Các phương pháp tìm kiếm có sử dụng thông tin

Nội dung

- Thuật toán Gen
- Các thành phần cơ bản của thuật toán gen
- Các khuyến cáo khi sử dụng thuật toán gen
- Ưu và nhược điểm của thuật toán gen

Thuật Toán Gene (GAs)

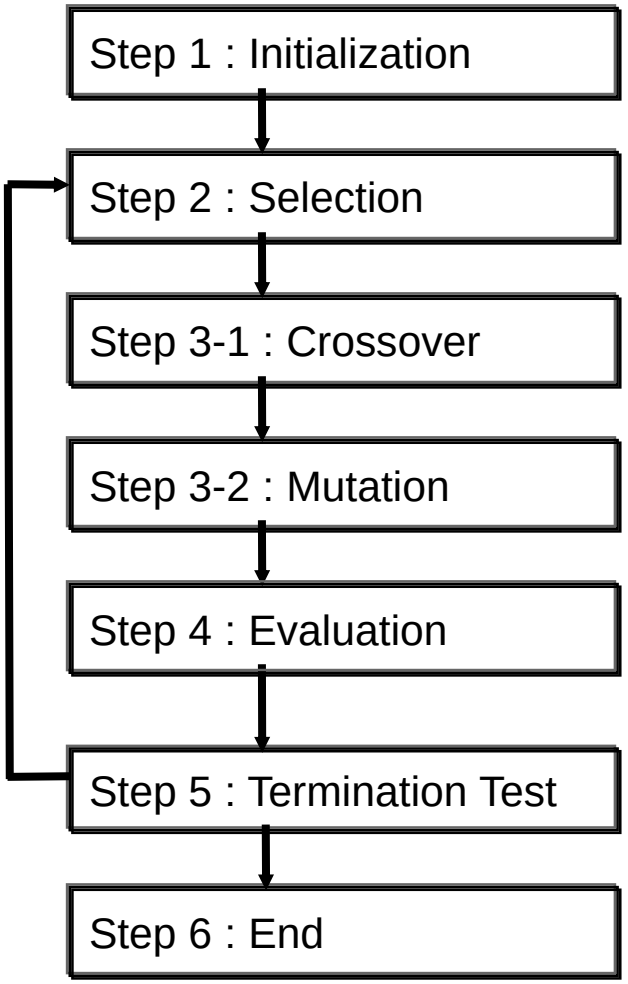
- **GAs** (John Holland, 1975) mô phỏng tiến hóa tự nhiên (Darwinian Evolution) ở mức gen sử dụng tư tưởng của **chọn lọc tự nhiên** (survival of the fittest)
- **Một cá thể (nhiễm sắc thể)** (chromosome) mô tả một lời giải ứng viên của bài toán.
- Một tập các cá thể "alive", gọi là **quần thể** (population) được tiến hóa từ thế hệ này tới thế hệ khác phụ thuộc vào sự thích nghi của các cá thể.
- **Kỳ vọng** (Hope): Thế hệ mới sinh ra sẽ chứa lời giải tốt của bài toán.

Mô tả thuật toán Gene

- Ban đầu, sinh ra thế hệ khởi tạo với quần thể $\mathbf{P(0)}$, chỉ số $\mathbf{i}$ chỉ ra thế hệ thứ $\mathbf{i}$.
- Lặp cho đến khi **quần thể hội tụ** hoặc **tiêu chuẩn kết thúc** đạt được.
  - Đánh giá độ thích nghi của mỗi cá thể trong $\mathbf{P(i)}$.
  - Lựa chọn các cha từ $\mathbf{P(i)}$ dựa trên độ thích nghi của chúng trong $\mathbf{P(i)}$.
  - Áp dụng các toán tử Gen (crossover, mutation) từ các cha đã chọn để sinh ra các con (offspring**)**
  - Đạt được thế hệ tiếp theo $\mathbf{P(i+1)}$ từ các con và các cá thể ở thế hệ $\mathbf{P(i)}$

Cài đặt GA

```
Procedure GA
    begin
        t := 0 ;
        initialize P(t) ;
        evaluate P(t) ;
        while (not termination-condition) do
        begin
            t := t + 1 ;
            select P(t) from P(t-1) ;
            alter P(t) ;
            evaluate P(t) ;
        end;
    end;
```



Các thành phần cơ bản của GAs

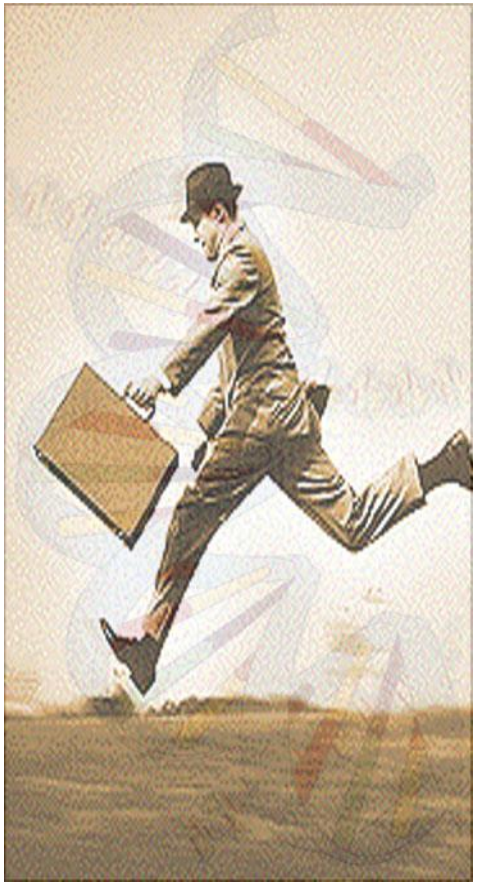
1. Giới thiệu một số bài toán
2. Mã hóa (encoding)
3. Khởi tạo quần thể (innitial population generation)
4. Hàm thích nghi (fitness function)
5. Phương pháp lựa chọn (Selection for recombination)
6. Lai ghép (Crossover)
7. Đột biến (Mutation)
8. Chiến lược thay thế (Replacement Strategy)
9. Tiêu chuẩn kết thúc (Termination Criteria)

Một số bài toán áp dụng

Bài toán Knapsack 01:

Mô tả bài toán:

- Bạn chuẩn bị đi picnic .
- Và bạn có một số các vật mà bạn có thể cầm theo
- Mỗi vật có một trọng lượng và một giá trị.
- Có một cái túi giới hạn trọng lượng bạn có thế cầm theo.
- Mỗi vật chỉ được chọn tối đa 1 lần.
- Bạn muốn cầm các vật mang theo với max giá trị.



Ví dụ về bài toán Knapsack 01

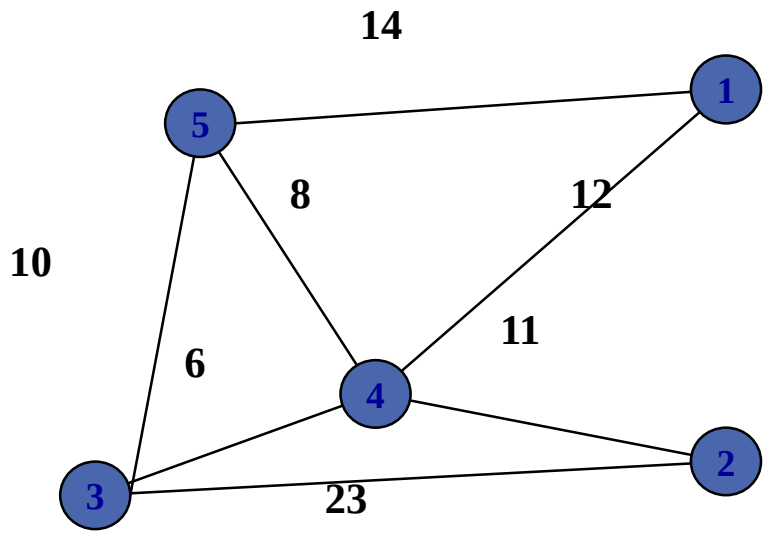
Ví dụ:

- Đồ vật: 1 2 3 4 5 6 7
- Giá trị: 5 8 3 2 7 9 4
- T.lượng: 7 8 4 10 4 6 4
- Khối lượng tối đa có thể mang là 22 đơn vị.
- Xếp đồ vật để có giá trị lớn nhất???

Bài toán TSP (người bán hàng)

Bài toán:

- Một người bán hàng cần ghé qua tất cả các thành phố, mỗi thành phố một lần và trở lại thành phố ban đầu. Có chi phí di chuyển giữa tất cả các thành phố. Tìm hành trình có tổng chi phí nhỏ nhất.



Mã hóa (Encoding)

- **Mã hóa nhị phân (Binary encoding)** là kiểu thông dụng nhất : nghiên cứu đầu tiên về thuật toán Gen sử dụng kiểu mã hóa này và bởi vì nó đơn giản
- Trong mã hóa nhị phân, mọi nhiễm sắc thể là chuỗi **bits** - **0** hoặc **1**.

  Cá thể (Chromosome) A: **10110010110010101100101**

  Cá thể (Chromosome) B: **11111110000011000011111**

- Mã hóa nhị phân đưa ra nhiều khả năng của nhiễm sắc thể với một số lượng nhỏ các gen đẳng vị.
- Các mã hóa này thường không tự nhiên cho nhiều bài toán và đôi khi có sai sau khi thực hiện các phép toán **crossover**, **mutation**.

Bài toán Knapsack 01

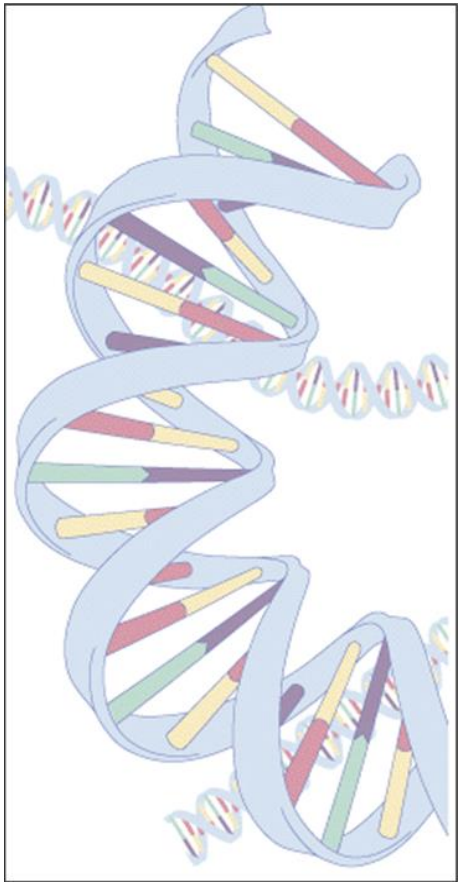
- Encoding: 0 = not exist, 1 = exist in the Knapsack

  Chromosome: 1010110

Item.	1	2	3	4	5	6	7
Chro	1	0	1	0	1	1	0
Exist?	y	n	y	n	y	y	n

  => Items taken: 1, 3 , 5, 6.

- Generate random population of *n* chromosomes:
  a) 0101010
  b) 1100100
  c) 0100011



Mã hóa hoán vị (Permutation Encoding)

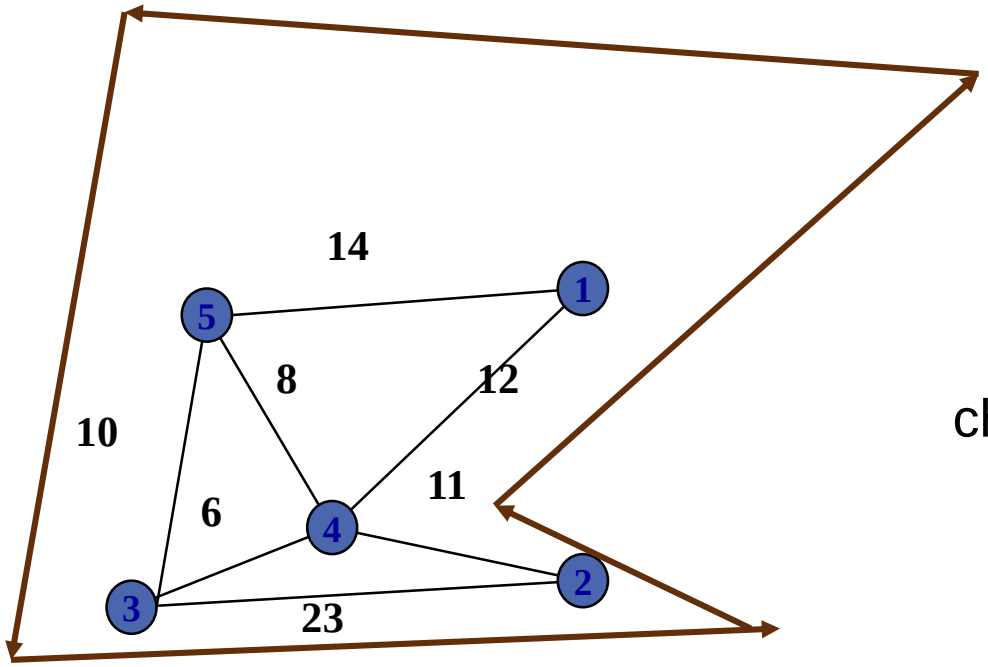
- **Permutation encoding** có thể sử dụng để giải quyết các bài toán có thứ tự như: Người bán hàng.
- Trong permutation encoding, tất cả các NST là chuỗi các số biểu diễn vị trí trong một dãy.

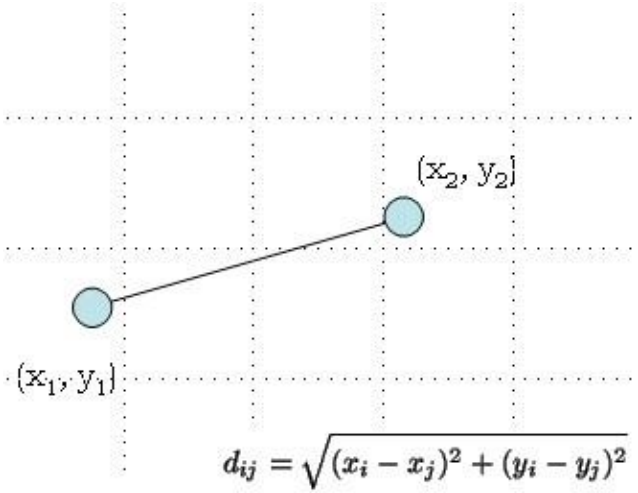
  NST A 1 5 3 2 6 4 7 9 8

  NST B 8 5 6 7 2 3 1 4 9

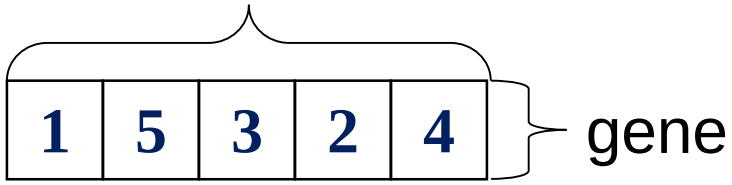
- Mã hóa hoán vị được sử dụng trong các bài toán có thứ tự.
- **Lưu ý**: Trong một vài trường hợp, việc hiệu chỉnh lai ghép và đột biến phải thực hiện để tạo ra NST phù hợp.

Biểu diễn đường đi





chromosome (individual)



Mã hóa giá trị (Value Encoding)

- **Value Encoding** được sử dụng trong các bài toán mà việc mã hóa nhị phân là khó thực hiện (Ví dụ như các bài toán mà giá trị là các số thực).
- Trong **VE** mỗi nhiễm sắc thể là một chuỗi các giá trị có thể nhận dạng bất kỳ tùy thuộc vào từng bài tóan cụ thể. Ví dụ giá trị có thể là số thực, ký tự, hoặc một đối tượng nào đó.

  Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545

  Chromosome B ABDJEIFJDHDIERJFDLDFLFEGT

  Chromosome C (back), (back), (right), (forward), (left)

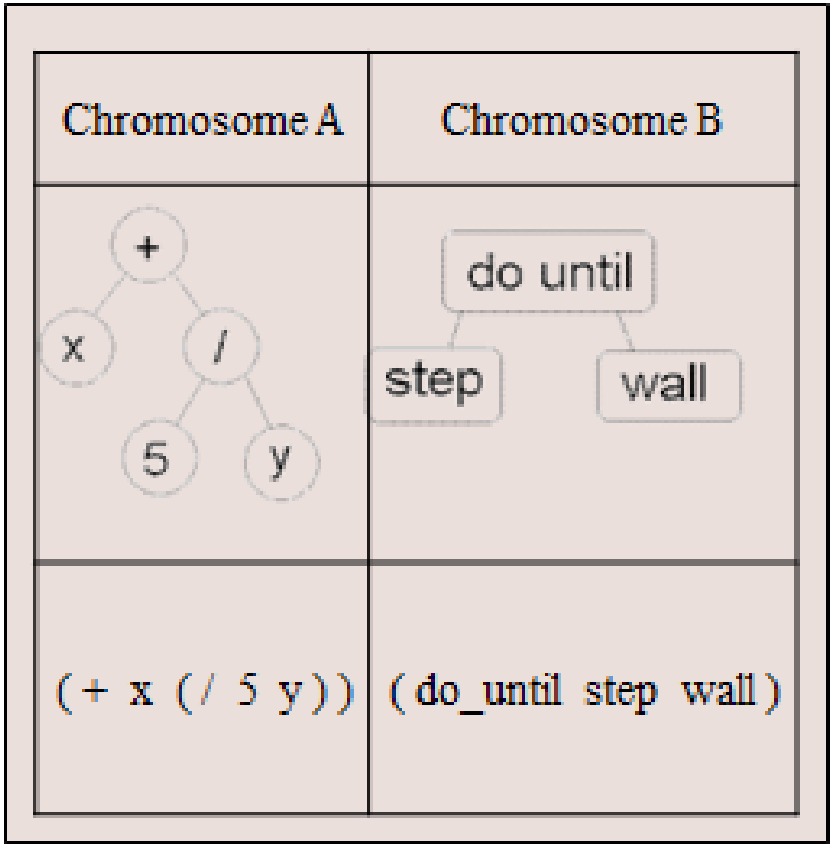
- VE là sự lựa chọn tốt đối với một số bài toán cụ thể.
- Với kiểu mã hóa này thường sẽ phải xây dựng một số phép lai ghép và đột biến cho các bài toán cụ thể.

Ví dụ về mã hóa giá trị

- **Bài toán:** Cho mạng neuron A có cấu trúc đã biết. Tìm trọng số giữa các neurons trong mạng để nhận được kết quả ra mong muốn.
- **Mã hóa:** Giá trị thực trong các chromosome biểu diễn các trọng số trong mạng.

Mã hóa dạng cây (Tree Encoding)

- **TE** thường được sử dụng trong các bài toán hoặc các biểu thức suy luận ví dụ như **genetic programming**.
- Trong TE, mỗi chromosome là một cây gồm các đối tượng như là các hàm hoặc các câu lệnh của ngôn ngữ lập trình.
- TE thường được dùng đối với các bài toán suy luận hoặc các cấu trúc có thể biểu diễn bằng cây.
- Các phép lai ghép và đột biến đối với kiểu mã hóa này cũng tương đối dễ thực hiện.



Khởi tạo quần thể - Bài toán Knapsack 01

- **Khởi tạo:**

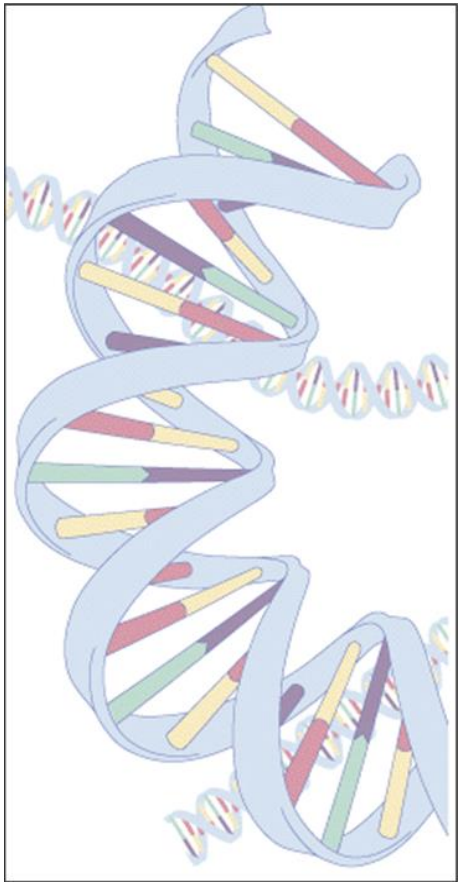
  - Encoding: 0 = not exist, 1 = exist in the Knapsack

    Chromosome: 1010110

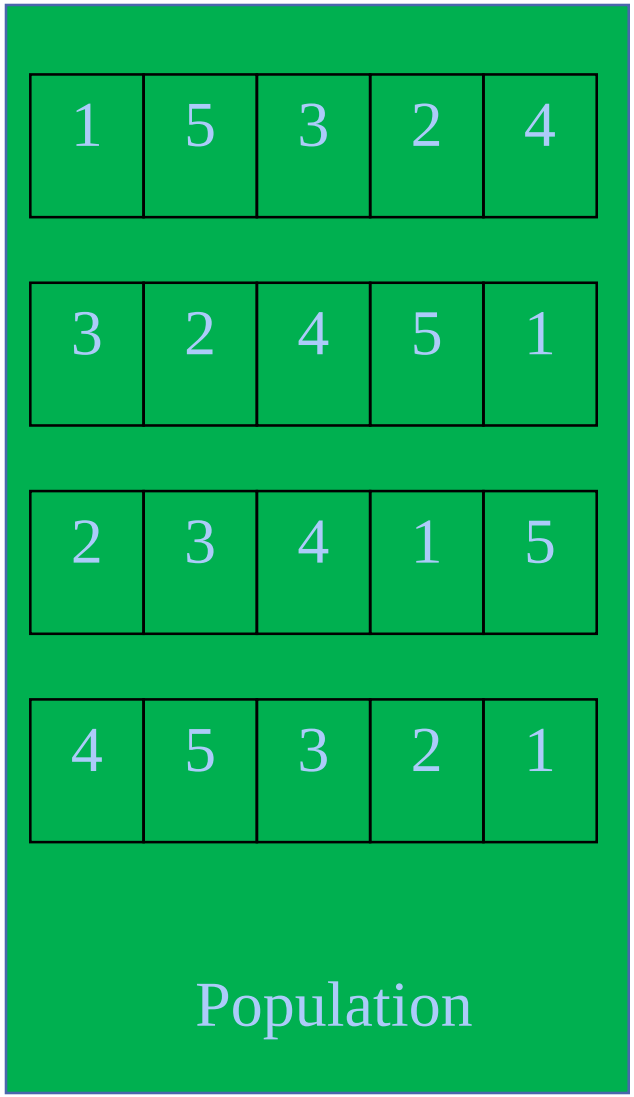
Item.	1	2	3	4	5	6	7
Chro	1	0	1	0	1	1	0
Exist?	y	n	y	n	y	y	n

    => Items taken: 1, 3 , 5, 6.

  - Generate random population of *n* chromosomes:
    a) 0101010
    b) 1100100
    c) 0100011



Khởi tạo quần thể - Bài toán TSP



Population size α

individual length

10.4.4.4. Hàm thích nghi - Bài toán Knapsack 01

- **Fitness**

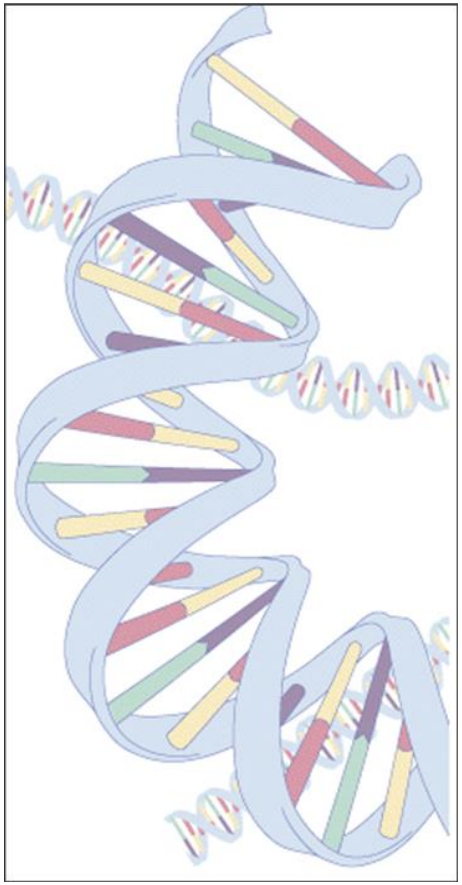
a) 0101010: Benefit= 19, Weight= 24

Item	1	2	3	4	5	6	7
Chro	0	1	0	1	0	1	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

b) 1100100: Benefit= 20, Weight= 19. ✓

c) 0100011: Benefit= 21, Weight= 18. ✓

=> We select Chromosomes b & c.



10.4.4.4. Hàm thích nghi - Bài toán TSP

1	5	3	2	4	58
3	2	5	1	4	56
2	3	4	1	5	55
4	5	3	2	1	57

$$f(indiv)=\sum_{1\le i<n} d_{i\,(i+1)}+d_{n1}$$

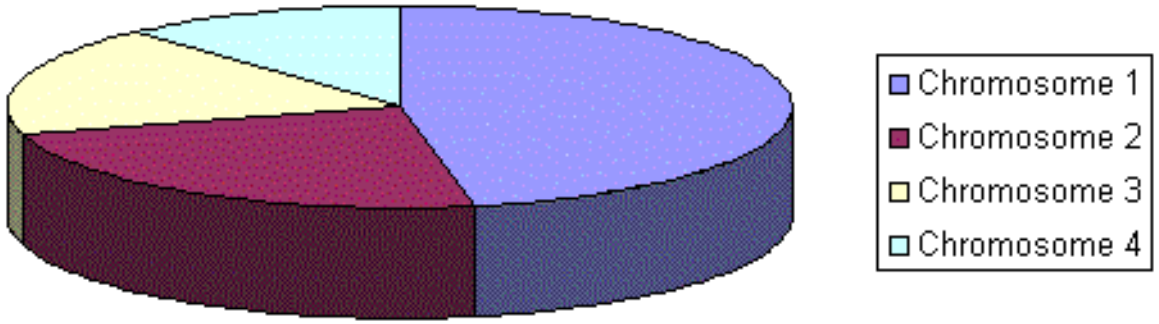
$$d_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}.$$

Phương pháp lựa chọn

- Các nhiễm sắc thể được chọn từ quần thể là các cha cho lai ghép (crossover). Vấn đề là lựa chọn các nhiễm sắc thể như thế nào?
- Theo thuyết tiến hóa Darwin's những cá thể tốt nhất sẽ được chọn để tạo ra các con.
- Có nhiều phương pháp để chọn nhiễm sắc thể tốt nhất.
- Ví dụ một số phương pháp chọn:
  - roulette wheel selection
  - Boltzman selection
  - tournament selection
  - rank selection
  - steady state selection
  - and many other sometimes weird methods

Roulette Wheel Selection

- Nhiễm sắc thể được chọn thông qua độ thích nghi của chúng.
- Nhiễm sắc thể tốt hơn (mang hàm ý xác suất) được chọn.
- Minh họa về phương pháp chọn:
  - Đặt các NST lên **roulette wheel**.
  - Kích thước của đoạn trong roulete wheel tương ứng với giá trị của hàm thích nghi.
  - Một hòn bi được lăn trong roulette wheel và NST nơi nó dừng lại được lựa chọn.
  - NST với giá trị thích nghi lớn sẽ được chọn nhiều hơn.



Roulette Wheel Selection

- Tiến trình trên có thể được mô tả bởi thuật toán sau:

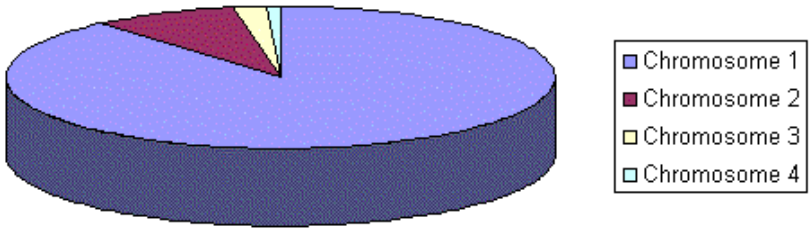
1. **[Tính tổng]** Tính tổng **S** của giá trị thích nghi của tất cả cá thể trong quần thể.
2. **[Lựa chọn]** Chọn ngẫu nhiên một giá trị **r** trong đoạn **(0,S)**.
3. **[Vòng lặp]** Với giá trị ngẫu nhiên **r** đã chọn, tương ứng với các thể nào đó khi hình thành **S** thì cá thể đó được chọn cho thế hệ kế tiếp.

Rank Selection

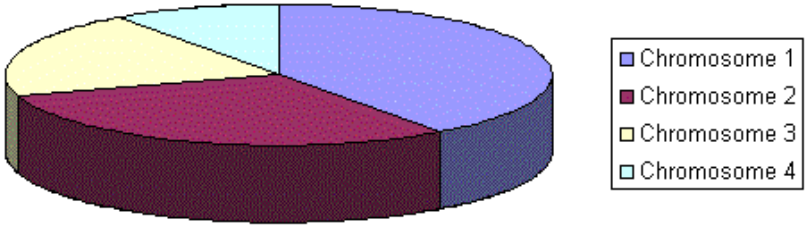
- Roulette Wheel Selection sẽ có vấn đề khi có sự khác nhau lớn giữa các độ thích nghi.
- Ví dụ: nếu NST tốt nhất có độ thích nghi là 90% thì các NST khác rất ít cơ hội được lựa chọn.
- Rank selection tính hạng quần thể đầu tiên và sau đó mọi NST nhận lại giá trị thích nghi được định nghĩa bởi hạng của chúng.
- NST tồi nhất sẽ có độ thích nghi là **1**, NST tồi thứ hai là **2** etc. và NST tốt nhất sẽ có độ thích nghi là **N** (Số nhiễm sắc thể trong quần thể).

Rank Selection

- Trạng thái sẽ thay đổi sau khi độ thích nghi sẽ được thay đổi bởi hạng của cá thể.
- Bây giờ tất cả các NST sẽ có một số ngẫu nhiên để lựa chọn.



Situation before ranking (graph of fitnesses)



Situation after ranking (graph of order numbers)

Steady-State Selection

- Trong tất các thế hệ sẽ có một vài NST tốt (Với độ thích nghi cao) được chọn để tạo NST con mới.
- Một vài NST tồi (với độ thích nghi thấp) sẽ bị xóa bỏ và các NST con mới sẽ thay thế chỗ của chúng.
- Phần còn lại của quần thể tạo ra thế hệ mới.

Chọn cá thể ưu tú - Elitism

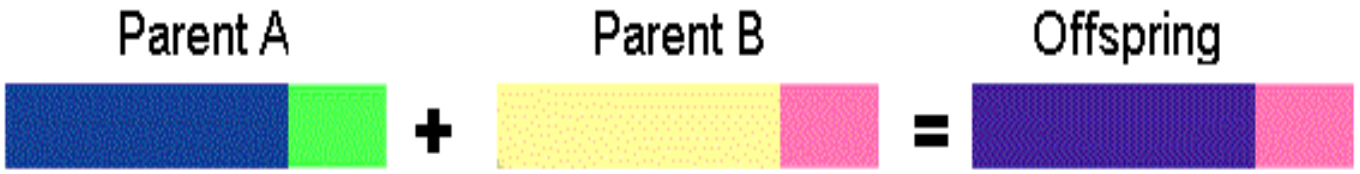
- Khi tạo quần thể mới bằng crossover hoặc mutation, có thể làm mất cá thể tốt nhất.
- Phương pháp **Elitism** cho phép copy những cá thể tốt (hoặc một vài cá thể tốt) sang quần thể mới.
- Thành phần còn lại của quần thể được tạo từ cách khác.
- Phương pháp **Elitism** có thể tăng tốc cho GA, vì nó ngăn không làm mất cá thể tốt.

Crossover và Mutation

- **Crossover** và **mutation** là hai phép toán cơ bản của thuật toán gen.
- Sự thi hành thuật toán Gen phụ thuộc rất nhiều vào crossover và mutation.
- Kiểu và sự thể hiện của các toán tử này phụ thuộc vào kiểu mã hóa và cũng phụ thuộc vào bài toán.
- Có nhiều cách để thực hiện toán tử crossover và mutation.

Crossover for Binary Encoding

- **Single point crossover** – một điểm crossover được lựa chọn, chuỗi nhị phân từ bắt đầu của NST tới điểm crossover được sao chép từ cha 1, phần cuối được sao chép từ cha còn lại
- **11001**011+11011**111** = **11001111**



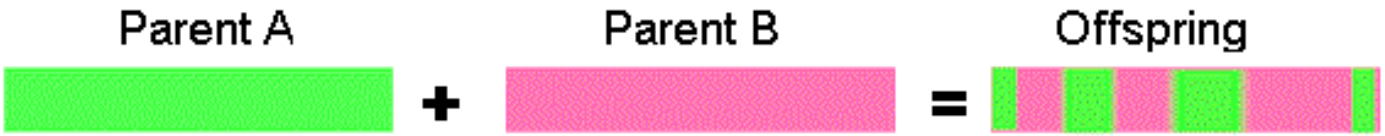
Two point crossover

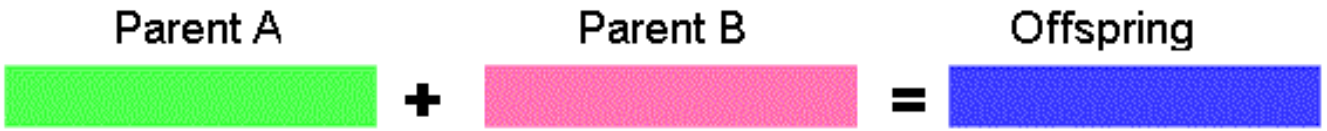
- Hai điểm crossover được chọn.
- Một chuỗi nhị phân từ bắt đầu của NST tới điểm crossover đầu tiên được sao chép từ cha thứ nhất, phần từ điểm đầu tiên tới điểm thứ hai crossover được sao chép từ cha thứ hai và phần cuối được sao chép từ cha đầu tiên
- **11**0010**11** + 11**0111**11 = **11011111**

Crossover for Binary Encoding

- **Uniform crossover -** Các bit được sao chép ngẫu nhiên từ cha thứ nhất hoặc cha thứ hai.
- **Arithmetic crossover** – Một phép toán được thực hiện để tạo ra con mới (new offspring)

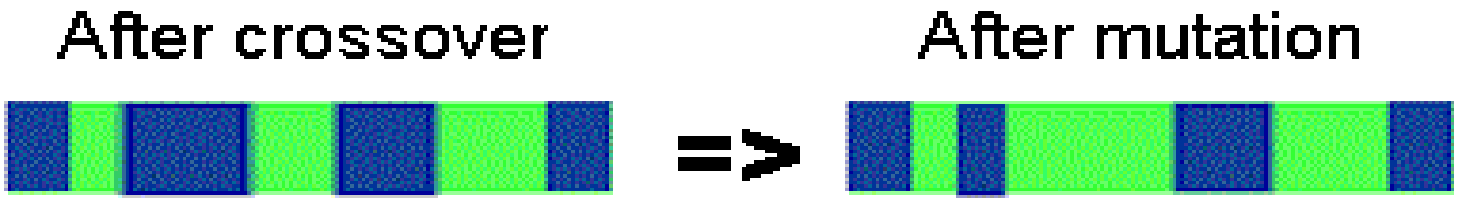
11001011 + 11011111 = 11001001 (LOGICAL AND)





Mutation for Binary Encoding

- **Đảo bit** – chọn các bits và đảo ngược giá trị của chúng
- 11001001 => 10001001



Crossover for Permutation Encoding

- **Single point crossover** – lựa chọn một điểm để lai ghép, permutation được sao chép từ điểm đầu tiên đến điểm lai ghép, sau đó cha còn lại được duyệt qua từng số và nếu số đó chưa có trong con, nó được thêm theo thứ tự của NST thứ hai
- (**1 2 3 4 5** | 6 7 8 9) + (4 5 3 **6** 2 **9** 1 **7 8**) = (**1 2 3 4 5** 6 9 7 8)

- *Chú ý: có nhiều cách để tạo ra phần sau điểm lai ghép.*

10.4.4.6.2. Crossover for Permutation Encoding

Two point crossover

- Đầu tiên, chọn hai điểm cắt, biểu thị bởi dấu “|”.
- Điểm cắt này được chen vào một cách ngẫu nhiên vào cùng một vị trí của mỗi mẫu cha mẹ

Ví dụ

- Có hai mẫu cho mẹ p1 và p2, với các điểm cắt sau thành phố thứ ba và thứ bảy

  p1 = (1 9 2 | 4 6 5 7 | 8 3)

  p2 = (4 5 9 | 1 8 7 6 | 2 3)

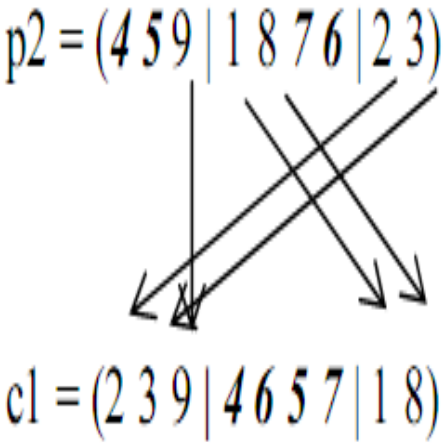
- Hai mẫu con c1 và c2 sẽ được sinh ra theo cách sau. Đầu tiên, các đoạn giữa hai điểm cắt sẽ được chép vào các mẫu con:

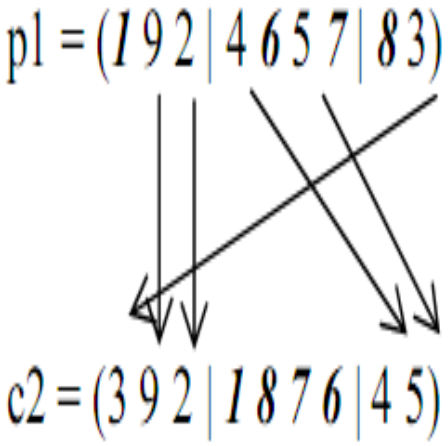
  c1 = (x x x | 4 6 5 7 | x x)

  c2 = (x x x | 1 8 7 6 | x x)

Ví dụ (tiếp)

- Bước kế tiếp là bắt đầu từ điểm cắt thứ hai của một trong hai mẫu cha mẹ, nếu ta đang muốn hoàn tất mẫu c1, thì ta sẽ bắt đầu từ điểm cắt thứ hai của mẫu p2, ta chép các thành phố từ điểm cắt này theo thứ tự vào các chỗ còn trống của c1, bỏ qua những thành phố mà c1 đã có. Khi đến cuối mẫu p2, thì quay lại đầu mẫu p2 tiếp tục chép sang c1 cho đến khi c1 đủ.





Mutation of Permuation Encoding

- **Order changing** – hai điểm được lựa chọn và thay đổi
- (1 **2** 3 4 5 6 **8** 9 7) => (1 **8** 3 4 5 6 **2** 9 7)

Crossover for Value Encoding

- Có thể sử dụng các lai ghép như kiểu mã hóa nhị phân
- Ví dụ:

(back), (back), (right), (forward), (left)

(right), (back), (left), (back), (forward)

=> (back), (back), (left), (back), (forward)

Mutation for Value Encoding

- Cộng vào hoặc trừ đi một số nhỏ từ các giá trị đã được chọn
- Ví dụ:

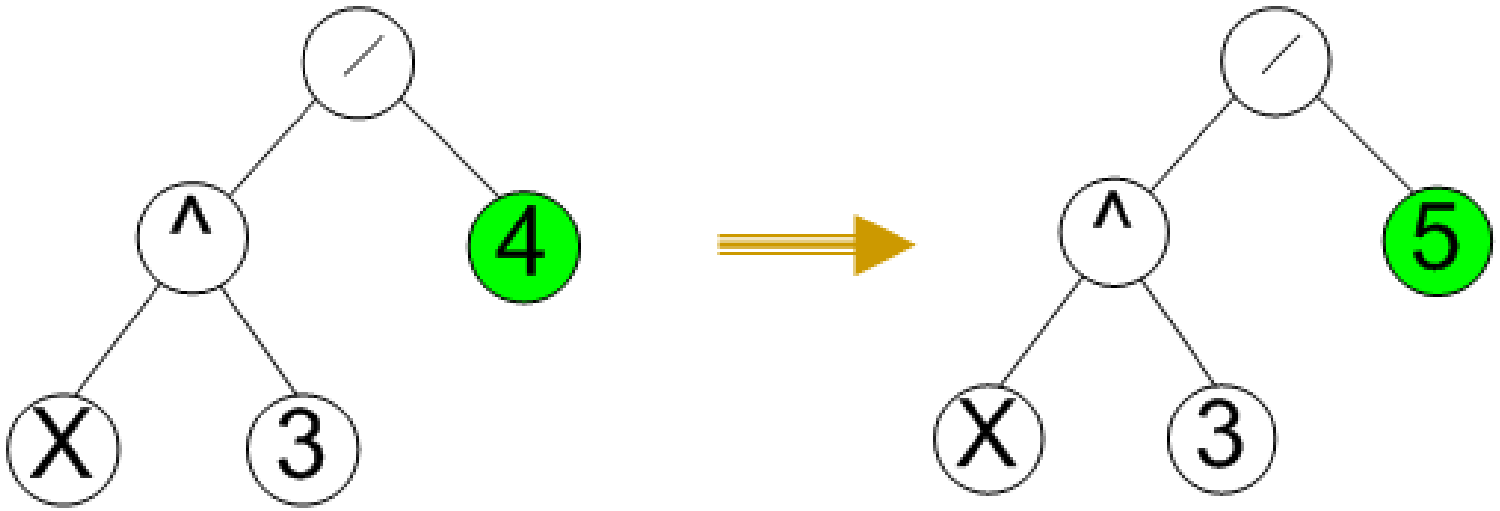
  (1.29 5.68 **2.86** **4.11** 5.55) =>

  (1.29 5.68 **2.73** **4.22** 5.55)

Crossover for Tree Encoding

- **Tree crossover** – Một điểm đột biến được chọn trong cả hai cha, các cha được phân chia bởi điểm đột biến và những phần sau điểm đột biến được thay đổi để tạo ra các con

Mutation for Tree Encoding



Crossover and Mutation Probability

- Có hai tham số cơ bản của các thuật toán gen - xác suất crossover và xác suất mutation.
- **Crossover probability**: mức độ thường xuyên crossover được thực hiện .
- Nếu không có crossover, thế hệ con được sao chép chính xác từ cha mẹ.
- Nếu có crossover, cá thể con được tạo ra từ một phần của NST cha.
- Nếu xác suất là **100%**, thì tất cả cá thể con được tạo ra bởi crossover.
- Nếu là **0%**, thế hệ mới tạo ra bằng cách sao chép nguyên các NST của thế hệ cũ.
- Crossover hy vọng tạo ra các NST mới sẽ chứa phần tốt của các NST cũ và như vậy NST mới sẽ tốt hơn.

Mutation Probability

- **Mutation probability**: Mức độ thường xuyên của các phần sẽ được đột biến.
- Nếu không có mutation, cá thể con sinh ra sẽ giống sau khi crossover (or directly copied).
- Nếu mutation được thực hiện, một hay nhiều phần của NST sẽ được thay đổi.
- Nếu xác suất là **100%**, thì NST sẽ được thay đổi, nếu là **0%**, không có thay đổi gì.
- Mutation sinh ra để tránh thuật toán gen rơi vào cực trị địa phương.
- Mutation không nên xẩy ra thường xuyên, vì nếu không GA sẽ thành phương pháp tính toán ngẫu nhiên (**random search**)

Các thành phần cơ bản của GAs

- Mã hóa (encoding)
- Khởi tạo quần thể(innitial population generation)
- Hàm thích nghi (fitness Function)
- Lựa chọn cho sự kết hợp lại (Selection for recombination)
- Lai ghép (Crossover)
- Đột biến (Mutation)
- **Chiến lược thay thế (Replacement Strategy)**
- Tiêu chuẩn kết thúc (Termination Criteria)

Tiêu chuẩn kết thúc

1. Thuật toán dừng khi quần thể hội tụ, i.e. cá thể tốt nhất trong quần thể giống với kết quả mong muốn.
2. Kết thúc khi số thế hệ sinh ra đạt đến số vòng lặp xác định trước.
3. Kết thúc khi các cả thể trở lên giống nhau.
4. Kết thúc khi cá thể tốt nhất trong quần thể không thay đổi theo thời gian.

Một số gợi ý cho GENs

- Thử nghiệm thuật toán gen cho bài toán cụ thể, bởi vì không có lý thuyết chung có thể làm hợp các tham số của thuật toán gen cho bất cứ bài toán nào.
- Quá trình có được là kết quả việc học kinh nghiệm của thuật toán gen, thông thường, phương pháp mã hóa nhị phân được sử dụng.

Một số gợi ý cho GENs

- **Crossover rate** Tốc độ lai ghép thường là cao, khoảng **80%-95%**. (Mặc dù vậy một vài kết quả cho một vài bài toán, tốc độ lai ghép khoảng 60% là tốt nhất.)
- **Mutation rate** Xác suất đột biến thường là rất thấp. Tỷ lệ này tốt nhất khoảng **0.5%-1%**.
- **Population size**
  - Kích thước quần thể rất lớn thường không cải tiến tốc độ của thuật toán gen.
  - Kích thước tốt khoảng **20-30**, mặc dù một vài trường hợp khoảng 50-100 thì tốt hơn.
  - Nhiều nghiên cứu cho thấy rằng kích thước của quần thể phụ thuộc vào kích thước của chuỗi mã hóa (chromosomes).
  - Nếu có NST 32 bits, thì kích thước quần thể nên cao hơn 16.

Một số gợi ý cho GENs

- **Lựa chọn (Selection):**
  - **roulette wheel selection** có thể được sử dụng, nhưng thỉnh thoảng rank selection có thể tốt hơn.
  - Có một số phương pháp phức tạp khác cho phép thay đổi tham số khi chọn trong khi thực hiện GA.
  - **Elitism** được chọn nếu không sử dụng cách nào khác để lưu thông tin của cá thể tốt nhất.
- **Encoding:** Phụ thuộc vào bài toán.
- Kiểu **Crossover và mutation:** Phụ thuộc vào mã hóa và bài toán

Ưu điểm

- Ưu điểm chính là khả năng song song của thuật toán .
- Gas duyệt qua không gian tìm kiếm sử dụng nhiều cá thể (and with genotype rather than phenotype) và ít mắc phải cực trị địa phương như các thuật toán khác.
- Dễ thể hiện.
- Khi đã có thuật toán gen cơ bản, chỉ cần viết một NST mới (just one object) để xử lý bài toán khác.
- Với cùng cách mã hóa, có thể thay đổi hàm thích nghi.
- Mặc dù vậy, trong một số trường hợp chọn và thể hiện mã hóa sẽ gặp khó khăn.

Nhược điểm

- Nhược điểm chính của Gas là thời gian tính toán.
- GAs có thể chậm hơn các thuật toán khác.
- Có thể kết thúc tính toán bất cứ lúc nào.

Tại Sao GAs Tốt?

- **Building blocks hypothesis and schema theorem** (Holland, 1975).

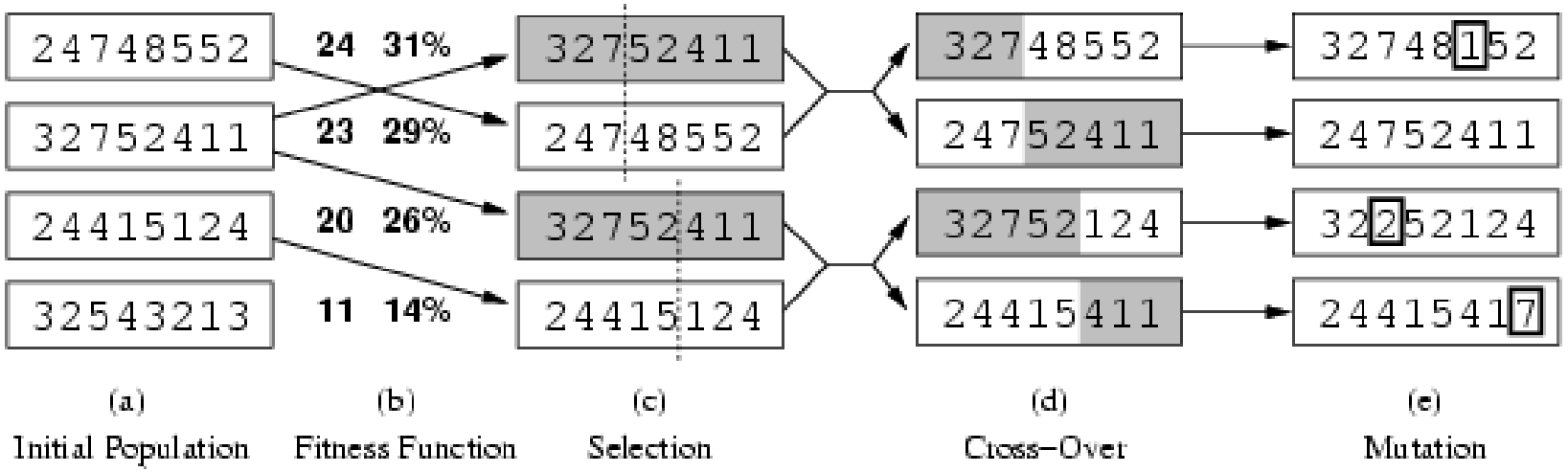
0010**0*001110*0**

0001011010011101001O

• "GAs operator set a bias towards short, low order and highly fit building blocks".

Tại Sao GAs Tốt?

Ví dụ cho biểu diễn phi nhị phân



- Fitness function: số lượng con hậu không ăn nhau (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Tại Sao GAs Tốt?

Minh họa