基于 DFS 的多 Agent 动态任务分配算法研究

张 瑜, 李凡长
(苏州大学计算机科学与技术学院, 江苏苏州 215006)

摘 要: 任务分配问题是 MAS 的重要研究内容之一, 对于任务分配这一复杂问题, 很多研究者从不同的角度提出各种行之有效的算法. 这些算法对于确定的环境是有效的, 对于不确定的动态的环境存在不足. 本文针对具有动态模糊特性的任务环境进行研究, 借助动态模糊集理论, 给出了相关的多 Agent 动态任务分配算法. 实例测试表明, 算法模型可以合理地模拟 MAS 系统中任务分配的运行过程, 并获得最优的任务分配策略和良好的任务实现效果.

关键词: 多 Agent; 任务分配; 动态模糊集; 强化机制; 遗传算法
中图分类号: TP18 文献标识码: A 文章编号: 0372-2112 (2009) 11-2551-06

Research on Multi-Agent Dynamic Task Allocation Algorithm and Based on Dynamic Fuzzy Set

ZHANG Yu, LI Fan-zhang
(*School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu* 215006, *China*)

Abstract: Task allocation problem is one of the import researches of multi-agent system. For the complex problem, many researchers from different points of view put forward a variety of effective algorithms. These algorithms are effective for the determinative environment, but for the uncertain and dynamic environment are deficient. In this paper, fuzzy characteristics of dynamic task environment are researched, using dynamic fuzzy set theory, given the relevant multi-Agent dynamic task allocation algorithm. Examples of the test show that the algorithm model can be a reasonable simulation of MAS system operation task allocation process, and have achieved the optimal task allocation strategy and a good effect.

Key words: multi-agent; task allocation; Dynamic Fuzzy Set (DFS); reinforcement mechanism; genetic algorithm

1 引言

在一个分布式多 Agent 系统中, 当某个 Agent 发现自己不具备独立的一项任务的能力时, 并且该任务本身根据不同的余数关系和计算要求, 一个任务往往可以被分解成多个子任务时, 它可以与其他 Agent 组成团队, 并通过团队合作来完成各子任务在 Agent 团队成员之间的分配. 任务分配问题定义为将分解好的若干个子任务分配到各个 Agent 中, 并根据给定的子任务间的数据约束关系合理的安排好每个 Agent 子任务执行顺序[1]. 一般的说, 多 Agent 任务分配团队合作需要经历 3 个任务阶段: 任务分配、问题求解和结果评价[2]. 假设多 Agent 系统中有 n 个 Agent, 每个 Agent 可以承担的任务有 m 种, 但每次只能承担一个任务, 并记作第 i 个 Agent 所承担的任务为 t_i.

定义 **1**(任务分配) 待分配的任务为 T, 完成任务需要具有 k 种技能 $C_t(t=1, 2, \cdots, k)$, 且对技能 C_t 的能力要求$(\underleftarrow{b}, \vec{b})(T, C_t)$, 任务共需要有 m 个 Agent 组成团队完成, 若 $m>1$, 任务分配问题就是根据这些任务属性和 Agent 状态把任务分配给最适合的 Agent 团队执行.

任务分配是一类公认的 NP 问题[3,4], 传统的方法大多采用一类集合划分和覆盖的理论[5]或线性规划逼近的方法[6,7]. 近年来, 许多学者将遗传算法等进化计算方法应用到任务分配问题中[1,9], 任务分配进化计算方法在大多数情况下比上述传统的方法可获得较好的问题求解结果. 但是目前所采用的进化方法中, 主要利用带某种知识的启发式方法或是将多种进化方法进行综合. 这些方法都是通过提高进化算法本身的效率来进行问题求解或对进化算法进行并行实现, 其共同点是单个种群的进化或多种群缺乏合作而并行独立进化. 当问题解是有多部分组成时, 个体解完整编码方式对好的部分解的利用可能会被其他较差的部分解所覆盖, 从而使得算法易于陷入局部最优点而过早收敛, 或使得算法的进化处于振荡状态而停滞不前.

收稿日期: 2008-01-03; 修回日期: 2009-01-21
基金项目: 国家自然科学基金(No. 60775045); 江苏省自然科学基金(No. BK2005027; No. BK2002040)

在这个问题上引入一种目标驱动的自适应能力的很强的强化学习技术, 从而使得遗传算法的本身体现出明晰的目标性. 在强化学习中, Agent 发现好的行动策略对应遗传算法中发现好的种群结构模式; Agent 动作选择对应遗传算法的选择算子; 多 Agent 并发行动选择对应着遗传算法中的交叉和变异.

在这个融合的过程中, 多 Agent 在任务分配环境中如何选择动作行为, 如何确定其行为策略以及各个 Agent 自身的状态都具有动态模糊性的特征. 因此, 本文选择动态模糊集(Dynamic Fuzzy Set, DFS)描述工具来描述, 从而解决任务分配问题, 目的是充分的保留遗传算法的通用性和鲁棒性以及强化机制的自适应性和目标驱动性的特点.

2 基于 DFS 的多 Agent 任务分配的概念

2.1 基于 DFS 的强化学习的基本概念

在离散时间 t 内, $t=1, 2, 3, \cdots$, Agent 获得当前的环境状态$(\overleftarrow{s}, \overrightarrow{s})_t \in (\overleftarrow{S}, \overrightarrow{S})$, $(\overleftarrow{S}, \overrightarrow{S})$ 是所有可能的状态的集合. 根据这个状态 Agent 采取不同的行为$(\overleftarrow{a}, \overrightarrow{a})_t \in (\overleftarrow{A}, \overrightarrow{A})(\overleftarrow{s}, \overrightarrow{s})_t$, 其中$(\overleftarrow{A}, \overrightarrow{A})(\overleftarrow{s}, \overrightarrow{s})_t$ 是在状态集合$(\overleftarrow{S}, \overrightarrow{S})$ 下所有可能行为的集合. 每次在状态$(\overleftarrow{s}, \overrightarrow{s})_t$ 下执行一动作$(\overleftarrow{a}, \overrightarrow{a})_t$ 便可以得到相应的即时奖赏值$(\overleftarrow{r}, \overrightarrow{r})_t$, $(\overleftarrow{r}, \overrightarrow{r})_t \in (\overleftarrow{R}, \overrightarrow{R})$, 同时环境状态变迁到一个新的状态. 强化学习的目的就是要获得一个行动选择策略$(\overleftarrow{\pi}, \overrightarrow{\pi})_t((\overleftarrow{s}, \overrightarrow{s}), (\overleftarrow{a}, \overrightarrow{a}))$ 是在状态$(\overleftarrow{s}, \overrightarrow{s})_t$ 采取动作$(\overleftarrow{a}, \overrightarrow{a})_t$ 的可能概率, 从而使得 Agent 所选择的动作能够获得最大的环境奖赏值[11].

在强化学习中, 简单地将整个任务空间当作是 Agent 的行动策略搜索空间, 根据每个 Agent 的对于任务空间环境的行动奖赏来确定任务分配, 这种方法面临着需要维持一个庞大的行为空间, 这样使得每一个行动不可能被访问足够多次, 因此需对任务空间进行分割. 允许 Agent 忽略与当前任务无关的其他细节, 将任务空间操作$(\overleftarrow{Op}, \overrightarrow{Op})$ 抽象成若干子任务空间操作$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$, 并将这些子$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 作为一种特殊的动作操作加入到原来的$(\overleftarrow{Op}, \overrightarrow{Op})$ 中去. 那么一个$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 可以理解为完成某个子目标而定义在状态子空间上的按一定策略执行的动作序列.

给出$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 形式化定义[12], $(\overleftarrow{Op}^+, \overrightarrow{Op}^+) = <(\overleftarrow{\Psi}, \overrightarrow{\Psi}), (\overleftarrow{\pi}, \overrightarrow{\pi}), (\overleftarrow{\beta}, \overrightarrow{\beta})>$ 三元组表示, 其中初始状态集$(\overleftarrow{\Psi}, \overrightarrow{\Psi}) \in (\overleftarrow{S}, \overrightarrow{S})$ 为状态空间的子集. 一个$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 被激活当且仅当当前状态$(\overleftarrow{s}, \overrightarrow{s}) \in (\overleftarrow{\Psi}, \overrightarrow{\Psi})$ 时, 通常$(\overleftarrow{\Psi}, \overrightarrow{\Psi})$ 包含且仅包含该$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 经历的所有可能状态. $(\overleftarrow{\pi}, \overrightarrow{\pi})$ 是$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 的内部策略, 用于定义$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 被触发是在某个状态下选择某动作的概率分布. 如果一个$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 被执行, 依据策略$(\overleftarrow{\pi}, \overrightarrow{\pi})$ 来选择动作, $(\overleftarrow{\beta}, \overrightarrow{\beta}): (\overleftarrow{S}, \overrightarrow{S}) \to [(\overrightarrow{0}, \overrightarrow{0}), (\overrightarrow{1}, \overrightarrow{1})]$ 是中止条件, 定义了每个子状态空间中每个状态作为$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 的终止条件成立的概率. 若$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 当前的状态$(\overleftarrow{s}, \overrightarrow{s})$, 系统根据$(\overleftarrow{\pi}, \overrightarrow{\pi})((\overleftarrow{s}, \overrightarrow{s}), (\overleftarrow{a}, \overrightarrow{a}))$ 来选择下一个动作$(\overleftarrow{a}, \overrightarrow{a})$ 作用于环境, 环境状态由$(\overleftarrow{s}, \overrightarrow{s})$ 转移到$(\overleftarrow{s'}, \overrightarrow{s'})$, 最后根据$(\overleftarrow{\beta}, \overrightarrow{\beta})((\overleftarrow{s'}, \overrightarrow{s'}))$ 判断是否继续执行, 若当前的$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 执行终止了, 可选择另一个继续执行. 在该方法中, 每个$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 都有自己的行动策略, 它能使 Agent 有效地达到与该$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 相应的子任务空间. 在这个过程中我们是通过 Agent 学习自动地在整个任务空间中找到一组合适的子任务空间, 进而构建相应的$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$.

2.2 基于 DFS 的动态任务分配的一般流程

一般来说, 多 Agent 动态任务分配需要经历 3 个阶段: 任务分配、问题求解和结果评价. 那么我们认为多 Agent 动态任务分配动态过程是一个受激励机制控制的有序过程, 如下及图 1 所示.

Step1: 对需要执行的目标任务$(\overleftarrow{So}, \overrightarrow{So})$ 中的某一个相关子集$(\overleftarrow{So}^+, \overrightarrow{So}^+)$, 依据$(\overleftarrow{So}^+, \overrightarrow{So}^+)$ 的公共特性激活一个操作机制$(\overleftarrow{Op}, \overrightarrow{Op})$ 的子集$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$, 在$(\overleftarrow{Op}^+, \overrightarrow{Op}^+)$ 的作用下, 形成一个任务目标$(\overleftarrow{Y}, \overrightarrow{Y})$ 的子集$(\overleftarrow{Y}^*, \overrightarrow{Y}^*)$.

Step2: 对于任务目标$(\overleftarrow{Y}, \overrightarrow{Y})$ 中的元素$(\overleftarrow{y}_0, \overrightarrow{y}_0)$, 在执行算法 ER 的作用下, 有一个偏差信息 $E(\overleftarrow{y}_0, \overrightarrow{y}_0)$, 激励机制$(\overleftarrow{C}, \overrightarrow{C})$ 依据激活评价机制$(\overleftarrow{V}, \overrightarrow{V})$ 的一个子集$(\overleftarrow{V}^*, \overrightarrow{V}^*)$, 并作用于任务环境, 环境响应 $N(\overleftarrow{y}_0, \overrightarrow{y}_0)$ 与偏差信息 $E(\overleftarrow{y}_0, \overrightarrow{y}_0)$ 在$(\overleftarrow{V}^*, \overrightarrow{V}^*)$ 的作用下, 对$(\overleftarrow{y}_0, \overrightarrow{y}_0)$ 进行修正.

Step3: 对于待执行任务中的任一子集$(\overleftarrow{So}^+, \overrightarrow{So}^+)$, 其操作与评价结果为: $(\overleftarrow{V}, \overrightarrow{V})[(\overleftarrow{Op}, \overrightarrow{Op})(\overleftarrow{So}, \overrightarrow{So})] \Rightarrow (\overleftarrow{Y}^*, \overrightarrow{Y}^*) \subseteq (\overleftarrow{Y}, \overrightarrow{Y})$

Step4: 重复上述过程, 直到任务执行过程达到精度要求为止.

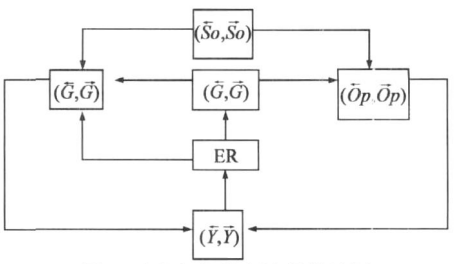

图1 动态任务分配流程描述图

3 基于 DFS 的多 Agent 任务分配算法

遗传算法是一种模仿生物进化过程的全局优化随机搜索算法, 具有在复杂空间求解问题近似最优的能力, 它的简单通用性和高度鲁棒性的特点, 使得它在任务分配问题中也得到了广泛的应用. 遗传算法是基于一个候选解群的迭代过程, 使用遗传算子和相关参数对任务空间进行搜索, 采用适应度来评价候选解的优劣, 但它并不能保证找到最优解. 强化学习是一种目标驱动的自适应能力很强技术, 具有强化机制的 Agent 能

够与环境进行交互, 根据环境的反馈来调整自己的行为策略. Agent 所采用的 ε- *greedy* 行动选择策略, 以 ε 的概率选择行动策略空间中的任一行动策略, 以 1- ε 的概率选择到目前为之最好的行动策略.

其实强化学习与遗传算法的结合自上世纪 80 年代后期就受到许多研究者的关注, 已有许多这方面的成果[13, 14]. 他们提出用强化学习的自适应控制遗传算子, 从深度、内在和本质上将这两者融合. 本文主要是根据这一思想将其应用到多 Agent 任务分配问题上了, 同时参考文献[14]中的算法结合任务分配中所面临的动态模糊性特性, 通过动态模糊集来表示算法的实现过程.

3.1　Agent 子任务分配执行算法

Step1: Initialize $D=(d_1, d_2, \cdots, d_n)$

Step2: For all $i=1, 2, \cdots, n; j=1, 2, \cdots, 2^{d_i}$

　　Initialize $(\overleftarrow{Q}, \overrightarrow{Q})_i(a_{i,j})=0$;

Step3: $k=0$;

Step4: Repeat

Step5: Choose$(\overleftarrow{a}, \overrightarrow{a})=((\overleftarrow{a}, \overrightarrow{a})_{1,jk}, (\overleftarrow{a}, \overrightarrow{a})_{2,jk}, \cdots, (\overleftarrow{a}, \overrightarrow{a})_{n,jk})$

　　using ε- *greedy*

　　GetRewards $(\overleftarrow{r}, \overrightarrow{r})=$ fitness(Agent$((\overleftarrow{a}, \overrightarrow{a}))$;

　　Rank$((\overleftarrow{r}, \overrightarrow{r})_1, (\overleftarrow{r}, \overrightarrow{r})_2, \cdots, (\overleftarrow{r}, \overrightarrow{r})_n)$;

Step6: If (terminal condition is satisfied)

　　execute step10

　　Else

　　execute step7

Step7: For all $i=1, 2, \cdots, n$

$(\overleftarrow{Q}, \overrightarrow{Q})_i(a_{i,jk})=(1-a_{i,jk})(\overleftarrow{Q}, \overrightarrow{Q})_i(a_{i,jk})+(a_{i,jk})(\overleftarrow{r}, \overrightarrow{r})+\gamma\max((\overleftarrow{Q}, \overrightarrow{Q})_i(a_{i,j}))$;

Step8: If (best_$(\overleftarrow{r}, \overrightarrow{r})>(\overleftarrow{r}, \overrightarrow{r})$)

　　Then $(\overleftarrow{r}, \overrightarrow{r})=$ best$(\overleftarrow{r}, \overrightarrow{r})$;

　　k= k+ 1;

　　execute step 4

Step9: If (terminal condition is satisfied)

Step10: Output best Agent

在 Step1 中对于任务空间进行划分, 因为简单的将整个任务空间当作是 Agent 的行动策略搜索空间, 根据每个 Agent 的对于任务空间环境的行动奖赏来确定任务分配, 这种方法面临着需要维持一个庞大的行为空间, 这样使得每一个行动不可能被访问足够多次, 因此对于任务空间进行分割. 任务空间分割直接反应出不同的结构搜索模式, 从而影响 Agent 的搜索结果, 与同时也直接决定着 Agent 行为策略空间的大小, 影响整个分配过程的时间和空间效率.

在给定的子任务空间中, 相应地创建 n 个 Agent 团队, 并且将子空间作为 Agent 团队的一个行为策略空间, 这样 n 个 Agent 团队的一次并发行动, 则构成了对各个子任务空间的一次搜索. 把该 Agent 团队所对应的平均奖赏作为行动策略的联合奖赏. 在该过程中搜索空间效率主要体现在 Agent 团队动作策略表所用的空间大小上, 定义 $d_{\max}=\max(d_1, d_2, \cdots, d_n)$, 其中 d_i 为子任务空间的大小, 则有 $2L \leqslant \sum_{i=1}^{n} 2^{d_i} \leqslant n2^{d_{\max}} \leqslant 2^L$, 与此对应的 Agent 动作策略表占用的空间分别约为 $2L$ 和 2^L.

在 Step5 对于每个 Agent 来说对动作效果短期奖赏函数 $Rr: (\overleftarrow{T_i}, \overrightarrow{T_i}) \times (\overleftarrow{A}, \overrightarrow{A}) \rightarrow (\overleftarrow{V}, \overrightarrow{V})$, 表示在当前目标任务状态对于 Agent 采用动作集合$(\overleftarrow{A}, \overrightarrow{A})$中的每个动作$(\overleftarrow{a_i}, \overrightarrow{a_i})$的执行效果评价, 当前任务空间大小为, $|(\overleftarrow{G}, \overrightarrow{G})(\overleftarrow{To}, \overrightarrow{To})|=|(\overleftarrow{G}, \overrightarrow{G})|^{|(\overleftarrow{A}, \overrightarrow{A})|} \times L$, 可见任务空间和动作集合之间的复杂度 $O(m^n)$ 是几何级数关系, 根据 Markov 过程无后效性的特点, 用平均奖赏函数 $Ar: (\overleftarrow{S_0}, \overrightarrow{S_0}) \times (\overleftarrow{A}, \overrightarrow{A}) \rightarrow (\overleftarrow{V}, \overrightarrow{V})$ 替换原有的短期奖赏函数, 则 $|(\overleftarrow{G}, \overrightarrow{G})(\overleftarrow{To}, \overrightarrow{To})|=|(\overleftarrow{G}, \overrightarrow{G})| \times |(\overleftarrow{A}, \overrightarrow{A})| \times L$, 则任务空间与动作集合的复杂度 $O(nma)=O(ka)$.

3.2　Agent 团队任务分配算法

在问题求解中给定一子任务空间集合, 根据上节算法产生的 Agent 团队的集合, 对于 Agent 团队的选择是通过竞争的方式获得执行权利的, 竞争主要通过 Agent 团队的能力值来衡量的, 那些能力值过低的 Agent 团队将会被消除, 在这个过程中的 Agent 团队彼此太相似的 Agent 也会被消除. 这个选择过程通过一定的学习和调整机制最终保留下一些能力强的 Agent 团队.

算法流程:

Step1: 初始化各个 Agent 对子状态空间的探测结果

Step2: 对于每个子状态空间数据执行以下操作:

1. 确定每个 Agent 的任务执行能力

2. 通过能力矩阵$(\overleftarrow{C}, \overrightarrow{C})$选择 n 个执行能力强的 Agent, 产生 Agent 团队, n 是根据子任务的个数确定, $(\overleftarrow{c}, \overrightarrow{c})_{i,j}=1/distance(Agent_i, (\overleftarrow{s}, \overrightarrow{s})_j)$, *distance* 为欧几里德距离

3. 复制 Agent 团队中每个成员, 全体 Agent 构成新的集合$(\overleftarrow{B}, \overrightarrow{B})=\{(\overleftarrow{b}, \overrightarrow{b})_j \mid j=1, 2, \cdots, n\}$, Agent 能力越强复制的$(\overleftarrow{b}, \overrightarrow{b})_j$值就越大, $(\overleftarrow{b}, \overrightarrow{b})_j=M \times (\overleftarrow{c}, \overrightarrow{c})_{ij} / \Sigma(\overleftarrow{c}, \overrightarrow{c})_{ij}$, M 为 Agent 复制的规模

4. 对 Agent 团队进行交叉变异产生新的集合$(\overleftarrow{B}, \overrightarrow{B})'=\{(\overleftarrow{b}, \overrightarrow{b})'_i \mid i=1, 2, \cdots, n\}$, $(\overleftarrow{b}, \overrightarrow{b})'_i=(\overleftarrow{b}, \overrightarrow{b})_i+1/(\overleftarrow{c}, \overrightarrow{c})_{i,j} \times (Agent_i-(\overleftarrow{b}, \overrightarrow{b})_j)$, Agent 能力越大变异越大

5. 在新的 Agent 团队中按照(ξ, ξ)比例选择高能力的 Agent 个体, 通过比较个体 Agent 之间的能力, 删除一些能力近似的 Agent 个体.

Step3: 将所有能力强的 Agent 个体组成团队.

Step4: 比较个体 Agent 之间的能力$(\overleftarrow{d}, \overrightarrow{d})_{tk}=distance(Agent_t-Agent_k)$

Step5: 清除所有$(\overleftarrow{d}, \overrightarrow{d})_{t,k}<(\overleftarrow{\delta}, \overrightarrow{\delta})_d$ 中所有的 Agent 个体

Step6: 直至条件满足, 否则转向 Step2

4　实例分析

实例 1: 在标准数据集 UCI data set* 中根据数据集

* http://www.ics.uci.edu/~mlearn/databases/

提供的不同的数据集的属性找到各自所属的样本类, Iris 数据集包含 virginica、versicolor 和 setosa 三类鸢尾花的信息, 其中共有 150 个样本点数据, 每个样本点有四个属性表示: 萼片长度、萼片宽度、花瓣长度、花瓣宽度. 将这 150 个样本点数据作为任务空间, 随机将其分割生成更小的子任务空间. 在这次实验中, 分割为 3 个子任务空间, 让 3 个 Agent 团队对其进行搜索, 贪心选择概率 $\varepsilon = 0.4$, 学习速率 $\alpha = 0.9$, $\gamma = 0.8$. 图 2 和图 3 给出了这三个 Agent 团队在其中两个子任务空间中的奖赏值函数曲线. 图 4~ 图 7 中分别随机的给出了一个 Agent 团队中不同的 Agent 对与任务空间搜索执行的奖赏值函数曲线. 我们在这四个 Agent 个体执行效果可以看出图 5 中的 Agent 的执行效果最好, 获得了稳定且较好的奖赏值.

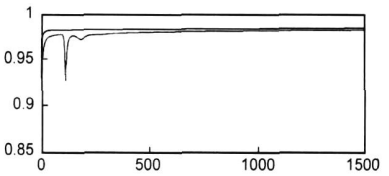

图2 Agent团队在子空间1的奖赏值曲线

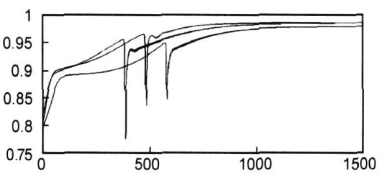

图3 Agent团队在子空间2的奖赏值曲线

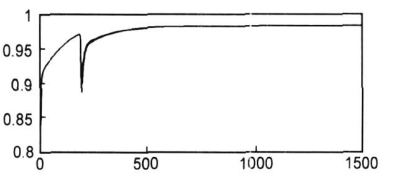

图4 Agent个体任务执行奖赏值曲线

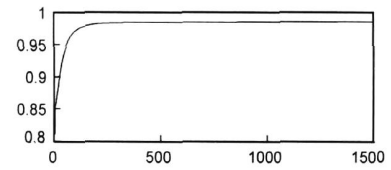

图 5 Agent个体任务执行奖赏值曲线

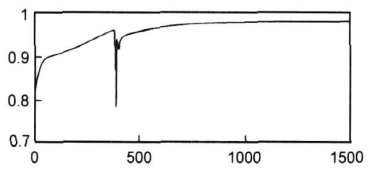

图6 Agent个体任务执行奖赏值曲线

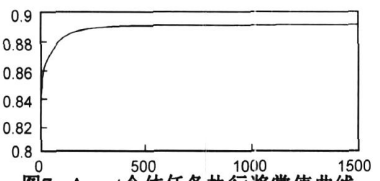

图7 Agent个体任务执行奖赏值曲线

实例 2: 为了验证算法的优越性, 选用文献[14]中常用的优化测试函数进行测试, 具体的函数如下:

$$f_1(x) = \sum_{i=1}^{3} x_i^2 \qquad (-5.12 \leqslant x_i \leqslant 5.12)$$

$$f_2(x) = \sum_{i=1}^{5} integer(x_i) \qquad (-5.12 \leqslant x_i \leqslant 5.12)$$

$$f_3(x) = \sum_{i=1}^{30} i x_i^4 + N(0,1) \qquad (-1.28 \leqslant x_i \leqslant 1.28)$$

$$f_4(x) = (x_1^2 + x_2^2)^{-0.25} \sin^2(50(x_1^2 + x_2^2)^{-0.1}) + 1) \qquad (-100 \leqslant x_i \leqslant 100)$$

$$f_5(x) = 0.5 + \frac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \qquad (-100 \leqslant x_i \leqslant 100)$$

$$f_6(x) = (4 - 2.1x_1^2 + x_1^4/3)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \qquad (-3 \leqslant x_i \leqslant 3)$$

注: 函数 f_2 中的 *interger*( ) 是取下整函数, 函数 f_3 中的 $N(0,1)$ 表示一个满足均值为 0 方差为 1 的正态分布随机数, 上述的所有函数均是求全局极小值.

在实验中算法的相关参数设置为: 贪心选择概率 $\varepsilon = 0.3$, 学习速率 $\alpha = 0.9$, $\gamma = 0.9$, 最大迭代数为 200. 在表 1 中对比了本文的算法和文献[14]的算法所得的最小值以及相应的变量 x 的取值. 将求六个函数的最小值当作是一个任务, 生成不同的 Agent 团队, 将任务空间划分成六个子任务空间, 即将六个函数随机的划分给不同 Agent 团队来执行. 在这六个子任务空间中, 每个 Agent 团队根据自己能力竞争任务, 图 8~ 图 13 是六个子任务空间中最好的 Agent 团队执行结果.

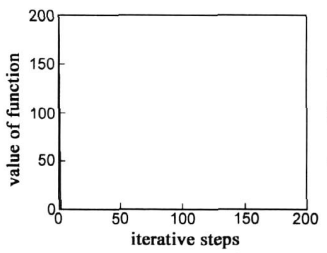

图8 算法在f_1中结果图

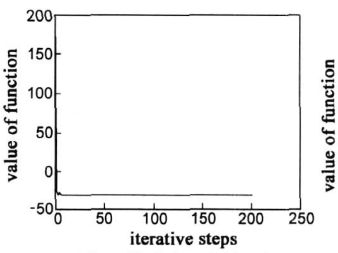

图9 算法在f_2中结果图

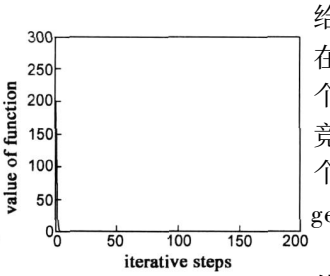

图10 算法在f_3中结果图

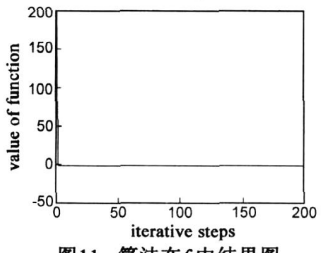

图11 算法在f_4中结果图

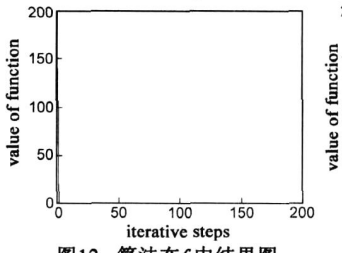

图12 算法在f_5中结果图

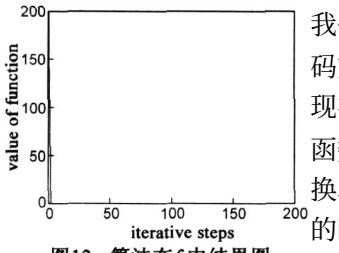

图13 算法在f_6中结果图

文献[14]中, 作者采用传统的二进制编码, 本文中我们采用十进制浮点数编码方式. 该方法的优越性表现在避免了文献[14]算法在函数极值计算中的数制转换、节约了大量的编码解码的时间, 提高了整个算法的实现速度. 鉴于二进制编码

具有一定的映射误差, 对于变量个数较多、精度要求较高的求解问题, 无法达到要求的精度缺陷, 本文的浮点数编码能够有效的提高运算精度. 根据表 1 中的实验结果可以看出本文的算法明显要优于文献[14]的算法. 根据图 8~ 图 13 所示, 在 Agent 团队求最小值的过程中, 每个 Agent 应用自身的经验值以及学习的其他 Agent 的经验, 能很快地找到相对较好的解. 同时将优秀的解空间基因分割代替原有算法中的随机交叉和变异, 能够将最优能力的 Agent 保留下来, 使其解的结构免遭破坏. 在这种情况下, 所划分的小基因样本块中较优的模式得以保持, 从而加速了极值的求解过程.

表 1　文献[14]中的算法与本文算法的运行结果比较

函数	文献[14]中的算法		本文的算法	
	最小值	X	最小值	X
f_1	0.008892	(0.01502, -0.05505, -0.07507)	0.000087	(-0.009331, -0.000030, -0.000258)
f_2	-28	(-5.12, -4.6324, -5.12, -4.6324, -5.12)	-30	(-5.12, -5.12, -5.12, -5.12, -5.12)
f_3	2.5456	(-0.18286, 0.18286, 0.91429, 0.18286, 0.18268, 0.54857, 0.18268, -0.91429, -0.54857, -0.54857, -0.18286, 0.18286, 0.18286, -0.54857, -0.54857, -0.18286, 0.54857, -0.18286, 0.54857, 0.54857, -0.54857, -0.18286, 0.18286, 0.18286, 0.54857, 0.18286, -0.18286, 0.18286, -0.18286, 0.54857)	1.289169	(0.224424, 0.020589, -0.003591, 0.068975, 0.240284, 0.033130, 0.043086, -0.149808, 0.078725, -0.056204, 0.020373, -0.084745, -0.064213, 0.004974, -0.133607, 0.128818, -0.56610, -0.081248, 0.030771, 0.021241, 0.032707, -0.1041418, -0.038263, 0.063907, -0.071752, -0.061127, -0.069906, -0.026636, -0.076240, -0.039672)
f_4	0.1986	(-92.517, -17.527)	0.086877	(-94.83652, -94.221507)
f_5	0.9921	(-0.009156, 0.088504)	0.009716	(0.5578440, -3.088510)
f_6	-1.0268	(-0.06473, -0.72796)	-1.031628	(0.089615, -0.712765)

5　相关工作比较

文献[2] 通过任务需求和 Agent 能力自信度, 给出了基于合同网协议的任务分配算法. 通过 Agent 对于任务做出的承诺, 以及承诺与付出的关系讨论了 Agent 联盟执行任务的过程. 但单纯的合同网协议忽略了 Agent 间的隐含和约束关系, 在选择执行者的有时并不是最有能力的, 从而导致问题求解不是全局最优. 同时每个 Agent 获得都是当前局部状态信息, 任务的状态信息的发布和个体 Agent 间覆盖式的交流增加系统的问题求解时间.

文献[8] 在处理任务分配问题上, 针对传统遗传算法中初始解结构和遗传算子的局限性, 在编码知识表示的基础上, 对初始解群的构造给出一种相对均衡的生成方法, 然后将标准的算子改进为内部杂交算子和类似变异的迁移算子; 同时在适应值的计算和参数的设定上有所改进, 从而提高算法的收敛速度. 但其在本质上仍未完全解决遗传算法在任务分配中所面临的本质问题. 本文在遗传算法中引入自适应的强化机制, 克服了局部最优和过早收敛的问题.

文献[14] 深入地分析和比较了强化学习和遗传算法, 提出了基于强化学习机制的遗传算法, 该算法通过基因空间分割将两者融合在一起. 然后算法在基因编码上采用的是二进制编码, 在时间和空间上对于算法的实现都有很大的局限性. 本文保留该算法的基本优越性, 改进了其基因编码方式, 采用十进制浮点数编码. 节约了大量的编码和解码时间, 同时在求解过程对于高精度要求的问题上也有一定优越性.

6　结论与展望

许多学者在研究任务分配问题时, 对问题描述所给出的假设是有差别的, 然而也不失一般性, 都假设一个大的应用任务已经被分解成了具有某种约束关系的多个子任务[1].

本文将具有强化机制的遗传算法引入到任务分配问题上来. 针对遗传算法固有的问题, 引入一种目标驱动的自适应能力很强的强化学习技术, 从而使得算法本身体现出明晰的目标性. 在强化学习 Agent 环境进行交互时, 能够根据环境的反馈来调整自己的行动策略. 在我们的算法中 Agent 采用的是 ε-*greedy* 行动选择策略, 即以 $1-\varepsilon$ 的概率选择到目前为止最好的行动策略, 以 ε 的概率选择行动策略空间中的任一行动策略.

同样也体现了对行动策略空间的发现和探索间的平衡. 在算法运行的初期通过增大 ε 值来提高 Agent 对于全局任务空间的搜索力度, 积累行动策略的经验知识; 而在算法运行的后期通过减小 ε 值来充分发现行动策略的经验知识, 提高算法的收敛性能. 如何根据应用任务的性质动态地将其分配成多个子任务是今后的研究工作.

参考文献:

[1] 钟求喜, 谢涛, 陈火旺. 任务分配与调度的共同进化方法[J]. 计算机学报, 2001, 24(3): 308- 314.
Zhong Qiu-xi, Xie Tao, Chen Huo-wang. Task allocation & scheduling by computational model of coevolution[J]. Chinese Journal of Computers, 2001, 24(3): 308- 314. (in Chinese)

[2] 曾广周, 杨公平, 王晓琳. 基于 Agent 能力自信度的任务分配问题研究[J]. 计算机学报, 2007, 30(11): 1922- 1929.
Zeng Guang-zhou, Yang Gong-ping, Wang Xiao-lin. Study of task allocation problem based on agent ability confidence[J]. Chinese Journal of Computers, 2007, 30(11): 1922- 1929. (in Chinese)

[3] Coffinan E G. Computer and Job-Shop Scheduling Theory [M]. New York: John Wiley & Sons, 1976.

[4] S Menon. Effective reformulations for task allocation in distributed systems with a large number of communicating tasks [J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(12): 1497- 1508.

[5] Vig L, Adams J A. Issues in multi-robot coalition formation [A]. Proceedings of International Workshop on Multi-Robot Systems[C]. Washington DC: Swarms to Intelligent Automata, 2005. 15- 26.

[6] Antonio Miranda Garcia. Approximation algorithms for multiprocessor task scheduling[D]. Texas A &M University, 1998.

[7] Spielman D A, Teng S H. Smoothed analysis: why the simplex algorithm usually takes polynomial time. [J]. Journal of the ACM, 2004, 51(3): 385- 463.

[8] 钟求喜, 谢涛, 陈火旺. 基于遗传算法的任务分配与调度[J]. 计算机研究与发展, 2000, 37(10): 1197- 1203.
Zhong Qiu-xi, Xie Tao, Chen Huo-wang. Task matching and scheduling by using genetic algorithms[J]. Journal of Computer Research and Development, 2000, 37(10): 1197- 1203. (in Chinese)

[9] 闵帆, 石兵, 杨国纬, 周明天. 分布式系统中任务分配的一种结点自适应算法[J]. 计算机学报, 2003, 26(3): 302- 309.
Min Fan, Shi Bing, Yang Guo-wei, Zhou Ming-tian. A self-learning algorithm for IPUs in distributed systems[J]. Chinese Journal of Computers, 2003, 26(3): 302- 309. (in Chinese)

[10] 蒋建国, 夏娜, 齐美彬, 木春梅. 一种基于蚁群算法的多任务联盟串行生成算法[J]. 电子学报, 2005, 33(12): 2178- 2182.
Jiang Jian-guo, Xia Na, Qi Mei-bin, Mu Chun-mei. An ant colony algorithm based multi-task coalition serial generation algorithm[J]. Acta Electronica Sinica, 2005, 33(12): 2178- 2182. (in Chinese)

[11] 高阳. 强化学习研究进展机器学习及其应用[M]. 北京: 清华大学出版社, 2006. 116- 133.

[12] R S Sutton, D Precup, S Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning[J]. Artificial Intelligence, 1999, 112(1/2): 181- 211.

[13] Pettinger J E, Everson R M. Controlling genetic algorithms with reinforcement learning[A]. Proceedings of the Genetic and Evolutionary Computation[C]. San Francisco: Morgan Kaufmann Publishers Inc, 2002. 692- 692.

[14] 王本年, 高阳, 陈兆乾, 谢俊元, 陈世福. RLGA: 一种基于强化学习机制的遗传算法[J]. 电子学报, 2006, 34(5): 856- 860.
Wang Ben-nian, Gao Yang, Chen Zhao-qian, Xie Jun-yuan, Chen Shi-fu. RLGA: A reinforcement learning based genetic algorithm[J]. Acta Electronica Sinica, 2006, 34(5): 856- 860. (in Chinese)

作者简介:

张 瑜 女, 1981 年生. 苏州大学计算机科学与技术学院硕士研究生, 主要研究方向为动态模糊逻辑、多 Agent 系统.
E-mail: 18952881521@189.cn

李凡长 男, 1964 年生, 博士生导师, 苏州大学计算机科学与技术学院教授, 中国人工智能学会理事, 中国人工智能学会的机器学习专委会常务委员, 中国计算机学会高级会员, 中国计算机学会的理论计算机科学专委会委员, 人工智能与模式识别专委会委员. 主要研究方向为动态模糊逻辑、李群机器学习等.
E-mail: lfzh@suda.edu.cn