第五章 結論與未來展望

5.1 結論

本系統有一項優點，那就是在加入新的書目資料格式時，不需要更改主程式內容，只須將新的書目資料格式轉為基因格式，再經過處理存入樣板資料庫即可。如此一來想要新增格式就不需要再撰寫程式，只須新增樣板序列，這不僅增加了系統的彈性，也使得系統更容易維護管理。或許會有人質疑 brief form 平均只有 14 個單位長，可以儲存的格式並不多，而且樣板資料庫的容量也不是很大，使用 dynamic programming 求最佳解應該就綽綽有餘，何必大費周章使用 BLAST 這麼龐大的程式。關於這個問題，一開始我們也是這麼認為。起初會使用 BLAST 其實就是想要偷懶，不想再寫額外的程式，所以才會將就使用。沒想到就是因為偷懶反而發現這是一個好用的工具。雖然 BLAST 在本系統是大材小用，但是這個方法卻可以應用到許多需要快速找尋特徵值的系統上。在一個資料庫龐大，而且必須比對長串資料的系統中，BLAST 就可以扮演一個非常重要的角色。例如現在要建立一個點歌系統，讓使用者只要唱一段歌曲或是哼一段歌曲，系統就可以自動把歌曲找出來。這樣的系統中想必資料庫一定很龐大，而且每一首歌的長度也不短，使用者唱出來的曲調也不一定正確。種種問題看起來，系統要快速找到正確的歌曲似乎不容易。這時候 BLAST 就可以派上用場。其實要使用者唱出正確音調並不容易，但是要唱出音調相對的變化就比較簡單。因此只要以蛋白質序列紀錄音調變化程度，再使用 BLAST 來找出最相似的片段，就可以很快對應到最相似的歌曲片段。由於 BLAST 容錯能力高，就算是使用者唱的旋律錯誤，BLAST 也可以自動加入 gap 來修正。不過這例子還只是個想法，是否可行還是要真正去作才知道。諸如此

類的應用還很多，想要建立一個新功能的系統其實很簡單。首先只須先擬定好基因轉換規則，然後再建立起計分表，最後再完成樣板資料庫就大公告成。

也許會有人懷疑使用 BLAST 是否真的可以正確分辨出每一個樣板，其實只要仔細計算重複出現的機率就可以明瞭了。以本系統的 brief form 為例，brief form 的平均長度只有 14.3 個單位長，理論上這樣的長度可以排出 4.02×10^{18} 種排列組合。而目前 OpCit 也只用了四百多種樣板。不過由於書目資料的格式會有一些特定規則存在，例如著作名稱通常會出現在標題名稱之後，類似這樣的規則會讓排列的方式大大減少。不過想要利用蛋白質序列排出 1000 種以上的格式應該是可難辦到。雖然目前還沒有辦法估計出所有書目資料格式的數目，但是我們相信以這個方法應該可以應付更多的格式。其實本系統最大的致命傷並不是格式不足，而是姓氏資料庫不足。能否正確辨別出姓氏對於系統準確率的影響程度很大。現階段只使用了中文姓氏資料庫，將來還會增加更多國家的姓氏資料庫，例如英國、日本、德國、法國、俄國、…，等等。

5.2 未來展望

現階段 BLAST 這項工具已經被我們應用在書目資料的後設資料解析之上，將來我們還會將這個方法引用到 call for paper 的搜尋上。如此就可以根據關鍵字，以及關鍵字之間的結構來作資料發掘(data mining)的動作。現在我們已經將生物資訊的工具應用在我們的程式上。我想這只是一個開端。以後我們還會利用生物資訊的其他工具來幫我們解決其它資訊領域的問題。雖然目前我們使用人工的方法來建立樣板資料庫，但是將來我們會從網際網路上蒐集結構化以及半結構化的書目資料。有了這些資料我們就可以讓電腦學習自動建

立様板資料庫。同時我們也可以從結構化的書目資料中擷取姓氏資料、期刊名稱、著作名稱等知識。將來這些知識也可當作系統解析後設資料的知識庫。再加上由本系統從半結構化書目資料解析出來的後設資料，也可以納入知識庫中。如此系統準確率一定可以比現在更好。