搜索方法中的剪枝优化

王　新
（常州轻工职业技术学院 信息工程系，江苏 常州 213164）

摘要：剪枝是搜索方法中最常见的一种优化技巧，而且主要以剪枝判断方法的设计为核心。文章分析了设计剪枝判断方法的三个原则：正确、准确、高效。并将常见的设计剪枝判断的思路分成可行性剪枝和最优性剪枝两大类，最后结合上述三个原则分别以一实例加以说明。

关键词：搜索；优化；剪枝
中图分类号：TP393　　文献标识码：A　　文章编号：1009-3044(2007)11-21398-02

The Branch Optimizes Scissors in Searching for Method

WANG Xin
(Information Project Department of Changzhou Institute of Light Industry Technology, Changzhou 213164, China)

Abstract: Pruning is the most familiar optimization technique in the way of search. Designs of pruning judgment method are especially critical. This paper analyzes three principles about the designs: validity, accuracy, efficiency. It also has the design ideas divided into two kinds: feasibility pruning and optimization pruning. Combining above three principles, both are validated by giving an example respectively at the end of paper.

Key words: Search; Optimization; Pruning

1 引言

搜索是人工智能中的一种基本方法。在建立一个搜索算法的时候，首要的问题不外乎两个：一是建立合适的算法模型；二是选择适当的数据结构。然而，搜索方法的时间复杂度大多是指数级的，简单的不加优化的搜索，其时间效率往往低的不能忍受。本文所讨论的主要内容就是建立了合适的算法模型之后，对程序进行优化的一种基本方法——剪枝。

首先应当明确的是："剪枝"的含义是什么？我们知道，搜索的进程可以看作是从树根出发，遍历一棵倒置的树（搜索树，参见图1）的过程。而所谓剪枝，顾名思义，就是通过某种判断，避免一些不必要的遍历过程。形象的说，就是剪去了搜索树中的某些"枝条"，故称"剪枝"。

在编写搜索程序的时候，一般都要考虑到剪枝。显而易见，应用剪枝优化的核心问题是设计剪枝判断方法，即确定哪些枝条应当舍弃，哪些枝条应当保留的方法。设计出好的剪枝判断方法，往往能够使程序的运行时间大大缩短；否则，也可能适得其反。那么，我们就应当首先分析一下设计剪枝判断方法的时候，需要遵循的一些原则。

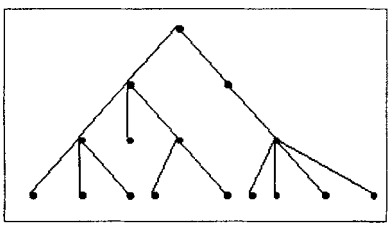

图1　搜索树

2 剪枝的原则

2.1 正确性

剪枝方法之所以能够优化程序的执行效率，正如前文所述，是因为它能够"剪去"搜索树中的一些"枝条"。这样，就可以保证所剪掉的枝条一定不是正解所在的枝条。当然，由必要条件的定义，没有被剪枝不意味着一定可以得到正解。

2.2 准确性

在保证了正确性的基础上，对剪枝判断的第二个要求就是准确性，即能够尽可能多的剪去不能通向正确解的枝条。剪枝方法只有在具有了较高的准确性的时候，才能真正收到优化的效果。

2.3 高效性

一般说来，设计好剪枝判断方法之后，对搜索树的每个枝条都要执行一次判断操作。然而，由于是利用出解的"必要条件"进行判断，所以，必然有很多不含正确解的枝条没有被剪枝。综上所述，我们可以把剪枝优化的主要原则归结为六个字：正确、准确、高效。当然，在应用剪枝优化的时候，仅有上述的原则是不够的，还需要具体研究一些设计剪枝判断方法的思路。我们可以把常用的剪枝判断大致分成以下两类：一是可行性剪枝；二是最优性剪枝（上下界剪枝）。下面就结合上述的三个原则，分别对这两种剪枝判断方法进行一些讨论。

3 可行性剪枝

我们已经知道，搜索过程可以看作是对一棵树的遍历。在很多情况下，并不是搜索树中的所有枝条都能通向我们需要的结果，很多的枝条实际上只是一些死胡同。如果我们能够在刚刚进入这样的死胡同的时候，就能够判断出来并立即剪枝，程序的效率往往会得到提高。而所谓可行性剪枝，正是基于这样一种考虑。下面我们举一个例子来说明。

例1 Betsy 的旅行

问题描述：

一个正方形的小镇被分成 N2 个小方格，Betsy 要从左上角的方格到达左下角的方格，并且经过每个方格恰好一次。编程对于给定的 N，计算出 Betsy 能采用的所有的旅行路线的数目。

问题分析：

我们用深度优先的回溯方法来解决这个问题：Betsy 从左上角出发，每一步可以从一个格子移动到相邻的没有到过的格子中，遇到死胡同则回溯，当移动了 N^2-1 步并达到左下角时，即得到了一条新的路径，继续回溯搜索，直至遍历完所有道路。但是，仅仅依照上述算法框架编程，时间效率极低，对 N=6 的情况就无法很好的解决。所以，优化势在必行。

对本题优化的关键就在于当搜索到某一个步骤时，能够提前判断出在后面的搜索过程中是否一定会遇到死胡同，而可行性剪枝正可以在这里派上用场。我们首先从"必要条件"，即合法的解所应当具备的特征的角度分析剪枝的方法，主要有两个方向：

(1)对于一条合法的路径，除出发格子和目标格子外，每一个中间格子都必然有"一进一出"的过程。所以在搜索过程中，必须保证每个尚未经过的格子都与至少两个尚未经过的格子相邻（除

收稿日期：2007-05-24
作者简介：王新（1975-），讲师，南京师范大学，研究方向：电子商务。

非当时 Betsy 就在它旁边)。这里,我们是从微观的角度分析问题;

(2)在第一个条件的基础上,我们还可以从宏观的角度分析,进一步提高剪枝判断的准确性。显然,在一个合法的移动方案的任何时刻,都不可能有孤立的区域存在。虽然孤立区域中的每一个格子也可能都有至少两个相邻的空的格子,但它们作为一个整体,Betsy 已经不能达到。我们也应当及时判断出这种情况,并避免之。

以上两个剪枝判断条件都是正确的,其准确度也比较高。但是,仅仅满足这两点还不够,剪枝判断的操作过程还必须力求高效。假如我们在每次剪枝判断时,都简单的对 N^2 个格子进行一遍扫描,其效率的低下可想而知。因此,我们必须尽可能的简化判断的过程。实际上,由于 Betsy 的每一次移动,只会影响到附近的格子,所以每次执行剪枝判断时,应当只对她附近的格子进行检查。

对于第一个剪枝条件,我们可以设一个整型标志数组,分别保存与每个格子相邻的没被经过的格子的数目,Betsy 每次移动到一个新位置,都只会使与之相邻的至多 4 个格子的标志值发生变化,只要检查它们的标志值即可;

而对于第二个剪枝条件,处理就稍稍麻烦一些。但我们仍然可以使用局部分析的方法, 即只通过对 Betsy 附近的格子进行判断,就确定是否应当剪枝,图 2 简要说明了剪枝的原理:

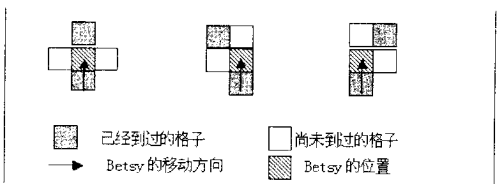

图 2 剪枝原理示意图

上图给出了可以剪枝的三种情况。由于 Betsy 到过的所有格子都一定是四连通的,所以每种情况下的两个白色的格子之间必定是不连通的, 它们当中必然至少有一个是属于某个孤立区域的,都一定可以剪枝。经过上述的优化,程序的时间效率有了很大的提高。

一般说来,可行性剪枝经常用于路径搜索类的问题。在应用可行性剪枝的时候,首先要多角度全面分析问题的特点(本题就是从微观和宏观两个角度设计剪枝方法), 找到尽可能多的可以剪枝的情况;同时,还必须注意提高剪枝的时间效率,所以我们使用了“局部判断”的方法,特别是在处理第二个剪枝条件时,更是通过局部判断来体现整体性质(是否有孤立区域),这一技巧不仅在设计剪枝方法的时候能够发挥作用,在其他方面也有着极为广泛的应用。

4 最优性剪枝

在我们遇到的问题中, 有一大类问题是所谓的 “最优化问题”,即所要求的结果是最优解。如果我们使用搜索方法来解决这类问题,那么,最优性剪枝是一定要考虑到的。

为了表述的统一,首先要作一些说明:我们知道,解的优劣一般是通过一个评价函数来评判的。这里定义一个抽象的评价函数——“优度”,它的值越大,对应的解也就越优(对于具体的问题,我们可以认为“优度”代表正的收益或负的代价等)。

然后,我们再来回顾一下搜索最优解的过程:一般情况下,我们需要保存一个“当前最优解”,实际上就是保存解的优度的一个下界。在遍历到搜索树的叶子节点的时候,我们就能得到一个新的解,当然也就得到了它的评价函数值,与保存的优度的下界作比较,如果新解的优度值更大,则这个优度值就成为新的下界。搜索结束后,所保存的解就是最优解。

那么,最优性剪枝又是如何进行的呢? 当我们处在搜索树的枝条上时,可以通过某种方法估算出该枝条上的所有解的评价函数的上界,即所谓估价函数 h。显然,h 大于当前保存的优度的下界,是该枝条上存在最优解的必要条件,否则就一定可以剪枝。所以,最优性剪枝也可以称为“上下界剪枝”。同时,我们也可以看到,最优性剪枝的核心问题就是估价函数的建立。下面举一个应用最优性剪枝的典型例题。

例 2 最少乘法次数

问题描述:

由 x 开始,通过最少的乘法次数得出 x^n,其中 n 为输入数据。

问题分析:

因为两式相乘等于方幂相加, 所以本题可以等效的表示为:构造一个数列$\{a_i\}$,满足:

$$a_i=\begin{cases}1 & (i=1)\\ a_j+a_k\ (1<=j,k<i) & (i>1)\end{cases}$$

要求 $a_t=n$,并且使 t 最小。

我们选择回溯法作为本程序的主体结构:当搜索到第 i 层时,a_i 的取值范围在 $a_{i-1}+1$ 到 $a_{i-1}*2$ 之间,为了尽快接近目标 n,应当从 $a_{i-1}*2$ 开始从大到小为 a_i 取值,当然,选取的数都不能大于 n。当搜索到 n 出现时,就得到了一个解,与当前保存的解比较取优。最终搜索结束之后,即得到最终的最优解。如果按上述结构直接进行搜索,效率显然很低。

这道题中所体现的最优性剪枝技巧是很多的,它们的优化效果也是非常显著的。

由本题并结合一些经验,我们可以简单总结一些应用最优性剪枝时需注意的问题:

(1)估价函数的设计当然首先得满足正确原则,即使用“必要条件”来剪枝。在此基础上,就要注意提高估价的准确性。本题的优化之二就是为了这个目的;

(2)与其他剪枝判断操作一样,最优性剪枝的估价函数在提高准确性的同时,也必须注意使计算过程尽量高效的原则。由于剪枝判断在运行时执行极为频繁,所以对其算法进行精雕细琢是相当必要的,有时甚至还可以使用一些非常手法,如前述的“逐步细化”技巧,就是一种寻求时间效率和精确度相平衡的方法;

(3)在使用最优性剪枝时,一个好的初始“优度”下界往往是非常重要的。在搜索开始之前,我们可以使用某种高效方法(如贪心法等)求出一个较优解,作为初始下界,经常可以剪去大量明显不可能的枝条。本题使用划分阶段的搜索方法进行初始定界,就带来了大幅度的优化;

(4)本文所举的是在深度优先搜索中应用最优性剪枝的例子。当然,在广度优先搜索中,也是可以使用最优性剪枝的,也就是我们常说的分枝定界方法。本题也可以使用分枝定界结合 A* 算法来解决(用改进的估价函数作为优度上界估计,即 h* 函数;前述的动态规划方法可以作为优度下界估计), 但时间效率和空间效率都要差一些。不过,在有些问题中,分枝定界和 A* 算法的结合以其较好的稳定性,还是有用武之地的。

5 总结

搜索方法,因其在时间效率方面“先天不足”,所以人们才有针对性的研究出了很多优化技巧。本文所论述的“剪枝”就是最常见的优化方法之一,几乎可以说,只要想使用搜索算法来解决问题,就必须考虑到剪枝优化。另外需要说明的是,本文所介绍的可行性和最优性两种剪枝判断, 其分类只是依据其不同的应用对象,而前面阐述的剪枝方法的三个原则——正确、准确和高效,才是贯穿于始终的灵魂。

本文还介绍了一些剪枝中的常用技巧,如“局部分析”、“逐步细化”等。恰当的使用它们,也能够使程序的效率(主要是时间效率)得到相当的提高。但是,剪枝方法无论多么巧妙,都不能从本质上降低搜索算法的时间复杂度,这是不争的事实。因此,我们在动手设计一个搜索算法之前,不妨先考虑一下是否存在着更为有效的方法。

总之,我们在实际设计程序的过程中,不能钻牛角尖,而更应当充分发挥思维的灵活性,坚持“具体问题具体分析”的思想方法。

参考文献:

[1]刘福生,王建德. 人工智能搜索与程序设计.