在动态开放环境中的一种跳步实时搜索算法*

罗 翊 胡 蓬 石纯一 王克宏

(清华大学计算机系 北京 100084)

摘要 本文在分析动态开放环境特性的基础上,提出了适应这些特性的搜索算法所应具有的性质,给出了一种适应动态环境的搜索算法——跳步算法.通过初步实验,对算法性能进行了分析与说明,表明算法具有较好的运行效率.

关键词 搜索算法,实时算法,动态开放问题,局部搜索.

搜索方法是人工智能中传统的通用求解方法[1],它通过尝试多种可能的解题步骤来达到求解目标.随着计算机系统与其环境进行着日益复杂、频繁的交互,象在大规模网络中进行网络寻址,处理路径断接、站点崩溃、新站点的加入等,提出了处理环境的动态开放性的要求.搜索算法必须能满足环境的实时要求,并能在不具有充分和确切的信息时作出合理的反应.传统的搜索方法着眼于预先找出一条求解路径,再按这条路径执行,规划与执行阶段完全分离.1990年Korf提出一种将执行阶段与规划阶段交替进行的实时启发式搜索算法(RTA*)[2],可以调节规划所花时间与执行时间的相对比例,满足动态环境的需要.由于将全部求解的阶段分成多个相隔离的规划、执行阶段,成功地解决了A*在实际计算机上所不能求解的25数码问题.

本文提出实时跳步搜索算法,具有比RTA*更高的搜索效率,而且可以较好地满足实时处理对搜索所提出的要求.

1 动态开放环境下的实时算法

传统的智能处理过程都是在封闭世界中进行,状态和条件只有当求解主体对它们施以动作之后才会改变.求解主体生成求解序列后,只要依照此序列执行便可以达到目标.但实际世界中的问题经常不满足封闭世界假设,不能假定当前存在的条件在若干时间之后依然成立,也不能假定当前不存在的条件在若干时间后就仍然不成立.甚至连系统目标在若干时间后也可能会改变.在这样一种环境中进行规划,若要生成完整的规划,就得考虑相当多种

* 本文研究得到国家自然科学基金资助.
作者罗翊,1970年生,博士生,主要研究领域为分布式人工智能,多Agent系统,智能Agent.胡蓬,1963年生,博士,副教授,主要研究领域为分布式人工智能,计算机体系结构.石纯一,1935年生,教授,博士导师,主要研究领域为人工智能应用基础,知识工程.王克宏,1941年生,教授,主要研究领域为知识工程,分布式人工智能.

本文通讯联系人:罗翊,北京 100084,清华大学计算机系
本文1995-12-18收到修改稿

可能性,这显然是不可能的.

因此,在动态开放环境中的求解技术需要基于部分规划,先根据当前条件和当前目标设计一个接近目标的部分解路径,然后执行这个部分解,使系统到达一个新状态,这个新状态与原来的状态相比更接近求解目标.在下一个规划/执行循环中根据新的当前条件和当前目标来生成进一步接近目标的部分解路径.经过多次循环后就可以达到目标.但是,条件和目标在生成部分解之后、执行部分解之前仍然可能变化.所以,要使部分规划方法真正可以解决动态开放环境问题,还得求助于自然的规律或假设:时间区间越小,在此区间内可能发生的变化也越少,即使有变化,其变化的幅度也会越小.这样,问题就变为如何使规划/执行循环的时间区间减小到可以忽略环境的变化,或者即使不能忽略这些变化,但要从新的状态到达目标也不会比从初始状态到达目标多花费很多代价.

对实时算法的要求之一就是用尽量少的时间完成任务.但这是所有算法追求的性能标准之一.真正对实时任务提出的要求是在规定的时间限度以内来完成一定的任务.通用实时算法中应该可以调整规划/执行循环的时间,从而可以适应不同的环境要求.

在动态开放环境中搜索求解,需要了解环境的最大响应时间限度和环境变化频度等2个基本环境参量,来确定可用的搜索时间和最长动作序列的长度.局部搜索可以适应环境的变化,随动态环境而不断修正、生成新的动作序列.

Korf在文献[2]中提出将规划与执行交错起来的RTA*实时算法,可以通过调整规划阶段所花时间来调整决策的准确性.另外还有限时A*算法,它对规划阶段设定阈值,当A*搜索的时间超出此阈值时就执行相应动作.

调整规划/执行循环的时间,可从规划时间和执行时间2方面来进行.规划时间主要由搜索的范围确定,而执行时间由执行的动作确定.从这条思路出发,本文提出跳步搜索算法.

2 跳步搜索算法

跳步搜索算法是对爬山法的推广,基本思想是在一次规划阶段中寻找一定范围内离当前目标最近的局部最优状态,并在执行阶段中执行一系列动作来达到该状态.在下一个规划/行动循环中再重复此过程,直到达到目标.由于它选取新状态不是只在当前状态的相邻状态中选择,而是在比较大的范围内选取一个距离比较远的状态,直接从此状态开始下一次的搜索,如同在搜索空间中跳跃,所以称为跳步搜索LS(leap search).

对局部最优状态的搜索是在受限的搜索空间中进行,当超出限度时,就转入执行阶段.可以用多种搜索方法和多种范围限度标准来进行局部搜索.最简单的是用固定最大深度的深度优先搜索方法,通过改变最大深度,就可以改变规划阶段所花时间.为了防止搜索过程陷入局部最优点无法退出,要维持一个已生成状态表,记忆以前各次循环中最后选定的局部目标.假如在当前搜索中遇到表中的状态,就不必再考虑此状态,而去寻找其它较优状态.局部目标不限于搜索前沿,而可以是搜索树中间状态,从而减少跳步的风险.为防止启发式函数的误差使搜索导向错误的方向,可将各个搜索出发点的直接后继状态及其启发式函数值保存在待扩展表中,作为局部搜索的选择对象.同时为减少待扩展表中状态数目,要对启发式函数值比较大的状态予以删除.

采用深度搜索为局部搜索方法的跳步搜索算法流程如下:

(0)设已生成状态表 GS 为空表,待扩展状态表 ES 为空表,初始状态为 C,当前局部最优状态 $S_{Min}=C$.

(1)设最小解代价 Min$S$ 为∞.

(1)从当前状态 C 作最大深度为 D 的深度优先搜索.生成新状态 S,或者从 ES 中选取状态 S.若所有状态都已生成,则转(2);

(2)判定 S 是否已在 GS 中,若是,则转(1);

(3)用启发式函数计算从 S 到当前目标 G 的距离 $h(S)$;

(4)判断是否 $h(S)<$Min$S$,若是,则令 Min$S=h(S)$,并记录从 C 到 S 的部分解路径;

(5)判断是否 $h(S)=$Min$S$,并且从 C 到 S 的解路径长度 $g(C,S)$ 是否小于 $g(C,S_{Min})$,若是,则记录从 C 到 S 的部分解.

(6)转(1).

(2)若 Min$S$ 为∞(表明周围所有状态都已被生成过),增大 D,再转到(1).

(3)执行:按照记录的从 C 到 S_{Min} 的部分解路径执行.若 $S_{Min}=G$,则结束,否则令 $C=S_{Min}$,$GS=GS+S_{Min}$,$ES=ES+\{s|s$ 可由 S_{Min} 直接生成$\}-\{s|h(s)-h(S_{Min})>R\}$,转(1).

可以通过设置不同的搜索深度 D 来调节规划阶段代价与执行阶段代价的平衡.若设置较大的 D,则规划阶段要花费更多的时间,但也缩短了到达目标的总的求解路径长度,缩短执行阶段所花费的总时间,还可以克服局部峰值的障碍,提高决策正确性.

提高搜索效率的方法之一是减少重复搜索.扩展新状态时可以利用上次搜索的结果,剪去部分搜索分支.最简单的方法是将通往原来状态的路径剪去.例如在 8 数码问题中,搜索树节点分支系数为 1.732[2],树根处分支系数为 2.732.剪去反向扩展的分支后,根的分支系数也为 1.732,节省了 36.6% 的计算量.例如当搜索深度为 5,而当前状态是上次搜索的前沿节点时,剪枝的效果可见图 1.图中 C 是起始状态,节点旁边的数字是相对于 C 的搜索深度.在 C 之上及同级的节点都是已在前一次搜索中被展开过的.因剪枝而遗漏的节点是深度在 5 以内且处于 C 之下的节点.由图可知,遗漏节点数为 $(b-1)*b^{(5-2)}+(b-1)*b*b^{(5-4)}$.8 数码问题的 $b=1.732$,计算得 5.999,而扩展总节点数为 $(1+b)(1+b+b^2+b^3+b^4)=54.439$,遗漏节点数占总数的 11%.节省的计算量为 36.6%.当前状态是上一次搜索的中间状态时,遗漏节点数更少.对于 8 数码这种有多条求解路径的问题,遗漏少数节点不会造成严重的影响.两相比较,剪枝是合理的.

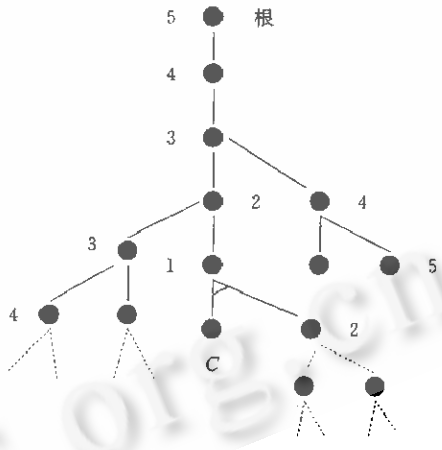

图1　在搜索树中剪枝的效果

局部搜索本身可以采取多分支跳步搜索,将搜索集中在最有希望的若干分支上,使计算资源得到有效的利用.先在浅层作搜索,找出若干较优点,再从这些较优点出发作局部搜索,找出更优状态点.如此重复,直至只剩下一个最优点.这样可以用较少的规划时间生成较长的局部解路径.

3　性能分析

决定搜索算法性能的主要因素是生成状态的数量和需存储的状态数.搜索算法是否能

保证找到解(如果有解)、是否能保证找到最优解,也是衡量性能的标准.

跳步搜索只存储生成的局部目标,总体搜索过程是直接目标导向,不考虑从初始状态到各候选状态的距离,不具有 A* 的整体上宽度优先特性.这一点利弊兼备.由于非宽度优先,LS 不保证找到最优解.另一方面,也使存储空间只需与规划/执行循环数(或最终解长度)成线性关系.对大规模问题,由于 A* 等有宽度优先特性的算法受限于存储空间而难以应用,LS 的这一缺陷显得并不严重.按 H. Simon 的有限合理性原理,实际系统只能追求近似最优而难以达到绝对最优.对实时应用的要求也并非追求最短执行时间,而是在时间限度之内作出反应.所以,跳步搜索的局部求优搜索策略是合理的.

如果要保证从最开始的初始状态到目标的搜索都一直沿着最优路径进行,那么就只能在生成整个路径并判定它为最优后才能去执行操作,这与实时环境所要求的实时响应特性是不相符合的.所以实时算法只能在不违反实时要求的前提下尽量提高决策的准确性,使最终求解代价与最优解尽量接近.这也就是跳步搜索中要将局部搜索的范围作为环境实时特性的函数的原因,以求在满足环境变化要求的前提下对解进行优化.

启发式搜索质量与启发式函数 $h(S)$ 精确程度相关.当 $h(S)$ 精确代表从 S 到目标的距离时,跳步搜索可以找到最优解.当 $h(S)$ 完全不能提供任何启发信息时(例如对所有状态都有 $h(s)=0$),跳步搜索变为受限深度搜索的多次反复,在无限状态空间中无法保证获得解,而在有限状态空间中仍可以获得解.我们可以用以下命题来说明这一点.

命题 1. 在有限状态空间中,若目标从任一状态皆可到达,则跳步搜索一定可以找到解.

证明:在有限状态空间中,只要搜索过程不陷入循环或死角,那么经过有限步搜索之后总可以穷尽一切状态而达到目标.跳步搜索中将所有曾经作为局部目标的状态排除在新候选目标之外,从而避免了循环.而对于启发式函数的"洼地",跳步搜索在其中摆动有限次之后,必定将此范围内的所有状态都作为局部目标状态生成过,再由算法步骤 2,可知搜索过程必将跳出此范围,而不会在其中无限摆动下去.由此 2 点,命题得证. □

当 $h(s)$ 可以提供部分启发信息时,即使在无限状态空间中,跳步搜索也可以保证获得解.有以下命题:

命题 2. 若 $h(s)$ 是可采用的,即 $h(s)\leqslant h^*(s)$,且对所有状态 s,存在启发式函数误差的上界,即 $\mathrm{Max}(h^*(s)-h(s))=E$ 是常数,若将算法步骤 3 中的常数 R 设为 E,则不论问题状态空间是否无限,只要 $h^*(C)$ 有限,即存在从初始状态到目标的有限长度解路径,则跳步搜索一定可以在有限次搜索后达到目标.

为证明此命题,要先证明命题 3.

命题 3. 在命题 2 的条件下,从当前状态 C 出发,经过有限次搜索后一定可以到达一个比当前状态更优的状态 S,即 $h^*(S)<h^*(C)$.

证明:因为 $h(s)\leqslant h^*(s)$,所以对于处在从 C 到目标 G 的最优路径 $P(C,G)$ 上的所有状态 s,$h(s)\leqslant h^*(s)<h^*(C)$.

在这条最优路径上,若某个状态 s' 成为局部最优而被选中,那么命题得证.

否则,必有某个状态 s' 处于待扩展表 ES 中,且从 s' 到目标 G 之间的所有状态都不在已生成状态表 GS 中.如若不然,假设有 $s''\in GS$,且 $s''\in P(C,G)$ 是离 G 最近的已生成状态,则根据算法步骤 3,s'' 通往 G 方向的下一状态 s''',要么在 GS 中,要么在 ES 中.因已假设 s'' 是离

G最近的已生成状态,故s'''只能在ES中.那么s'''就是所求的s'.有$h(s')<h^*(C)$.

因为$h(s)\leqslant h^*(s)$,且$\mathrm{Max}(h^*(s)-h(s))=E$,所以$h(s)\leqslant F$的状态集合最大为$\{s|h^*(s)\leqslant F+E\}$,而这个集合是有限的.

在搜索过程中,若将s'从ES中删除,则此时新的当前状态C'应有$h(C')<h(s')-E$,又由$\mathrm{Max}(h^*(s)-h(s))=E$,有$h^*(C')\leqslant h(C')+E<h(s')\leqslant h^*(s')\leqslant h^*(C)$,故$C'$是优于$C$的新状态,命题得证.

若在搜索中一直没有找到这样的C',则至多在搜索完整个$\{s|h(s)\leqslant h(s')\}$集合后,将以$s'$作为跳步的出发点.按照上面的结论,这个集合的元素数目有限,故从当前状态开始搜索,经过有限次搜索后必定可以将s'作为局部最优进行扩展.而s'是优于C的新状态,命题得证.□

下面来证明命题2.

证明:从初始状态C出发,经过有限次搜索后必定到达较优的状态.重复此过程,则经过至多$h^*(C)$次迭代后可以到达目标,而每次迭代的搜索次数亦为有限,总的搜索次数显然是有限的.命题得证.□

根据以上类似的思路,可以提出命题4.

命题4. 若对所有状态s,存在启发式函数误差的上界,即$\mathrm{Max}(|h^*(s)-h(s)|)=E$是常数,若将算法步骤3中的常数$R$设为$2E$,则不论问题状态空间是否无限,只要$h^*(C)$有限,即存在从初始状态到目标的有限长度解路径,则跳步搜索一定可以在有限次搜索后达到目标.

根据以上命题,可知由于设置了待扩展状态表,使搜索能够最终导向目标,避免了"越界"和"死角"等情形对问题求解的障碍.

跳步搜索每个循环所生成的状态数是由D确定的常数,总状态数是此常数乘以达到解所需循环数.因为一次循环可以生成多个动作,使循环数小于最终的解路径长度.用一个简化的表达式表示为$S(D)=E(D)*P(D)/L(D)$,其中$E(D)$为每次规划生成状态数,$P(D)$为最终解路径长度,$L(D)$为每次生成的局部解长度.考虑$S(D)$随D的变化情形:$E(D)$与D基本成指数关系;当D增大时,由于局部规划更优,$P(D)$总值亦应减小(可参照下面实验结果);$L(D)$随D增大而增大.综合可得$P(D)/L(D)\propto 1/D^a$,$E(D)\propto b^D$.于是

$$S(D)=k\times b^D/D^a \tag{1}$$

b为搜索树分支系数,k,a为由问题类型而定的常数.对于8数码问题,$b=1.732$.[2]

待扩展状态表和已生成状态表中需要存储的状态总数至多为$b\times P(D)/L(D)$,与最终解路径长度之间成多项式关系,而非指数关系,并且随D的增大而减少.

我们在Intel Pentium微机上针对8数码问题实现了跳步搜索算法,作为对照,还实现了RTA*算法.为比较算法的一般性能,我们对每种条件组合下的实验运行了500个随机生成的实例,实例是从目标状态随机走50步之后到达的状态.对500次运行的结果的平均值来进行分析.

按图2中LS生成结点数拟合(1)式,得到$k=234.7$,$a=1.0$.由图2看到,理论值与实际生成值是较接近的.从生成状态数看,LS优于RTA*,这是因为在$S(D)$式中有$1/D$项.

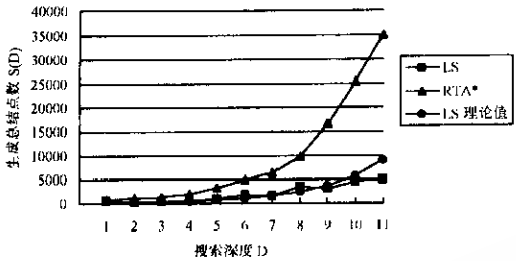

图 2 生成总状态数与搜索深度的关系图

从实验结果看出,LS 算法比 RTA * 算法的效率提高了数倍之多. 如果它们生成解的长度相同,由于 LS 算法经过若干步才进行一次局部搜索,比起每执行一步都要进行局部搜索的 RTA *,总的生成状态数要少若干倍. 从 2 种算法生成路径长度的实验结果(见图 3)来看,RTA * 和 LS 所生成的解长度基本相当. RTA * 对局部最优点采取最小承诺策略,每次只执行通往最优方向的一步,然后对局势再进行评估. 然而当评估函数不完全精确时,评估次数增多会增大引入错误的可能性,使求解偏离正确路径,增加总的求解路径长度. LS 采取最大承诺,直接转到局部最优点,带有一定风险,但下一步规划中可以在较大范围内寻找最优状态,大范围的搜索可以重新找到正确的路径. RTA * 每次只执行一步,在求解过程中发生振荡的概率会比 LS 大.

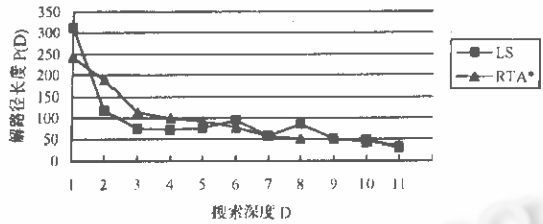

图 3 最终解路径长度与搜索深度的关系

4 结 语

对处理动态问题的搜索算法的研究从 RTA * 算法开始. RTA * 中将搜索划分为规划与执行阶段,采取遗忘部分历史的方法来克服 A * 的空间缺陷. 但它每次执行只限于一个动作,效率比较低,而且也未明确提出对动态目标的处理.

文献[3]提出对动态移动目标的搜索 MTS(moving target search). MTS 算法是对自学习 RTA * (LRTA *)的扩展,可以学习搜索空间中任意 2 点间的精确距离. MTS 在文献[4]中被扩展为 IMTS(intelligent moving target search),在规划阶段执行前探搜索(Lookahead),在前探的计算代价和到达目标的执行代价之间进行平衡. 对移动目标的搜索算法有轨迹搜索算法[5],记忆搜索主体已搜索过的状态空间及从这个空间中任一状态到当前状态的最短路径值,如果目标到达已搜索空间中的任意点,就利用这些信息来决定搜索方向. 这几种搜索算法着眼于通过保存运行中获得的信息,以便更精确地对再次出现的状态进行处理. 跳步搜索则着眼于如何控制规划与执行阶段的时间比例,以满足实时要求.

与爬山法相比,跳步搜索不局限于只在相邻状态中寻找更优状态,可以更快地达到目

标,不容易陷入局部极值的陷阱中.与 RTA * 相比,它每一次规划生成不止一个动作,而是多个行动组成的序列,具有更强的实时性,可以在一个执行阶段中完成更多的任务.这不仅有益于满足需要立即采取行动的实时应用,也可以用更少的执行环节迅速达到目标.与传统搜索方法相比,由于它在每个规划阶段中可以采用不同的搜索目标,因此除开快速的特点外,还具有实时响应特性,可以适应目标不断变化的应用环境,如交通控制、战场战术规划与控制等.

实时搜索算法如何学习环境的变化规律、如何预测环境状态随动作序列进行而发生的变化,对提高系统求解能力具有重要意义.对动态开放环境的定量建模也是今后进一步工作的方向.

参考文献

1 Korf R E. Search. Encyclopedia of artificial intelligence, 2nd ed. 1992. 1460～1467.

2 Korf R E. Real-time heuristic search. Artificial Intelligence, 1990, **42**:189～211.

3 Ishida T, Korf R E. Moving target search. In: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 1991. 204～210.

4 Ishida T. Moving target search with intelligence. In: Proceedings of the Tenth National Conference on Artificial Intelligence, 1992. 525～532.

5 Chimura F, Tokoro M. The trailblazer search: a new method for searching and capturing moving targets. In: Proceedings of the Eleventh National Conference on Artificial Intelligence, 1994. 1347～1352.

A LEAP REAL TIME SEARCH ALGORITHM IN DYNAMIC AND OPEN ENVIRONMENTS

LUO Yi　HU Peng　SHI Chunyi　WANG Kehong

(Department of Computer Science　Tsinghua University　Beijing　100084)

Abstract　The dynamic and open environment has its own unique characteristics, and the search algorithms that want to survive in this environment and handle the real time problems must deal with these characteristics. What is the requirement to such search algorithms? This paper gives out some hints to the answer. It describes a search algorithm suited for dynamic environment, leap algorithm, which has realized some ideas in the analysis. The algorithm can run more efficiently than a famous real time algorithm RTA *, and the experiment data support this point.

Key words　Search algorithm, real time algorithm, dynamic and open environment, local search.