使用不同的博弈树搜索算法解决计算机围棋的吃子问题

张培刚,陈克训
(Department of Computer Science, University of North Carolina at Charlotte, Charlotte NC 28223, USA)

摘 要:使用 Alpha-Beta 搜索和 proof-number (pn) 搜索解决计算机围棋的吃子问题. 对吃子问题形式化并给出了简单有效的评估函数. Alpha-Beta 搜索使用了包括置换表在内的各种扩展技术. pn 搜索使用了包括 df-pn 在内的 4 种变体. 研究结果显示,对于解决吃子问题 pn 搜索优于 Alpha-Beta 搜索. 并且搜索过程中所产生的数据的一些模式可以帮助在结果未知的情况下对结果进行预测. 所设计的算法可以用于解决单独的吃子问题或者计算机围棋比赛中的吃子计算.

关键词:计算机围棋;博弈树搜索;启发式搜索;Alpha-Beta 搜索;proof-number 搜索;吃子问题
中图分类号:TP18 **文献标识码**:A **文章编号**:1673-4785(2007)03-0084-07

Using different search algorithms to solve computer Go capturing problems

ZHANG Pei-gang, CHEN Keh-hsun
(Department of Computer Science, University of North Carolina at Charlotte, Charlotte NC 28223, USA)

Abstract: The capturing problem is an important tactical sub-problem in computer Go. Alpha-Beta ( ) search and proof-number (pn) searches are used to solve computer Go capturing problems with the same evaluation function. The performance of the capturing algorithm is favorable and practical. The algorithm could be used to solve post game capturing problems or to perform real time capturing calculations in computer Go tournament matches. Our results show that pn-search is superior to-search at solving capturing problems. Some patterns can be found in the searching process that could help predict the search outcome when it is still unknown. Our work also provides a framework that could be used to solve other computer Go sub-problems and other games.

Keywords: computer Go; game tree search; Heuristic search; Alpha-Beta search; proof - number search; block capturing

博弈树搜索算法是博弈程序的核心组成部分. Alpha-Beta 搜索是最广泛使用的博弈树搜索算法. pn 搜索对于解决非对称博弈树问题也非常有效.

面对博弈程序设计人员的一些问题包括:

1) 每种搜索算法的特性是什么? 2) 对于解决一类特定问题哪种搜索算法比较好? 3) 如何将搜索算法嵌入博弈程序用于实时地解决问题? 计算机围棋是当前人工智能领域的一大难题. 计算机围棋包括很多战术问题例如死活问题[1-2]、吃子问题[3]和连接问题[4]. 构建一个强大的计算机围棋程序需要很好地解决这些战术问题.

文中尝试使用不同的树搜索算法解决围棋的吃子问题. 首先设计和实现了 Alpha-Beta 搜索及其扩展和 pn 搜索及其变体去解决吃子问题. 进一步对这些算法的结果进行比较并总结不同算法的特征. 最后对搜索过程中产生的数据进行分析并发现一些模式可以用于对搜索结果进行预测. 这个吃子算法的表现是出色和实用的. 这个算法可以用于解决单独的吃子问题或者计算机围棋比赛中的吃子计算. 结果显示,对于解决吃子问题,pn 搜索优于 Alpha-Beta 搜索. 并且发现搜索过程中的一些模式可以帮

收稿日期:2006-08-21.

助在结果未知的情况下对结果进行预测. 本文的工作也为解决其他计算机围棋问题和其他博弈问题提供了一个框架.

1　吃子问题

1.1　计算机围棋吃子问题

围棋是一项非常古老而且受欢迎的棋类游戏. 计算机围棋是人工智能中一个很吸引人的研究领域[5-6]. 由于其内在复杂性,计算机围棋程序的棋力与人相比在目前仍然大大落后. 计算机围棋遇到各种类型的问题,为人工智能的各种方法提供了极佳的试验场. 它包括许多有趣的战术问题,例如死活、吃子和连接问题. 构建一个强大的计算机围棋程序需要很好地解决这些战术问题[7].

吃子计算的目的是要得到一块棋能否被吃死或者能否逃脱以及如何吃死或者逃脱. 吃子计算对于计算机围棋程序是基本和关键的[3]. 几乎所有的计算机围棋程序都包括吃子模块. 吃子算法可以被用于解决单独的吃子问题或者被嵌入游戏引擎进行实时的吃子计算. 由于在实时计算中时间是非常有限的,所以吃子算法的效率非常重要.

吃子问题为博弈树搜索算法提供了很好的试验场. 一些研究者提出了一般用途的算法并且以解决吃子问题作为示例[8-9]. 这些算法解决吃子问题的速度非常慢. 为了得到更好的和更实际的结果,需要进一步的研究.

在作者以前的一项研究中,基于 Alpha-Beta 搜索并高度性选择的启发式搜索被用于解决围棋的吃子问题,得到了很好的结果[3]. 文中使用不同的搜索算法利用相对简单的评估函数并且考虑更多的候选着法以处理各种不同的可能情况.

面对吃子问题,高水平的围棋选手一般能够很快发现棋子的弱点并进行深而且窄的搜索. 对于大多数情况,基于以前经验所形成的知识,他们能够发现棋形的弱点.

在下面的几节将介绍候选着法的产生和排序的

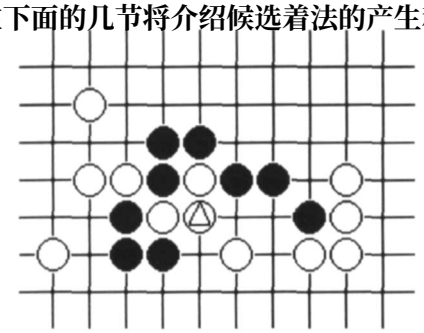


图 1　如图标出的白子是吃子的目标,
如果轮黑棋下能够被吃住

Fig. 1　The marked white stone is the target, it could be captured if black move next

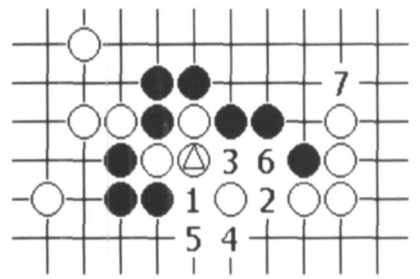


图 2　数字标出的点是候选招法,
2 是吃住目标的正确着法

Fig. 2　The candidate moves to capturing the marked tone, location 2 is the right move to capture the target

策略、终止状态检测、中间节点和叶子节点评估. 然后将使用同样的方法来产生候选着法、节点评估、终止状态检测并将它们用于不同的搜索算法. 关于计算机围棋和吃子问题的一些术语可以见文献[3,6].

1.2　候选着法产生

候选着法一般靠近目标棋块或者上一手棋. 一块棋的相关选点被定义为这块棋的气和二次气以及对手相邻这块棋并且不大于三气的棋块的气. 这里使用一个简单的评估函数,将着这手棋下在棋盘上并且计算:

Point Evaluation = # Block's First liberties * 4 + # Block's Second liberties

然后选择评估值比较大的相关选点.

考虑 5 种类型的棋块:

1) 目标棋块;

2) 目标棋块的关键链;

3) 上一手棋所属棋块;

4) 对手相邻棋块 ;

5) 上一手棋的相邻棋块.

每一种类型的棋块都能产生一些相关选点,在数目上设定了上限和下限. 基本上从这 5 种类型的棋块的相关选点中选择候选着法.

使用 2 种类型的目标去帮助选择候选着法:吃子目标和逃脱目标. 如果下一手棋轮目标棋块方下是逃脱目标,否则就是吃子目标. 对于不同目标候选着法的选择策略也略有不同. 例如,对于逃脱目标,候选着法不能使目标棋块被直接征子吃掉.

算法中也使用一些其他条件去排除候选着法. 候选着法不能填自己的真眼. 候选着法的总数被限制在 15 以下,对于绝大多数情况这是足够的.

1.3　节点评估

使用对目标棋块的关键链的评估值作为中间节点和叶子节点的评估值:

Node Evaluation = # Crucial chain's First liberties *4 + # Crucial chain's Second liberties

1.4 终止状态测试

算法中设定一个节点评估值的上限来进行终止状态测试.如果节点的评估值大于这个上限,则目标棋块被认为可以逃脱.如果目标棋块可以被直接吃掉或者被征子吃掉则被认为可以吃掉.

2 博弈树搜索算法

对于二人、零和、完备信息的棋类游戏,例如围棋、象棋、国际象棋和跳棋,博弈树算法对构建一个强大的博弈程序是关键的.博弈树搜索的目标是要找到极大极小博弈树或者与和树的最佳值以及最佳着法序列.一般博弈树非常大以至于不能被当前的计算机所求解.降低博弈树大小的策略包括缩小候选着法的数目以缩小展开因子、当搜索到某一深度时使用评估函数以及使用搜索路径选择策略.选择候选着法以及使用评估函数很大程度上取决于领域知识.搜索路径选择策略则一般与领域知识无关.深度优先搜索和最佳优先搜索是最主要的2种搜索路径选择策略.

Alpha-Beta搜索是深度优先搜索算法而且是当前博弈游戏程序最广泛使用的博弈树搜索算法[10-11].候选着法的排序对于Alpha-Beta搜索的效率是关键的.Alpha-Beta搜索有很多扩展,其中包括迭代加深、置换表、历史启发函数等,大多数用于得到更好的候选着法排序.使用迭代加深的Alpha-Beta搜索可以得到某一深度的最优值,这可以作为限制时间搜索下的近似解.Alpha-Beta搜索的缺点是很难搜索到很深的深度.

pn搜索是最佳优先搜索算法.对于非均衡树的求解,pn搜索是非常强大的[12].pn搜索需要将整个搜索树保存在内存中,所以内存消耗很大.近年来,出现了一些pn搜索的深度优先搜索的变体,例如pn *[13]、df-pn和df-pn+[14].使用proof number和disproof number的阈值,这些算法可以以深度优先的方式去搜索博弈树,但是所展开的博弈树与pn搜索是一样的.这些深度优先的变体可以减少内存的使用,但是效果与pn搜索相似.

目前Alpha-Beta搜索及其扩展仍然是最广泛使用的博弈树搜索算法[11].pn搜索及其变体变得越来越流行[1].

3 Alpha-Beta搜索

Alpha-Beta搜索的思想是利用Alpha值——对于极大节点的最差可能值和Beta值——对于极小节点的最差可能值.使用这些值,很多分支可以被裁减掉[15].

在儿子节点被最好排序的情况下,Alpha-Beta搜索为求解问题需要访问的节点数目约为$w^{d/2}$,w是平均候选着法的数目,d是搜索的深度.在最坏情况下,为了求解它需要搜索整个博弈树.

文中使用Alpha-Beta搜索及迭代加深、置换表.每次迭代深度增加为2.关于置换表使用Zobrist哈希[16]和线性探查.

几种常用的用于博弈树搜索算法比较的指标有:CPU时间、访问的叶子节点数、访问的所有节点数[17].文中使用访问的所有节点数来衡量算法的表现.

这里使用Kano围棋系列的第3册中的吃子问题来检验算法效果.每一个问题的搜索时间被限制在200 s内.表1包含了Alpha-Beta搜索不使用置换表和使用置换表的测试结果.

使用置换表的算法表现比不使用置换表的算法表现好:使用的时间较少、访问的节点较少.将节点的最佳值存储在置换表中可以避免对同样的局面重新计算.在迭代加深中使用置换表也可以帮助候选着法排序.

候选着法排序的质量对Alpha-Beta搜索的效率是关键的.但对于不同问题很难给出统一的评估函数去得到候选着法的最佳排序.进一步的工作可以使用模式数据库去帮助候选着法排序.

结果发现Alpha-Beta搜索不能解决的问题主要由于以下3个原因:

1)在时间限制下Alpha-Beta搜索很难搜索到很深的深度.对于那些需要很深深度搜索才能求解的问题,Alpha-Beta搜索一般用光时间从而失败.

2)候选着法没有包括所有可能着法:正确着法缺失.

3)解决此问题需要其他知识如连接和死活知识.

4 proof-number搜索

pn搜索是最佳优先搜索算法.对于非均衡树的求解pn搜索是非常强大的.pn搜索的思想是总是展开那些用最小代价就可以证明或者否证与或博弈树的节点[12].pn搜索将节点分为2种:与节点和或节点.在每个与节点努力去否证节点,在每个或节点努力去证明节点.在每个节点上使用2个数值:proof number——最少需要展开的叶子节点去证明此节点的数目;disproof number——最少需要展开的叶子节点去否证此节点的数目.对于博弈树有时

可以看成是与或树,有时可以看成是极大极小树,与节点通常与极小节点是一样的,或节点通常与极大节点是一样的.

近年来出现的 pn*、df - pn 和 df - pn + 等变体使用哈希表存储中间结果. 内存消耗可以减小到哈希表的大小.

对于非终止节点 pn 搜索和 df - pn 将 proof number 和 disproof number 设为 1. Df - pn + 使用评估函数 h 和 cost 来计算 proof number 和 disproof number. 类似的也可以对 pn 搜索使用评估函数来计算 proof number 和 disproof number,可以称这种算法为 pn + .

对于 pn + ,

$$pn = E^2, dn = \left(\frac{1}{E} \times 25\right)^2.$$

对于 df - pn + ,

$$cost = 0, h.pn = E^2, h.dn = \left(\frac{1}{E} \times 25\right)^2.$$

式中:E 是节点评估值,pn 是 proof number,dn 是 disproof number.

这里使用 pn 搜索、pn + 、df - pn 和 df - pn + 解决吃子问题. 候选着法产生和终止状态测试与 Alpha - Beta 搜索是一样的. 算法中设定了访问的节点总数和搜索时间来限定搜索. 对于 df - pn 和 df - pn + ,一些候选着法产生但没有被访问,所以所访问的节点计数显得比 pn 搜索和 pn + 要少.

这里使用 Kano 围棋系列的第 3 册中的吃子问题来检测算法的效果. 每一个问题的搜索时间被限制在 200 s 内. 表 1 包含了测试结果.

表 1　不同算法解决 Kano 系列第 3 册中吃子问题的结果

Table 1　Results of different algorithms to solve capturing problems in Kano book 3)

算法	求解成功问题	求解失败问题	时间/ ms	节点数	深度
αβ无置换表	52	9	5 502	28 746	5. 6
αβ有置换表	53	8	2 785	13 187	5. 8
pn	55	6	2 148	18 607	7. 2
pn +	55	6	1 303	10 931	7. 0
df - pn	55	6	2 259	5 792	7. 4
df - pn +	55	6	1 648	2 175	6. 7

表 1 中,时间是求解问题所用平均时间(ms);节点数是求解问题所访问平均节点数;深度是平均深度.

pn 搜索及其变体比 Alpha-Beta 搜索解出了更多的问题、花费了更少的时间、访问了更少的节点和搜索到更深的深度. pn 搜索及其变体未能解的问题是 Alpha-Beta 搜索未能解的问题的子集.

使用 pn 搜索及其变体的另外一个好处是它们不需要对候选着法的排序. 这也会节省相当时间.

结果发现那些没有解决的问题主要由于以下 2 个原因:

1) 候选着法没有包括所有可能着法:正确着法缺失.

2) 解决此问题需要其他知识如连接和死活知识.

5　结果分析

5. 1　算法比较

对于大多数问题,pn 搜索及其变体比 Alpha-Beta 搜索效果好. 大多数问题可以在 1 s 内被解出. pn 搜索、pn + 、df - pn 和 df - pn + 没有解出的 6 个问题是相同的问题,他们是 Alpha-Beta 搜索未能解出的问题的子集. 在所有解出的问题中,其中 13 个问题 Alpha-Beta 搜索比 pn 搜索效果好;29 个问题 pn 搜索比 Alpha-Beta 搜索效果好;13 个问题 pn 搜索和 Alpha-Beta 搜索表现相当. 从上表也可以看到,pn + 表现比 pn 搜索好,df - pn + 比 df - pn 表现好. 使用评估函数计算 proof number 和 disproof number 的算法,证明和否证根节点所访问的节点数比较少.

图 3～5 是在限定时间下各种算法的表现比较. 可以看到搜索时间如果限定在 100 ms,pn + 的表现最好.

从图 4 和图 5 可以看到 pn 搜索及其变体有基本相同的表现而且比 Alpha-Beta 搜索表现好.

5. 2　吃子问题分析

Kano 系列第 3 册中的 61 个吃子问题的搜索结果为:49 个问题成功吃死,6 个问题成功逃脱,6 个问题未成功求解.

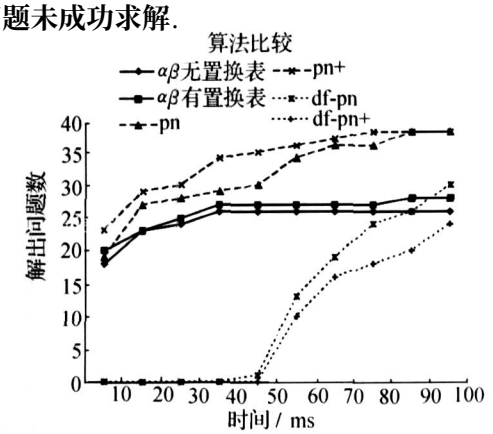


图 3　所解问题数(100 ms 内限制)

Fig. 3　Problems solved within 100 milliseconds

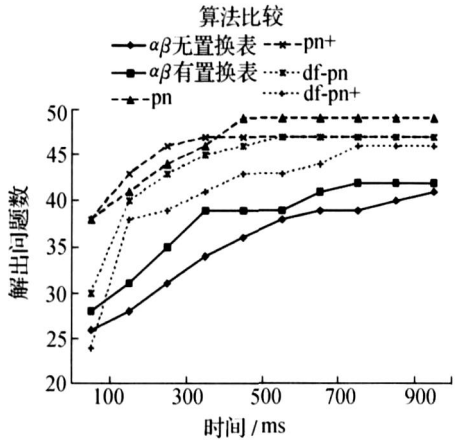


图4 所解问题数(1 s内限制)

Fig. 4 Problems solved within 1 second

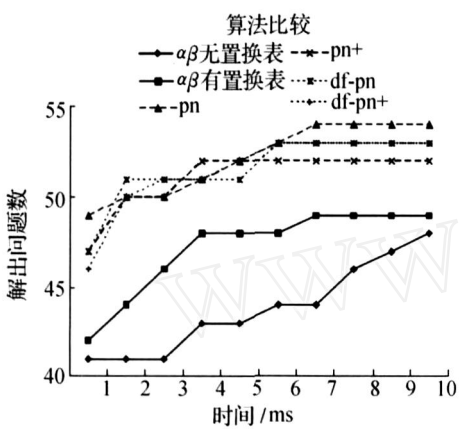


图5 所解问题数(10 s内限制)

Fig. 5 Problems solved within 10 seconds

对于其中的13个问题,Alpha-Beta搜索比pn搜索求解时间少.图6所示问题Alpha-Beta搜索使用640 ms访问3 139个节点成功求解,pn搜索使用3 384 ms访问16 791个节点成功求解.

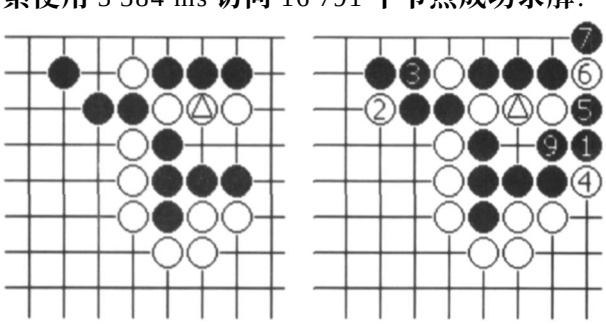


图6 一个Alpha-Beta搜索求解快于pn搜索的问题

Fig. 6 A problem $\alpha\beta$ search solved faster than pn-search

对于其中的29个问题,pn搜索比Alpha-Beta搜索求解时间少.图7所示问题Alpha-Beta搜索使用33 s访问133 384个节点成功求解,pn搜索只使用260 ms访问2 094个节点成功求解.pn搜索的求解过程很像人的思考方法:进行深而且窄的搜索.

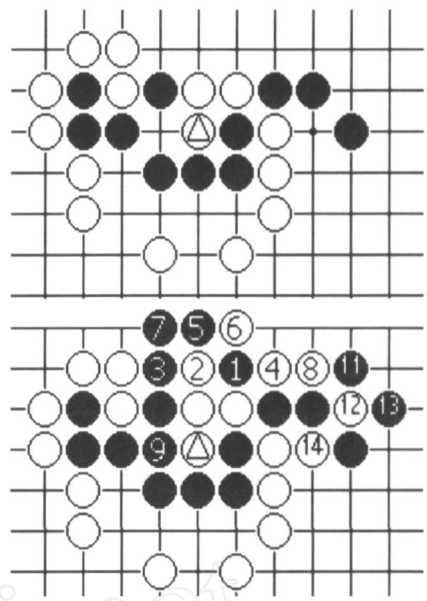


图7 一个pn搜索求解快于Alpha-Beta搜索的问题

Fig. 7 A problem pn-search solved faster than $\alpha\beta$ search

5.3 Alpha-Beta搜索中的数据分析

程序记录了Alpha-Beta搜索过程中每一层迭代加深所得到的最佳值.发现对于不同种类的问题,每类问题所得到的最佳值的变化存在着相似的模式.图6～8是最佳值随深度增加而变化的几种示例.

某一深度的最佳值是在搜索到此深度所得到目标棋块关键链的一阶和二阶气的评估.值越小越有利于吃子方,值越大越有利于逃脱方.当目标被吃住时值变为0.

对于那些成功吃死的问题,最佳值可能会先有所增加然后逐渐降为0.对于那些成功逃脱的问题,最佳值一般越来越大.对于那些未能求解的问题,最佳值在一定范围内徘徊.

需要进一步的研究这种模式是否可以帮助在结果未知的情况下对结果进行预测.对于实际的博弈程序分配给某一问题的搜索时间通常非常有限,很难进行全面的搜索.如果这种预测可信,那么它将非常实用.

5.4 pn搜索中的数据分析

程序记录了pn搜索过程中根节点的proof number和disproof number的变化.可以发现对于每种类型的问题根节点的proof number和disproof number的变化也存在着一定的模式.图9和10是

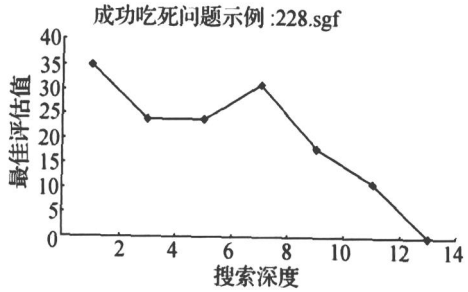


图 8　最佳值随深度变化(成功吃死问题示例)

Fig. 8　The best value of $\alpha\beta$ search with search deepening, captured problem example

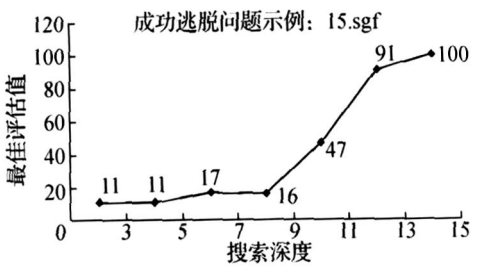


图 9　最佳值随深度变化(成功逃脱问题示例)

Fig. 9　The best value of $\alpha\beta$ search with search deepening, escaped problem example

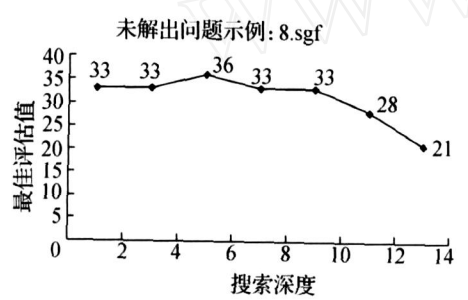
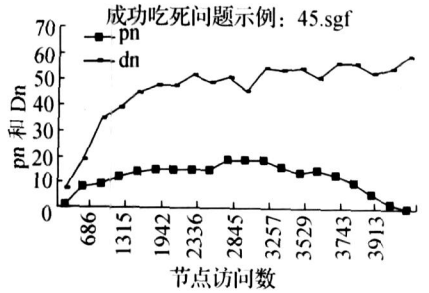


图 10　最佳值随深度变化(未能求解问题示例)

Fig. 10　The best value of $\alpha\beta$ search with search deepening, unsolved problem example

随搜索访问节点数的增加根节点的 proof number 和 disproof number 的变化情况.

proof number 是最少需要展开的叶子节点去证明此节点的数目;disproof number 是最少需要展开的叶子节点去否证此节点的数目. 如果根节点的 proof number 变为 0 则目标被吃死. 如果根节点的 disproof number 变为 0 则目标逃脱. 对于那些成功吃死的问题,proof number 一般增加到一定的峰值然后逐渐下降直到 0,disproof number 一般持续增长. 对于那些成功逃脱的问题,proof number 一般持续增长,disproof number 一般增加到一定的峰值然后逐渐下降直到 0. 对于那些未解出的问题,proof number 和 disproof number 一般持续增长.



11　pn 搜索过程中根节点的 proof number 和 disproof number 的变化(成功吃死问题示例)

Fig. 11　The variation of proof number and disproof number during pn-search, captured problem example

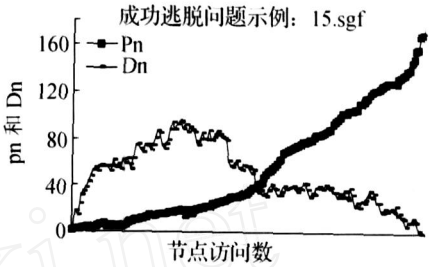


图 12　pn 搜索过程中根节点的 proof number 和 disproof number 的变化(成功逃脱问题示例)

Fig. 12　The variation of proof number and disproof number during pn-search, escaped problem example

需要进一步的研究这种模式是否可以帮助在结果未知的情况下对结果进行预测.

6　结束语

与其他相关工作比较,本文的吃子算法的表现是出色的. pn 搜索及其变体在解决吃子问题中有更好的表现. 大多数问题可以在不到 1 s 的时间内求解成功. 这一成果可以被用于计算机围棋程序在比赛中使用.

那些未能求解的问题基本上可以归为 2 个原因:评估函数的质量不高和需要其他类型的领域知识如连接和死活. 这需要提高评估函数的质量和使用其他围棋战术知识.

进一步的工作包括:

1) 进一步优化吃子问题的评估函数. 包括优化候选着法的产生、节点评估和用于 pn + 和 df - pn + 中的评估函数. 机器学习的方法例如遗传算法可以引入以帮助优化评估函数.

2) 进一步研究 Alpha-Beta 搜索和 pn 搜索中的模式及其对搜索结果的预测.

3)比较 Alpha-Beta 搜索和 pn 搜索所访问的博弈树.研究将 Alpha-Beta 搜索和 pn 搜索结合以得到更好的搜索效果的可能性.

4)使用本搜索框架去解决其他计算机围棋中的战术问题.

参考文献:

[1]KISHIMOTO A. Correct and efficient search algorithms in the presence of repetitions[D]. Edmonton: University of Alberta, 2005.

[2]WOLF T. Forward pruning and other heuristic search techniques in tsume go[J]. Information Sciences, 2000, 122 (1) :59 - 76.

[3]CHEN K, ZHANG P. A heuristic search algorithm for capturing problems in go[J]. ICGA Journal, 2006, 29 (4) : 183 - 190.

[4]CHEN K. Soft and hard connectivity in go[A]. Proceedings of the 8th International Conference on Computer Science and Informatics[C]. Salt Lake City, USA, 2005.

[5]BOUZY B, CAZENAVE T. Computer go: an AI oriented survey[J]. Artificial Intelligence, 2001, 132(1) :39 - 103.

[6]MULLER M. Computer go[J]. Artificial Intelligence, 2002, 134(1 - 2) :145 - 179.

[7]CHEN K. Computer go: knowledge, search, and move decision[J]. ICGA Journal, 2001, 24(4) :203 - 215.

[8]CAZENAVE T. Abstract proof search, computers and games 2000[A]. LNCS 2063[C]. Hamamatsu, Japan, 2000.

[9]THOMSEN T. Lambda-Search in game trees-with application to go[J]. ICGA Journal, 2000, 23(4) :203 - 217 .

[10]CAMPBELL M, HOANE A J, HSU F. Deep blue[J]. Artificial Intelligence, 2002, 134(1 - 2) : 57 - 83.

[11]JUNGHANNS A. Are there practical alternatives to alpha-beta[J] . ICCA Journal, 1998, 21(1) :14 - 32.

[12]ALLIS L V, MEULEN M, HERIK H J. Proof-number search[J]. Artificial Intelligence, 1994, 66(1) :91 - 124.

[13]SEO M, IIDA H, UITERWIJK J W. The PN *-search algorithm: application to Tsume-Shogi[J]. Artificial Intelligence, 2001, 129(1 - 2) :253 - 277.

[14]NAGAI A. Df-pn algorithm for searching and/or trees and its applications[D]. Tokyo: University of Tokyo, 2002.

[15]KNUTH D, MOORE R. Analysis of alpha-beta pruning[J]. Artificial Intelligence, 1975, 6(4) :293 - 326.

[16]ZOBRIST A L. A new hashing method with application for game playing[R]. Techn. Rep. # 88, Madison: Univ. of Wisconsin, 1970.

[17]CAMPBELL M, MARSLAND T. A comparison of minimax tree-search algorithms[J]. Artificial Intelligence, 1983, 20 (4) : 347 - 367.

作者简介:

张培刚,男,1973 年生,博士研究生,主要研究方向为计算机围棋、人工智能、启发式搜索和机器学习等.

E-mail:pzhang1 @uncc. edu.

陈克训,男,1945 年生,教授,博士. 主要研究方向为启发式搜索、人工智能和计算理论等. 所研发的电脑围棋程序“棋慧”曾两获电脑围棋世界冠军和七次夺得计算机奥林匹亚竞赛金牌. 发表论文 40 余篇.

E-mail:chen @uncc. edu.