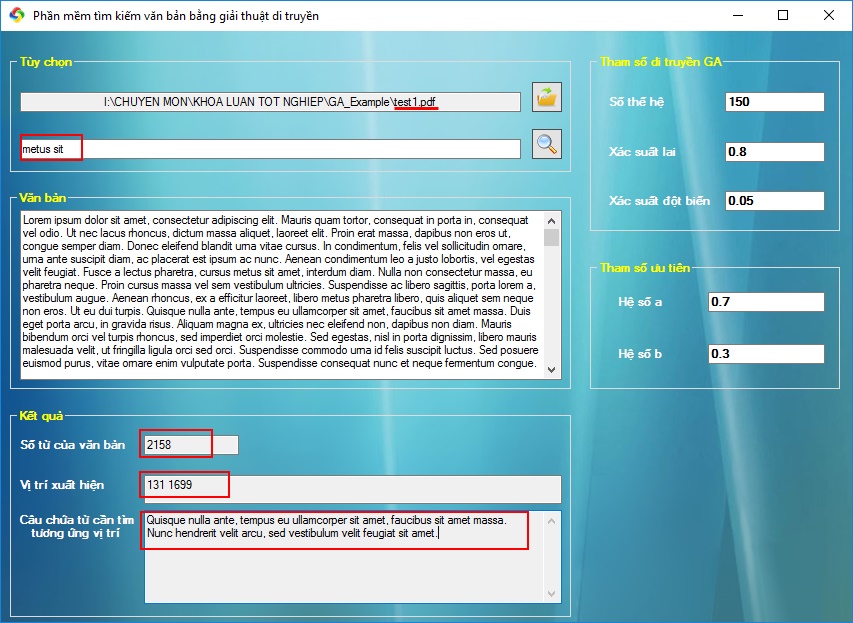
***Hình 3.6.*** *Kết quả chạy thử nghiệm bài toán 1 lần thứ 1.*

* *Lần thứ 2:*

**

***Hình 3.7.*** *Kết quả chạy thử nghiệm bài toán 1 lần thứ 2.*

* *Lần thứ 3:*

**

***Hình 3.8.*** *Kết quả chạy thử nghiệm bài toán 1 lần thứ 3.*

*Kết quả thu được sau khi chạy thử nghiệm:*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Số từ trong văn bản*** | ***Lần thử***  ***thứ*** | ***Vị trí***  ***xuất hiện*** | ***Câu chứa từ khoá cần tìm*** |
| *2158* | *1* | *1061* | *Fusce* ***sit*** *amet interdum dui, mattis efficitur nisl.* |
| *2* | *131* | *Quisque nulla ante, tempus eu ullamcorper* ***sit*** *amet, faucibus* ***sit*** *amet massa.* |
| *3* | *131* | *Quisque nulla ante, tempus eu ullamcorper* ***sit*** *amet, faucibus* ***sit*** *amet massa.* |
| *1699* | *Nunc hendrerit velit arcu, sed vestibulum velit feugiat* ***sit*** *amet.* |

***Bảng 3.2.*** *Kết quả chạy thử nghiệm bài toán 1*

* + 1. *Bài toán 2*

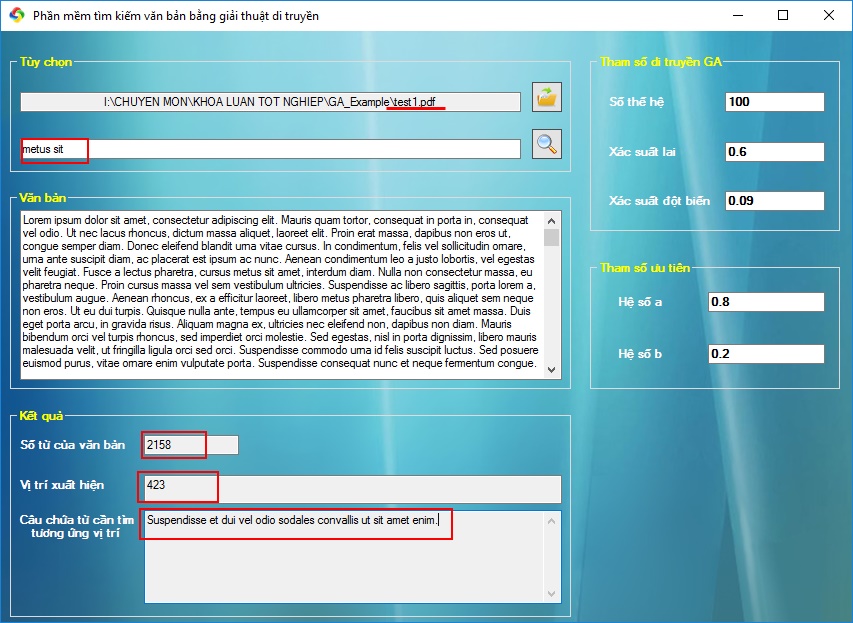
Vẫn bài văn bản mẫu thử trên, nhưng thay đổi các tham số di truyền (số thế hệ, xác suất lai, xác suất đột biến) và tham số ưu tiên (hệ số a, hệ số b).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Số thế hệ** | **Xác suất lai** | **Xác suất**  **đột biến** | **Hệ số a** | **Hệ số b** | **Văn bản**  **thử** | **Từ khoá** |
| 100 | 0.6 | 0.09 | 0.8 | 0.2 | test1.pdf | *metus sit* |

***Bảng 3.3.*** *Tham số đầu vào của bài toán 2*

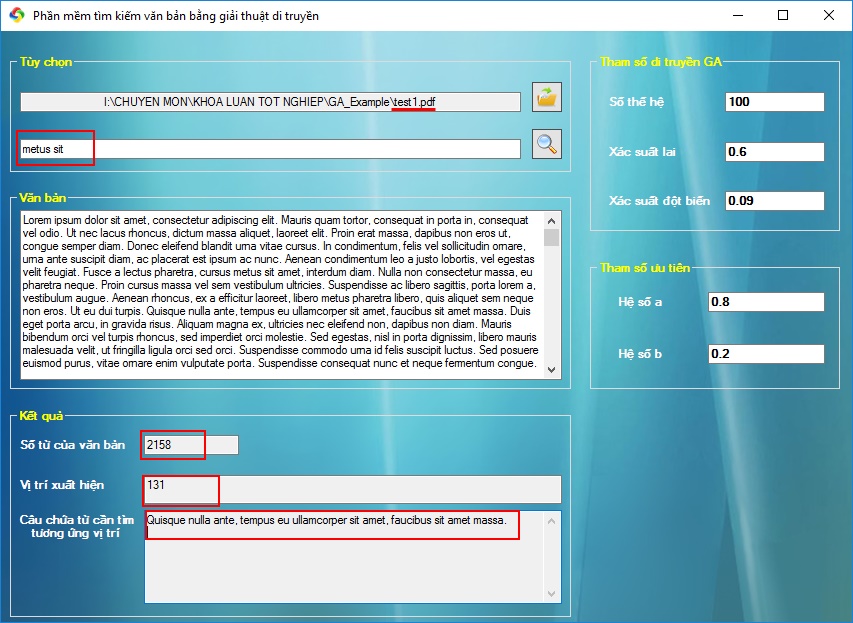
Hình ảnh kết quả thu được sau khi chạy thử nghiệm bài toán 2:

* Lần chạy thứ 1:



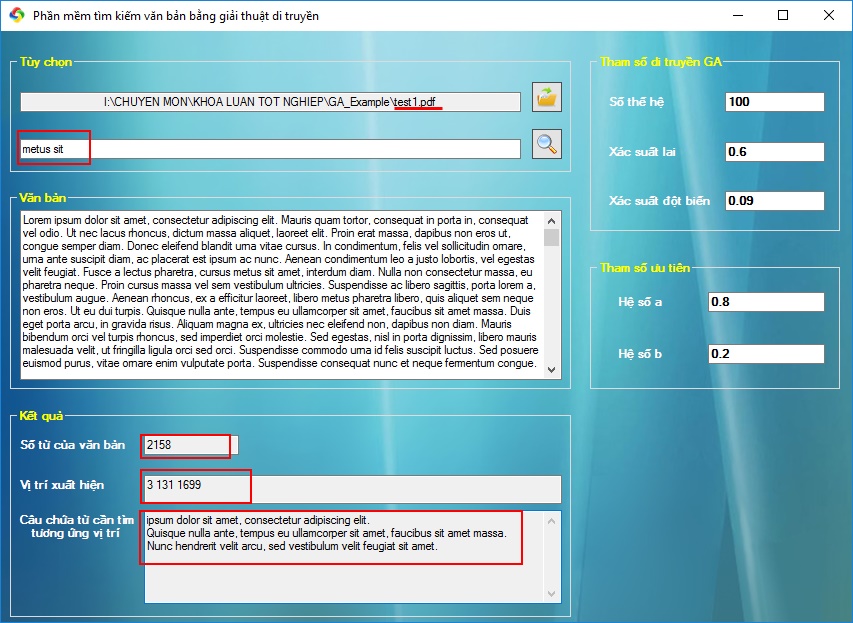
***Hình 3.9.*** *Kết quả chạy thử nghiệm bài toán 2 lần thứ 1.*

* Lần chạy thứ 2:



***Hình 3.10.*** *Kết quả chạy thử nghiệm bài toán 2 lần thứ 2.*

* Lần chạy thứ 3:



***Hình 3.11.*** *Kết quả chạy thử nghiệm bài toán 2 lần thứ 3.*

Kết quả thu được sau khi chạy thử nghiệm:

|  |  |  |  |
| --- | --- | --- | --- |
| **Số từ trong văn bản** | **Lần thử**  **thứ** | **Vị trí**  **xuất hiện** | **Câu chứa từ khoá cần tìm** |
| 2158 | 1 | 423 | Suspendisse et dui vel odio sodales convallis ut ***sit*** amet enim. |
| 2 | 131 | Quisque nulla ante, tempus eu ullamcorper ***sit*** amet, faucibus ***sit*** amet massa. |
| 3 | 3 | Lorem ipsum dolor ***sit*** amet, consectetur adipiscing elit. |
| 131 | Quisque nulla ante, tempus eu ullamcorper ***sit*** amet, faucibus ***sit*** amet massa. |
| 1699 | Nunc hendrerit velit arcu, sed vestibulum velit feugiat ***sit*** amet. |

***Bảng 3.4.*** *Kết quả chạy thử nghiệm bài toán 2*

* + 1. *Bài toán 3:*

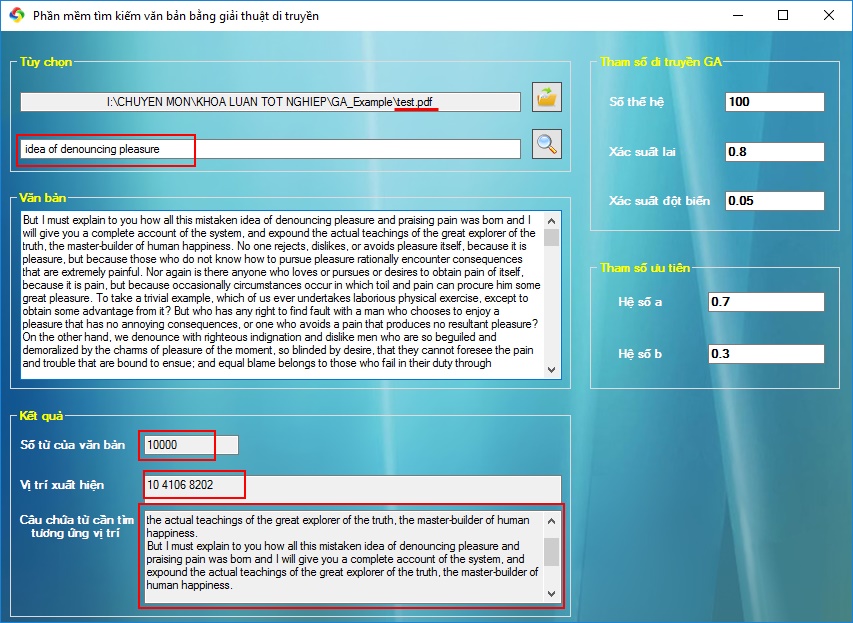
Với việc chạy thử nghiệm bài toán thứ 3 này, văn bản thử sẽ chứa số lượng cá thể lớn (10000) tức văn bản chứa 10000 từ.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Số thế hệ** | **Xác suất lai** | **Xác suất**  **đột biến** | **Hệ số a** | **Hệ số b** | **Văn bản**  **thử** | **Từ khoá** |
| 100 | 0.8 | 0.05 | 0.7 | 0.3 | test.pdf | *idea of denouncing pleasure* |

***Bảng 3.5.*** *Tham số đầu vào của bài toán 3*

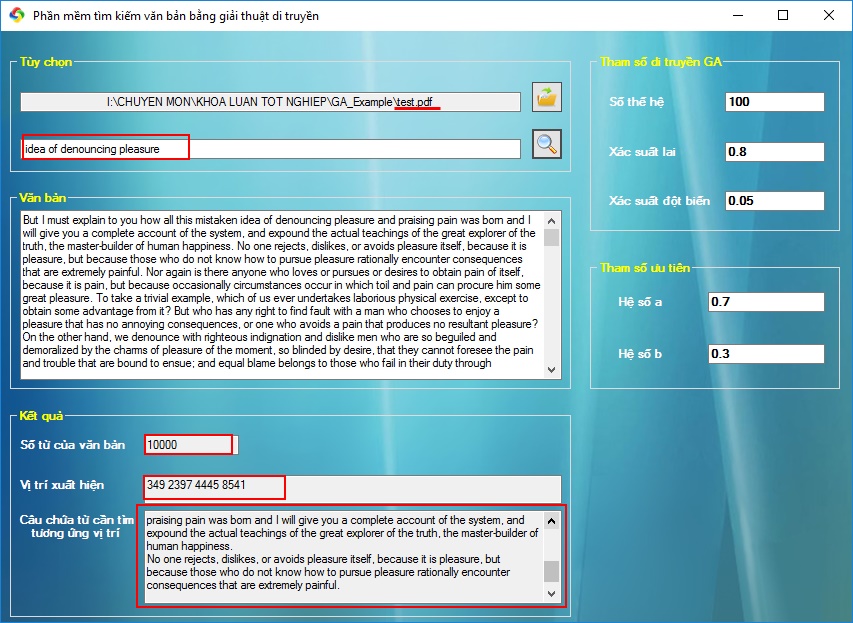
Hình ảnh kết quả thu được sau khi chạy thử nghiệm bài toán 3 với từ khoá " *idea of denouncing pleasure"* :

* Lần chạy thứ 1:



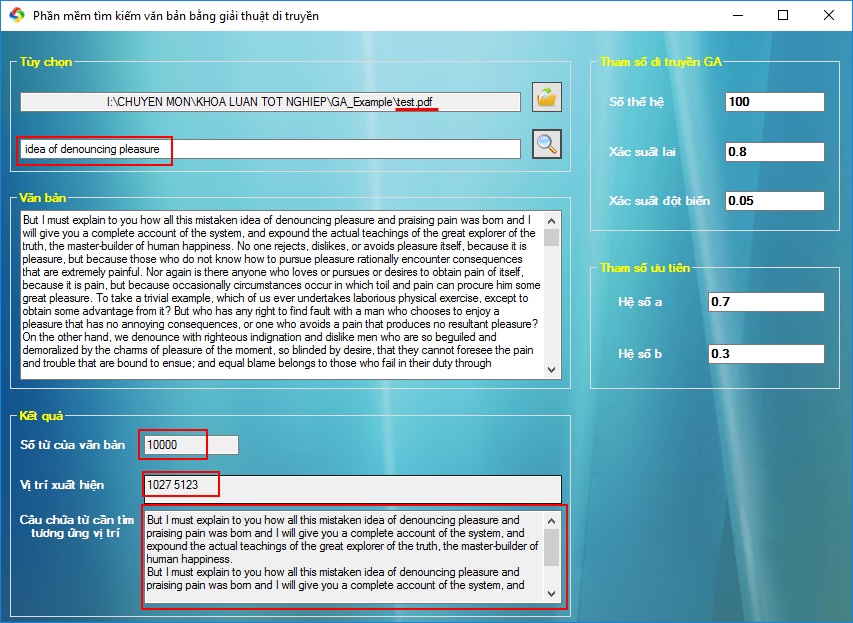
***Hình 3.12.*** *Kết quả chạy thử nghiệm bài toán 3 lần thứ 1.*

* Lần chạy thứ 2:



***Hình 3.13.*** *Kết quả chạy thử nghiệm bài toán 3 lần thứ 2.*

* Lần chạy thứ 3:



***Hình 3.14.*** *Kết quả chạy thử nghiệm bài toán 3 lần thứ 3.*

Kết quả thu được sau khi chạy thử nghiệm với từ khoá " *idea of denouncing pleasure"*:

|  |  |  |  |
| --- | --- | --- | --- |
| **Số từ trong văn bản** | **Lần thử**  **thứ** | **Vị trí**  **xuất hiện** | **Câu chứa từ khoá cần tìm** |
| 10000 | 1 | 10 | But I must explain to you how all this mistaken ***idea of denouncing pleasure*** and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. |
| 4106 | But I must explain to you how all this mistaken ***idea of denouncing pleasure*** and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. |
| 8202 | No one rejects, dislikes, or avoids ***pleasure*** itself, because it is ***pleasure***, but because those who do not know how to pursue ***pleasure*** rationally encounter consequences that are extremely painful. |
| 2 | 349 | But I must explain to you how all this mistaken ***idea of denouncing pleasure*** and …, the master-builder of human happiness. |
| 2397 | But I must explain to you how all this mistaken ***idea of denouncing pleasure*** and …, the master-builder of human happiness. |
| 4445 | But I must explain to you how all this mistaken ***idea of denouncing pleasure*** and …, the master-builder of human happiness. |
| 8541 | No one rejects, dislikes, or avoids ***pleasure*** itself, because it is ***pleasure***, … ***pleasure*** rationally encounter consequences that are extremely painful. |
| 3 | 1027 | But I must explain to you how all this mistaken ***idea of denouncing pleasure*** and …, the master-builder of human happiness. |
| 5123 | But I must explain to you how all this mistaken ***idea of denouncing pleasure*** and …, the master-builder of human happiness. |

***Bảng 3.6****. Kết quả chạy thử nghiệm bài toán 3.*

**Nhận xét:**

Như vậy, thông qua các kết quả đạt được từ việc chạy thử nghiệm các bài toán ta có thể thấy rằng giải thuật di truyền mang lại lợi ích rất lớn trong việc giải quyết những bài toán có không gian tìm kiếm lớn, thuận tiện hơn cho người dùng khi họ không nhớ chính xác từ khoá và có thể đưa ra vị trí cụ thể của từ khoá đó.

# **Chương 4**

# *TỔNG KẾT*

* 1. **Nhận xét về giải thuật di truyền trong bài toán tìm kiếm văn bản**
     1. *Ưu điểm*
        + - Có thể tìm kiếm trong không gian tìm kiếm rộng.
          - Có thể tìm kiếm nhiều lời giải tối ưu cùng một lúc trong cùng một lần chạy.
     2. *Khuyết điểm*
        + - Thuật giải di truyền nói chung là phương pháp tìm kiếm xác suất nên cần phải thực nghiệm nhiều lần, với sự thay đổi thông số đầu vào khác nhau để có thể nhận được những kết quả tối ưu mong muốn.
          - Điều căn bản và quan trọng nhất trong thuật giải di truyền nói chung và ứng dụng trong bài toán tối ưu đa mục tiêu nói riêng là phương pháp gán độ thích nghi để có thể đánh giá những cá thể có tính tối ưu hơn và từ đó thực hiện quá trình tái sinh để tiếp tục phát sinh ra những cá thể mới trong quá trình tiến hóa. Việc tìm được hàm ánh xạ từ hàm mục tiêu thành hàm tính độ thích nghi là không hề đơn giản.
  2. **Đánh giá kết quả nghiên cứu**
* Khoá luận đã bước đầu đề xuất phương pháp ứng dụng giải thuật di truyền vào việc giải quyết bài toán tìm kiếm văn bản.
* Tìm hiểu và cài đặt được các thuật toán tìm kiếm văn bản, qua đó làm cơ sở so sánh nghiên cứu của đề tài.
* Xây dựng được các hàm tính toán cho bài toán và phát biểu bài toán tìm kiếm văn bản để có thể áp dụng giải thuật di truyền.
* Các chương trình và kết quả chạy thử nghiệm đã chứng minh hứng tiếp cận giải thuật di truyền giải quyết bài toán tìm kiếm văn bản là đúng đắn và có hiệu quả. Đặc biệt chương trình ứng dụng đã chỉ ra được các vị trí xuất hiện đoạn văn bản giống văn bản mẫu hoặc gần giống với văn bản mẫu (trong trường hợp văn bản không chứa văn bản mẫu) trong thời gian cho phép.
  1. **Kiến nghị hướng phát triển**
* Hiện tại ứng dụng đang trong quá trình phát triển dựa vào các nghiên cứu đã được thực hiện ở trên. Do thời gian hạn chế, còn nhiều lỗ hỏng trong kiến thức nên ứng dụng chưa được phát triển đáng kể.
* Chúng em sẽ cố gắng hoàn thiện ứng dụng hơn nữa, tiến hành tìm và sữa chữa những lỗi mà ứng dụng gặp phải để trong thời gian sớm nhất có thể đưa vào áp dụng trong thực tế.
* Sau khi phát triển thành công ứng dụng, hướng nghiên cứu tiếp theo là tìm hiểu ứng dụng giải thuật di truyền cho nhiều dạng bài toán tìm kiếm, chẳng hạn bài toán tìm kiếm trên các file dữ liệu có cấu trúc đặc biệt
* Cho phép tìm kiếm tất cả các file văn bản trong máy tính có chứa một từ, cụm từ hay một đoạn văn bản; hoặc chứa nội dung gần giống với nội dung văn bản cần tìm. Các kiểu file văn bản có thể tìm kiếm là các tệp tin định dạng .txt, .doc, .xls, .ppt và một số tệp tin văn bản khác trên môi trường Windows.
* Ngoài ra chương trình còn có các tham số lựa chọn khác thuận tiện theo mục đích người sử dụng, chẳng hạn như: tham số kích thước quần thể, số thế hệ tiến hoá, lựa chọn độ chính xác (ngưỡng) với từ tìm kiếm, số vị trí xuất hiện.. sẽ đáp ứng được phần lớn nhu cầu sử dụng cách thiết thực và hiệu quả.
* Hướng nghiên cứu mở rộng ra tìm trên môi trường Internet.
  1. **Kết quả đạt được**
* Hiểu được giải thuật di truyền và các bước thực hiện.
* Hiểu được bài toán áp dụng giải thuật di truyền vào tìm kiếm văn bản và những phương pháp tiếp cận để giải bài toán.
* Xây dựng đượng một ứng dụng đơn giản trên nền tảng ngôn ngữ lập trình C# để thực hiện giải bài toán vào những ví dụ cụ thể.
* Đã tiến hành thử nghiệm một số bài toán cụ thể và đạt được kết quả như mong đợi.
  1. **Hạn chế**
* Giao diện của ứng dụng còn khá đơn giản.
* Các thông số được gán cố định cho từng bài toán.
* Việc nghiên cứu chỉ dừng ở việc tìm kiếm trên tập tin định dạng \*.pdf, không thể mở rộng tìm kiếm các loại file khác.
* Ứng dụng chỉ tìm cục bộ không thể tìm kiếm rộng hơn trên Internet.
* Chỉ giải quyết các bài toán trong toán học, còn thiếu tính ứng dụng cụ thể.

# **TÀI LIỆU THAM KHẢO**

**Tài liệu Tiếng Việt**

[1]. Đoàn Đình Thọ, Nguyễn Văn Khiêm, Nguyễn Thị Đang Tâm, *Các toán tử và kỉ thuật nâng cao trong thuật giải di truyền*, Trường Đại Học Khoa Học Tự Nhiên, 2011.

[2]. Mai Ngọc Tùng, *Nghiên cứu ứng dụng thuật giải di truyền để tìm kiếm thông tin trên văn bản*, Đại Học Quốc Gia Thành Phố Hồ Chí Minh, 2012.

[3]. Phan Thị Cúc, *Ứng dụng giải thuật di truyền trong bài toán tìm kiếm văn bản*, Trường Đại Học Công Nghệ Thông Tin Và Truyền Thông, 2012.

[4]. Nguyễn Văn Quyết, *Bài toán tìm kiếm văn bản sử dụng giải thuật di truyền*, Trường Đại Học Thái Nguyên, 2009.

[5]. Nguyễn Quang Lập, *Thuật toán bầy đàn POS, giải thuật di truyền và ứng dụng giải các bài toán tối ưu đa mục tiêu*, Trường Đại Học Công Nghệ Thông Tin Và Truyền Thông, 2013.

[6]. David E.GoldBerg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, 1989.

**Tài liệu trực tuyến**

[7]. https://vi.wikipedia.org/wiki/Giải\_thuật\_di\_truyền

[8]. <https://www.tutorialspoint.com/genetic_algorithms/index.htm>

# **PHỤ LỤC**

1. **Hàm tạo gen**

public class Genome

{

public double[] m\_genes;

public string[] m\_word\_genes;

private string s\_genes;

public string S\_genes

{

get { return s\_genes; }

set { s\_genes = value; }

}

private int m\_length;

private double m\_fitness;

static Random m\_random = new Random();

public Genome()

{

}

public Genome(int position, double stringLength)

{

int bitLength = (int)Math.Log(stringLength, 2);

string binary = Convert.ToString(position, 2);

int i = 0;

string prefix = "";

while (i <= bitLength - binary.Length) {

prefix += "0";

i++;

}

s\_genes = prefix + binary;

}

public Genome(int length, bool createGenes)

{

m\_length = length;

m\_genes = new double[ length ];

if (createGenes)

CreateGenes();

}

public Genome(ref double[] genes)

{

m\_length = genes.GetLength(0);

m\_genes = new double[ m\_length ];

for (int i = 0 ; i < m\_length ; i++)

m\_genes[i] = genes[i];

}

public Genome(string gene)

{

m\_length = gene.Length;

m\_word\_genes = new string[m\_length];

CreateCharacterGenes(gene);

}

private void CreateGenes()

{

// DateTime d = DateTime.UtcNow;

for (int i = 0 ; i < m\_length ; i++)

m\_genes[i] = m\_random.NextDouble();

}

public void CreateCharacterGenes(string gene)

{

for (int i = 0; i < m\_length; i++)

{

m\_word\_genes[i] = gene[i].ToString();

}

}

1. **Hàm lai ghép**

public void Crossover(ref Genome genome2, out Genome child1, out Genome child2)

{

int pos = (int)(m\_random.NextDouble() \* (double)m\_length);

child1 = new Genome(m\_length, false);

child2 = new Genome(m\_length, false);

for(int i = 0 ; i < m\_length ; i++)

{

if (i < pos)

{

child1.m\_genes[i] = m\_genes[i];

child2.m\_genes[i] = genome2.m\_genes[i];

}

else

{

child1.m\_genes[i] = genome2.m\_genes[i];

child2.m\_genes[i] = m\_genes[i];

}

}

}

public void Crossover\_s(ref Genome genome2, out Genome child1, out Genome child2)

{

Random rd = new Random();

child1 = new Genome();

child2 = new Genome();

int pos = (int)rd.Next(0, genome2.s\_genes.Length - 1);

pos = (int)genome2.s\_genes.Length / 2;

int lenght = genome2.S\_genes.Length;

string genome\_t1 = s\_genes;

string genome\_t2 = genome2.s\_genes;

child1.S\_genes = genome\_t1.Substring(0, pos + 1) + genome\_t2.Substring(pos + 1, lenght - pos - 1);

genome\_t1 = s\_genes;

genome\_t2 = genome2.S\_genes;

child2.S\_genes = genome\_t2.Substring(0, pos + 1) + genome\_t1.Substring(pos + 1, lenght - pos - 1);

}

1. **Hàm đột biến**

public void Mutate()

{

for (int pos = 0 ; pos < m\_length; pos++)

{

if (m\_random.NextDouble() < m\_mutationRate)

m\_genes[pos] = (m\_genes[pos] + m\_random.NextDouble())/2.0;

}

}

public void Mutate\_s()

{

Random rd = new Random();

int pos = (int)rd.Next(0, s\_genes.Length - 1);

StringBuilder sb = new StringBuilder(S\_genes);

if (m\_random.NextDouble() < m\_mutationRate)

{

if (S\_genes[pos] == '1')

{

sb[pos] = '0';

}

else

{

sb[pos] = '1';

}

}

S\_genes = sb.ToString();

}

public string Genes()

{

return s\_genes;

}

public void Output()

{

for (int i = 0 ; i < m\_length ; i++)

{

System.Console.WriteLine("{0:F4}", m\_genes[i]);

}

System.Console.Write("\n");

}

public void GetValues(ref double[] values)

{

for (int i = 0 ; i < m\_length ; i++)

values[i] = m\_genes[i];

}

private static double m\_mutationRate;

private string p;

public double Fitness

{

get

{

return m\_fitness;

}

set

{

m\_fitness = value;

}

}

public static double MutationRate

{

get

{

return m\_mutationRate;

}

set

{

m\_mutationRate = value;

}

}

public int Length

{

get

{

return m\_length;

}

}

}

1. **Hàm so sánh các gen**

GenomComparer.cs

public sealed class GenomeComparer : IComparer

{

public GenomeComparer()

{

}

public int Compare( object x, object y)

{

if ( !(x is Genome) || !(y is Genome))

throw new ArgumentException("Not of type Genome");

if (((Genome) x).Fitness > ((Genome) y).Fitness)

return 1;

else if (((Genome) x).Fitness == ((Genome) y).Fitness)

return 0;

else

return -1;

}

}

1. **Lớp thuật toán di truyền (gồm các hàm tính các thế hệ lai tiếp theo và đột biến)**

GA.cs

public delegate double GAFunction(string gene, string[] S, string[] Sm, double hsA, double hsB);

/// <summary>

/// Genetic Algorithm class

/// </summary>

public class GA

{

/// <summary>

/// Default constructor sets mutation rate to 5%, crossover to 80%, population to 100,

/// </summary>

///

private double m\_mutationRate;

private double m\_crossoverRate;

private int m\_populationSize;

private int m\_generationSize;

private int m\_genomeSize;

private double m\_totalFitness;

private string m\_strFitness;

private bool m\_elitism;

double hsA;

double hsB;

private ArrayList m\_thisGeneration;

public ArrayList ThisGeneration

{

get { return m\_thisGeneration; }

set { m\_thisGeneration = value; }

}

private ArrayList m\_nextGeneration;

private ArrayList m\_fitnessTable;

static Random m\_random = new Random();

static private GAFunction getFitness;

private double xacSuatLai;

private System.Windows.Forms.TextBox txtXSDotBien;

private int p;

private int soTheHe;

public GAFunction FitnessFunction

{

get

{

return getFitness;

}

set

{

getFitness = value;

}

}

// Properties

public int PopulationSize

{

get

{

return m\_populationSize;

}

set

{

m\_populationSize = value;

}

}

public int Generations

{

get

{

return m\_generationSize;

}

set

{

m\_generationSize = value;

}

}

public int GenomeSize

{

get

{

return m\_genomeSize;

}

set

{

m\_genomeSize = value;

}

}

public double CrossoverRate

{

get

{

return m\_crossoverRate;

}

set

{

m\_crossoverRate = value;

}

}

public double MutationRate

{

get

{

return m\_mutationRate;

}

set

{

m\_mutationRate = value;

}

}

public string FitnessFile

{

get

{

return m\_strFitness;

}

set

{

m\_strFitness = value;

}

}

/// <summary>

/// Keep previous generation's fittest individual in place of worst in current

/// </summary>

public bool Elitism

{

get

{

return m\_elitism;

}

set

{

m\_elitism = value;

}

}

public GA()

{

InitialValues();

m\_mutationRate = 0.05;

m\_crossoverRate = 0.80;

m\_populationSize = 100;

m\_generationSize = 2000;

m\_strFitness = "";

}

public GA(double crossoverRate, double mutationRate, int populationSize, int generationSize, int genomeSize)

{

InitialValues();

m\_mutationRate = mutationRate;

m\_crossoverRate = crossoverRate;

m\_populationSize = populationSize;

m\_generationSize = generationSize;

m\_genomeSize = genomeSize;

m\_strFitness = "";

}

public GA(double crossoverRate, double mutationRate, int populationSize, int generationSize, double a, double b)

{

InitialValues();

m\_mutationRate = mutationRate;

m\_crossoverRate = crossoverRate;

m\_populationSize = populationSize;

m\_generationSize = generationSize;

this.hsA = a;

this.hsB = b;

m\_strFitness = "";

}

public GA(int genomeSize)

{

InitialValues();

m\_genomeSize = genomeSize;

}

public void InitialValues()

{

m\_elitism = false;

}

/// <summary>

/// Method which starts the GA executing.

/// </summary>

public void Go(String[] populationInput, string[] Sm)

{

if (getFitness == null)

throw new ArgumentNullException("Need to supply fitness function");

//if (m\_genomeSize == 0)

// throw new IndexOutOfRangeException("Genome size not set");

// Create the fitness table.

m\_fitnessTable = new ArrayList();

m\_thisGeneration = new ArrayList(populationInput.Length);

m\_nextGeneration = new ArrayList(populationInput.Length);

Genome.MutationRate = m\_mutationRate;

CreateGenomes(populationInput);

RankPopulation(populationInput, Sm);

StreamWriter outputFitness = null;

bool write = false;

if (m\_strFitness != "")

{

write = true;

outputFitness = new StreamWriter(m\_strFitness);

}

for (int i = 0; i < m\_generationSize; i++)

{

CreateNextGeneration();

RankPopulation(populationInput, Sm);

}

if (outputFitness != null)

outputFitness.Close();

}

/// <summary>

/// After ranking all the genomes by fitness, use a 'roulette wheel' selection

/// method. This allocates a large probability of selection to those with the

/// highest fitness.

/// </summary>

/// <returns>Random individual biased towards highest fitness</returns>

private int RouletteSelection()

{

double randomFitness = m\_random.NextDouble() \* m\_totalFitness;

int idx = -1;

int mid;

int first = 0;

int last = m\_populationSize -1;

mid = (last - first)/2;

// ArrayList's BinarySearch is for exact values only

// so do this by hand.

while (idx == -1 && first <= last)

{

if (randomFitness < (double)m\_fitnessTable[mid])

{

last = mid;

}

else if (randomFitness > (double)m\_fitnessTable[mid])

{

first = mid;

}

mid = (first + last)/2;

// lies between i and i+1

if ((last - first) == 1)

idx = last;

}

return idx;

}

/// <summary>

/// Rank population and sort in order of fitness.

/// </summary>

private void RankPopulation(string[] S, string[] Sm)

{

m\_totalFitness = 0;

for (int i = 0; i < S.Length; i++)

{

Genome g = ((Genome) m\_thisGeneration[i]);

g.Fitness = FitnessFunction(g.Genes(), S, Sm, hsA, hsB);

m\_totalFitness += g.Fitness;

}

m\_thisGeneration.Sort(new GenomeComparer());

// now sorted in order of fitness.

double fitness = 0.0;

m\_fitnessTable.Clear();

for (int i = 0; i < S.Length; i++)

{

fitness += ((Genome)m\_thisGeneration[i]).Fitness;

m\_fitnessTable.Add((double)fitness);

}

}

/// <summary>

/// Create the \*initial\* genomes by repeated calling the supplied fitness function

/// </summary>

private void CreateGenomes(String[] populationInput)

{

for (int i = 0; i < populationInput.Length; i++)

{

Genome g = new Genome(i, populationInput.Length);

m\_thisGeneration.Add(g);

}

}

private void CreateNextGeneration()

{

m\_nextGeneration.Clear();

Genome g = null;

if (m\_elitism)

g = (Genome)m\_thisGeneration[m\_populationSize - 1];

for (int i = 0; i < m\_populationSize; i += 2)

{

int pidx1 = RouletteSelection();

int pidx2 = RouletteSelection();

Genome parent1, parent2, child1, child2;

parent1 = ((Genome)m\_thisGeneration[pidx1]);

parent2 = ((Genome)m\_thisGeneration[pidx2]);

if (m\_random.NextDouble() < m\_crossoverRate)

{

parent1.Crossover\_s(ref parent2, out child1, out child2);

}

else

{

child1 = parent1;

child2 = parent2;

}

child1.Mutate\_s();

child2.Mutate\_s();

m\_nextGeneration.Add(child1);

m\_nextGeneration.Add(child2);

}

if (m\_elitism && g != null)

m\_nextGeneration[0] = g;

m\_thisGeneration.Clear();

for (int i = 0 ; i < m\_populationSize; i++)

m\_thisGeneration.Add(m\_nextGeneration[i]);

}

public void GetBest(out int values, out double fitness)

{

Genome g = ((Genome)m\_thisGeneration[m\_populationSize-1]);

//values = new double[g.Length];

//g.GetValues(ref values);

values = int.Parse(Convert.ToInt32(g.S\_genes, 2).ToString());

fitness = (double)g.Fitness;

}

public void GetWorst(out double[] values, out double fitness)

{

GetNthGenome(0, out values, out fitness);

}

public void GetNthGenome(int n, out double[] values, out double fitness)

{

if (n < 0 || n > m\_populationSize-1)

throw new ArgumentOutOfRangeException("n too large, or too small");

Genome g = ((Genome)m\_thisGeneration[n]);

values = new double[g.Length];

g.GetValues(ref values);

fitness = (double)g.Fitness;

}

}

1. **Lớp giao diện (gồm các hàm tính độ thích nghi và xử lý trên giao diện)**

Form.cs

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

string[] words;

string[] words\_search;

private ArrayList words\_result;

private void Form1\_Load(object sender, EventArgs e)

{

System.Windows.Forms.ToolTip ToolTip1 = new System.Windows.Forms.ToolTip();

ToolTip1.SetToolTip(this.btnMoVB, "Mở văn bản");

System.Windows.Forms.ToolTip ToolTip2 = new System.Windows.Forms.ToolTip();

ToolTip2.SetToolTip(this.btnTim, "Tìm");

txtSoTheHe.Text = "10";

txtXSLai.Text = "0.8";

txtXSDotBien.Text = "0.05";

txtHeSoA.Text = "0.7";

txtHeSoB.Text = "0.3";

}

private void btnMoVB\_Click(object sender, EventArgs e)

{

OpenFileDialog dialog = new OpenFileDialog();

dialog.Filter = "Text files | \*.pdf"; // file types, that will be allowed to upload

dialog.Multiselect = false; // allow/deny user to upload more than one file at a time

if (dialog.ShowDialog() == DialogResult.OK) // if user clicked OK

{

String path = dialog.FileName; // get name of file

//string content;

PdfReader reader = new PdfReader(path);

string text = string.Empty;

for (int page = 1; page <= reader.NumberOfPages; page++)

{

text += PdfTextExtractor.GetTextFromPage(reader, page);

}

reader.Close();

txtVanBan.Text = text;

txtDuongDan.Text = path;

char[] delimiterChars = { ' ', '\t' };

string[] splitText = text.Split(delimiterChars);

List<string> result = new List<string>();

foreach (string item in splitText)

{

if (item != "")

result.Add(item);

}

words = result.ToArray();

txtSoKyTu.Text = words.Count().ToString();

}

}

private void btn\_Search\_Click(object sender, EventArgs e)

{

executeSearch();

}

private void executeSearch()

{

if (checkInput() == "")

{

char[] delimiterChars = { ' ', ',', '.', ':', '\t' };

words\_search = txtTim.Text.Split(delimiterChars);

int soTheHe = 0;

int.TryParse(txtSoTheHe.Text, out soTheHe);

double xacSuatLai = 0.8;

double.TryParse(txtXSLai.Text, out xacSuatLai);

double xacSuatDotBien = 0.05;

double.TryParse(txtXSDotBien.Text, out xacSuatDotBien);

double hsA = 0.7;

double.TryParse(txtHeSoA.Text, out hsA);

double hsB = 0.3;

double.TryParse(txtHeSoB.Text, out hsB);

GA ga = new GA(xacSuatLai, xacSuatDotBien, words.Length, soTheHe, hsA, hsB);

ga.FitnessFunction = new GAFunction(theActualFunction);

//ga.FitnessFile = @"H:\fitness.csv";

ga.Elitism = true;

ga.Go(words, words\_search);

this.words\_result = ga.ThisGeneration;

List<StoreResult> list = getResultPosition();

txtViTriXuatHien.Clear();

txtResult.Clear();

foreach (StoreResult item in list)

{

txtViTriXuatHien.AppendText(item.position + " ");

int tmpPos = item.position;

bool flag = false;

int i = tmpPos;

if (tmpPos == 0)

{

while (flag == false)

{

if (words[i].Contains(".") || words[i].Contains("?") || words[i].Contains("!"))

{

txtResult.AppendText(words[i]);

flag = true;

}

else

{

txtResult.AppendText(words[i] + " ");

i++;

}

}

}

else

{

i = tmpPos;

int startSen = i;

while (flag == false)

{

if (words[i].Contains(".") || i == 0 || words[i].Contains("?") || words[i].Contains("!"))

{

startSen = i == 0? 0 : i + 1;

flag = true;

}

else

{

i--;

}

}

flag = false;

while (flag == false)

{

if (words[startSen].Contains(".") || words[i].Contains("?") || words[i].Contains("!"))

{

txtResult.AppendText(words[startSen]);

flag = true;

}

else

{

txtResult.AppendText(words[startSen] + " ");

startSen++;

}

}

}

txtResult.AppendText("\n");

//txtResult.Text = textResult;

}

}

else

MessageBox.Show(checkInput());

}

private List<StoreResult> getResultPosition()

{

List<StoreResult> list = new List<StoreResult>();

int searchTextLength = words\_search.Length;

Genome g\_last = ((Genome)words\_result[words\_result.Count - 1]);

int position\_last = int.Parse(Convert.ToInt32(g\_last.S\_genes, 2).ToString());

StoreResult result\_last = new StoreResult();

result\_last.position = position\_last;

result\_last.fitness = g\_last.Fitness;

list.Add(result\_last);

for (int i = words\_result.Count - 2; i > 0; i--)

{

Genome g = ((Genome)words\_result[i]);

int position = int.Parse(Convert.ToInt32(g.S\_genes, 2).ToString());

if ((list[list.Count - 1].position + searchTextLength) < position && g.Fitness > 0)

{

StoreResult sr = new StoreResult();

sr.position = position;

sr.fitness = g.Fitness;

list.Add(sr);

}

}

return list;

}

private string checkInput()

{

string result = "";

if (words == null)

result += "Vui lòng chọn tệp tin văn bản\n";

if (txtTim.Text == "")

result += "Vui lòng nhập văn bản cần tìm\n";

return result;

}

// optimal solution for this is (0.5,0.5)

public static double theActualFunction(string gene, string[] S, string[] Sm, double hsA, double hsB)

{

double a = hsA;

double b = hsB;

//F(x) = a.G(x) + b.H(x)

int position = int.Parse(Convert.ToInt32(gene, 2).ToString());

double test = a \* G(position, S, Sm) + b \* H(position, S, Sm);

return a \* G(position, S, Sm) + b \* H(position, S, Sm);

}

public static int G(int x, string[] X, string[] Y)

{

int[,] L = new int[Y.Count(), Y.Count()];

for (int i = 0; i < Y.Count(); i++)

{

L[i, 0] = 0;

}

for (int j = 0; j < Y.Count(); j++)

{

L[0, j] = 0;

}

//if (x == 0)

// x = 1;

for (int i = 1; i < Y.Count() && (i + x - 1) < X.Count(); i++)

{

for (int j = 1; j < Y.Count(); j++)

{

string tmp = Regex.Replace(X[i + x - 1], "[^0-9a-zA-Z]+", "");

if (tmp == Y[j])

{

L[i, j] = L[i - 1, j - 1] + 1;

}

else

{

L[i, j] = Math.Max(L[i - 1, j], L[i, j - 1]);

}

}

}

return L[Y.Count() - 1, Y.Count() - 1];

}

public static int H(int positionX, string[] S, string[] Sm)

{

int count = 0;

for (int i = 0; i < Sm.Length; i++)

{

for (int j = positionX; j < (positionX + Sm.Length) && j < S.Length; j++)

{

string tmp = Regex.Replace(S[j], "[^0-9a-zA-Z]+", "");

if (Sm[i] == tmp)

count++;

}

}

return count;

}

private void txtDuongDan\_TextChanged(object sender, EventArgs e)

{

}

private void txtTim\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyValue == 13)

{

executeSearch();

}

}

private void groupBox1\_Enter(object sender, EventArgs e)

{

}

private void label10\_Click(object sender, EventArgs e)

{

}

class StoreResult {

public int position { get; set; }

public double fitness { get; set; }

}

}