

Artificial Intelligence Term Project Report

Wei Li 李韓

April 30, 2017

Contents

1	Overview	1
1.1	Define Pattern Matching	2
1.2	Define Significant Pattern	2
1.3	Search Strategy	2
2	Detailed Description	3
2.1	Starting State	3
2.2	Goal Test	3
2.3	Tricks & Heuristics	3
2.3.1	Hash Table	3
2.3.2	Score's Calculation	4
2.3.3	Priority Queue's tricks	4
2.3.4	Question 2's Max Mutation	4
2.3.5	Queue's Generate	4
3	Test Results	5

1 Overview

在此Project的Q2 以及 Q3中，我使用了**基因演算法(Genetic Algorithms)**來解決此問題，並在此基礎上加上了一些 tricks 以及 heuristics。對於Q1，我使用了傳統的暴力方法，因為它已經足夠解決該問題。

我的程式使用IDE code::blocks進行compile，參數為C++11 以及 -O3。

因為演算法的隨機性質，我進行了多次測試：

對於 ex1-3, 平均在30s左右能找到最佳解，且最佳解與助教所給完全相同。

對於 ex4-5, 平均在120s左右能找到最佳解，且最佳解與助教所給完全相同。

對於 Q2，平均在30s左右能找到理想的解(我認為是最佳解)。

對於 Q3，平均在120s左右能找到理想的解(我認為是最佳解)。

Q1的code見Question1.cpp，Q2與Q3的code見Question23.cpp。

另外，請務必使用code::blocks的C++11，並加上-O3的優化參數進行compile

1.1 Define Pattern Matching

這裡，我所說的匹配(Matching)，在Q2中指的是兩個pattern中的15個基因，至多出現5處不相同，在Q3中，則是至多7處。

1.2 Define Significant Pattern

首先，我們知道，candidate patterns(候選pattern)表示，在所有control sequences均能匹配上的pattern。那麼，我們再給出significant pattern(最佳pattern)的定義：

在滿足candidate patterns(候選pattern)的情況下，在每個control sequences中，找出能與其匹配，並且突變次數最少的pattern，記錄它的突變數，這樣我們能夠得到50個數，對這50個數進行加總，得到的結果，我們稱它為該candidate patterns的score。

而score最少的candidate patterns，我們稱它為significant pattern(最佳pattern)理由很簡單，因為它突變的總次數最少(即score最小)

Algorithm 1 Using Genetic Algorithms to find *Significant Pattern*

```
Priority Queue  $\leftarrow$  original pattern
min score  $\leftarrow \infty$ 
best pattern  $\leftarrow$  empty
while true do
  random select an pattern
  random mutation the pattern
  calculate the pattern's score
  if score < min score and matched in all control sequences then
    min score  $\leftarrow$  score
    best pattern  $\leftarrow$  pattern
  end if
  Add the pattern to Priority Queue
  if Priority Queue's len > limit then
    pop the worst one
  end if
end while
```

1.3 Search Strategy

如上圖所示，根據基因演算法(GA)的適者生存，優勝劣汰遺傳機制，我們有如下思考：

在50個control sequences中，每個control sequences均存在一個或者多個 pattern，經過若干次突變(Q2中至多5次，Q3中至多7次)，變回我們要尋找的significant pattern。

那麼，我們先計算出這50*985個pattern與50個control sequences匹配的score，並將它們都放入一個Priority Queue(優先佇列)中，按照匹配的score遞增排序，score越小，排在越前。

其次，隨機地從排在優先佇列前1500位的pattern中取出一個pattern，並對其進行隨機突變(random mutation)操作，變異後，再次與所有control sequences匹配，計算score，並放入Priority Queue中。

如此，循環往復，隨機選擇pattern，進行突變，Priority Queue中pattern的數量會不斷擴大，並且匹配次數越少的pattern(說明越有可能繼續突變為最佳pattern)，會被放在Priority Queue的越前面，也就有機會再次被取出，進行再次突變，而匹配次數多，表現差的pattern，根據適者生存的規律，將會被淘汰(我們有設置優先佇列的最大長度，如果超出這個長度，pattern將會被拋棄)。

2 Detailed Description

2.1 Starting State

如前所述，最開始我們將50個control sequences所能產生的所有pattern，計算score後，全部放入Priority Queue中去，這是我們最開始的狀態空間，一共為 $50 * 985 = 49250$ 個(這裡，我們放入Priority Queue中的State，是一個包含pattern以及score的pair)。

2.2 Goal Test

對於任何一個從Priority Queue中取出並進行隨機突變的pattern，首先我們會確定它是否滿足candidate patterns(候選pattern)的要求，這個在計算score的時候就會完成，如果滿足，再將它和當前的significant pattern(最佳pattern)進行比較，如果其score低於當前最佳pattern，它將會成為新的最佳pattern。

如此看來，我們的程序並不會有明顯的停止時間，因為我們並不知道最佳pattern究竟是什麼，所以我們會不斷找到比當前最佳pattern更好的pattern。

當然，如果我們認為當前的最佳pattern已經是最後我們所要的，那我們便可以停止我們的程式，亦或是，長時間沒有找到比當前pattern更好的pattern，我們也可以停止程式。

2.3 Tricks & Heuristics

在不斷優化程式的過程中，我加入了大量的Tricks以及Heuristic。

2.3.1 Hash Table

突變是隨機的，所以一定存在某個突變後的pattern在先前已經被test過了，所以我們需要避免訪問重複的state。這裡我使用了C++ STL中的set進行重複判斷，每訪問過一個state，就將其加入到set中。另外，在程式一開始，我們就將genome.data檔案中的所有sequence產生的pattern全部加入到set中，這樣在後續的搜尋過程中，我們可以保證找到的pattern一定不會出現在genome.data中。這樣處理即方便，又省去後面check所需要耗費的時間。

2.3.2 Score's Calculation

在計算score的時候，一定會遇到一個問題：
對於當前突變的pattern，在某一個control sequences中，並沒有找到任何一個能

與其匹配pattern。此時我們不再與下一個control sequence匹配，而是直接返回一個很大的分數(假設為MAX SCORE)，表示這個pattern並不是(候選pattern)。這樣，程式可以加速不少。

但是，這樣Priority Queue中就會出現大量分數一樣的原始pattern，而大的分數將會導致其很快被淘汰。前面提及，這些原始pattern中必定存在能夠突變成最佳pattern的pattern，所以，我的處理辦法是，如果是原始pattern，如果它的score是MAX SCORE，則將它的分數減半，這樣，它存活在Priority Queue的機會將會增大。

2.3.3 Priority Queue's tricks

我們知道，Priority Queue沒加入一個state，就需要重新sort，這很耗時。我的做法是，並不適用STL中真正的Priority Queue，而是使用vector來存儲，程式每跑1000個iteration，才進行一次sort，而在sort之後，如果當前Priority Queue的總量超過了50000，我們會將50000之後的state剔除掉，這樣做是合理的，同時也是保證sort的速度，排在後面的state，是生存能力弱的state(或者說是不太可能突變為最佳pattern的state)，所以它們將會被我們無情拋棄。

另外，前面提到我們的Priority Queue的數量限制是50000，但是，我們並不會在這50000個state中進行隨機選取，我們僅僅在前1500位中進行隨機選取，這是合理的，因為排在前面的state，理應更有可能突變為最佳pattern。

2.3.4 Question 2's Max Mutation

對於Q3，最大突變數量為7，這很容易找到候選pattern，所以Priority Queue中的score更新得會很快，而對於Q2，最大突變數量為5，這很難找到候選pattern，這將會導致整個Priority Queue中都會出現score相同的state，這很可能導致存活機率更大的state排在1500位之後，那麼我們可能不會訪問到它們。

這裡，對於Q2，在最開始計算原始pattern的score的時候，我將其最大突變數量設置為6，這樣，我們會找到一些滿足最大突變數量為6的pattern，它們的score將不會是MAX SCORE的一半，而會是一個更小的數值，這意味著它將排在Priority Queue的前面。儘管我們用的最大突變數量為6，但是，很明顯可以意識到，這些pattern完全可以再通過一次突變，而變回滿足最大突變數為5的情況。當然，在計算非原始pattern的時候，我們必須將最大突變數量設置為5，因為我們需要找到真正的最佳pattern。經過測試，使用此tricks後，Q2的解題速度加快不少。

2.3.5 Queue's Generate

前面有提到，最開始，我們的Priority Queue中一共有 $50 \times 985 = 49250$ 個state。

但事實上，計算這49250個pattern的score本身是一件非常耗時的事情，而根據隨機理論，這些能夠經過突變變為最佳pattern的pattern，應該均勻(或者說相對來說均勻)地分佈在這50個有control sequence上，那麼，我們沒有必要將這49250個state全部放入Priority Queue，適當選取一些即可。

經過測試，在Q2中，我只將前10個control sequence的9850個pattern計算score後放入Priority Queue中。而在Q3中，我只將前20個control sequence的9850個pattern計算score後放入Priority Queue中。這樣的做法均能夠找到最佳解，而且還能夠進一步加快速度。

3 Test Results

程式執行後的答案，見資料夾ans。
程式執行後的log，見資料夾test_log。

```
ITERATION: 129000  MIN MUTATION: 197  CUR TIME: 36.679000
-----
- FIND PATTERN: GTTATATCGGCTGAC
- SCORE: 160
- CUR TIME: 36.692000
-----
```

ex1 ans

```
ITERATION: 84000  MIN MUTATION: 186  CUR TIME: 26.249000
-----
- FIND PATTERN: AGATCATACTGCGCT
- SCORE: 160
- CUR TIME: 26.278000
-----
```

ex2 ans

```
GENERATE QUEEN: 10 CUR TIME: 13.488000
GENERATE USED TIME: 14.716000
-----
ITERATION: 153000  MIN MUTATION: 206  CUR TIME: 36.354000
-----
- FIND PATTERN: ACTGACACCTGGTTA
- SCORE: 161
- CUR TIME: 36.384000
-----
```

ex3 ans

```
-----
ITERATION: 38000  MIN MUTATION: 212  CUR TIME: 108.141000
-----
- FIND PATTERN: CATACTGTAGCTACG
- SCORE: 211
- CUR TIME: 109.295000
-----
```

ex4 ans

```
ITERATION: 12000  MIN MUTATION: 220  CUR TIME: 65.801000
-----
- FIND PATTERN: AGATCTACTACTAGG
- SCORE: 210
- CUR TIME: 65.824000
-----
```

ex5 ans

```
-----
ITERATION: 40000  MIN MUTATION: 222  CUR TIME: 116.653000
-----
- FIND PATTERN: TTTTAATATAGTTAA
- SCORE: 221
- CUR TIME: 117.406000
-----
ITERATION: 267000  MIN MUTATION: 221  CUR TIME: 548.531000
-----
```

ex6 ans

對於ex_datasets的測試，如上圖所示，我們在ex1,ex2,ex3,ex4以及ex5中，均找到了唯一的正確答案，和助教提供的完全相同，並且程式的執行時間也在理想

的範圍內。在ex6中，我們找到的答案有些許偏差，這可能和隨機性已經程式的參數設定上有關係(我的Priority Queue總量只有50000，且我只在前1500位中進行隨機選擇，最佳答案有可能被放在後面而被淘汰了)。

```
-----  
- FIND PATTERN: CTGTTAGATCAACTG  
- SCORE: 165  
- CUR TIME: 29.809000  
-----  
ITERATION: 143000  MIN MUTATION: 165  CUR TIME: 31.242000
```

question 2 answer

```
-----  
- FIND PATTERN: ATCTATAGCCTGATG  
- SCORE: 209  
- CUR TIME: 98.775000  
-----  
ITERATION: 34000  MIN MUTATION: 209  CUR TIME: 110.655000
```

question 3 answer

對於ex_datasets的測試，如上圖所示，我們同樣得到了相當理想的答案，從score上推測，Q2的 $score = 165$, Q3的 $score = 209$ ，這兩個答案應該均為最佳答案或者接近最佳答案。

更詳細的測試信息，見資料夾 ans 以及 test_log。