第五章 結論與未來研究方向

第一節 結論

透過改良方法一，我們將 Crafty 20.14 的效能提升 20%左右。由於每個棋類程式的平均子節點分支度，與 fail high 的機率是不同的，所以我們也提出一套分析及驗證方法，方便讓其它棋類程式可快速找出一個適合的 CPU 分配法則。而改良方法二是較普遍適用於各種棋類軟體，主要層數的最合適數值，則是要看 CPU 的處理速度，來決定每個 CPU 較適合的工作量大小，及決定到底控制在那個層數才允許使用 DTS 搜尋演算法做分割。而透過我們的實驗分析，我們又將原本 Crafty 20.14 的效能提升 31%左右。

綜合以上兩種改良方法，測試了 12 個盤面（如附錄 A），我們將 Crafty 20.14 的效能提升將近 40%左右，如下表 5-1 所示。

	原始 Crafty 執行時間（T1）	法一 + 法二 執行時間（T2）	提升效能 （T1 / T2 %）
盤面 1	26.64	18.42	144.625%
盤面 2	45.58	34.98	130.303%
盤面 3	34.69	24.86	139.541%
盤面 4	45.08	32.84	137.271%
盤面 5	67	47.04	142.432%
盤面 6	15.63	9.97	156.77%
盤面 7	40.76	32.96	123.665%
盤面 8	10.29	6.82	150.879%
盤面 9	14.71	11.22	131.105%
盤面 10	12.99	8.78	147.949%
盤面 11	46.32	37.66	122.995%
盤面 12	29.02	19.3	150.362%
			平均為 139.825%

表 5-1：改良方法一+改良方法二的提升效能

相信每個使用 DTS 搜尋演算法的棋類軟體，都可以再提升整體的效能。我們也將此兩種改良方法實作在電腦象棋程式“深象”中，其中改良方法一由於分割點造成的額外負擔所花費的時間比節省的總搜索節點數所賺到的時間是差不多的，所以若是改成 8 個分 6 個，效能是沒有多大的提升。而在改良方法二的部分，我們也是將深象控制在第 4 層才允許做 DTS 搜尋演算法，會獲得最好的效能。其實驗數據如下表 5-2 所示。

測試盤面	控制在第 2 層的執行時間	控制在第 3 層的執行時間	控制在第 4 層的執行時間	控制在第 5 層的執行時間	控制在第 6 層的執行時間
盤面 1	75.7	64.0	50.2	64.3	78
盤面 2	88.2	79.0	67.1	77.2	90.2
盤面 3	86.8	76.3	63.4	74.9	87.9
盤面 4	69.5	58.1	46.6	57.4	66.2
盤面 5	89.0	75.8	63.8	74.2	88.9
盤面 6	66.4	52.0	40.9	50.8	61.6
盤面 7	74.6	68.4	55.4	66.8	80.2
盤面 8	58.9	49.6	38.6	46.4	56.7
盤面 9	80.1	70.8	56.8	69.2	78.0
盤面 10	73.5	62.3	47.7	60.1	72.2

表 5-2：改良方法二的實驗在深象的結果

我們測試的盤面中（如附錄 B），將其固定完整搜索到 13 層，而執行時間為最少的絕大多數是落於控制在第 4 層才允許做 DTS 搜尋演算法，所以深象的參數便設定在 4，以便取得最好的效能。

第二節　未來研究方向

DTS 搜尋演算法的作者 Robert M. Hyatt 一直朝著降低分割點與挑選好的分割點為方向做努力。雖然我們的改良方法一是透過提高了分割點數，卻降低整體搜尋節點數來換得提升整體效能，但將來棋類軟體的走法排序與審局函數若能做得更好，使整體 fail high 的機率提高到接近於 100%，則改良方法一可能會變成不適用，所以繼續朝著降低分割點與挑選好的分割點仍為方向來提升效能的正確之路。在此也提出一個最理想的假設，要是能夠事先預測出每個節點的分支度，有多少分支度才去分配多少個 CPU，相信這樣便能將每個 CPU 的工作效能達到最高。