基于多值表示的并行规划方法

史晶晶 刘大有 蔡敦波 吕 帅 江 鸿
(吉林大学计算机科学与技术学院 长春 130012)
(吉林大学符号计算与知识工程教育部重点实验室 长春 130012)

摘 要 Fast Downward规划系统是第四届国际规划竞赛的冠军。以高效的串行规划系统Fast Downward为基础,设计并实现了并行规划系统Parallel Downward。首先提出4个并行规划的相关定义;之后提出多值规划任务下动作互斥的定义、充要条件,并实现了动作互斥判断算法;在此基础上设计了候选并行动作集的生成算法;然后为提高系统求解质量重新设计了新的搜索控制策略;最后,给出剪枝策略来抑制并行规划状态空间的指数级膨胀。通过对国际规划竞赛测试问题的实验,Parallel Downward表现出良好的规划效率和规划质量,相比Sapa规划系统Parallel Downward具有较好的可扩展性。

关键词 并行规划,多值规划任务,状态空间启发式搜索,因果图启发式

Parallel Planning Based on Representation with Multi-valued State Variables

SHI Jing-jing LIU Da-you CAI Dun-bo LU Shuai JIANG Hong
(College of Computer Science and Technology, Jilin University, Changchun 130012, China)
(Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China)

Abstract Fast Downward had won the classical track of the 4th International Planning Competition at ICAPS 2004, our work extended the Fast Downward planning system to a parallel setting, and implemented a parallel planning system called Parallel Downward. Several techniques were proposed in this paper. Firstly, four definitions related to parallel plan were given. Secondly, we proposed a method to decide mutual exclusive actions in multi-valued planning tasks, including definitions, the necessary and sufficient condition and the deciding algorithm. Thirdly, we designed an efficient generation of successor states with low cost of generating all the legal sets of actions. Finally, we proposed an effective search control policy of parallel planning to enhance the quality of this valid plan. In addition, we proposed the pruning policy to restrain state space of parallel planning exploding exponentially. We tested Parallel Downward on planning benchmarks used in the International Planning Competitions. The experimental results show that Parallel Downward is excellent in both planning efficiency and planning quality. Parallel Downward is superior in scalability compared with Sapa parallel planning system.

Keywords Parallel planning, Multi-valued planning task, Planning as heuristic search in state space, Causal graph heuristic

智能规划是人工智能研究的一个重要领域,同时也是一门涵盖知识表示、知识推理、非单调逻辑、人机交互和认知科学等方面的多领域交叉学科。近10年来,有关智能规划的研究在问题描述和问题求解两方面得到了新的突破。

为了使智能规划技术从理论走向实用,研究者不断提出复杂的新的问题描述方法:从经典规划模型发展到概率规划、一致性规划、感知规划、时态规划问题和资源规划问题等。其中,概率规划建模动作具有随机型不确定性;一致性规划问题需要考虑到初始状态和动作的不确定性;感知规划建模带有观测能力的规划任务;时态规划问题和资源规划问题在生成规划解时需要考虑时态约束和资源约束。

按照问题求解方式智能规划方法又可分为:基于逻辑演绎的规划方法、基于可满足性的规划方法、基于模型检测的规划方法、基于CSP的规划方法、基于线性规划的规划方法以及基于状态空间启发式搜索的规划方法等。其中,基于启发式搜索的规划方法[1]是当前规划研究的热点,近几年很多该类高效的问题求解方法被提出。它通过设计领域无关的启发函数和剪枝策略,在国际智能规划竞赛[2]中取得了优异的成

到稿日期:2008-10-31 返修日期:2009-02-25 本文受国家自然科学基金重大项目(60496321),国家自然科学基金项目(60573073,60503016,60603030,60773099,60703022,60873149),国家863高技术研究发展计划项目(2006AA10Z245,2006AA10A309),吉林省科技发展计划重点项目(20060213),欧盟项目TH/Asia Link/010(111084)资助。

史晶晶(1985—),女,硕士研究生,主要研究领域为智能规划与数据挖掘;刘大有(1942—),男,教授,博士生导师,主要研究领域为知识工程与专家系统、数据挖掘等,E-mail:liudy@jlu.edu.cn;蔡敦波(1981—),男,博士研究生,主要研究领域为智能规划、约束满足问题;吕 帅(1981—),男,博士研究生,主要研究领域为智能规划与自动推理;江 鸿(1985—),女,硕士研究生,主要研究领域为智能规划与自动推理、数据挖掘。

绩，如 HSP 在 IPC1 中表现优秀，Fast-Forward 获得了 IPC2 次优规划域的冠军，LPG 获得了 IPC3 次优规划域的冠军，Fast Downward[3] 获得了 IPC4 次优规划域的冠军，SGPlan$_t$ 在经典规划、数值规划和灵活规划竞赛中都表现出优秀的性能。

实际环境中的大量任务存在并行的可能，如探测型火星车的移动器件和观测器件能够并行执行；不同部门的商业操作能够并行执行等。在这种情况下，串行规划解将降低工作效率并延迟任务完成时间；并行规划解能够合理地同时调度多个部件，以最快速度完成规划任务。本文主要研究基于启发式搜索的构造并行规划解的高效方法。

本文首先介绍 Fast Downward 串行规划系统；然后以高效的串行规划系统 Fast Downward 为基础，设计并实现基于多值表示的并行规划系统 Parallel Downward；最后，给出 Parallel Downward 系统的实验结果。

1 Fast Downward 串行规划系统

1.1 Fast Downward 规划系统框架

Fast Downward(FD)系统最主要的两个特点是多值规划任务和基于因果图分析的启发函数，它将基于布尔变量的规划任务转化为基于多值变量的规划任务，之后采用适用于多值变量的规划求解过程。FD 系统设计了一种从布尔规划任务向多值规划任务的自动转换方法，提出了因果图启发式函数。该启发式函数逐步地递归调用自顶向下的遍历因果图，直到所有的子目标都是基本的图搜索任务，再采用层次化分解生成搜索状态的目标距离估计。

FD 系统主要分为 3 部分：转换、知识编译和搜索，如图 1 所示。

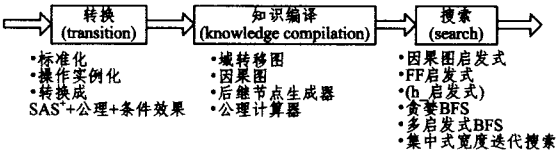

图 1 Fast Downward 系统

1.2 因果图启发式

因果图启发式(CG heuristic)是 FD 规划系统中启发式搜索过程的中心部件，它通过分析以剪枝的因果图中的一个小“窗口”而导出的一系列子问题来估计从某个给定状态到目标状态的开销。

对每一个变量 v 及它的每一对取值 $d,d'\in D_v$，CG 启发式计算利用因果图和域转移图计算变量 v 从 d 到 d' 取值变化的开销 $\text{cost}_v(d,d')$，作为变量 v 的启发值。一个给定状态的 CG 启发式估计值是所有在目标条件 $s_*(v)$ 中定义的变量 v 的开销 $\text{cost}_v(s(v),s_*(v))$ 的总和。

1.3 搜索控制策略

FD 规划系统默认的搜索算法是带 closed 表的贪婪最好优先算法。贪婪最好优先算法见文献[4]。使用的算法和常用的贪婪最好优先算法有以下 3 点不同：偏好动作、延迟的启发式计算、多启发式最好优先搜索。本文主要介绍延迟的启发式计算。

延迟的启发值计算(deferred heuristic evaluation)意味着不立即计算当前状态 s 后继的启发式估计值，而是用状态 s 的估计值代替扩展状态的启发值作为后继状态排序的依据，存储在 open 表中，后继状态仅在其扩展的同时再计算其自身的启发值。

例 1 当前 open 表中有 4 个状态：s_0，s_1，s_2 和 s_3，它们父节点的启发值分别为 4,6,5 和 4。搜索算法会选择其父节点的启发值最小的节点进行扩展，在此例中，将在 s_0 和 s_3 中随机选择一个顶点扩展，不妨假设选择 s_0 来计算其启发值，假设计算结果为 3，s_0 有 3 个后继 s_4，s_5 和 s_6，则扩展之后的 open 表如图 2 所示。

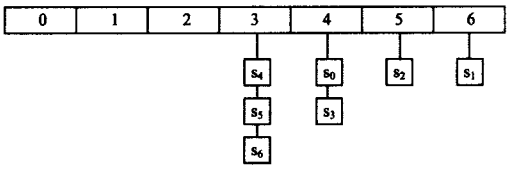

图 2 open 表

延迟的启发式计算和偏好动作一起考虑会节省大量的时间：如果状态 s 的一个后继 s' 是一个偏好后继，那么它很可能在其它后继节点扩展之前通过偏好动作的 open 表进行扩展，这样当存在一个从 s' 到目标顶点的非增路径 * 时，s 的大部分后继节点的启发值都不需要被计算。

1.4 Fast Downward 的不足及其扩展到并行规划的优势

在实际环境中，多个智能体的并行执行和单智能体的多个组件之间的并行执行能够大幅度提高系统效率。因此，并行规划方法的研究具有重要价值。FD 规划系统只能给出一个规划任务的串行规划解，致使其不能在实际环境中广泛应用。

以往的并行规划方法大多基于命题可满足思想，易于处理命题逻辑描述的规划问题，在处理数值变量等方面存在建模复杂等不足。因此，许多研究者尝试了基于状态空间搜索的并行规划方法。

FD 规划系统在串行规划领域的成功证明了它与其它的状态空间搜索的规划系统相比有先天的优势，将其扩展到并行领域有较大的意义。本文的工作就是研究扩展 FD 系统处理并行规划问题的原理、方法和系统设计。

2 Parallel Downward 并行规划系统

目前，能够求解并行规划问题的规划系统有 SAPA，LPG，SATPLAN2006 和 AltAltp[5] 等。Fast Downward 规划系统虽然效率较高，但是仅能求解串行规划。本节给出以 Fast Downward 为基础设计的并行规划系统 Parallel Downward 的技术细节及具体实现。

2.1 基本定义和表示

根据文献[6]中串行规划问题对应的状态空间，定义并行规划问题对应的状态空间。

定义 1（并行规划问题对应的状态空间） 一个状态模型 $\Pi=\langle S,S_I,S_G,A,f\rangle$，其中：

- S 是一个有限非空的状态集合；
- $S_I\in S$ 是初始状态；

* 在非增路径上，所有前驱顶点的启发值都不大于后继顶点的启发值。

• $S_G \subseteq S$ 是一个非空的目标状态集合；

• $A(s) \subseteq A$ 表示每一个状态 $s \in S$ 的可用动作集合；

• $f(AP(s),s)$定义了一个状态转移函数 $AP(s) \subseteq A(s)$，并且 $AP(s)$中的任何两个动作都不互斥，$AP(s)$是状态 s 下合法的并行动作集合。

定义 2(状态转移) 令 $AP(s)$是状态 s 上的一个合法的并行动作集，状态转移函数 $f(AP(s),s)$是动作集 $AP(s)$在状态 s 下的执行结果，其中 $f(AP(s),s)=\bigcup_{i=0}^{n-1} s+add(a_i)-del(a_i), a_i \in AP(s)$。

定义 3(并行规划解) 并行规划问题的规划解是一个合法的并行动作集的序列：$\pi=\langle AP(s_0),AP(s_1),\cdots,AP(s_n)\rangle$，使得 $s_1=f(AP(s_0),s_0),\cdots,s_{n+1}=f(AP(s_n),s_n)$，并且 $s_{n+1} \in S_G$。

定义 4(最优并行规划解) 一个并行规划解 $\pi=\langle AP(s_0),AP(s_1),\cdots,AP(s_n)\rangle$是最优的当且仅当 n 最小。

基于状态空间搜索求解并行规划问题的基本思想为：从初始状态出发，使用合法动作集扩展后继状态，以某种策略在后继状态集中继续扩展，直到发现一个目标状态。由于状态空间大小存在指数级膨胀的问题，目前高效的搜索算法大都利用启发式函数。因此，性能优良的启发式函数在基于状态空间搜索的并行规划方法中具有关键作用。

2.2 动作互斥的判断

参考规划图中两种不同角度的动作互斥定义[6,7]，给出多值规划任务中的动作互斥的定义 5 和定义 6，其中定义 6 也可看作是多值规划任务中动作互斥的充要条件。

定义 5(动作互斥) 两个动作是互斥的，当且仅当不存在任何一个合法的并行规划解在同一个时间步包含这两个动作。

定义 6(动作互斥的充要条件) 令 a 和 b 为状态 s 的两个可用动作，称它们互斥，当且仅当满足下列条件之一：

a)冲突(Interference)：如果一个动作的删除效果与另一个动作的前提或添加效果中包含有相同的变量；

b)需求竞争(Competing Needs)：如果两个动作的前提条件在当前状态下是矛盾的，即存在一个变量的两个不同取值分别出现在两个动作的前提条件中。

对定义 6 作进一步解释，给出动作不互斥的充要条件的形式化表示。

性质 1(动作不互斥的充要条件) 多值规划任务中的动作 a 和 b，同时满足以下 3 条，则 a 和 b 不互斥(记多值变量取值约束集[1] x 所涉及的变量为 $Vars(x)$)。

1)动作 a,b 的前提条件不矛盾；

2)$(Vars(\text{pre}(a)) \cup Vars(\text{eff}(a))) \cap Vars(\text{eff}(b))=\emptyset$；

3)$(Vars(\text{pre}(b)) \cup Vars(\text{eff}(b))) \cap Vars(\text{eff}(a))=\emptyset$。

基于性质 1，本文提出判断多值规划任务中的两个动作是否互斥的算法 1，描述如下。

算法 1 mpt-exclusive(a,b) //a,b 是两个动作

```
BEGIN
1. for each condi∈conds(a) do
       p1[var(condi)]:=i;
2. for each effi∈effs(a) do
   a) if P[var(effi)]=0
          P[var(effi)]:=-1;
   b) for each pcondj∈pconds(effi) do
          P[var(pcondj)]:=i
3. for each effi(effs(b) do
       if P[var(effi)]! =0
           return true;
4. for each condi∈conds(b) do
       Q[var(condi)]:=i;
5. for each effi∈effs(b) do
   a)if Q[var(effi)]=0
       Q[var(effi)]:=-1;
   b)for each pcondj∈pconds(effi) do
       Q[var(pcondj)]:=i;
6. for each effi∈effs(b) do
       if Q[var(effi)]! =0
           return true;
7. for each variable vi do
           if P[vi]>0 and Q[vi]>0
       1)if var(cond P[vi])=vi
                 var1:= value(cond P[vi]);
           else
               for each pcondj∈pconds(effP[vi]) do
                   if var(pcondi)=vi
                       var1:= value(pcondi);
                       break;
   2)if var(cond Q[vi])=vi
             var2:= value(cond Q[vi]);
         else
             for each pcondj∈pconds(effQ[vi]) do
                 if var(pcondi)=vi
                     var2:= value(pcondi);
                     break;
   3)if(var1! =var2)
             return true;
8.   return false;
END
```

其中，P 和 Q 两个数组分别记录动作 a 和 b 的前提与效果涉及变量集。以数组 P 为例，若 P 的元素值 $P[i]$为正数，则表示动作 a 的条件 cond_i 包含变量 i 或者动作 a 的效果 eff_i 中的某一个前提中包含变量 i；若数组变量 $P[i]$的取值为负数，则表示只在效果中出现。这样做的目的是在步骤 7 中检查前提条件冲突时是否可方便地找到该变量在前提中的取值。

步骤 1—步骤 3 检查$(Vars(\text{pre}(a)) \cup Vars(\text{eff}(a))) \cap Vars(\text{eff}(b))$是否为 $\emptyset$；步骤 4—步骤 6 检查$(Vars(\text{pre}(b)) \cup Vars(\text{eff}(b))) \cap (Vars(\text{eff}(a))$是否为 $\emptyset$；步骤 7 检查动作 a 与动作 b 条件和效果前提中是否矛盾。其中步骤 1 和步骤 2 查找同时在 a,b 中出现的变量的取值，如果取值不相等，则矛盾，返回 true。经过步骤 1—步骤 7 的所有检查，说明动作不互斥，返回 false。

2.3 候选并行动作集的生成

生成一个状态的所有候选并行动作集通常需要较高的计算代价。构造候选并行动作集朴素的方法是枚举该状态可用

[1] 变量取值约束是变量及值域内的一个取值组成的序偶。

动作集的所有非空子集，然后判断每个子集的合法性。假设可用动作集的规模为 N，则需要对 2^N-1 个集合做合法性判断。假设一个集合的元素数为 n，那么判断该集合的合法性需要判断 $n*(n+1)/2$ 次动作互斥。如此庞大的计算量，必然降低系统的求解效率。

本文提出一种生成候选动作集的高效方法，并不需要枚举可用动作集的每个非空子集。例如，已知动作集 $A=\{a_1, a_2, a_3\}$ 不是合法的并行动作集；若动作集 $B \supseteq A$，则 B 也不是合法的并行动作集。基于该事实，可以采取逐步生成并行动作集的方法。首先，空集和单元素集合一定不包含互斥动作对，它们是合法的并行动作集。任取一个元素，做成单元素集合，它与空集作为初始基准集合 P 的元素；然后，依次加入其它动作，对于当前欲加入的动作 a_i，将其逐个加入基准集合 P 的每一个元素 A_i 中，又组成若干个新动作集 NA_i，舍弃所有存在互斥动作对的新动作集，其余加入基准集合中。

这种办法不仅减少了大量的不必要的动作集的合法性判断，而且由于 A_i 中的所有动作集均不互斥，新集合的合法性只需判断 a_i 是否与 A_i 的每个元素互斥。假设新集合 NA_i 的元素数是 n_i，每个集合的合法性判断由原来需要的 $n_i*(n_i+1)/2$ 减少到 n_i-1 次互斥动作对的判断。下面通过例 2 予以说明。

例 2　当前状态下有动作集 $A=\{a_1, a_2, a_3, a_4, a_5, a_6\}$，互斥的动作对有 $\langle a_1, a_2\rangle$，$\langle a_1, a_4\rangle$，$\langle a_2, a_3\rangle$，$\langle a_2, a_4\rangle$，$\langle a_2, a_5\rangle$，$\langle a_2, a_6\rangle$，$\langle a_3, a_4\rangle$ 和 $\langle a_4, a_5\rangle$。初始基准集合 $P=\{\emptyset, \{a_1\}\}$，插入过程如表 1 所列。

表 1　并行动作集生成过程

当前插入动作	基准集合 P(插入后)	动作集合法性检查次数	互斥动作对判断次数
a_2	$\{\emptyset, \{a_1\}, \{a_2\}\}$	2	1
a_3	$\{\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_3\}\}$	3	2
a_4	$\{\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_3\}, \{a_4\}\}$	5	5
a_5	$\{\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_3\}, \{a_4\}, \{a_5\}, \{a_1, a_5\}, \{a_3, a_5\}, \{a_1, a_3, a_5\}\}$	6	6
a_6	$\{\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_3\}, \{a_4\}, \{a_5\}, \{a_1, a_5\}, \{a_3, a_5\}, \{a_1, a_3, a_5\}, \{a_6\}, \{a_1, a_6\}, \{a_3, a_6\}, \{a_1, a_3, a_6\}, \{a_4, a_6\}, \{a_5, a_6\}, \{a_1, a_5, a_6\}, \{a_3, a_5, a_6\}, \{a_1, a_3, a_5, a_6\}\}$	10	14

原方法需要判断 $2^6=64$ 次动作集的合法性，改进后则只需要判断 26 次；原方法需要判断 190 次动作互斥，改进后则只需要判断 28 次。

下面给出逐步生成候选动作集的算法 2。

算法 2　step-generate-pacts (all_applicable_actions, n)

```
BEGIN
P:={∅,{all_applicable_actions[1]} };
for i from 2 to n do
    P':=∅;
    for each Ai∈P do
        if ∀ai∈Ai 与 all_applicable_actions[i]不互斥
            Ai:={all_applicable_actions[i]}∪Ai;
            P':=P(∪Ai;
     P:=P∪P';
END
```

内层的 for 循环生成由插入动作增加的新并行动作集，并判断它们的合法性，将合法的集合加入 P' 中。外层 for 每循环一次计算插入一个新动作产生的所有新的动作集，并将其中合法的集合在插入下一个动作之前加入基准动作集 P。最后得到的基准集 P 即为所有合法的并行动作集的集合。

2.4　搜索控制策略

上述方法使 Parallel Downward 具有并行规划的能力，但新系统求解质量和求解效率都有待提高。Fast Downward 系统的搜索控制策略是通过当前节点的父节点启发值来选择下一个进行扩展的节点，当用它引导并行规划系统求解问题时，求解问题的时间比较长，给出的并行解较差，几乎与串行解等长。因此，需要重新设计使用于并行规划的搜索控制策略。

Fast Downward 系统搜索控制策略应用在并行规划中失败的主要原因是：在并行规划的搜索空间中有数量庞大的节点拥有相同启发值的父节点，该搜索控制策略按照这些节点的生成顺序选择，不能保证搜索会向着一个较优的方向扩展。

重新设计适用于并行规划的搜索控制策略：根据并行空间规模大的特点，重新计算每一个节点的启发值，使用该节点自身的启发值来作为选择扩展节点的依据。此时候选的扩展节点数目虽然比原搜索控制策略有较大减少，但是仍然存在大量相同启发值的节点。本文进一步考虑在自身启发值最优的节点中，再做选择。

从父节点到达当前节点的动作集，称作是该节点的动作集。从当前节点的父节点通过单个动作达到的节点的启发值，称作当前状态下该动作的启发值。本文在自身启发值最优的节点中，选择该节点的动作集中单个动作的启发值的最优节点作为下一个扩展的节点。这样就可以大大减小并精确了候选扩展节点的选择范围。

由于在并行规划的搜索空间中，启发值最优节点的数目也比较大，计算每一个节点的动作集中的每一个动作的启发值，工作量非常庞大，并且计算之后，还要进行统计与比较，这会大大降低系统的求解效率。因此本文考虑在生成候选的并行动作集之前，求出每个可用的动作的启发值，然后根据启发值对它们进行排序，再生成候选的并行动作集。这样，既能避免动作的启发值的重复计算，又能保证单个动作启发值最优的节点先生成，直接按照节点的生成顺序选择即可，省略了统计和比较的工作。

算法 3 是搜索控制策略的算法。

算法 3　searching-control-strategy (all_applicable_actions, n)

```
BEGIN
for i from 1 to n do
    h:=get_heuristic(all_applicable_actions[i]);
    act_openlist. insert(h,all_applicable_actions[i]);
for j from min_heuristic(act_openlist)
to max_heuristic(act_openlist) do
    while ! empty(act_openlist[j]) do
        a:=act_openlist[j]. pop();
        all_order_actions. push_back(a);
step-generate-pacts (all_order_actions,n);
END
```

在算法 3 中两个 for 循环是对所有的可用动作集排序，然后将排序好的动作存储在动作数组 all_order_actions 中。

为了避免时间复杂度较高的排序算法，利用一个与状态 open 有相同结构的动作 open 表 act_openlist 来对动作进行排序。第一个 for 循环中，求解每个动作的启发值，并插入到 act_openlist 中。第二个 for 循环按照动作启发值的大小依次将动作从 act_openlist 表取出插入到 all_order_actions。

2.5 剪枝策略

基于状态空间搜索的并行规划一个主要的难点是搜索空间的指数级膨胀，剪枝策略成为影响并行规划系统解质量和求解时间的关键因素。设计剪枝策略需要遵循两个策略：第一，在不影响解质量的前提下，尽量压缩搜索的空间；第二，应使剪枝需要的时间代价尽量小。

剪枝策略首先考虑剪掉每次扩展的节点中启发值较小的部分节点，这样裁剪对解质量的影响非常小。此方法有两个步骤：第一，找到每次扩展的节点中启发值比较小的节点，将所有节点的启发值都求出并保存；其次，对所有节点的启发值进行统计和比较，舍弃启发值较差的节点，将其余节点插入到 open 表中。由此易知，这种方法的时间代价非常大，大量的时间浪费在剪枝策略上，虽然搜索的空间减小了，但没有提高系统的求解效率。

根据搜索控制策略的特点，即使节点启发值是最优的，但节点的动作集中单动作启发值较差，也不会成为下一个被扩展的节点，本文考虑剪掉单动作启发值较差的节点。在搜索控制策略中，对所有可用动作按照启发值排序后，舍弃排在后面的半数动作，只利用其余可用动作生成候选的并行动作集合。

此种策略有如下 3 种优势：首先，无需冗余地剪枝开销；其次，单动作启发值较差的节点其自身启发值一般也较差，对解质量的影响较小；最后，对搜索空间规模的压缩很显著，如果原节点共有 $2n$ 个可用动作，至多有 2^{2n} 个后继节点，而剪枝后至多有 2^n 个后继节点，压缩了 2^n 倍。

算法 4 是带剪枝策略的搜索控制算法。

算法 4 searching-control-strategy-c(all_applicable_actions,n)

```
BEGIN
for i from 1 to n do
    h:=get_heuristic(all_applicable_actions[i]);
    act_openlist. insert(h,all_applicable_actions[i]);
for j from min_heuristic(act_openlist)
to max_heuristic(act_openlist) do
    while ! empty(act_openlist[j]) do
        a:=act_openlist[j]. pop();
        all_order_actions. push_back(a);
step-generate-pacts (all_order_actions,n/2);
END
```

本文的剪枝策略将剪枝开销减少到最低，只在原算法的基础上调用候选并行动作集合的生成算法时稍作修改。3.1 节的系统剪枝前后求解问题的实验结果说明了该简单易行的剪枝策略，对于绝大部分问题没有影响到其解质量，并且对求解时间有指数级别的提高。

3 实验结果及分析

本节给出并行规划系统 Parallel Downward 的实验结果。规划测试域为：Logistic 域、ZenoTravel 域、Pipesworld 域、Satellite 域和 Driverlog 域。这 5 个规划域是国际规划竞赛中使用的并行规划域。

Logistics 域建模一类是使用运输工具运送货物到指定地点的物流问题。运输工具为卡车和飞机，卡车只能在一个城市内的不同地点移动，飞机只能在城市之间的飞机场运输货物。每个具体的问题都涉及一些城市，每个城市都包括一些地点：卡车停靠站和飞机场。问题的目标是将处于不同地点的一些包裹运到指定地点。

ZenoTravel 域建模一类用飞机来运送乘客到达目的地的问题。飞机可以选用快速飞行和慢速飞行两种方式，快速飞行比慢速飞行消耗的燃油的速度快，问题的目标是在有限条航线上合理地分配燃油，使燃油尽量少。每个具体的问题中包括每架飞机的油量的描述及飞机和乘客的初始地点的描述。每个问题的目标都是将不同的乘客运送到不同的目的地。

Satellite 域建模一类收集若干卫星图像数据的问题。此类问题目标是规划和调度多卫星间的观察任务。

Driverlog 域建模一类利用卡车在不同地点间运输包裹的问题。卡车司机为了开动卡车必须在卡车之间走动，小路只供司机行走，公路只供卡车行驶。

实验平台如下：TCL PC 机一台，CPU：Intel® Celeron® M 1.50GHz，内存：1G，操作系统：Linux，编程语言：C++语言，编程环境：gcc。在以下各表中，problem 均为规划问题名称，length 为规划解长度，time 为规划求解时间。时间精确到 0.01 秒。求解时间限制为 900 秒；超出该限制的求解过程用横线“—”标记。

3.1 Parallel Downward 系统剪枝前后的比较

表 2 是 Parallel Downward 在 Logistic 域上的剪枝前后对比的实验结果。从表 2 中可以看出，剪枝后系统 Parallel Downward 给出的 Logistic 域上绝大规划问题的规划解与剪枝前系统给出的规划解相同，而剪枝后的求解时间较剪枝前的呈指数级减小，并且此优势随着问题规模的增大表现得更加明显。

表 3 是 Parallel Downward 在 ZenoTravel 域上剪枝前后对比的实验结果。从表 3 中可以看出，剪枝后的 Parallel Downward 在 ZenoTravel 域上除个别问题受到较小的影响外，对其它问题的规划解均无影响，并且剪枝后的求解时间较剪枝前的有明显提高。对于由于搜索空间规模过大剪枝前的系统不能求解的问题，剪枝后的系统均能顺利求解。

表 2 和表 3 的实验结果表明：剪枝后的 Parallel Downward 较剪枝前的系统相比，规划解质量受到的影响很小，求解效率有明显的提高。随着问题规模的增大，求解速度呈现稳定的指数级提高。以上结果说明本文的剪枝策略是合理且高效的。

表 2 Parallel Downward 在 Logistic 域的测试结果

problem	剪枝前的 Parallel Downward		Parallel Downward	
	length	time(s)	length	time(s)
L01	12	0.17	12	0.02
L02	25	95.99	25	0.76
L03	8	0.08	8	0.02
L04	14	0.74	21	0.03
L05	16	0.46	16	0.04
L06	16	1.02	22	0.07

L07	24	172.41	24	2.56
L08	26	777.01	26	3.87
L09	33	33.96	33	0.95
L10	18	92.87	18	1.04
L11	13	27.73	13	0.68
L12	14	63.17	14	0.76
L13	22	803.05	22	4.08
L14	—	—	20	27.83
L15	19	316.17	19	3.50

表 3 Parallel Downward 在 ZenoTravel 域的测试结果

problem	剪枝前的 Parallel Downward		Parallel Downward	
	length	time(s)	length	time(s)
Z01	1	0.00	1	0.00
Z02	5	0.00	5	0.00
Z03	5	0.01	5	0.00
Z04	7	0.01	7	0.00
Z05	10	0.02	13	0.01
Z06	8	0.03	8	0.02
Z07	10	0.03	10	0.01
Z08	8	0.12	8	0.04
Z09	14	0.17	14	0.04
Z10	12	0.80	12	0.10
Z11	9	1.44	9	0.15
Z12	11	1.21	11	0.14
Z13	12	43.34	12	0.36
Z14	—	—	19	253.99
Z15	—	—	22	84.22

3.2 与 Sapa 的比较

Sapa 规划系统[8]是前向状态空间搜索的规划系统。对于状态空间，它采取启发式估值的形式来选择下一个要扩展的状态，它的启发式估值是先从忽略动作的删除效果和所有数值资源的影响的放宽式来获得，然后再调整这些估值使得它们与资源约束相对应。Sapa 规划系统曾参加 2002 年的第三界国际规划竞赛，本文测试使用的系统是 2004 年 6 月完成的版本，此版本修改了若干错误，并且采用了更加高效的数据结构[9]。

表 4 是 Sapa 与 Parallel Downward 在 Logistic 域的测试结果对比。实验结果表明，在 Logistic 域中 Parallel Downward 较 Sapa 系统所需求解问题的时间较少，解质量较优，并且求解能力更强。

表 5 是 Sapa 与 Parallel Downward 在 Zenotravel 域的测试结果对比。由该表可知，在 Zenotravel 域中 Sapa 较 Parallel Downward 求解问题所需的时间更少。Parallel Downward 给出的解中有 4 个问题的解较 Sapa 更优，有 5 个问题较 Sapa 更差，6 个问题中两个规划器给出的规划解的质量均相同。

表 6 是 Sapa 与 Parallel Downward 在 Satellite 域的测试结果对比。实验结果表明，在 Satellite 域中两个规划器给出的规划解质量相当，但 Parallel Downward 所需的求解时间较少，且求解能力较强。

表 7 是 Sapa 与 Parallel Downward 在 Driverlog 域的测试结果对比。实验结果表明，在 Satellite 域中 Sapa 给出的规划解质量较好，但 Parallel Downward 所需的求解时间较少，且求解能力更强。

表 4 Sapa 与 Parallel Downward 在 Logistic 域的测试结果比较

problem	Sapa		Parallel Downward	
	length	time(s)	length	time(s)
L01	14	0.17	12	0.02
L02	—	—	—	—
L03	11	0.20	8	0.01
L04	20	0.48	21	0.04
L05	20	0.69	16	0.04
L06	—	—	22	0.05
L07	—	—	25	2.50
L08	—	—	26	3.84
L09	—	—	33	0.90
L10	—	—	18	1.01
L11	—	—	13	0.64
L12	—	—	14	0.73
L13	—	—	22	4.06
L14	—	—	15	27.84
L15	—	—	19	3.46

表 5 Sapa 与 Parallel Downward 在 ZenoTravel 域的测试结果比较

problem	Sapa		Parallel Downward	
	length	time(s)	length	time(s)
Z01	1	0.00	1	0.00
Z02	5	0.00	5	0.00
Z03	5	0.00	5	0.01
Z04	7	0.00	7	0.01
Z05	9	0.00	10	0.01
Z06	7	0.00	8	0.02
Z07	9	0.00	10	0.03
Z08	11	0.01	8	0.05
Z09	13	0.01	14	0.05
Z10	13	0.01	12	0.10
Z11	9	0.02	9	0.14
Z12	12	0.02	11	0.12
Z13	16	0.04	12	0.35
Z14	14	0.46	19	254.81
Z15	22	3.28	22	84.57

表 6 Sapa 与 Parallel Downward 在 Satellite 域的测试结果比较

problem	Sapa		Parallel Downward	
	length	time(s)	length	time(s)
S01	8	0.01	8	0.00
S02	12	0.02	12	0.00
S03	10	0.08	10	0.01
S04	17	0.11	17	0.02
S05	14	1.45	15	0.30
S06	17	0.74	10	0.07
S07	14	3.47	20	4.28
S08	—	—	23	19.68
S09	—	—	17	138.04
S10	—	—	20	283.11

表 7 Sapa 与 Parallel Downward 在 Driverlog 域的测试结果比较

problem	Sapa		Parallel Downward	
	length	time(s)	length	time(s)
D01	7	0.01	8	0.00
D02	16	0.02	14	0.00
D03	8	0.00	10	0.00
D04	12	0.04	22	0.02
D05	16	0.04	13	0.02
D06	9	0.04	8	0.02
D07	9	0.09	14	0.02
D08	17	0.13	17	0.04
D09	—	—	16	0.03
D10	13	1.93	19	0.06
D11	—	—	25	0.07

D12	—	—	42	1.23
D13	—	—	26	0.12
D14	—	—	32	0.35
D15	—	—	33	19.82

以上实验结果表明，Parallel Downward 规划系统能够求解出以上 4 个域中的绝大部分的并行规划问题，并且表现出良好的规划效率和规划质量。与 Sapa 系统相比，Parallel Downward 系统在多数问题上求解时间更少，并且求解能力更强。Parallel Downward 规划系统的扩展性优于 Sapa 系统。

结束语 智能规划是人工智能的重要研究领域。基于状态空间启发式搜索的规划方法已成为智能规划研究的热点。然而，基于状态空间启发式搜索的并行规划方法由于面临状态空间指数级爆炸的问题，有待深入研究。

本文设计的并行规划系统 Parallel Downward 继承了 Fast Downward 的高效性，并且扩展了其求解能力。出于扩展的需要，本文首先提出了 4 个并行规划的相关定义；之后又提出了多值规划任务下动作互斥的定义、充要条件，并实现了动作互斥判断算法；在此基础上设计了并行动作集的生成算法；然后为了提高系统求解质量，本文重新设计了新的适应于并行规划搜索的控制策略；最后，给出剪枝策略，克服了并行规划搜索空间指数级膨胀给求解问题带来的困难。基于 Linux 系统和 gcc 开发环境，实现了 Parallel Downward 规划系统。通过在国际通用测试问题上的大量实验，Parallel Downward 表现出良好的求解效率、求解质量和较强的规划求解能力。通过与 Sapa 规划系统的对比，Parallel Downward 表现出较优的可扩展性。

为了进一步提高系统的规划质量和规划效率，下一步工作可以考虑改进启发式函数。将当前系统的启发式函数估计由当前状态到达目标状态所需的动作数改进为估计由当前状态到达目标状态所需的“动作步”数，以适应求解并行规划问题的需求。

参 考 文 献

[1] Bonet B, Geffner H. Planning as heuristic search[J]. Artificial Intelligence, 2001, 129(1): 5-33

[2] 智能规划竞赛(International Planning Competition)官方网站. 2006. http://zeus.ing.unibs.it/ipc-5/

[3] Helmert M. The Fast Downward Planning System[J]. Journal of Artifical Intelligence Research, 2006, 26: 191-246

[4] Russell S, Norvig P. Artificial Intelligence: A Modern Approach (Second Edition)[M]. New Jersey: Prentice-Hall, 2003

[5] Nigenda R S, Kambhampati S. AltAlt^p: Online Parallelization of Plans with Heuristic State Search[J]. Journal of Artificial Intelligence Research, 2003, 19: 631-657

[6] Blum A L, Furst M L. Fast planning through planning graph analysis. Artificial Intelligence, 1997, 90: 281-300

[7] 李天际，姜云飞. 图规划及其扩展的分析和研究[J]. 计算机科学，2001，7(28)：69-73

[8] 姜云飞，吴康恒. 智能规划的研究和应用[J]. 计算机科学，2002，2(29)：100-103

[9] Sapa 规划系统主页. 2002. http://rakaposhi.eas.asu.edu/sapa/

(上接第 177 页)

由$\langle f(3,3,3), f(0,3,0), f(3,1,3), f(0,1,0)\rangle = \langle 0,1,2,2\rangle$，可知 $f \notin L_{G_{4,2}}$。

(6)对两个拟线性函数集 L_P：

由$\langle f(3,3,3), f(0,3,0), f(3,1,3), f(0,1,0)\rangle = \langle 0,1,2,2\rangle$，可知 $f \notin L_p$。

(7)由定义可知，$f \notin P_4 \cup \{*\}$。

综上所述，$G_4 = \{\langle 0,0,1,1\rangle, \langle 0,0,2,2\rangle, \langle 0,0,3,3\rangle, \langle 1,1,2,2\rangle, \langle 1,1,3,3\rangle, \langle 2,2,3,3\rangle, \langle 0,1,2,3\rangle\}$必是部分四值逻辑中准完备集之最小覆盖的成员，同时也证得与之相似的 3 个准完备集均为最小覆盖的成员。

(二)对于第 2 类至第 22 类的 G_4，同理，只需为每一类的某一个准完备集构造一个函数 $f_i(x,y,z)$，其中 $i=2,\cdots,22$，使得函数 $f_i(x,y,z)$属于对应的 $T(G_4)$，且 $f_i(x,y,z)$不属于除对应 $T(G_4)$外的其它 270 准完备集。而这 21 个函数都是存在可求的，所以第 2 类至第 22 类的 39 个准完备集也均为最小覆盖成员。

结束语 本文证明了 $m=4$ 时的 42 个不可剔除的正则可离函数集都是最小覆盖成员，从而定出了部分四值逻辑中保四元正则可离关系的准完备集之最小覆盖成员。至此，部分四值逻辑中准完备集的最小覆盖已彻底解决，这为 P_4^* 中 sheffer 函数的判定与构造奠定了良好的基础，并为部分 K (>4)值逻辑理论的研究提供了帮助。

参 考 文 献

[1] Schofield P. Independent conditions for completeness of finite algebras with a single generator[J]. J. London Math, 1969, 44: 413-423

[2] Kudrjavcev V B. The coverings of precomplete classes of K-valued logic(Russian)[J]. Diskr—etnyi. Analiz, 1970, 17: 32-44

[3] 刘任任. 部分三值逻辑中准完备集之最小覆盖[J]. 湘潭大学：自然科学学报，1991，13(2)：158-164

[4] Liu Renren. Some results on the decision for Sheffer functions in partial K-valued Logic(II)[C]// Proceedings of the 28th International Symposium on Multiple-Valued Logic. IEEE Computer Society Press, 1998: 77-81

[5] Liu Renren. Some results on the minimal coverings of precomplete Classes in partial K-valued logic functions[C]// IEEE international conference on systems, Man & Cybernetics. 2003, 1: 2645-2650

[6] Liu Renren. On the categorizing of simply separable relations in partial four-valued logic[C]// Advances in Natural Computation, Lecture Notes in Computer Science. Springer-Verlag, 2005, 3612: 1251-1256

[7] Liu Renren. On the Categorizing of Fully Symmetric Relations in Partial Four-valued Logic[C]// Advances in Natural Computation, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2006, 4223: 286-289

[8] 刘玉珍，刘任任. 部分 K 值逻辑中正则可离函数集的一些结果[J]. 计算机工程与应用，2006，9：48-49，72

[9] 刘玉珍，刘任任. 部分 K 值逻辑中最小覆盖之判定的一些结果[J]. 计算机工程与应用，2007，43(23)：38-39，50