ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.9, September 2009, pp.2352−2365
doi: 10.3724/SP.J.1001.2009.03563
© by *Institute of Software*, *the Chinese Academy of Sciences*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 BDD 的增量启发式搜索*

徐艳艳[1,2], 岳伟亚[3+]

[1](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

[2](中国科学院 研究生院 信息科学工程学院,北京 100190)

[3](Department of Computer Science, College of Engineering, University of Cincinnati, Cincinnati 45219, USA)

BDD-Based Incremental Heuristic Search

XU Yan-Yan[1,2], YUE Wei-Ya[3+]

[1](State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](School of Information Science and Engineering, Graduate University, The Chinese Academy of Sciences, Beijing 100190, China)

[3](Department of Computer Science, College of Engineering, University of Cincinnati, Cincinnati 45219, USA)

+ Corresponding author: E-mail: weiyayue@hotmail.com

Xu YY, Yue WY. BDD-Based incremental heuristic search. *Journal of Software*, 2009,20(9):2352−2365.
http://www.jos.org.cn/1000-9825/3563.htm

Abstract: Incremental search reuses information from previous searches to find solutions to a series of similar search problems. It is potentially faster than solving each search problem from scratch. This is very important because many artificial intelligence systems have to adapt their plans continuously to changes in the world. If the changes are small, incremental search will be very efficient. BDD (binary decision diagram)-Based heuristic search combines the advantages of BDD-based search and heuristic search. Heuristic search impacts the size of the resulting search trees and BDDs can be used to efficiently describe the sets of states based on their binary encodings. This article first introduces BDD-based heuristic search and incremental search. Combining the two methods, it then gives a BDD-based incremental heuristic search algorithm BDDRPA*. The experimental results show that BDDRPA* is a very efficient incremental heuristic search algorithm. It can be used to solve many problems like symbolic replanning and robot navigation problems and so on.

Key words: incremental search; heuristic search; BDD (binary decision diagram); replanning

摘 要: 增量搜索是一种利用先前的搜索信息提高本次搜索效率的方法,通常可以用来解决动态环境下的重规划问题.在人工智能领域,一些实时系统常常需要根据外界环境的变化不断修正自身,这样就会产生一系列变化较小的相似问题,此时应用增量搜索将会非常有效.另外,基于BDD(binary decision diagram)的启发式搜索,结合了基于BDD的搜索和启发式搜索这两种方法的优点.它既用 BDD 这一紧凑的数据结构来表示系统的状态空间,又通过使用启发信息来进一步压缩搜索树的大小.在介绍基于 BDD 的启发式搜索和增量搜索之后,结合这两种方法给出了基于BDD 的增量启发式搜索算法——BDDRPA*.大量的实验结果表明,BDDRPA*算法是非常有效的,它可以被广泛地

* Supported by the National Natural Science Foundation of China under Grant Nos.60833001, 60721061, 60725207 (国家自然科学基金); the National Basic Research Program of China under Grant No.2010CB328103 (国家重点基础研究发展计划(973))

Received 2008-07-19; Accepted 2009-01-15

© 中国科学院软件研究所 http://www.jos.org.cn

应用到智能规划、移动机器人问题等领域中.

关键词: 增量搜索;启发式搜索;BDD(binary decision diagram);重规划
中图法分类号: TP181 文献标识码: A

在人工智能领域的许多动态重规划问题中,当发现真实情形与原先设想的情形不同或者系统的状态格局随时间变化不断发生改变时,我们需要对新的情形重新规划[1].一般的处理方法是对新的格局再次进行彻底的搜索,从而得到新的规划继续执行;然而我们发现,在很多情况下,每次发生的改变都只是一小部分,因此再进行一次彻底的搜索就显得冗余且效率低下,好的处理方法是利用先前的搜索信息来提高本次搜索的效率,这就是增量搜索(incremental search 也称为渐进搜索,)[2]的基本思想.在增量搜索的基础上加入启发信息[3,4]来选择下一个扩展节点的搜索方法就是增量启发式搜索[5].

从 1980 年代以来,在算法文献[6−9]以及人工智能文献[10,11]中提出了许多增量搜索算法.这些算法的不同之处主要是解决单对最短路径问题还是多对最短路径问题、使用的解决方案有何区别、图的拓扑结构变化和边的权重变化的约束有何不同等等.对图的拓扑结构和边的权重变化不加任何约束的问题称为完全动态最短路径问题[12].主要的增量搜索算法有 LPA*算法[13]以及在它的基础上发展的 D* Lite 算法[14]和 DD* Lite 算法[15]等.LPA*算法的实质是在 DynamicSWSF-FP 算法[16]的基础上加入了 A*算法的思想,D* Lite 则是结合了 D*[17]和 LPA*算法的优势.

在过去的十几年中,基于 BDD(binary decision diagram)[18]强大的搜索技术在符号化模型检测领域[19]得到了很好的应用.鉴于 BDD 表达紧凑性以及化简后唯一性的优点,利用它可以在一定程度上缓解模型检测的状态爆炸问题,从而使得被检测的状态个数大为增加.在人工智能领域的搜索问题中,随着搜索深度的增加,状态空间也呈指数爆炸,经典的避免状态爆炸的方法是使用启发函数来引导搜索路径至目标节点.但是 A*算法需要保存所有已经扩展的节点,因此,搜索过程往往会因为内存被耗完而提前结束.如果能把 BDD 这种紧凑的数据结构应用到 A*算法中,将会压缩搜索空间从而提高算法的处理规模.到目前为止,基于 BDD 的搜索算法有 BDDA*算法[20]和 SetA*算法[21,22].

文献[23]提出了基于 BDD 的增量搜索算法——BDDRPA*,它综合了基于 BDD 的搜索和增量搜索这两种方法的优点.它既用 BDD 作为数据结构以提高搜索的空间效率,又结合了增量搜索的思想来提高重搜索的效率,文中通过大量实验证明了当搜索问题的规模越来越大时,BDDRPA*将会非常有效.但是,文献[23]并没有考虑搜索问题的边的权重不断变化的情况,且启发函数 h 被设为 0,这样,BDDRPA*算法就退化为基于 BDD 的宽度优先增量搜索算法.因此,本文扩充了 BDDRPA*算法的功能,添加了权重不断发生改变的情况,且使用了有效的启发函数,使得 BDDRPA*可以利用启发信息来进一步压缩问题的搜索空间.

本文第 1 节介绍基于 BDD 的启发式搜索方法,即如何把 BDD 这种数据结构引入到启发式搜索方法中.第 2 节介绍基于 BDD 的增量搜索方法,即如何用搜索空间的旧迁移关系 T 构造新迁移关系 T'.第 3 节讨论 BDDRPA*算法的运行过程.第 4 节对算法进行分析.第 5 节通过大量实验比较,证明 BDDRPA*算法的高效性.第 6 节总结全文并说明下一步要做的工作.

1 基于 BDD 的启发式搜索

OBDDs(ordered binary decision diagrams)[18]是一种布尔公式的标准表示方式.这种表示方式通常比合取范式或析取范式更紧凑,具有化简后唯一性这个重要的特点,而且有非常有效的方法对它们进行操作(apply function).由于 OBDD 可以很紧凑地描述由电路、协议或控制系统所决定的状态空间(Kripke structure),所以符号化模型检测能够验证非常大的系统[19].本节将介绍如何用 OBDD 描述一般的启发式图搜索问题.

定义 1(基于 BDD 的搜索问题). 基于 BDD 的搜索问题可以用一个四元组$\langle S,T,i,G\rangle$来表示,其中:

- S 表示搜索空间所有状态的集合;
- $T\subseteq S\times N^{+}\times S$ 定义了搜索空间所有状态之间的迁移关系,其中$(s,c,s')\in T$ 当且仅当从 s 到 s'存在一个权重值

© 中国科学院软件研究所 http://www.jos.org.cn

为 c 的迁移($c\in N^+$,N^+表示正整数集合,c 的取值可以扩展到小数);

- i 表示初始状态;
- G 表示目标状态集合.

搜索问题的一个解就是一条路径$\pi=s_0,\ldots,s_n$,其中 $s_0=i,s_n\in G$ 并且当 $0\le i\le n-1$ 时,$(s_i,c(s_i,s_{i+1}),s_{i+1})\in T$.搜索问题的最优解即最小代价路径解是所有解中$(c(s_0,s_1)+\ldots+c(s_i,s_{i+1})+\ldots+c(s_{n-1},s_n))$最小的解,其中 $c(s_i,s_{i+1})$表示从 s_i 到 s_{i+1} 的权重为 c.

为了用 BDD 表示搜索问题,先介绍如何用 BDD 表示有限域上的关系.设 Q 是一个{0,1}上的 n 元关系,那么 Q 的特征函数$\Phi_Q(x_1,\ldots,x_n)=1$ iff $Q(x_1,\ldots,x_n)$可以用一个 BDD 来表示.若 Q 是一个有限域 D 上的 n 元关系,则为了用一个 BDD 表示 Q,要先对 D 中的元素进行编码.假设 D 有 x 个元素($2^{m-1}<x\le 2^m$,$m>1$),那么需要 m 个布尔变量通过双射 Ψ:$\{0,1\}^m\to D$ 对 D 中的元素进行编码.通过编码 Ψ,就构造了一个 $m\times n$ 的布尔关系 Q':$Q'(x_1,\ldots,x_n)=Q(\Psi(x_1),\ldots,\Psi(x_n))$,其中 $x_1,\ldots,x_n$ 用 m 个布尔变量编码.这样,Q 就可以通过 Q'的特征函数$\Phi_{Q'}$用一个 BDD 表示.因为集合可以被看成一元关系,所以集合也可以用 BDD 表示.

现在用 BDD 表示搜索问题$\langle S,T,i,G\rangle$.S 是一个状态的集合,假设它有 2^m 个元素,通过 Ψ:$\{0,1\}^m\to S$ 把 S 中的每个元素用布尔变量编码,S 就可以通过它的特征函数Φ_S用一个 BDD 来表示.对于迁移关系 T,除了用与用来编码 S 的相同的一组布尔变量之外,还需要两组新的布尔变量,一组用来表示 s 的下一个状态 s',另一组用来表示权重值 c.通过这 3 组布尔变量 $\boldsymbol{s},\boldsymbol{s'}$和 $\boldsymbol{c}$**,T 可以被编码为一个布尔关系 $T(\boldsymbol{s},\boldsymbol{c},\boldsymbol{s'})$,这样,它就可以通过特征函数$\Phi_T$用一个 BDD 来表示.

在启发式搜索中,还需要用 BDD 表示启发函数 h 和 open 表.启发函数 h 可以被看成一个二元关系 $h(s,h\text{-value})$,即状态 s 的启发函数值为 h-value,它可以通过特征函数Φ_h用一个 BDD 表示.Open 表实质上是状态以及它们所对应的 f 值的集合,可以用二元关系$(s,f\text{-value})$来表示,即状态 s 的 f 值是 f-value.Open 表在搜索过程中不断改变,通过 $ExistAnd()$操作$(\exists\boldsymbol{x}.T(\boldsymbol{x},\boldsymbol{c},\boldsymbol{x'})\wedge\Phi_S^i(\boldsymbol{x}))[\boldsymbol{x'}/\boldsymbol{x}](\boldsymbol{x},\boldsymbol{x'}$和 $\boldsymbol{c}$ 各表示一组布尔变量)再抽象去 $\boldsymbol{c}$ 得到Φ_S^{i+1},这样一步步扩展状态集合Φ_S^i得到新扩展的状态集合Φ_S^{i+1}.在计算新扩展的状态的 f 值时,如果用 BDD 作算术运算,那么既有局限性又慢,所以本文采用 BDD 的分裂和合并来计算 f 值.从 open 表中取出 f 值最小的状态 s,同时得到它的 f 值 $f(s)$.扩展 s 得到新的状态集合 S',对 S'中的每一个状态 s',通过 $ExistAnd(\mathrm{BDD}_{s'},\mathrm{BDD}_h)$操作得到 s'的启发函数值 $h(s')$,通过 $ExistAnd(\mathrm{BDD}_s,\mathrm{BDD}_h)$得到 $h(s)$,再从 BDD_T 中得到 $c(s,s')$,然后通过数学运算 $f(s')=f(s)+c(s,s')-h(s)+h(s')$计算出 $f(s')$.构造 BDD_f,把 $\mathrm{BDD}_{s'}$和 BDD_f进行∧操作得到 $\mathrm{BDD}_{f\wedge s'}$,最后将其与 open 表进行∨即得到插入 s'后新的 open 表***.

下面用一个例子(如图 1 所示)来说明如何编码,如何用 BDD 表示 h,T 和一步步扩展的 open 表以及如何进行启发式搜索.图 1 是一个有向图,边的权重以及每个节点的启发函数值已在图中标明.图中一共有 6 个节点,也就是 6 个状态,因此需要 3 个布尔变量($x_0x_1x_2$).这 6 个状态的编码分别是:$s_0=000=\neg x_0\wedge\neg x_1\wedge\neg x_2$,$s_1=001=\neg x_0\wedge\neg x_1\wedge x_2$,$s_2=010=\neg x_0\wedge x_1\wedge\neg x_2$,$s_3=011=\neg x_0\wedge x_1\wedge x_2$,$s_4=100=x_0\wedge\neg x_1\wedge\neg x_2$ 和 $s_5=101=x_0\wedge\neg x_1\wedge x_2$.$s_0$ 是初始节点 i,s_5 是目标节点 G,我们的目的是找一条从 s_0~s_5 的最短路径.根据图中标的启发函数值以及权重值,可以算出最小的 h 值为 0,最大的 f 值为 6,所以一共需要 3 个布尔变量($c_0c_1c_2$)来编码权重值 c,h 值以及 f 值(这 3 组值使用同一组布尔变量).根据这种编码方法,得到启发函数 h 和迁移关系 T 对应的两个 BDDs,如图 2 所示****.

开始搜索时,open 表中只有一个初始节点 s_0,它的 h 值是 3,因此它的 f 值也是 3.扩展 s_0,得到 s_1 和 s_3,它们的 f 值都是 3,因此一起扩展.扩展 s_1 和 s_3,得到 s_2 和 s_4,s_2 的 f 值是 3,s_4 的 f 值是 4.扩展 s_2,得到 s_5.此时,open 表中有两个节点 s_4 和 s_5,$(f(s_4)=4)<(f(s_5)=6)$,因此扩展 s_4,得到 f 值等于 5 的新的 s_5.这时,open 表中 f 值最小的节点是 s_5,它是目标节点,因此搜索过程结束,得到一条代价为 5 的最短路径 $s_0\to s_3\to s_4\to s_5$.整个搜索过程一步步得到的

** 本文在使用黑体 $\boldsymbol{s},\boldsymbol{s'},\boldsymbol{c},\boldsymbol{x}$ 和 $\boldsymbol{x'}$时,表示它们代表的是一组布尔向量.

*** 本文的∧∨¬都是指在 OBDD 上的布尔运算.

**** 本文中出现的 BDDs 都是一种简写形式,只画出了到 1 的路径,到 0 的路径省略.这样做可以使这些图看起来更清晰、更直接,阅读起来也更方便.在图中,虚线表示变量取 0 值,实线表示变量取 1 值.

© 中国科学院软件研究所 http://www.jos.org.cn

open 表如图 3 所示.

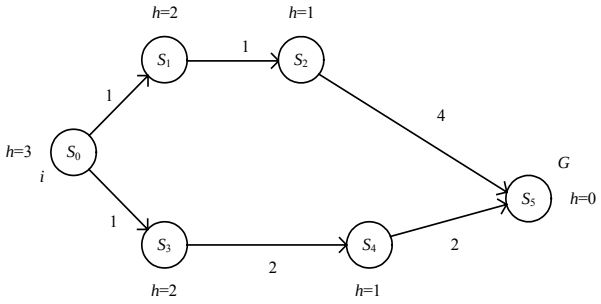

Fig.1 A simple example

图 1 一个简单示例

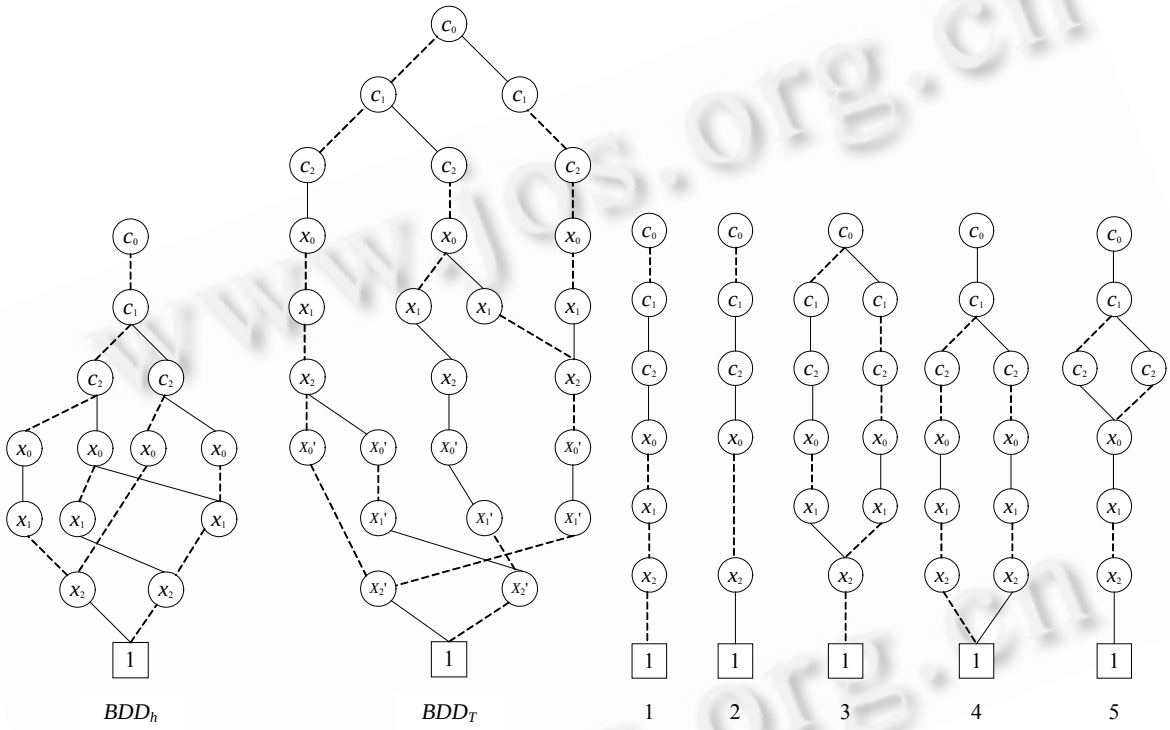

Fig.2 BDD_h and BDD_T

图 2 BDD_h 和 BDD_T

Fig.3 BDDs for open

图 3 open 表(BDD 表示)

2 基于 BDD 的增量搜索

在重规划的过程中,系统的状态格局会随着时间不断发生改变.如果每次格局改变后都不利用先前的信息而完全重新搜索,效率就会非常低下.增量搜索是一种利用之前的搜索信息来提高本次搜索效率的非常有效的搜索方法.基于 BDD 的增量搜索思想主要体现在系统格局发生变化后重新构造新格局的迁移关系的 BDD 上,即如何利用旧格局的迁移关系 BDD_T 构造新格局的迁移关系 $BDD_{T'}$.

给定一个搜索问题$\langle S,T,i,G\rangle$,基于 BDD 的增量启发式算法 BDDRPA*先用第 1 节介绍的基于 BDD 的启发式搜索方法搜索出一条从 s_i 到 s_G 的最短路径;接着,问题的状态格局发生改变,算法将利用本节介绍的方法根据

© 中国科学院软件研究所 http://www.jos.org.cn

旧格局的迁移关系 BDD_T 构造出新格局的迁移关系 $BDD_{T'}$,然后再用启发式方法搜索路径,如此一直反复下去.首先给出两条引理,然后根据这两条引理给出 3 条定理.根据这 3 条定理,就可以利用旧格局的迁移关系 BDD_T 构造出新格局的迁移关系 $BDD_{T'}$.

引理 1. 给定一个 OBDD T,如果从 T 上删除一条用 cube c 表示的路径 p,那么可以通过运算 $T\wedge\neg c$ 得到删除路径 p 后的新 OBDD T'.*****

证明:首先,我们知道 T 是若干条路径的集合,即 $T=c_1\vee c_2\vee\ldots\vee c_n$.令 $C=\{c_i|1\le i\le n\}$,则删除路径 p 可以分以下两种情况讨论:

1) 若 T 中没有路径 p,即 c 不属于 C,则 $T'=T=T\wedge\neg c$;

2) 若有 $c_i(1\le i\le n)$满足 $c_i=c$,则 $T'=T\wedge\neg c=c_1\vee c_2\vee\ldots\vee c_{i-1}\vee c_{i+1}\vee\ldots\vee c_n$,即 p 已被删去,而其他路径不受影响.

综上,通过 $T\wedge\neg c$ 操作确实可以得到删除路径 p 后的 OBDD T'. □

引理 2. 给定一个 OBDD T,如果往 T 中添加一条用 cube c 表示的路径 p,那么可以通过运算 $T\vee c$ 得到添加路径 p 后的新 OBDD T'.

证明:与引理 1 一样,T 是若干条路径的集合,即 $T=c_1\vee c_2\vee\ldots\vee c_n$.令 $C=\{c_i|1\le i\le n\}$,则增加路径 p 同样可以分下面两种情况讨论:

1) 若 T 中已有路径 p,即 $c\in C$,则 $T'=T=T\vee c$;

2) 若 T 中没有路径 p,则 $T'=T\vee c=c_1\vee c_2\vee\ldots\vee c_n\vee c_{n+1}$,其中 $c_{n+1}=c$,即 p 已被加上,而其他路径不受影响.

综上,通过 $T\vee c$ 操作确实可以得到加上路径 p 后的 OBDD T'. □

定理 1. 设搜索空间图 G 的迁移关系的 OBDD 的表示为 BDD_T,将 G 上的一个节点 s 及其相应的边删除后得到的新的图 G'的迁移关系的 OBDD 表示设为 $BDD_{T'}$,那么 $BDD_{T'}$ 可以由 BDD_T 通过以下两步得到:

1) 将 BDD_T 与被删除节点 s 的 OBDD BDD_s 的“¬”作∧运算得到 BDD_O,即 $BDD_O=BDD_T\wedge\neg BDD_s$;

2) 将 BDD_O 与被删除节点 s 的后继表示形式 s'的 OBDD $BDD_{s'}$的“¬”作∧运算就得到 $BDD_{T'}$,即 $BDD_{T'}=BDD_O\wedge\neg BDD_{s'}$.

证明:s 被删除意味着从 s 不能到达其他任何节点并且其他任何节点也不能到达 s,图 G 上的其他与 s 不相邻的节点将不受任何影响,这就是 $BDD_{T'}$ 的新性质.

1) 将 BDD_T 与 BDD_s 的“¬”作∧运算得到 BDD_O,则由引理 1 可知,在 BDD_O 中,s 将不能到达其他任何节点;

2) 将 BDD_O 与 $BDD_{s'}$的“¬”作∧运算得到 $BDD_{T'}$,则由引理 1 可知,任意节点将不能再到达 s.

图 G 上的其他与 s 不相邻的节点不受这两步操作的影响,也就是说,通过以上两步操作,得到的确实是 s 被删除后的图 G'的迁移关系 $BDD_{T'}$. □

定理 2. 设搜索空间图 G 的迁移关系的 OBDD 的表示为 BDD_T,将往 G 上添加一个节点 s 以及与它相邻的边后得到的图 G'的迁移关系的 OBDD 表示设为 $BDD_{T'}$,那么 $BDD_{T'}$ 可以由 BDD_T 通过以下两步得到:

1) 将 BDD_T 与 s 和 $succ(s)$(s 的后继节点集合)之间的迁移关系的 OBDD 表示 $BDD_{s\to succ(s)}$作∨运算得到 BDD_O,即 $BDD_O=BDD_T\vee BDD_{s\to succ(s)}$;

2) 将 BDD_O 与 $pred(s)$(s 的前驱节点集合)到 s 之间的迁移关系的 OBDD 表示 $BDD_{pred(s)\to s}$ 作∨运算得到 $BDD_{T'}$,即 $BDD_{T'}=BDD_O\vee BDD_{pred(s)\to s}$.

证明:s 以及它的相邻边被添加到 G,则它可以到达它的后继节点并且它的前驱节点也能到达它,图 G'上的其他与 s 不相邻的节点将不受任何影响.

1) 将 BDD_T 与 $BDD_{s\to succ(s)}$作∨运算得到 BDD_O,由引理 2 可知,在 BDD_O 中,s 能够到达它的后继节点;

2) 将 BDD_O 与 $BDD_{pred(s)\to s}$ 作∨运算得到 $BDD_{T'}$,由引理 2 可知,在 $BDD_{T'}$ 中,s 的前驱节点能够到达 s.

图 G 上的其他与 s 不相邻的节点不受这两步操作的影响,也就是说,通过以上两步操作,得到的确实是添加

***** 本文中,我们把有且只有一条路径通向真值 1 的 BDD 称为 cube;而一条由几个布尔变量取特定值构成的路径 p 显然可以用一个 cube 表示.

© 中国科学院软件研究所 http://www.jos.org.cn

s 后的图 G'的迁移关系 $BDD_{T'}$. □

定理 3. 设搜索空间图 G 的迁移关系的 OBDD 的表示为 BDD_T,将 G 上的一条边的权重发生变化后(比如,$(s,c,s')\rightarrow(s,c',s')$)得到的图 G'的迁移关系的 OBDD 表示设为 $BDD_{T'}$,那么 $BDD_{T'}$可以由 BDD_T通过以下两步得到:

1) 将 BDD_T 与权重发生变化前的边的 OBDD 表示 $BDD_{(s,c,s')}$的"¬"作∧运算得到 BDD_O,即 $BDD_O=BDD_T\wedge\neg BDD_{(s,c,s')}$;

2) 将 BDD_O 与权重发生变化后的边的 OBDD 表示 $BDD_{(s,c',s')}$作∨运算得到 $BDD_{T'}$,即 $BDD_{T'}=BDD_O\vee BDD_{(s,c',s')}$.

证明:图 G 上的一条边 $s\rightarrow s'$的权重由 c 变成 c',则我们需要从图 G 的迁移关系 T 的 BDD 表示中删除旧的迁移(s,c,s')并添加新的迁移(s,c',s'),图 G'上的其他边将不受此变化的影响.

1) 将 BDD_T与 $BDD_{(s,c,s')}$的"¬"作∧运算得到 BDD_O,由引理 1 可知,在 BDD_O 中,我们已经删除了带旧的权重的边(s,c,s').

2) 将 BDD_O 与 $BDD_{(s,c,s')}$作∨运算得到 $BDD_{T'}$,由引理 2 可知,在 $BDD_{T'}$中,我们已经添加了带新的权重的边(s,c',s').

图 G'上的其他边不受这两步操作的影响,也就是说,通过以上两步操作,我们确实得到了边 $s\rightarrow s'$的权重由 c 变成 c'后的图 G'的迁移关系 $BDD_{T'}$. □

下面举例说明(如图 4 所示)如何根据旧迁移关系的 BDD 构造出新迁移关系的 BDD.初始的格局如图 4 中的左图所示,设它的迁移关系的 BDD 表示为 BDD_T;然后删除节点 s_4 并把边 $s_0\rightarrow s_3$ 的权重由 1 改为 2 从而得到图 4 中间的图,设它的迁移关系的 BDD 表示为 $BDD_{T'}$;最后,再加上 s_4 以及和它相邻的两条边得到图 4 的右图,设它的迁移关系的 BDD 表示为 $BDD_{T''}$.首先运用定理 1 根据 BDD_T(如图 2 所示)构造 BDD_{temp}(如图 5 所示).进行定理 1 的第一步操作,即通过 $BDD_T\wedge\neg BDD_{s4}$ 得到 BDD_O;再进行定理 1 第 2 步操作,通过 $BDD_O\wedge\neg BDD_{s4'}$得到 BDD_{temp}.接着运用定理 3 根据 BDD_{temp} 构造 $BDD_{T'}$(图 6).先进行定理 3 的第 1 步操作,$BDD_{O'}=BDD_{temp}\wedge\neg BDD_{(s_{0,1},s_3)}$;再进行定理 3 的第 2 步操作得到 $BDD_T=BDD_{O'}\vee BDD_{(s_{0,2},s_3)}$.最后运用定理 2 根据 $BDD_{T'}$构造 $BDD_{T''}$(图 7).进行定理 2 的第 1 步操作,即通过 $BDD_{T'}\vee BDD_{s_4\rightarrow succ(s_4)}$ 得到 $BDD_{O''}$;再进行定理 2 的第 2 步操作,通过 $BDD_{O''}\vee BDD_{pred(s_4)\rightarrow s_4}$ 得到迁移关系 $BDD_{T''}$.到这里为止,就完成了如何随着图格局的变化利用旧迁移关系构造新迁移关系的整个演示流程.

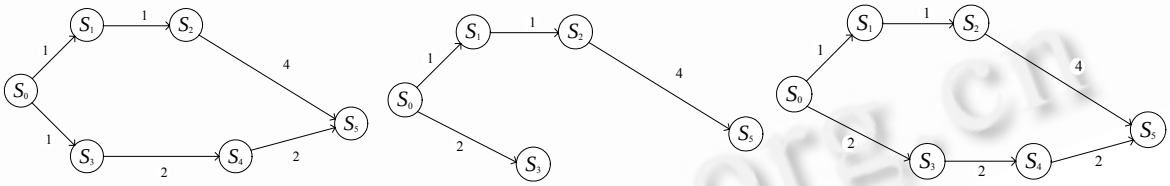

Fig.4 Changed graphs

图 4 逐渐变化的图

3 BDDRPA*算法

在前两节的基础上,本节讨论基于 BDD 的增量启发式搜索算法 BDDRPA*.其伪代码如下所示:

Procedure *Main*()

{01} BDD_i,BDD_G,BDD_T,BDD_h;
{02} BDD_{open},BDD_{closed};
{03} BDD_{trace}; /* BDD used for memorizing the path */
{04} while (*update_graph*){ /* update the topology of the graph */
{05} $BDD_{open}\leftarrow\exists x.(BDD_h\wedge BDD_i)$;
{06} if (update) *Reconstruct_transition*();

© 中国科学院软件研究所 http://www.jos.org.cn

```
        while (1){
{07}        if (BDD_open is empty)    return 0;                          /* no path, exit */
{08}        Choose_shortest_cost(BDD_open) and conjoin BDD_temp and BDD_closed;
{09}        if (∃x.(BDD_temp∧BDD_G)) return 1;                           /* find path, exit */
{10}        update_BDD_open (BDD_temp);                                  /* conjoin BDD_succ and BDD_open */
{11}    }
{12} }
```

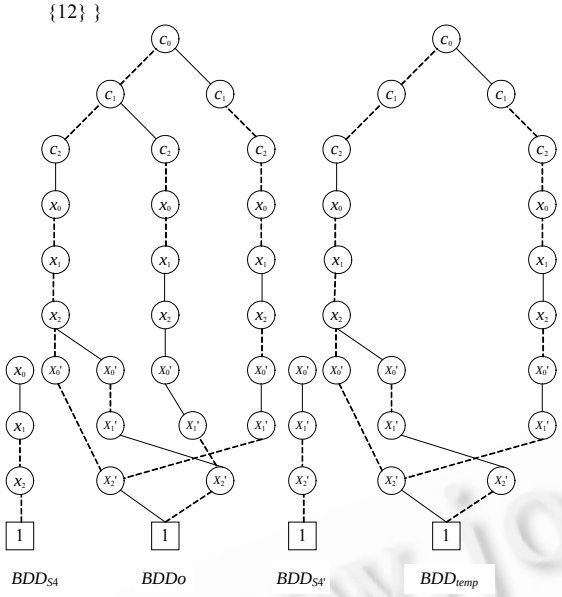

Fig.5 $BDD_T \rightarrow BDD_{temp}$

图 5 $BDD_T \rightarrow BDD_{temp}$

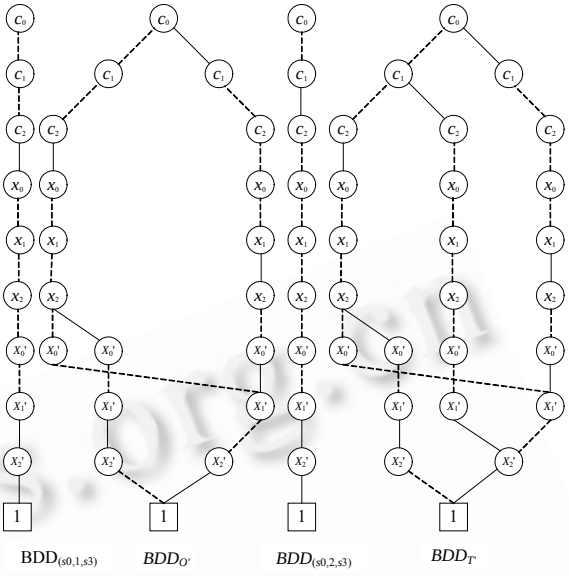

Fig.6 $BDD_{temp} \rightarrow BDD_{T'}$

图 6 $BDD_{temp} \rightarrow BDD_{T'}$

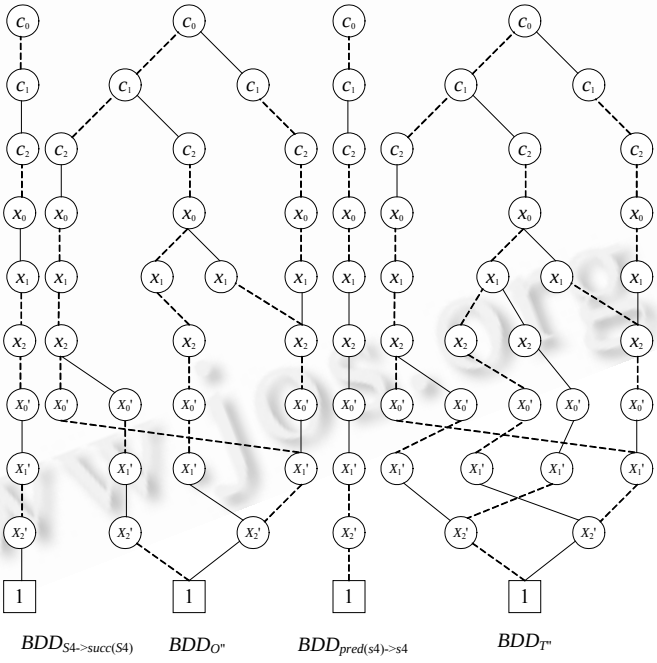

Fig.7 $BDD_{T'} \rightarrow BDD_{T''}$

图 7 $BDD_{T'} \rightarrow BDD_{T''}$

© 中国科学院软件研究所 http://www.jos.org.cn

Procedure *Reconstruct_transition*()

{13} for (every changed state s in G){
{14} if (s is deleted){
{15} $BDD_T \leftarrow BDD_T \wedge \neg BDD_s$;
{16} $BDD_T \leftarrow BDD_T \wedge \neg BDD_{s'}$;
{17} }else{ /* s is added */
{18} $BDD_T \leftarrow BDD_T \vee BDD_{s\to succ(s)}$;
{19} $BDD_T \leftarrow BDD_T \vee BDD_{pred(s)\to s}$;
{20} }
{21} }
{22} for (every changed cost c in G){
{23} $BDD_T \leftarrow BDD_T \wedge \neg BDD_{Tc}$;
{24} $BDD_T \leftarrow BDD_T \vee BDD_{Tc'}$;
{25} }

Procedure *Choose_shortest_cost*(BDD_{open})

{26} ($f_{\min}$,BDD_{temp},BDD_{open})←*dequeue*(BDD_{open})

Procedure *update_BDD*$_{open}$(BDD_{temp})

{27} $BDD_{succ}=\exists x.(BDD_T \wedge BDD_{temp})$; /* expand BDD_{temp} */
{28} for (every successor s in BDD_{succ}){
{29} bool *adjustment*=FALSE;
{30} if (s is in BDD_{closed}){
{31} name s in BDD_{closed} s';
{32} if ($g(s)<g(s')$){
{33} *adjustment*=TRUE;
{34} remove s' from BDD_{closed};
{35} remove s' from BDD_{trace};
{36} }
{37} }else if (s is in BDD_{open}){
{38} name s in BDD_{open} s';
{39} if ($g(s)<g(s')$){
{40} *adjustment*=TRUE;
{41} remove s' from BDD_{open};
{42} remove s' from BDD_{trace};
{43} }
{44} } else { /* it is the first time to reach s */
{45} *adjustment*=TRUE;
{46} }
{47} if(adjustment){
{48} construct $BDD_{f'}$ with f' ; /* $f'=f+c(p(s),s)-h(p(s))+h(s)$ */
{49} $BDD_{open}=BDD_{open}\vee(BDD_s\wedge BDD_{f'})$; /* add s to BDD_{open} */
{50} $BDD_{trace}=BDD_{trace}\vee(BDD_{temp}\wedge BDD_s)$; //used to search back for the path
{51} }
{52} }

在 *main*()中,首先判断图的格局有没有发生改变{05},如果改变{07},重构迁移关系 BDD_T{15-27};BDD_{open} 被初始化为 s_i 和它的 f 值{06};搜索开始,如果 BDD_{open} 为空,表示没有路径,算法失败退出{09};否则,从 BDD_{open} 中选取 f 值最小的节点的集合 BDD_{temp},并把 BDD_{temp} 中的节点放入 BDD_{closed}{10},选择 f 值最小的节点的集合可以通过函数 *dequeue*()来实现{28};判断 BDD_{temp} 是否包含目标节点,如果包含,算法成功退出{11};否则,扩展 BDD_{temp},并把扩展出来的节点插入 BDD_{open}{12}.在扩展 BDD_{temp} 的函数 *update_BDD*$_{open}$()中{29−54},新生成的节点分为 3 种情况:第 1 种已经在 BDD_{open} 中,这时需要对比新旧 g 值的大小,如果新 g 值较小,则删除旧的添加新的;第 2 种,已经在 BDD_{closed} 中,这时也要对比新旧 g 值的大小,如果新的 g 值较小,则把旧的从 BDD_{closed} 中删除,并把新的添加到 BDD_{open} 中;第 3 种是初次生成的节点,这种节点直接添加到 BDD_{open} 中即可.

BDDRPA*算法用 BDD_{trace} 记录搜索树以输出路径.在 BDD_{trace} 中,一个节点可有若干个后续节点,但却只有一个父节点,即用最小 g 值扩展到它的节点.对于要添加到 BDD_{open} 的节点 s,构造一条从 s 的父节点到 s 的路径并把它添加到 BDD_{trace} 中({52}行 BDD_s 中的布尔变量用的是 s 的后继表示形式);如果以后发现有更小 g 值可以扩展到 s,那么就删除旧路径{37,44}添加新路径{52}.搜索完成后,从目标节点 s_G 开始,可以通过 *ExistAnd*()操作一步步从 BDD_{trace} 中得到搜索路径.举例说明:对于 $s_0\to s_1\to s_2$ 这样的一个搜索问题,搜索结束后,得到 $BDD_{trace}=(\neg x_0\wedge\neg x_1\wedge\neg x_0'\wedge x_1')\vee(\neg x_0\wedge x_1\wedge x_0'\wedge\neg x_1')$,目标节点是 $x_0'\wedge\neg x_1'$.首先用 $x_0'\wedge\neg x_1'\wedge BDD_{trace}$ 操作得到

© 中国科学院软件研究所 http://www.jos.org.cn

$(\neg x_0\wedge x_1\wedge x_0'\wedge\neg x_1')$,然后用$\exists x_i.e(x_0,\ldots,x_n)=e(x_0,\ldots,x_n)[x_i/0]\vee e(x_0,\ldots,x_n)[x_i/1]$操作从$(\neg x_0\wedge x_1\wedge x_0'\wedge\neg x_1')$中消去 x_0'和 x_1',得到目标节点 $x_0\wedge\neg x_1$ 的父节点$\neg x_0\wedge x_1$,如此反复,直到父节点为初始节点$\neg x_0\wedge\neg x_1$ 为止,这样就得到了搜索路径 $s_0\rightarrow s_1\rightarrow s_2$(这个过程还需要替换 x 为 x'或替换 x'为 x).和 BDD_T 一样,BDD_{trace} 也能记录路径的代价 c,且在搜索完成后,能用 *ExistAnd*()操作从 BDD_{trace} 中得到路径的代价.

4 算法分析与证明

定理 4. 给定启发函数 $h(s)\leq h^*(s)$,BDDRPA*算法是可采纳的.也就是说,如果问题有解,那么 BDDRPA*一定能找到最优解.($h^*(s)$表示节点 s 到目标节点的实际距离).

证明:BDDRPA*算法的单次搜索满足 A*算法搜索的条件,也就是说,BDDRPA*的单次搜索实质就是用 BDD 作数据结构的 A*搜索.对于 BDD_{open} 中的节点,每次用 *dequeue*()函数选出 f 值最小的节点集合 BDD_{temp} 来扩展,新扩展出的节点分为 3 类:

(1) 已在 BDD_{open} 中.这类节点需要比较它们的新旧 g 值,然后取 g 值较小的节点;如果新扩展的 g 值较小,则删除 BDD_{open} 中旧的节点,并把新扩展的节点按照新的 f 值再次插入到 BDD_{open} 中.

(2) 已在 BDD_{closed} 中.这类节点可以和(1)中的节点做同样的处理.如果新的 g 值较小,则直接从 BDD_{closed} 中删去,重新计算 f 值插入到 BDD_{open} 中(这样处理可以省去修改新扩展节点的子节点的指针的开销).

(3) 第 1 次扩展到的节点.这类节点直接计算 f 值添加到 BDD_{open} 即可.

经过这样的处理,可以证明,若问题有解,那么 BDDRPA*一定能终止,且在终止之前,BDD_{open} 中必存在一个节点 s,它位于从 s_i 到 s_G 的最短路径上并且 $f(s)\leq f^*(s_i)$(A*算法有类似的证明).

下面用反证法证明 BDDRPA*找到的路径 $s_i=n_0,n_1,\ldots,n_k=s_G$ 是最短路径.假设这条路径不是最短路径,则 $f(n_k)>f(s_i)$.但是已知 BDDRPA*终止前,BDD_{open} 中一定存在一个节点 s,使得 $f(s)\leq f^*(s_i)$,因此 $f(s)\leq f(n_k)$,那么算法应该选 s 而不是 n_k,产生矛盾.所以,BDDRPA*终止时,找到的一定是最优解. □

下面分析 BDDRPA*的复杂性.算法中主要有 BDD_{open},BDD_{closed},BDD_T,BDD_h 和 BDD_{trace} 这 5 个 BDDs,所用空间是它们的大小之和.因为 BDD 用布尔变量对状态编码,所以可以指数缩减节点的个数,比如状态空间有 10 000 个节点,只需用 $\log_2 10000$(向上取整)=14 个布尔变量编码即可,而传统的搜索算法可能需要存储 10 000 个节点;又因为 BDD 有紧凑性优点,所以可以进一步压缩所需空间.在规模较大的例子中(见实验部分),不用启发函数的 BDDRPA*算法(即 h=0,BDDRPA*退化为基于 BDD 的宽度优先)的单次搜索时间远远小于宽度优先搜索(BFS),这正是因为 BDD 的压缩比率较大,从而在运算中节省了时间.设 BDD 的大小为 n,BDD 上的逻辑运算($\neg\vee\wedge$)的最坏时间复杂度都是 $O(n)$.另外,与 A*算法一样,BDDRPA*的时间复杂度还和启发函数 h 的定义以及计算密切相关.

5 实验结果与分析

本文用两个实验来验证 BDDRPA*的效率,一个是 gridworld 例子,另一个是推箱子 sokoban 智力游戏.用来做实验的机器其配置如下:操作系统 fedora linux 7.0,CPU intel pentium Dcpu 2.80GHZ,内存 1GB DDR,BDD 包 cudd 2.4.1.

5.1 gridworld

四方向的 gridworld 例子如图 8 所示.图上空白的格子表示此格子可通,阴影的格子表示此格子被阻塞,不可通,s_{start} 表示初始节点,s_{goal} 表示目标节点.我们的目的是寻找一条从初始节点到目标节点的最短路径,路径已在图上用灰色线条标出.每个格子的可通性可随时间改变,在当前格局某个格子可通,到下一个格局就可能被阻塞,反之亦然.比如,从格局 1(图 8 左图)到格局 2(图 8 右图),一共有 6 个格子的可通性发生了改变.在右图中,可通性发生改变的格子已经在它上面标记了 changed.如果一个格子是可通的,设定通过它的代价为 1.这是一个简单的线路规划问题,随着时间的改变,gridworld 的格局会不断改变,我们的任务就是要随着格局改变一次次地找到

© 中国科学院软件研究所 http://www.jos.org.cn

任意两个格子之间的最短路径.显然,BDDRPA*算法能够很好地满足这个需求.

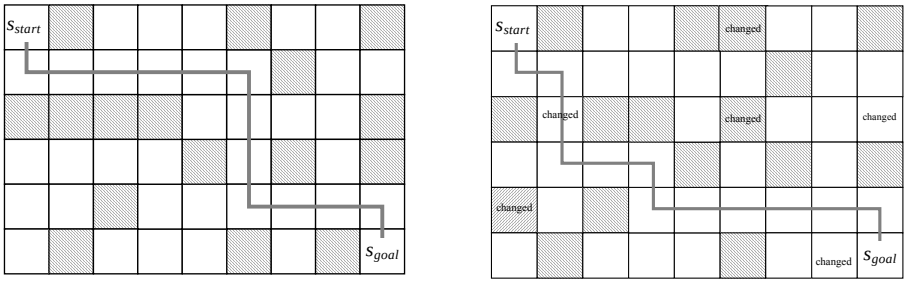

Fig.8 gridworld

图 8 gridworld

在实验中,首先创建一个 $n\times n$ 的完全没有阻塞节点的 gridworld,也就是说,所有的格子都是可通的.然后,每次随机阻塞一定百分比的节点(1%,2%,5%,10%,20%和 30%)直到从初始节点到目标节点找不到路径为止;再然后每次按照同样的百分比开通节点,直到所有节点全部被开通.对于每个规模每次变化某个特定百分比的 gridworld($n\times n$,x%),至少测试 100 次,最后取的是这几百次的平均时间.实验结果见表 1.在表 1 中,第 1 列“maze”是 gridworld 的规模,比如,100×100 表示长和宽各有 100 个节点,共 10 000 个节点;第 2 列“percentage”表示每次阻塞和开通的百分比;第 3 列“reuse”(reuse_time)表示利用旧迁移关系 T 计算新迁移关系 T' 所需要的时间;第 4 列“non_reuse”(non_reuse_time)表示不用旧迁移关系 T 而完全重新构造新迁移关系 T' 所需要的时间;第 5 列“heuristic”(heuristic_time)表示 BDDRPA*使用启发函数 Manhattan distance(两点 (x_1,y_1) 和 (x_2,y_2) 之间的 Manhattan distance 等于 $|x_1-x_2|+|y_1-y_2|$)的搜索时间;第 6 列“non_heuristic”(non_heuristic_time)表示 BDDRPA*不使用启发信息的搜索时间,这时,BDDRPA*退化为基于 BDD 的宽度优先搜索;第 7 列“BFS”(BFS_time)表示宽度优先 BFS 算法的搜索时间.表中的时间单位是秒(s);“–”表示程序在 1 小时内没有结束.

下面分析表 1 的数据.图 9 是 reuse_time 和 non_reuse_time 随着百分比的变化的对比,用来对比是否使用增量构造迁移关系的两种方法.为了节省版面,我们只给出了规模是 500×500 和 1000×1000 时的对比,这是因为这两个规模的实验结果的区分度最大.从这两个图可以看出,当每次变化的百分比小于 20%时,使用增量构造迁移关系方法明显优于不使用增量的构造方法,并且问题的规模越大,优势越明显;而当每次变化的百分比达到 30%时,增量构造方法开始丧失优势.所以,BDDRPA*算法适用于每次变化不太大的情况,这也正是增量搜索的显著特点.

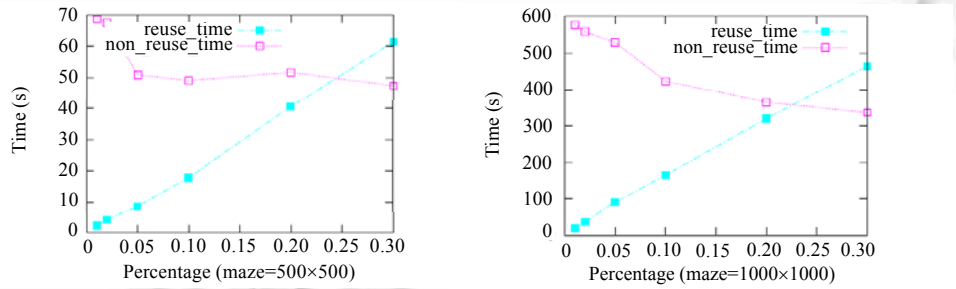

Fig.9 reuse_time and non_reuse_time according to percentage

图 9 reuse_time 和 non_reuse_time 随着百分比变化的对比

图 10 是根据 reuse_time,non_heuristic_time 和 BFS_time 这 3 列数据画出的,BDDRPA*_non_heuristic_time 用的是 reuse_time 和 non_heuristic_time 的和.对于 500×500 的规模,因为 BFS 在 1 小时内得不到结果,所以我们给了一个最保守的估计值 3600s;又因为 BFS 在 1000×1000 时所用时间难以预料,所以我们只比较到 500*500.为了节省版面,只取了变化百分比为 2%和 20%这两种情况.因为 non_heuristic_time 没有使用启发信息,所以 BDDRPA*已经退化为基于 BDD 的宽度优先搜索.从这两个曲线图可以看出,即使不用任何启发信息,BDDRPA*

© 中国科学院软件研究所 http://www.jos.org.cn

和 BFS 相比也非常有优势,而且规模越大,优势越明显.

Table 1 Experimental results for the gridworld

表 1 gridworld 实验结果

Maze_Size	Percent	Reuse_Time	Non_Reuse_Time	Heuristic_Time	Non_Heuristic_Time	BFS_Time
5×5	0.01	0	0.000 413 367	0.000 733 373	0.002 320 15	0.000 173 34
5×5	0.02	1.33333e−5	0.000 426 697	0.000 826 72	0.002 320 15	0.000 106 67
5×5	0.05	2.66667e−5	0.000 333 36	0.000 973 397	0.002 000 13	6.66667e−5
5×5	0.1	5.334e−5	0.000 306 677	0.001 080 08	0.002 306 82	0.000 106 67
5×5	0.2	0.000 226 677	0.000 333 353	0.000 840 05	0.002 000 12	5.334e−05
5×5	0.3	0	0.000 266 667	0.001 333 37	0.001 733 37	0.000 133 367
10×10	0.01	8e−05	0.002 780 2	0.006 340 4	0.014 500 9	0.002 160 12
10×10	0.02	0.000 160 01	0.003 120 19	0.006 100 38	0.014 220 9	0.001 680 12
10×10	0.05	0.000 280 005	0.003 160 2	0.005 280 31	0.013 960 9	0.001 840 15
10×10	0.1	0.000 480 025	0.002 780 21	0.005 020 3	0.012 280 8	0.001 080 06
10×10	0.2	0.001 140 06	0.002 620 18	0.005 860 4	0.012 800 8	0.001 320 05
10×10	0.3	0.001 714 36	0.002 285 86	0.006 571 93	0.012 286 6	0.002 000 14
20×20	0.01	0.000 540 04	0.027 961 7	0.033 802 1	0.094 385 9	0.018 821 1
20×20	0.02	0.001 520 11	0.026 781 8	0.029 021 7	0.080 985	0.018 061 1
20×20	0.05	0.002 285 89	0.023 663 8	0.026 495 2	0.066 915 3	0.017 507 5
20×20	0.1	0.003 267 81	0.016 339	0.023 466 3	0.070 330 9	0.016 395 4
20×20	0.2	0.008 222 75	0.016 778 8	0.028 112 9	0.055 957	0.015 889 8
20×20	0.3	0.011 400 9	0.016 801	0.025 801 6	0.069 804 4	0.016 600 9
30×30	0.01	0.001 740 11	0.070 104 4	0.066 944 2	0.209 193	0.067 024 2
30×30	0.02	0.002 680 14	0.061 583 9	0.037 982 3	0.182 045	0.069 584 4
30×30	0.05	0.005 371 12	0.043 688 1	0.050 722 3	0.145 526	0.045 935 4
30×30	0.1	0.011 500 7	0.048 503	0.042 541 2	0.120 695	0.051 657
30×30	0.2	0.021 779 1	0.041 224 8	0.064 003 8	0.165 296	0.044 558 4
30×30	0.3	0.033 113 2	0.049 781	0.059 337	0.153 609	0.047 780 7
50×50	0.01	0.005 940 38	0.186 452	0.128 468	0.607 805	0.382 324
50×50	0.02	0.009 966 73	0.186 758	0.155 874	0.569 062	0.277 306
50×50	0.05	0.026 501 7	0.202 058	0.176 966	0.610 244	0.367 023
50×50	0.1	0.052 643 3	0.185 292	0.192 332	0.522 128	0.352 502
50×50	0.2	0.113 644	0.195 83	0.177 284	0.521 532	0.348 749
50×50	0.3	0.150 009	0.196 513	0.209 514	0.568 036	0.323 52
100×100	0.01	0.038 205 8	1.188 89	0.840 561	3.403 02	6.710 72
100×100	0.02	0.065 115 2	1.021	0.869 299	2.933 32	6.715 44
100×100	0.05	0.162 01	1.067 27	0.670 043	2.613 95	6.447 88
100×100	0.1	0.314 953	1.052 07	0.748 58	2.131 83	4.711 76
100×100	0.2	0.550 923	0.917 613	0.728 157	2.432 16	3.189 98
100×100	0.3	0.784 449	0.883 255	0.761 248	1.670 1	4.073 85
200×200	0.01	0.178 291	6.003 46	2.696 87	17.545 2	84.395 4
200×200	0.02	0.350 022	5.629 54	4.825 45	14.319 1	92.075
200×200	0.05	0.858 942	6.356 62	6.203 94	13.378 6	99.404 6
200×200	0.1	1.530 71	4.971 85	1.945 82	9.0915 6	64.184 8
200×200	0.2	2.933 79	4.656 29	5.087 32	10.100 1	83.458 8
200×200	0.3	4.149 25	4.239 77	2.754 17	10.610 2	56.507 5
500×500	0.01	2.109 97	68.759 2	50.205 1	251.1	–
500×500	0.02	4.097 01	67.531 4	27.289 5	253.182	–
500×500	0.05	8.608 08	51.027 3	28.588 7	201.708	–
500×500	0.1	17.735 9	49.179 9	50.118 7	201.461	–
500×500	0.2	40.615 4	51.790 9	7.589 9	156.087	–
500×500	0.3	61.540 6	47.437 6	17.952 4	119.863	–
1000×1000	0.01	18.481 2	577.258	355.757	1 642.53	–
1000×1000	0.02	35.792 2	558.477	294.932	1 638.22	–
1000×1000	0.05	89.918 8	529.19	223.198	1 425.94	–
1000×1000	0.1	165.579	423.793	94.593 9	1 011.96	–
1000×1000	0.2	319.982	364.887	196.761	763.431	–
1000×1000	0.3	464.748	336.245	183.65	774.218	–

图 11 是根据 heuristic_time 和 non_heuristic_time 两列数据按照不同的变化百分比绘出来的,用于对比是否使用启发式搜索所用的时间.同样,为了节省版面,只取了变化百分比为 2%和 20%这两种情况.从这两个曲线图可以看出,使用启发信息明显优于不使用启发信息.

© 中国科学院软件研究所 http://www.jos.org.cn

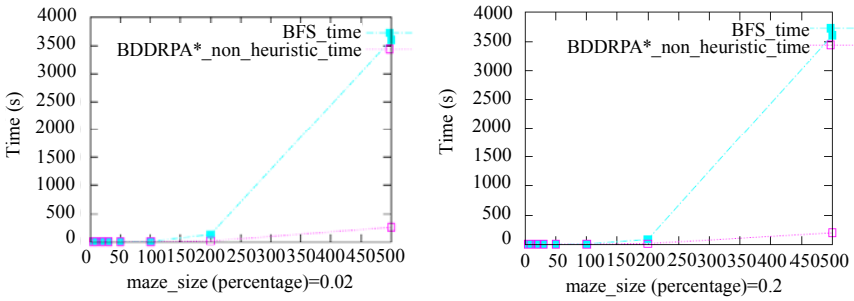

Fig.10 BDDRPA* and BFS without using heuristics

图 10 不使用启发信息的 BDDRPA*和 BFS 的对比

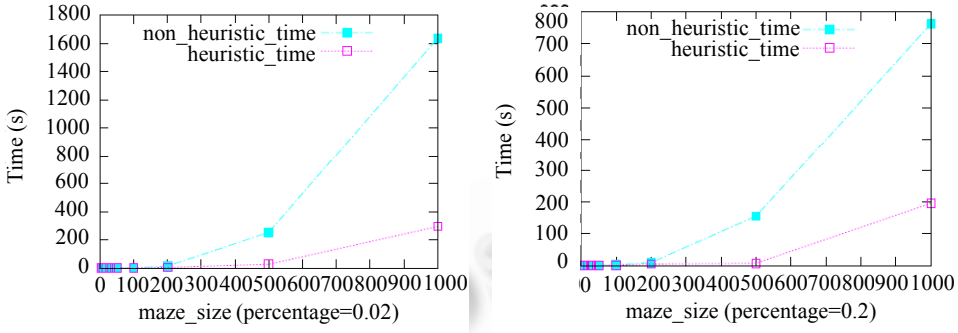

Fig.11 Heuristic_time and non_heuristic_time

图 11 Heuristic_time 和 non_heuristic_time

5.2 推箱子(sokoban)

推箱子(如图 12 所示)是一个非常经典的智力游戏.玩游戏的人假设自己是一个搬运工,他需要在迷宫般的仓库中把箱子全部推到指定位置.人可以在迷宫中可通的位置任意走动,但是他每次只能推动 1 个箱子,并且每次只能推动 1 步.现有的推箱子求解算法在构造搜索空间的时候面临状态爆炸问题,因为大小为 20 的空间 5 个箱子(规模 4×5×5)一共就有 $C(20,6)=38760$ 个状态.BDDRPA*的优势之一就是可以用 BDD 这一紧凑数据结构压缩状态空间.图 12 中的两个格局非常相似,右图和左图相比只多了两个障碍物.因此,假设需要求左图→右图→左图这 3 个格局的解,使用本文介绍的增量搜索算法必然会节省时间.这个例子可以用来模拟每次系统状态格局变化很小的动态机器人线路规划问题.

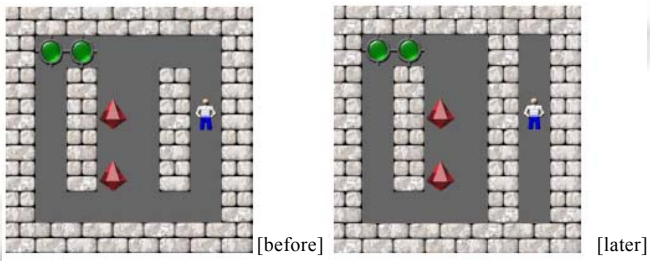

Fig.12 Sokoban

图 12 推箱子游戏

本文做了 3 个规模不同的推箱子游戏例子,每个规模的游戏有一个初始的格局,对初始格局先搜索解,然后让它变化 4 次(规模不变,障碍物、箱子和目标位置改变),每次再搜索解(变化后可能无解,如果无解,表 2 中路径

© 中国科学院软件研究所 http://www.jos.org.cn

长度 length 用∞表示).对这 3 个规模的每个格局,得到一组数据,见表 2.表中的第 1 列 maze 表示例子规模的大小,比如 7×7×2 表示矩阵大小为 7×7,箱子数目为 2;第 2 列 n 表示一次次的变化,0 表示初始格局(初始格局没有 reuse_time,所以表中用"–"表示),1、2、3、4 分别表示第 1 次、第 2 次、第 3 次和第 4 次改变;length 表示搜索路径的长度;nodes 表示搜索过程中扩展出的节点的个数;reuse(reuse_time)表示使用旧迁移关系 T 构造迁移关系 T' 需要的时间;non_reuse(non_reuse_time)表示不用旧迁移关系而完全重新构造新迁移关系需要的时间;search_time 表示搜索时间.启发函数用的是所有不在目标位置的箱子和距离它最近的目标位置的距离之和,也就是在没有障碍物的情况下,人至少需要推多少步才能完成游戏.表中的时间单位是秒(s).

从表 2 可以看出,当迁移关系很大时,使用增量(reuse_time)的效果要远远好于不使用增量(non_reuse_time)的效果,也就是说,BDDRPA*算法在迁移关系很大并且每次变化不大时非常有效.值得注意的是,开通(即删去障碍物)和阻塞(即添加障碍物)的 reuse_time 相差很大(见表中第 5 列数据),这是因为开通是作一系列∨运算,而阻塞只需要作简单的∧运算,这与 BDD 的操作特性完全吻合.

Table 2 Experimental results for sokoban

表 2 推箱子实验结果

Maze_Size	n	Length	Nodes	Reuse_Time	Non_Reuse_Time	Search_Time
6×6×2	0	29	298	–	3.556 22	0.192 012
6×6×2	1	29	273	0.024 001	2.644 17	0.168 011
6×6×2	2	∞	7	0.008 001	2.084 13	0.004
6×6×2	3	29	266	0.508 032	2.336 15	0.168 01
6×6×2	4	29	298	0.400 025	2.748 17	0.184 012
7×7×4	0	55	12 668	–	337.241	31.566
7×7×4	1	49	13 640	138.397	559.711	36.086 3
7×7×4	2	45	14 822	243.095	735.11	44.650 8
7×7×4	3	45	12 391	0.728 046	578.524	34.470 2
7×7×4	4	45	12 668	0.848 053	434.835	31.93
9×9×3	0	134	111 683	–	2 292.61	939.543
9×9×3	1	134	111 683	3.008 19	2 722.68	664.518
9×9×3	2	134	111 683	3.668 23	2 819.47	667.206
9×9×3	3	82	144 592	1 313.21	3 668.75	1120.52
9×9×3	4	68	193 241	1 501.56	4 825.33	1533.26

6 总 结

本文对基于 BDD 的增量启发式算法 BDDRPA*进行了全面而深入的探讨.BDDRPA*综合了基于 BDD 的搜索、启发式搜索和增量搜索这 3 种方法的优点,文中通过大量实验表明,它是一种非常高效的重搜索算法.BDDRPA*算法能够被应用到智能交通、计算机网络以及移动通信网络等线路规划问题中.除此之外,在人工智能的机器人动态线路规划、机器学习和智能控制领域也可以考虑使用 BDDRPA*算法,这也是我们下一步的研究工作.

References:

[1] desJardins M, Durfee E, Ortiz C, Wolverton M. A survey of research in distributed, continual planning. Artificial Intelligence Magazine, 1999,20(4):13−22.

[2] Frigioni D, Marchetti-Spaccamela A, Nanni U. Fully dynamic algorithms for maintaining shortest paths trees. Journal of Algorithms, 2000,34(2):251−281.

[3] Stuart Russell, Peter Norvig. Artificial Intelligence: A modern Approach. Pearson Education, Inc., 2002.

[4] Korf RE. Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 1985,27(1):97−109.

[5] Koenig S, Furcy D, Bauer C. Heuristic search-based replanning. In: Ghallab M, Herbzberg J, Traverso P, eds. Proc. of the 6th Int'l Conf. on Artificial Intelligence Planning and Scheduling. AIPS 2002. Menlo Park: AAAI Press, 2002. 294−301.

[6] Ausiello G, Italiano G, Marchetti-Spaccamela A, Nanni U. Incremental algorithms for minimal length paths. Journal of Algorithms, 1991,12(4):615−638.

© 中国科学院软件研究所 http://www.jos.org.cn

[7] Feuerstein E, Marchetti-Spaccamela A. Dynamic algorithms for shortest paths in planar graphs. Theoretical Computer Science, 1993,116(2):359−371.

[8] Frigioni D, Marchetti-Spaccamela A, Nanni U. Fully dynamic output bounded single source shortest path problem. In: Proc. of the 7th Annual ACM-SIAM Symp. on Discrete Algorithms. 1996. 212−221.

[9] Rohnert H. A dynamization of the all pairs least cost path problem. In: Mehlhorn K, ed, Proc. of the Symp. on Theoretical Aspects of Computer Science. New York: Springer-Verlag, 1985. 279−286.

[10] Edelkamp S. Updating shortest paths. In: Prade H, ed. Proc. of the European Conf. on Artificial Intelligence. 1998. 655−659.

[11] Koenig S, Likhachev M, Furcy D. Lifelong Planning A*. Artificial Intelligence Journal, 2004,155(1-2):93−146,.

[12] Frigioni D, Marchetti-Spaccamela A, Nanni U. Fully dynamic algorithms for maintaining shortest path trees. Journal of Algorithms. 2000,34(2):251−281.

[13] Koenig S, Likhachev M, Liu Y, Furcy D. Incremental heuristic search in artificial intelligence. Artificial Intelligence Magazine, Summer, 2004,25(2):99−112.

[14] Koenig S, Likhachev M. Improved fast replanning for robot navigation in unknown terrain. In: Proc. of the 2002 IEEE Int'l Conf. on Robotics and Automation. ICRA-2002. 2002. 968−975.

[15] Ayorkor Mills-Tettey G, Anthony Stentz, Bernardine Dias M. DD* Lite: Efficient incremental search with state dominance. Technical Report, CMU-RI-TR-07-12, Robotics Institute, Carnegie Mellon University, 2007.

[16] Ramalingam G, Reps T. An incremental algorithm for a generalization of the shortest-path problem. Journal of Algorithms, 1996,21:267−305.

[17] Stentz A. The focused D* algorithm for real-time replanning. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. IJCAI. San Fransisco: Morgan Kanfmann Publishers, 1995. 1652−1659.

[18] Bryant RE. Symbolic manipulation of Boolean functions using a graphical representation. In: Ofek H, O'Neill LA, eds. Proc. of the 22nd ACM/IEEE Design Automation Conf. DAC. New York: ACM, 1985. 688−694.

[19] Clarke EM, Jr. Grumberg O, Peled DA. Model Checking. Cambridge: The MIT Press, 1999.

[20] Edelkamp S, Reffel F. OBDDs in heuristic search. In: Herzog O, Gunter A, eds. Proc. of the 22nd Annual German Conf. on Advances in Artificial Intelligence (KI-98). LNCS 1504, New York: Springer-Verlag, 1998. 81−92.

[21] Jensen RM, Bryant RE, Manuela M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In: Proc. of the 18th National Conf. on Artificial Intelligence and 14th Conf. on Innovative Applications of Artificial Intelligence. AAAI-2002. Menlo Park: AAAI Press, 2002. 668−673.

[22] Jensen RM, Veloso MM, Bryant RE. State-Set branching: Leveraging BDDs for heuristic search. Artificial Intelligence, 2008,172: 103−139.

[23] Yue WY, Xu YY, Kaile Su. BDDRPA*: An efficient BDD-based incremental heuristic search algorithm for replanning. In: Sattar A, Kang BH, eds. Proc. of the AI 2006: Advances in Artificial Intelligence, the 19th Australian Joint Conf. on Artificial Intelligence. New York: Springer-Verlag, 2006. 627−636.

徐艳艳(1981－),女,河南焦作人,博士,主要研究领域为形式化方法,模型检测,人工智能.

岳伟亚(1985－),男,博士,主要研究领域为可满足性问题,算法设计与分析,人工智能.

© 中国科学院软件研究所 http://www.jos.org.cn