文章编号:1001 - 9081(2005)11 - 1720 - 03

围棋死活问题的计算机求解

廖 里
(乐山师范学院 计算机科学系,四川 乐山 614004)
(liaoli_lstc@163.com)

摘 要:提出了一种基于搜索的围棋死活问题的求解方法,并实现了一个围棋死活问题求解程序 SharpSense。对比实验表明,SharpSense 的性能明显优于同类程序,对封闭围棋死活问题的解题能力达到了围棋专业棋手的水平。SharpSense 还发现了围棋死活问题经典著作《围棋死活大全》中的两个错误。

关键词:搜索算法;博弈;博弈树;围棋;死活问题
中图分类号: TP181 **文献标识码**:A

Solving life and death problems in Go

LIAO Li
(*Department of Computer Science, Leshan Teachers College, Leshan Sichuan* 614004, *China*)

Abstract: An approach to solve life and death problems in Go, which was based on search algorithm, was presented. With this approach, the program SharpSense was implemented. Experiments show that SharpSense outperforms other programs solving life and death problems in Go. And in solving enclosed problems, SharpSense has the capability of professional Go player. SharpSense detects two errors in All about Life and Death Problems in Go, which is the classic in this domain.

Key words: search algorithms; game; game tree; Go; life and death problems

0 引言

计算机博弈是人工智能中历史最长,也是最受公众关注的研究领域之一。计算机博弈为人工智能提供一个实验平台,将人工智能的一些理论与方法应用于计算机博弈,可通过博弈水平的高低来检验这些理论与方法的有效性;研究计算机博弈所得到的成果也可推广至人工智能的其他领域。因此,计算机博弈被认为是人工智能的"果蝇"。如同生物学家通过研究果蝇取得许多生物学上的重大成果,研究计算机博弈也会取得人工智能上的重大成果。计算机博弈的传统研究重点是国际象棋,随着 IBM 公司研制的"深蓝"电脑战胜了人类国际象棋世界冠军,计算机博弈研究者将注意力转向了复杂度更高的围棋。人工智能学科创始人之一 John McCarthy 认为,围棋是人工智能的新"果蝇"[1]。

尽管近十几年来人们对计算机围棋做了大量的研究,目前最好的围棋程序的棋力仍然很弱[2,3]。现有围棋程序的一个弱点是不能准确地判断棋子的死活。与棋子死活相关的问题即为死活问题,在死活问题中,一方为进攻方,另一方为防守方。如果是进攻方先走,死活问题就是给出一个杀死防守方的走法;如果是防守方先走,死活问题就是给出一个救活防守方的走法。围棋死活问题大多数情况下是封闭死活问题,在封闭死活问题中,进攻方将防守方完全包围。死活问题的求解是高性能围棋人机对弈程序的关键问题之一,同时专门的死活问题求解程序也有一定的实用价值。利用死活问题求解程序对现有的死活问题及其解答进行验证,可发现其中的错题和错解。解答死活问题是围棋棋手必须进行的训练,可利用死活问题求解程序构造交互式死活问题训练系统,在这种系统中,训练者可以和计算机在死活问题范围内进行人机对弈,使训练者更深刻地理解死活问题,从而提高训练效果。

GoTools 是近十年来最强的封闭死活问题求解程序[4],GoTools 用 alpha-beta 剪枝算法进行搜索,使用启发式围棋规则进行走法排序,并用遗传算法调整这些启发式围棋规则的参数[5]。GoTools 的解题能力达到了业余高段棋手的水平,但 GoTools 对复杂死活问题的解答时间仍然过长。

本文提出了一种基于搜索的围棋死活问题的求解方法,该方法使用变深度 NegaScout 搜索算法对死活问题所对应的博弈树进行搜索,根据问题的规模动态地设置置换表的容量,并根据搜索过程中所获得的信息进行走法排序。基于这种方法实现了一个封闭死活问题求解程序 SharpSense。

1 基于搜索的围棋死活问题的求解方法

1.1 搜索算法

NegaScout 搜索算法是对 alpha-beta 剪枝算法的一种有效改进[6],它的基本思想是试图用最小窗 alpha-beta 剪枝算法去证明一个走法的价值要低于当前最好的走法,最小窗 alpha-beta 剪枝算法的代价要小于一般的 alpha-beta 剪枝算法。如证明成功,即该走法的价值确实低于当前最好的走法,则可直接考虑下一个走法。如证明不成功,即该走法的价值要高于当前最好的走法,则要对该走法进行一次一般的 alpha-beta 剪枝算法。对走法进行排序后,在大多数情况下是先对价值高的走法进行搜索,再对价值低的走法进行搜索,用最小窗 alpha-beta 剪枝算法证明成功的可能性要就要高于不成功的可能性,从而降低了总的搜索代价。

NegaScout 搜索算法最初用于国际象棋程序,它有一个参数是最大搜索深度,当对博弈树进行的搜索达到规定的最大搜索深度后即对当前局面用估值函数进行估值。本文用变深

收稿日期:2005 - 05 - 09
作者简介:廖里(1973 -),男,湖南邵阳人,讲师,硕士,主要研究方向:人工智能、算法设计与分析.

度 NegaScout 搜索算法求解围棋死活问题,该算法不再设置最大搜索深度,对每一深度的局面都用估值函数进行估值;如估值函数已能够判断防守方的死活,则停止搜索;否则对该局面生成全部合法的走法,继续进行搜索。对死活问题的求解表明变深度 NegaScout 搜索算法的最大搜索深度是随着问题的难度变化的。对于容易和中等难度的题目,最大搜索深度一般在 10 和 30 之间,对于复杂的问题,最大搜索深度一般在 30 和 60 之间。变深度 NegaScout 搜索算法 VD_NegaScout 描述如下:

```
procedure VD_NegaScout (P, alpha, beta)
v = evaluate (P)
if v≠UN_DECIDED then
  return v
end if
M = generateMoves (P)
make (P, m1)
best = - VD_NegaScout (P, - beta, - alpha)
undo (P, m1)
for all m∈M except m1 do
  if best≥beta then
    return best
  end if
  alpha = max (best, alpha)
  make (P, m)
  v = - VD_NegaScout (P, - alpha - 1, - alpha)
  if v > best then
    if v > alpha and v < beta then
      best = - VD_NegaScout (P, - beta, - v)
    else
      best = v
    end if
  end if
  undo (P, m)
end for
return best
```

1.2　置换表

对于围棋死活问题而言,博弈树中的一些局面会通过不同的路径重复遇到。将已搜索局面的有关信息保存在置换表中,可以避免重复搜索同一局面,或者能够加快对该局面的搜索。为加快置换表访问速度,置换表是由哈希表实现的。每个局面按照以下方法转换为一个 64 位二进制哈希数:在搜索开始之前为黑棋在棋盘上的每个位置设置一个 64 位二进制随机数,为白棋在棋盘上的每个位置设置一个 64 位二进制随机数。此外,设置两个 64 位二进制随机数分别对应于轮黑棋走和轮白棋走两种情况。对于是否发生劫争,劫争的位置也设置相应的随机数。每个局面按照各个棋子在棋盘上的位置、轮哪一方走和劫争情况将上面设置的 64 位二进制随机数按位进行异或运算即得到 64 位二进制哈希数。根据置换表的容量,取这个哈希数的低位数字即为这个局面在置换表中的位置。如置换表的容量的为 512K 个局面,则取哈希数的低 19 位数字。

在国际象棋程序中,增大置换表的容量能提高搜索速度[7],这是因为增大置换表的容量能减少置换表冲突(即两个不同的局面分配到置换表的同一位置)的可能性。对于围棋死活问题的求解,则要根据问题的难易程度动态地设置置换表的容量。对于容易的问题,求解过程中搜索的局面较少,用小容量的置换表即可使冲突的可能性减少到接近于零的程度,再增加置换表的容量不仅不会提高速度,反而会降低内存访问的局部性和 Cache 的命中率,导致解题速度的下降。可用死活问题的规模,即死活问题中的空白点的个数,来衡量问题的难易程度。空白点的个数越多,双方可供选择的走法也越多,问题也就越复杂。本文对置换表的容量按以下方法进行设置:令空白点的个数为 x,$x \leqslant 9$ 时,置换表的容量为 8K 个局面;$10 \leqslant x \leqslant 13$ 时,置换表的容量为 32K 个局面;$x = 14$ 时,置换表的容量为 64K 个局面;$x = 15$ 时,置换表的容量为 128K 个局面;$x = 16$ 时,置换表的容量为 256K 个局面;$x = 17$ 时,置换表的容量为 512K 个局面;$x \geqslant 18$ 时,置换表的容量为 2M 个局面。

1.3　走法排序

走法排序是将某一局面的所有合法的走法按照其价值的高低进行排序,价值高的走法先进行搜索。走法排序对 NegaScout 搜索算法的性能具有重大的影响,成功的走法排序方案能在大多数情况下将价值高的走法排在价值低的走法的前面进行搜索,从而提高最小窗 alpha-beta 剪枝算法证明成功的可能性。本文根据搜索过程中所获得的信息进行走法排序,其中包括置换表中走法信息[7]、杀手启发[8]、兄弟提升启发[9]和历史启发[10]。置换表中保存已搜索局面的最好走法,当在搜索中又碰到同一局面,则将置换表中的最好走法作为优先搜索的走法。杀手启发将博弈树中每一层引起剪枝的走法(杀手走法)保存起来,当对博弈树其他部分的搜索到达这一层时,如果杀手走法是该局面的合法走法,则优先搜索杀手走法。兄弟提升启发来源自围棋中的格言:“敌之要点,我之要点”。对于一个局面 P,我方走了 $m1$ 后,对手走 $m2$ 即可引起剪枝,则 $m2$ 可能是该局面的要点,应优先搜索。历史启发为每个走法记录了一个历史分数,当一个走法引起剪枝或者是一个局面中最好的走法时,将会增加它的历史分数,走法排序即根据每个走法的历史分数进行排序。

本文将以上四种搜索过程中所获得的信息结合起来。对于每个局面,先搜索置换表中保存的走法,再搜索杀手走法,最后按照历史分数的高低搜索其他走法。对每个走法所得到的兄弟提升走法,将作为下一个搜索的走法。

2　性能测试

根据以上方法实现了一个封闭死活问题求解程序 SharpSense。

2.1　与 GoTools 的比较

用文献[4]中的 64 个问题作为测试集,将 SharpSense 的性能与 GoTools 进行了对比。在 Celeron1.1G/128M 机器上,GoTools 全部正确解答上述 64 个问题共需要 71.5s,SharpSense 全部正确解答上述 64 个问题共需要 4.50s,解题速度是 GoTools 的 15.9 倍。

2.2　解题能力测试

用《秀行死活题杰作集》[11]中的封闭死活问题测试了 SharpSense 的解题能力。《秀行死活题杰作集》为日本著名棋手藤泽秀行所著,其中的死活问题有四种难度:初级、中级、高级和高段。其中高段难度的死活问题非常复杂,围棋专业棋手解答也需要用相当长的时间。对这些死活问题的解答情况见表 1。对初级、中级和高级三种难度的死活问题以 10 min 作为时间限制,如解题时间超出 10 min 则为解题失败;对高段难度的死活问题以 120 min 作为时间限制。

表 1 SharpSense 对四种难度死活问题的解答情况

难度	问题个数	其中封闭问题个数	SharpSense 正确解答的个数	正确解答率(%)	最长正确解答时间/s	最短正确解答时间/s	平均正确解答时间/s
初级	40	37	37	100	9.508	0.001	0.709
中级	39	36	35	97.2	8.414	0.001	0.569
高级	61	54	53	98.1	25.98	0.004	1.805
高段	23	21	19	90.5	4245.7	0.01	293.7

高段难度的死活问题中解题时间最长的五个问题的解答情况见表 2。对于其中的 163 题,藤泽秀行认为围棋专业棋手需要思考几个小时才能解答[11]。

表 2 解题时间最长的高段难度问题的解答情况

题数	问题中的空白点	搜索的局面	最大搜索深度	解答时间/s
162	25	936 218 537	75	4 245.7
154	22	143 763 141	74	706.4
163	21	82 111 266	55	365.2
161	20	42 815 704	99	193.7
149	22	6 568 311	53	27.6

3 对《围棋死活大全》的验证

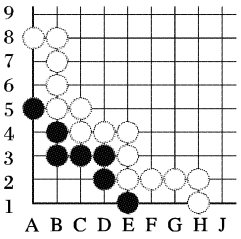

图1 一个死活题

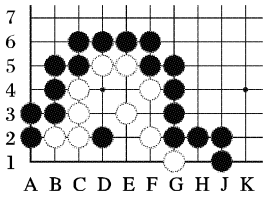

图2 一个解说图

《围棋死活大全》是围棋死活问题经典著作。用 SharpSense 对《围棋死活大全》中封闭死活问题的解答进行了验证,发现了其中两个错误。对于图 1 所示的问题,《围棋死活大全》认为即使白棋先走,黑棋也是无条件活棋[12]。但是 SharpSense 发现将形成劫争,黑棋不能无条件活棋。SharpSense 所发现的形成劫争的走法是:白 B1,黑 A4,白 F1,黑 D1,白 B2,黑 A2,白 A1。《围棋死活大全》忽略了白 F1 这步棋。对于图 2 所示的问题[13],《围棋死活大全》认为白棋走 F3 可活棋,但 SharpSense 发现这是错误的走法。白 F3,黑 D4,白 E4,黑 B1 后,白棋已是死棋。图 2 所示的问题,白棋走 D1 才能活棋。

致谢:在此感谢加拿大 Brock 大学 Thomas Wolf 教授为本文的研究提供了 GoTools 软件。

参考文献:

[1] MCCARTHY J. Chess as the drosophila of AI [A]. Computers, Chess and Cognition[C]. Springer-Verlag, 1990. 227 - 237.

[2] BOUZY B, CAZENAVE T. Computer Go: An AI oriented survey [J]. Artificial Intelligence, 2001, 132(1): 39 - 102.

[3] MUELLER M. Computer Go[J]. Artificial Intelligence, 2002, 134 (1/2): 145 - 179.

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

[5] PRATOLA M, WOLF T. Optimizing GoTools' Search Heuristics using Genetic Algorithms[J]. ICGA Journal, 2003, 26 (1): 28 - 49.

[6] REINEFELD A. An Improvement to the Scout Tree Search Algorithm[J]. ICGA Journal, 1983, 6(4): 4 - 14.

[7] BREUKER DM, UITERWIJK JWHM, HERIK HJVD. Replacement Schemes for Transposition Tables [J]. ICCA Journal, 1994, 17 (4): 183 - 193.

[8] AKL SG, NEWBORN MM. The principle continuation and the killer heuristic [A]. ACM Annual Conference Proceedings[C]. Seattle: ACM, 1977. 466 - 473.

[9] DYER D. Searches, tree pruning and tree ordering in Go [A]. Proceedings of the Game Programming Workshop in Japan[C]. Tokyo: Computer Shogi Association, 1995. 207 - 216.

[10] SCHAEFFER J. The History Heuristic and the Performance of Alpha-Beta Enhancements [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1989, 11(11): 1203 - 1212.

[11] 藤泽秀行. 秀行死活题杰作集[M]. 北京: 华夏出版社, 1987.

[12] 赵治勋. 围棋死活大全(上)[M]. 成都: 蜀蓉棋艺出版社, 1989. 72 - 73.

[13] 赵治勋. 围棋死活大全(下)[M]. 成都: 蜀蓉棋艺出版社, 1989. 100 - 101.

(上接第 2704 页)

3) 注册表的监控和维护

实现目标:按层次结构输出注册表中的注册信息;能阻塞或者恢复向某个注册资源转发服务请求;能强制注销某项服务。

具体实现:这个部分的功能需要直接操纵注册表数据。除了系统自动维护注册表之外,还应当支持手工校正系统问题,以维持系统连续运转。

5 结语

面向服务的消息中间件的显著特点就是实现了从服务描述到服务发现等面向服务的特性,极大增强了系统提供服务的灵活性,支持经典的 C/S 服务模式。同时,简单的负载均衡策略有助于充分利用系统中的冗余设备,避免单个服务器负载过大。

当然,提高服务的灵活性也增加了处理服务参数的复杂性,服务器和客户端必须自行处理消息中的服务参数并调用相应服务模块。不过,目前的某些语言已经支持动态调用函数并动态传入各个参数,例如 Java,这将使得通过消息驱动服务可以完全自动化处理,进一步提高了系统集成的开发效率。

参考文献:

[1] VINOSKI S. Is Your Middleware Dead? [DB/OL]. IEEE Computer Society, 2004 - 10.

[2] Object Management Group. The Common Object Request Broker: Architecture and Specification (3.0 Edition) [EB/OL], 2002 - 06.

[3] Java Message Service (JMS) Specification. Version 1.1[EB/OL]. http://java.sun.com/products/jms/docs.html, 2004.

[4] BOOTH D, HAAS H, MCCABE F, *et al*. Web Service Architecture [EB/OL]. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf, 2004.

[5] MAHESHWARI P, TANG H, LIANG R. Enhancing Web services with message-oriented middleware[DB/OL]. IEEE Computer Society, 2004 - 07.

[6] 郭红, 陈锐, 胡黎明. 基于 XML、CORBA 和 Agent 技术的集成模型研究[J]. 小型微型计算机系统, 2003, 24(9): 1646 - 1649.