多agent规划领域中的观察信息约简

伍 选 文中华 汪 泉 常 青
(湘潭大学信息工程学院 湘潭 411105)

摘 要 观察信息约减是近年来不确定规划中的研究热点,但研究集中于单个agent的环境,在多agent规划环境下的研究不足。面对多agent环境下的规划问题,设计了一种用于不确定规划领域中多agent求解协同规划解的ORMAP算法。该算法首先根据基于模型检测的不定规划中的状态分层思想,将问题领域的所有状态进行分层,以此来减少不同的agent的冲突,再利用以最小代价优先的回溯法搜索协同规划解,同时在解的搜索过程中选择最小的观察信息集,使求出的协同规划解在众多符合条件的协同规划解中所需要的观察信息最少或接近最少,这样就达到了信息约简的目的。最后通过实验证明,在考虑了观察信息约简的限制条件后,这种算法的效率较高。

关键词 多agent,智能规划,不确定规划,观察信息约简,状态分层
中图法分类号 TP18 **文献标识码** A

Observation Reduction in Multi-agent Domain

WU Xuan WEN Zhong-hua WANG Quan CHANG Qing
(College of Information Engineering, Xiangtan University, Xiangtan 411105, China)

Abstract Observation information reduction is a hot area of the uncertainty planning research in recent years, but these researches concentrate on the single agent's environment, and the planning problem related to observation information reduction in the multi-agent domain is lack of researching. Confronted with the planning problem in the multi-agent domain, this paper designed an ORMAP algorithm which can find a collaborative planning in the nondeterministic multi-agent domain. At first, the ORMAP algorithm layers all states in the problem domain according to the model-based hierarchical states thought in order to avoid the conflicts between different agents. Then, it searches the collaborative planning solution with the method of the backtracking prior to minimum cost, meanwhile reduces the observation information. At last, a cooperative planning solution can be obtained and it is the one which needs least amount of observation information in all cooperative planning solution to the problem domain, so that it reaches the point. Finally, the experiment shows the efficiency of this algorithm is higher after considering the constraints of the observation information reduction.

Keywords Multi-agent, Intelligent planning, Uncertainty planning, Observation information reduction, Hierarchical state

1 引言

智能规划问题的研究,其范围不断在扩大。在2010年的ICAPS上专门讨论了多agent规划(MAP)问题。多agent规划问题的主要困难在于agent间的合作,在寻找其规划解的过程中,单个agent往往只考虑自身的执行情况,缺乏全局观,且系统环境也在动态变化。因此解决让多个agent在互不干扰的情况下,能够有一定的合作关系,共同完成一个任务这一难题具有重要的意义。目前,在多agent规划领域中的研究都集中在其确定动作的研究,即执行一个动作可达到的状态是确定的。主要的方法有:以A*、WHCA*、LRA*为代表的启发式搜索[1],类似蚁群算法、遗传算法的智能算法[2],与博弈论结合的算法[3],引入一定社会规则的算法[4],基于协商机制的合作求解算法[5]等等。然而对于不确定动作路径协同规划的求解还是有待研究的问题。由于规划领域中存在不确定动作,agent在执行不确定动作时到达的状态不唯一,这使得求协同规划的过程更加复杂。

2007,IJCAI上第一次提出了观察信息约简的概念[6],后来又有研究者在这方面进行了研究。其中有研究者提出了构造最小观察变量集和最优观察变量集的通用方法[7];基于一致性规划的构造最小观察变量集的方法[8];基于动作对的观察信息约简[9]和基于上下文的观察信息约简[10]。在不确定规划领域中,由于存在不确定动作,因此控制器在执行一个不确定动作后不知道agent当前所在状态,从而就不知道当前应该执行的动作,使得控制器不能继续执行,导致执行失败。通过观察信息就可以判断控制器在执行一个不确定动作后agent的所在状态,然而在执行过程中,并不需要所有观察信息。去除不必要的观察信息,达到减少开销的目的。

到稿日期:2013-08-06 返修日期:2014-01-04 本文受国家自然科学基金项目(61070232,61272295)资助。
伍 选(1988-),男,硕士生,主要研究方向为智能规划,E-mail:raty01@sina.com;文中华(1966-),男,博士,副教授,主要研究方向为智能规划、图论;汪 泉(1989-),男,硕士生,主要研究方向为智能规划、云计算;常 青(1987-),男,硕士生,主要研究方向为智能规划。

在规划解未知的不确定多 agent 规划领域中，就观察信息约简这一问题展开研究。在不确定多 agent 规划领域中，求解协同规划解是一个非常热点的问题。由于执行过程中存在不确定动作，agent 在执行不确定动作后会到达不同的状态，这使得协同规划的过程更加复杂，而在此基础上进行观察信息约简也将非常复杂。本文针对此问题设计了 ORMAP 算法，算法利用状态分层的思想对整个领域进行状态分层，找出可能发生冲突的状态，提高搜索效率。再利用以最小代价优先的回溯法搜索协同规划解，并且在搜索解的过程中进行观察信息约简，最后得到的协同规划解是众多符合条件的协同规划解中所需观察信息较少的，并且通过实验和分析可以看出，算法的执行效率是比较高的。

2 相关定义

定义 1(不确定多 agent 规划领域) 规划领域是一个不确定的状态转移系统，即执行动作后的状态是不确定的，可能是一个状态，也可能是一个状态集合。

$\Sigma=\{S,n,A_{i=1\cdots n},R\}$

其中，S 是有限状态集，n 是 agent 的数目，A_i 代表 agent$_i$ 的有限动作集，$R\in S'\times A\times S''$是一个不确定的转移关系，其中 $A=A_1\times\cdots\times A_n$，$S''\subseteq S$ 为当前所有 agent 的状态集合，$S''\subseteq S$ 为所有 agent 执行相应动作后的状态集合。

定义 2(不确定多 agent 规划问题) 规划问题 $P=\{\Sigma, I_{i=1\cdots n},G_{i=1\cdots n}\}$，$\Sigma$ 是一个规划领域，$I_i\in S$ 是 agent$_i$ 的可能初始状态集合，$G_i\in S$ 是 agent$_i$ 的目标状态集合。

定义 3(不确定多 agent 规划解) 对于一个规划问题 P 的规划解 $\pi=\{\pi_1,\pi_2,\cdots,\pi_n\}$，其中 π_i 是 agent$_i$ 的规划解，规划解 π 保证任意一个 agent 在执行动作序列的过程中不会出现冲突的情况，并且能使每个 agent 都能到达自己的目标状态(所有 agent 是在同一时间执行动作，没有先后顺序)。

定义 4(观察函数) 设 S 是一个状态转换系统的状态集，V 是一个有限的观察变量集合。对观察变量 $v\in V$ 的赋值是函数 $\chi:S\times V\rightarrow\{T,\perp\}$。

在完全可观察的情况下，任何不同的状态之间都是可分辨的，即$(\exists v\in V)(\chi(s,v)\neq\chi(s',v))$。而在完全不可观察的情况下，任何状态之间都是不可分辨的，即$(\forall v\in V)(\chi(s,v)=\chi(s',v))$。

定义 5(需要区分的状态对集合) 在执行规划解的过程中，控制器需要根据观察变量的值通过区分必要的状态对以确定当前 agent 所在状态，这些状态对组成的集合就是需要区分的状态对集合 fsp。

定义 6(最小观察变量集合) 能够用最少的观察变量保证规划解的执行的观察变量集合，就称为最小观察集合 V_{obs}。

直观地说，假设 fsp 是 n 个状态对的集合，V 是 m 个观察变量的集合。存在 V_{obs}(保证规划解的正确执行)$\subseteq V$，若对于任意一个 V'(保证规划解的正确执行)$\subseteq V$，都有 $|V_{obs}|<|V'|$，则 V_{obs} 是一个最小观察变量集合。

图 1 是一个简单不确定多 agent 领域 Σ，假设有一个规划问题 $P=\{\Sigma,I_{1,2},G_{1,2}\}$，其中 $I_1=\{s_1\}$，$I_2=\{s_2\}$，$G_1=\{s_6\}$，$G_2=\{s_7\}$。那么 $\pi=\{\pi_1,\pi_2\}$是这个规划问题的一个协同规划解，其中 $\pi_1=\{(s_1,a_2),(s_3,a_7),(s_4,a_8)\}$，$\pi_2=\{(s_2,a_4),(s_5,a_9)\}$。

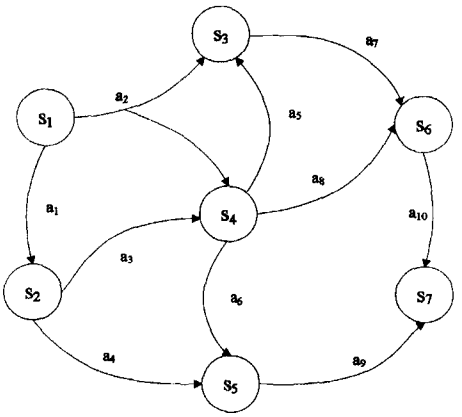

图 1 一个简单不确定多 agent 领域

3 观察信息约简

观察信息约简即对信息筛选，将多余的信息去除，只留下有用的信息。在已知规划解 π 的前提下，观察信息约简分两步完成，第一步是寻找需要区分的状态对集合，可通过不确定动作所到达的状态集合来寻找，控制器之所以不能判断当前 agent 所在的状态，是因为 agent 执行某个不确定动作后到达的是一个状态集合而不是一个状态，状态集合中的任意一个状态都有可能是 agent 执行不确定动作后所在的状态，所以通过区分这个集合中两两状态组合的状态对集合就能确定当前 agent 所在状态；第二步是利用需要区分的状态对集合找出最小或近似最小的观察变量集合。在第一步找到 fsp 集合后，利用贪心法对观察变量集合进行约简，得到最小观察变量集合 V_{obs}。贪心法求解的主要思路是优先选择可以区分状态对最多的观察变量，如此循环，直到所有需要区分的状态对都被区分完为止，得到的就是一个近似最小的观察变量集合。贪心法求解并不能保证所求出的观察变量集合是最小的，得到的只是一个近似最优集合。

在执行 π 的过程中，只需要获取 V_{obs} 中观察变量的值就能保证 π 的正确执行，这样就能达到减少开销的目的。

以上是观察信息约减的基本思想。文献[6]首次提出了观察信息约减的概念，但是其只涉及单个 agent 的强规划问题。其后的研究扩展到了不定规划的强循环规划[11]、弱规划。而我们尝试在多 agent 领域讨论这个问题。由于涉及到多 agent 的冲突与合作，问题也变得更加复杂。

4 ORMAP 算法

在不确定多 agent 规划领域中，求解规划解是一个很复杂的问题，而在此基础上进行观察信息约简就更加复杂。因此本文设计了一个求解协同规划解的算法 ORMAP (Observation Reduction for Multi-agent Planning)，并在求解过程中采用边求规划解边约束观察变量的方法来对观察信息进行约简，最后得到一个多 agent 规划领域的协同规划解，并且执行这个协同规划解所需要的观察信息较少。

4.1 算法基本思想

ORMAP 算法主要分为两个阶段，第一个阶段是在多 agent 规划领域中进行状态分层[12]，分层后就能快速地找出无冲突的协同规划。第二个阶段是根据状态分层过程中求出的当前状态 agent 可能冲突列表，通过以最小代价优先的回溯

法找出一个使观察变量集合近似最小的协同规划解，并保存近似最小观察变量集合，返回这个协同规划解。

在多 agent 规划领域中，agent 在执行规划解的过程中如何避免冲突情况是必须要解决的问题，其中不确定性动作又会带来更大的变数。因此首先进行状态分层，文献[12]介绍了在单个 agent 领域如何进行状态分层，我们将其扩展到多 agent 领域，设计出多 agent 的状态分层算法，找出多 agent 的可能发生冲突的状态。

分层原则就是将到达目标状态步数最短并且相同的状态划为同一层，主要核心思路如下。

1. 把 agent 的目标状态集合定为第一层，即 $S_1=S_g$。

2. 第二层是从不属于 S_1 的状态集合里找出状态 s，并且使得 s 在执行某个动作 a 后，所到达的状态集合是 S_1 的子集，即 $S_2=\{s|s\notin S_1 \text{ and } \gamma(s,a)\subseteq S_1\}$。

3. $S'=S_1\cup\cdots\cup S_{i-1}$，第 i 层则是从不属于 S' 的状态集合中找出状态 s，并且使得 s 在执行某个动作 a 后，所到达的状态集合是 S' 的子集，即 $S_i=\{s|s\notin S' \text{ and } \gamma(s,a)\subseteq S'\}$。

4. 若找出的状态集合满足 I 属于 S'，则返回成功。

通过以上的分层步骤可以求出单个 agent 的状态分层，再把每个 agent 的状态分层进行合成，得出整个领域的状态分层。用 $agent_i-layered[i][j]$ 存储 $agent_i$ 在第 j 层的分层信息，用 $layered[j]$ 存储整个领域第 j 层的分层信息，用 $heuristic[layers][s]$ 存储领域状态分层第 $layers$ 层中状态 s 可能存在的 agent 个数，也就是说当 $heuristic[layers][s]$ 大于 1 时，可能有 $heuristic[layers][s]$ 个 agent 同时到达状态 s，具体步骤如下。

5. 先对每个 agent 分别进行状态分层，把 agent 所在的层数的状态信息存储在对应的 $agent_i-layered[i][j]$ 中。

6. 为了方便后面计算，将每个 agent 的初始状态定为第一层，交换每个 agent 的分层信息。

7. 把每个 agent 的分层信息合并成整个领域的分层信息。在合成过程中，若某个 agent 当前层的状态 s 已经出现在领域的对应层数上，那么就说明这个状态 s 可能会引起 agent 之间的冲突，所以把对应的代价信息 $heuristic[layers][s]$ 值增加 1，在搜索规划解时能够使用最小代价优先的思路加速搜索。

得到整个领域的状态分层信息后，就可以利用 *heuristic* 中的值搜索协同规划解，*heuristic* 的值越小就代表发生冲突的概率越小，所以在搜索过程中，优先选择 *heuristic* 值小的路径能提高搜索效率。

如果在该领域中，假设 agent 个数为 m，状态个数为 n，可知该算法平均时间复杂度为 $O(m*\log n)$。最坏情况下时间复杂度为 $O(n*m)$。

通过进行状态分层，有利于迅速求解规划解，主要是利用上面求得的 *heuristic*，将其作为最小代价优先选择的依据，并进行回溯搜索，实现函数为 Multi-agent-Planning。函数可以求出多 agent 规划领域的协同规划解，并保证所求解是使得观察变量集合最小的规划解。

对观察变量集合进行约简的方法是以不确定动作来寻找需要区分的状态对集合以及利用贪心法找到最小观察变量集合。控制器之所以不知道当前 agent 的状态，是因为在执行不确定动作后，agent 到达的是一个状态集合而不是一个状态，那么就可以通过区分这个状态集合中两两状态组成的状态对，确定当前 agent 的状态。这些状态对组成的集合就是需要区分的状态对集合，利用贪心法优先选择能够区分状态对数目多的观察变量，直到所有需要区分的状态对都已经被区分完后，得到的就是一个近似最小观察变量集合。

4.2 算法实现

该算法的框架结构如下。

```
1. Algorithm ORMAP(P)
2. Multi-Hierarchical(P);
3. π_F=Multi-agent-Planning(agent,V,χ);
4. return π_F;
```

函数 Muti-agent-Planning 采用多次搜索规划解的形式，把每次搜索的规划解与已求出的最优解比较，若当前的规划解比已求出的最优解好，则保存结果，继续搜索，否则退出搜索，返回最优解。

其主要函数的实现如下所示。

```
1. Function Multi-agent-Planning(agent)
2.   while(true)
3.     while(true)
4.       if(all agents arrive target) break;
5.       searchagent=Search(searchagent);
6.     π'=GetAction(agent);
7.     if(V_obs=(φ || |curV|<|V_obs|)
8.       V_obs=curV;
9.       π=π';
10.    else
11.      break;
12.  return π;
```

在冲突时就进行回溯，直到无冲突状态出现为止。跳出循环体后，在第二层循环内得到规划解 π' 后，求出其对应的最小观察变量集合 *curV*，若 *curV* 集合的大小小于 V_{obs} 的大小，则重新给 V_{obs} 赋值，并更新规划解 π，继续执行第二层循环体内的搜索，搜索下一个规划解，否则 V_{obs} 就是最小观察变量集合，跳出第一层循环体，并返回对应观察变量集合 V_{obs} 的整个领域协同规划解 $\pi=\{\pi_1,\pi_2,\cdots,\pi_i,\cdots\}$。

函数 Search 是整个算法的核心部分，通过搜索和最小代价优先选择，找出当前的最优解，再通过回溯找出整个领域的规划解，并在搜索过程中判断选择的动作是否能使观察变量集合减小或者比已求出的最小观察变量集合小。其伪代码如下所示。

```
1. Function Search(searchagent)
2.   if(searchagent.index arrive target)
3.     return nextagent(searchagent);
4.   i=searchagent.index;
5.   j=searchagent.layer;
6.   for each s∈agent[i][j].source
7.     for each a∈Heuristic(s,j+1)
8.       if (γ(s,a)⊆agenti-hierarchical[j+1]&& γ(s,a)∩agent[i]
         [j].usedStatus=φ)
9.         if (|γ(s,a)|>1 && V_obs=φ && |curV|>=|V_obs
           |))
10.          continue;
11.        agent[i][j].usedStatus ∪= γ(s,a);
12.        agent[i][j].selectAction ∪= (s,a);
```

```
13.          break;
14.      if(it has conflicted)
15.        FindConflictingagent(s,i,j);
16.        if(the conflict is sloved)
17.          return Search(searchagent);
18.        else
19.          return false;
20.    store the source to agent[i][j+1]. source;
21.    store the target to agent[i][j+1]. target;
22. return nextagent(searchagent);
```

在伪代码中，$\gamma(s,a)$为在状态 s 执行动作 a 后得到的状态集合。$agent_i$-*hierarchical*[j +1]表示 $agent_i$ 在第 j+1 层的分层信息，具体过程如下。

第 2、3 行：若当前搜索的 agent 已经到达目标状态，则继续搜索下一个 agent。函数 nextagent 的作用就是找出下一个应该搜索的 searchagent，即当前搜索的 agent 标号为 n 时，下一个就搜索标号为 1、搜索层数加 1 的 agent；当 agent 标号不为 n 时，下一个搜索的是标号为 n+1、层数不变的 agent。

第 4—19 行：对于 $agent_i$ 第 j 层搜索初始状态集合中的每一个状态 s，都必须找到一个对应的动作 a 执行。动作 a 需要满足两个条件，一个是状态 s 执行动作 a 后所得到的状态集合必须是 $agent_i$ 在状态任务分层后第 j+1 层状态集合的一个子集；另一个就是状态 s 执行动作 a 后所得到的状态集合必须与 *agent*[i][j]已用过的状态集合不重复。函数 Heuristic(s,j+1)的作用是求出状态 s 按照 *heuristic* 进行筛选后，根据代价值得到一个优先选择动作列表，排在该列表前面的动作是 *heuristic*[j][s]中值较低的。选出一个动作 a 后，判断这个动作是否是不确定动作，若是不确定动作，则判断当加入这个不确定动作后，对当前需要区分的状态对集合进行观察变量集合约简，将得到的观察变量集合与最小观察变量集合比较，若是大于或等于，则不选择这个动作，继续寻找下一个动作。其中 Pairs(S)是集合 S 中两两状态组合成的状态对，并返回这个状态对集合，即 Pairs(S)＝{(s,s')|s∈S,s'∈S,s≠s'}((s,s')和(s',s)视为同一对状态对)。若找到了满足条件的动作 a，则把执行动作 a 后的状态集合加到已用过的状态集合中，并把这个动作加入到 $agent_i$ 的第 j 层已使用过的动作集合中。如果没有一个能够满足条件的动作 a，那么通过 FindConflictingagent(s,i,j)找出与 $agent_i$ 在 j+1 层相冲突的 agent，并从该 agent 开始搜索，判断是否能解决该冲突情况。若能够解决该冲突就重新搜索 $agent_i$ 的第 j 层，否则，返回失败，表示 $agent_i$ 的第 j 层在状态 s 上找不到一个可执行的动作 a。

第 20—22 行：将 $agent_i$ 的第 j 层所有初始状态 s 执行对应动作 a 到达的所有状态的集合定为下一层搜索的初始状态集合，并把目标状态集合复制到下一层，返回下一个搜索的 agent，即 nextagent(*searchagent*)。

设 agent 个数为 m，状态个数为 n，显然最坏情况下该算法的时间复杂度为 $O(m*n^2)$。

函数 FindConflictingagent 是找出 agent 状态 s 上，执行动作 a 后与其出现冲突的 agent，通过修改其执行的动作，判断是否能够解决冲突情况，其具体过程如下所示。

```
1. Function FindConflictingagent(s,agent_k,depth)
2.   for each s_j∈γ(s,a_i)
3.     if(flag[depth+1][s_j]≠0)
4.       agent=flag[depth+1][s_j];
5.       flag[depth+1][s_j]=0;
6.       DeepSearch(agent,depth);
7. if(search is successful) return success;
8. flag[depth+1][s_j]=agent;
9. return DeepSearch(agent_k,depth-1);
```

函数 FindConflictingagent 有 3 个初始化参数，表示第 k 个 agent 在第 *depth* 层状态 s 上执行某个动作后，会出现冲突情况。主要思路是对于每一个属于执行对应动作 a 后得到状态集合中的状态 s，通过标记数组 *flag*，找出导致冲突的 agent，从这个 agent 的第 *depth* 层重新搜索以求避开冲突情况。若搜索成功则返回，否则回溯到 $agent_k$ 上一层，重新选择一个新的动作执行。

通过这 3 个函数，可以得到一个最小观察变量集合和一个多 agent 协同规划解。

5 实验及分析

根据伪代码设计实验求解不确定多 agent 规划领域中的协同规划解，并且使得求出的协同规划解所需要的观察信息较少。通过实验可以验证 ORMAP 算法的正确性和有效性。

本实验的实验环境为双核 T2300 1.66GHz CPUs，3.00GHz处理器，2GB 内存，Microsoft Windows XP Professinal 操作系统，Microsoft Visual Studio 2008，算法用 C++语言编程实现。

使用随机程序生成规划领域、规划问题和对应的观察变量集合，保证每次所生成的规划领域是不同的。现分别对状态数目在 50,70,80,100,150，agent 数目为 2,5,6,8,10 时，执行算法 ORMAP 来求解规划领域中的协同规划解，并记录求解协同规划解所用的时间。每类规划领域即(状态数目，agent数目)随机生成 300 组数据，所消耗的时间为执行过程中的平均时间。

表 1 记录了不同状态数对应不同 agent 数目执行 300 组数据后所用的平均时间，其中第一行代表状态数目，第一列代表 agent 数目，行与列的交叉处表示规划领域(状态数目，agent 数目)执行 300 组数据所用的平均时间(单位：ms)。

表 1 算法 ORMAP 执行时间表

agent \ 状态	50	70	80	100	150
2	10.65	17.96	21.42	30.56	47.19
5	28.67	38.82	43.82	73.85	121.20
6	35.70	54.21	61.06	82.73	145.06
8	44.16	66.41	75.15	103.28	203.66
10	50.07	77.77	89.97	123.25	267.18

从表 1 可以看出在 agent 数目一定的条件下，状态数目越多，算法 ORMAP 耗时就越多；同理，在状态数目一定的条件下，agent 数目越多，算法 ORMAP 耗时也越多。规划领域的规模越大，算法执行的时间也就越长。

算法执行的时间分为两个部分，一部分是状态分层的时间，另一部分是求解规划解的时间。在状态数目不是很多的情况下，效果不是很明显。但是在状态数目很多的情况下，进行状态分层后可以在很大程度上减少重复计算，提高搜索规划解的效率。

(下转第 192 页)

素的影响作用，突出对底层细节方面的策略支持。实际操作时，可将此方法与传统方法相结合，两种方法互相辅佐，各取所长，从高到低制定企业的规划策略。

结束语 本文应用 ANP 方法对企业应用系统的云迁移适合度进行评价，并结合 ANP 中的超级矩阵提出了挖掘系统迁移过程中关键影响因素的分析方法，用于支持完善企业演进策略。其评价模型会因决策人认识的不同而有所差异。本文的重点是将 ANP 方法与企业战略相结合，提出一种支持企业战略演进的新思路。但由这一方法所提取的分时段关键影响因素与企业演进路线之间的契合性还有待进一步论证；同时，由于方法本身缺乏实践，未能与企业进行互动，因此缺少了相关数据来证明方法给企业所带来的效益，需要在后期对相关方法进行更深层次的探索，这些也都是下一步研究工作的重点。

参 考 文 献

[1] 邓仲华，汪宜晟，李志芳，等. 信息资源云服务的质量评价指标研究[J]. 图书与情报，2012(4)：12-15

[2] 付超，桂鹏飞. 基于证据推理的云计算服务适应性评估[J]. 计算机应用研究，2012，29(11)：4304-4308

[3] Davoudi M R，Sheykhvand K. Enterprise Architecture Analysis using AHP and Fuzzy AHP[C]// Proceedings of 2011 4th IEEE International Conference on Computer Science and Information Technology(ICCSIT 2011). VOL07，2007：202-207

[4] Ginty R M，Carrasco R，Oddershede A，et al. Strategic foresight using an analytic hierarchy process：environmental impact assessment of the electric grid in 2025[C]// The 12th International Symposium on the Analytic Hierarchy Process，2013. Kuala Lumpur，Malaysia，2013

[5] Xu Jun-li，Liu Lei，Shao Zeng-zhen. Research on the evolution of enterprise clusters based on theory of MAS[C]// Proceedings of 2011 4th IEEE International Conference on Computer Science and Information Technology(ICCSIT 2011). VOL01，2011：676-680

[6] Ke Xing，Duan Ning-dong. Analysis on the evolution mechanism of enterprise organizational capability system from the perspective of CAS theory[C]//第七届中国管理学年会组织与战略分会场论文集. 天津，2012

[7] 冯秀珍，郝鹏. 云计算环境下的信息资源云服务模式研究[J]. 计算机科学，2012，39(10)：110-114

[8] Saaty R. A validation of the effectiveness of inner dependence in an ANP model[C]// The 12th International Symposium on the Analytic Hierarchy Process，2013. Kuala Lumpur，Malaysia，2013

[9] 孙宏才，田平，王莲芬. 网络层次分析法与决策科学[M]. 北京：国防工业出版社，2011

[10] Saaty T L. Principia Mathematic Decernendi：Mathematical Principles of Decision Making[M]. Pittsburgh：RWS Publication，2010

[11] 孙超平，杨善林. 战略 SWOT 决策模型的构建及其实证研究[J]. 系统仿真学报，2009，21(3)：868-872

(上接第 179 页)

结束语 在规划解未知的多 agent 规划领域进行观察信息约简，主要面临两大难题，即如何求出对应的协同规划解和如何进行观察信息约简。针对此难题，本文设计了算法 OR-MAP，首先对多 agent 领域进行状态分层，将分层信息通过以最小代价优先的回溯法求出协同规划解。然后进行观察信息约简，对已经搜索到的协同规划解，记录其观察变量集合大小，若在下次搜索过程中，部分规划解所需要的观察变量集合的大小大于或等于已经存在的观察变量集合的大小，则中止当前搜索。如此反复，直到找不到协同规划解为止。OR-MAP 算法可以在规划解未知的条件下，找出多 agent 规划领域的一个协同规划解，并使得执行这个协同规划解所需要的观察信息较少。

将来还可以继续在这几个方面进行研究：1)在求多 agent 规划领域的协同规划解时，分层的方法有待提高，算法中使用的方法不能求出含有循环结构的情况，在今后的工作中可以继续这方面的研究；2)本文是在完全可观察条件下进行观察信息约简，那么如何在部分可观察条件下进行观察信息约简可以做进一步的研究；3)若通过观察变量得出的观察信息是错误的，那么如何设计一个具有容错能力的算法来进行观察信息约简。

参 考 文 献

[1] Standley T. Finding Optimal Solutions to the Multi-agent Path-finding Problem Using Heuristic Search[C]// Proceedings of the 2010 AAAI. Atlanta，2010：173-178

[2] Jansen R，Sturtevant N. A new approach to cooperative path-finding[C]// Proceedings of the 2008 AAMAS. Estoril，2008：1401-1404

[3] Larbi R B，Konieczny S，Marquis P. Extending Classical Planning to the Multi-agent Case：A Game-theoretic Approach[C]// Proceedings of the ECSQARU-07. Hammamet，2007：731-742

[4] Wang K-H C，Botea A. Fast and Memory-Efficient Multi-agent Pathfinding[C]// Proceedings of the ICAPS-08. Sydeny，2008：380-387

[5] Huang Wei，Zhang Dong-mo，Zhang Yan，et al. Bargain over Joint Plans[C]// Proceedings of the PRICAI-10. Hanoi，2010：608-613

[6] Huang Wei，Wen Zhong-hua，Jiang Yun-fei，et al. Observation reduction for strong plans[C]// Proceedings of the 20th International Joint Conference on Artificial Intelligence(IJCAI-07). Hyderabad，2007：1930-1935

[7] 饶东宁，蒋志华，姜云飞，等. 对不确定规划中观察约简的进一步研究[J]. 软件学报，2009，20(5)：1254-1268

[8] 周俊萍，殷明浩，谷文祥，等. 部分可观察强规划中约减观察变量的研究[J]. 软件学报，2009，20(2)：290-304

[9] Huang Wei，Peng Hong. Observation Reduction for State-action Tables[C]// Proceedings of the International Conference on Computational Intelligence and Security. Beijing，2009：10-14

[10] Huang Wei，Wen Zhong-hua，Jiang Yun-fei，et al. Structured Plans and Observation Reduction for Plans with Context[C]// Proceeding of 21th International Joint Conference on Artificial Intelligence(IJCAI 09). Pasadena，2009：1721-1727

[11] 常青，文中华，胡雨隆，等. 强循环规划的观察信息约简[J]. 计算机工程与应用，2012，48(2)：148-150

[12] 文中华，黄巍，刘任任，等. 模型检测规划中的状态分层方法[J]. 软件学报，2009，20(4)：858-869