求解二元约束满足问题的多智能体进化算法*

钟伟才 刘 静 焦李成

西安电子科技大学智能信息处理研究所，西安 710071

摘要 基于智能体对环境的感知与反作用的能力提出了一种新的求解二元约束满足问题的方法. 该方法将多智能体系统与进化算法有机地结合起来，每个智能体固定在网格的一个格点上，而它为了增加自身能量将与其邻域展开竞争. 同样，智能体也可利用自身的知识进行自学习来增加能量. 根据二元约束满足问题的特点，设计了智能体的竞争行为与自学习行为. 为了克服已有编码方式的缺点，为智能体设计了最小冲突编码. 理论分析证明算法具有全局收敛性. 实验中用250个不同难度的标准问题对算法的两个参数进行了系统的分析. 结果表明该算法的性能非常稳定，参数少，易于使用. 与4个著名方法的比较结果表明该方法获得的解的质量是最高的，其性能优于其他4种方法.

关键词 多智能体系统 进化算法 约束满足问题 最小冲突编码 竞争行为 自学习行为

大量的人工智能问题以及计算机其他领域里的问题都可以归结为约束满足问题(constraint satisfaction problems, CSPs). 文献[1]说明任何一个约束满足问题都能转化为一个二元约束满足问题，因此，研究求解二元约束满足问题的方法对于求解约束满足问题具有很重要的意义. 对于约束满足问题，已提出了很多基于进化算法的方法，如 GlassBox[2]，H-GA[3]，SAW[4]等. 这些方法从不同角度利用进化算法来处理约束满足问题，推动了这一领域的发展.

分布式人工智能中基于智能体的计算已经应用于计算机学科的各个领域[5—8]. 文献[7]设计了一种基于能量的多智能模型，求解了高达7000个皇后的问题. 文献[8]将多智能体系统和遗传算法相结合，能够快速求解高达上万维的函数优化问题. 所有这些结果表明智能体与进化算法相结合来求解NP难题的巨大潜力，也为求解约束满足问题提供了新的思路.

1 用于二元约束满足问题的智能体

1.1 二元约束满足问题

一个二元约束满足问题由以下3部分构成：

(1) 一个有限的变量集合，$\boldsymbol{x}=\{x_1, x_2, \cdots, x_n\}$；

(2) 定义域集合 $\boldsymbol{D}$，它由每个变量有限离散的定义域构成，其中 $|\cdot|$ 为集合的基数：

$$\boldsymbol{D}=\{D_1, D_2, \cdots, D_n\}, x_i \in D_i = \{d_1, d_2, \cdots, d_{|D_i|}\}, i=1,2,\cdots,n, \quad (1)$$

(3) 约束集合 $\boldsymbol{C}=\{C_1(x_1^1, x_2^1)=\langle d_1^1, d_2^1\rangle, C_2(x_1^2, x_2^2)=\langle d_1^2, d_2^2\rangle, \cdots, C_m(x_1^m, x_2^m)=\langle d_1^m, d_2^m\rangle\}$，其中 $x_1^i, x_2^i \in \boldsymbol{x}$，$d_1^i$，$d_2^i$ 属于 x_1^i，x_2^i 的定义域，$C_1(x_1^i, x_2^i)=\langle d_1^i, d_2^i\rangle$ 表示 x_1^i，x_2^i 不能同时取 d_1^i，d_2^i 这两个值，$i=1, 2, \cdots, m$.

一个二元约束满足问题的搜索空间，$\boldsymbol{S}$，就为 n 个有限定义域的 Descartes 乘积，即 $\boldsymbol{S}=D_1\times D_2\times\cdots\times D_n$. 而一个二元约束满足问题的解，$\boldsymbol{S}=\langle x_1, x_2, \cdots, x_n\rangle \in \boldsymbol{S}$，就是满足所有约束的变量取值方

2004-03-25 收稿，2004-07-22 收修改稿

* 国家自然科学基金重点项目(批准号：60133010)和西安电子科技大学青年科研工作站基金资助

E-mail：neouma@163.com

式. 下面看一个简单的二元约束满足问题的例子：

例1：一个二元约束满足问题可以描述成如下的形式：

$$\begin{cases} \boldsymbol{x}=\{x_1,x_2,x_3\} \\ \boldsymbol{D}=\{D_1,D_2,D_3\},D_i=\{1,2,3\},i=1,2,3 \\ \boldsymbol{C}=\{C_1(x_1,x_2)=\langle 1,3\rangle,C_2(x_1,x_2)=\langle 3,3\rangle, \\ \quad C_3(x_1,x_3)=\langle 2,1\rangle,C_4,(x_1,x_3)=\langle 2,3\rangle, \\ \quad C_5(x_1,x_3)=\langle 3,1\rangle,C_6(x_1,x_3)=\langle 3,3\rangle, \\ \quad C_7(x_2,x_3)=\langle 1,1\rangle,C_8(x_2,x_3)=\langle 1,2\rangle, \\ \quad C_9(x_2,x_3)=\langle 1,3\rangle,C_{10}(x_2,x_3)=\langle 2,1\rangle, \\ \quad C_{11}(x_2,x_3)=\langle 3,1\rangle\} \end{cases}, \tag{2}$$

满足该问题的所有解为〈1, 2, 2〉, 〈1, 2, 3〉, 〈2, 2, 2〉, 〈2, 3, 2〉, 〈3, 2, 2〉.

1.2 智能体

一个用于二元约束满足问题的智能体定义成如下的形式.

定义1 一个用于二元约束满足问题的智能体 $\boldsymbol{a}$，是该问题搜索空间中的一个元素，它的能量见(3)式，其中 $\chi(\boldsymbol{a},C_i)=\begin{cases}1 & \boldsymbol{a}\text{违反约束}C_i \\ 0 & \text{否则}\end{cases}$.

$$\forall \boldsymbol{a}\in \boldsymbol{S},Energy(\boldsymbol{a})=-\sum_{i=1}^{m}\chi(\boldsymbol{a},C_i), \tag{3}$$

智能体的目的是尽可能的增大自身能量，它可采用一定的行为来达到它的目的.

在二元约束满足问题中，因为每个变量的定义域是一个有限的离散集合，等价于自然数的集合，即 $D=\{d_1,d_2,\cdots,d_{|D|}\}$ 等价于 $D=\{1,2,\cdots,|D|\}$. 因而最直接的编码方法就是整数编码，将每个个体表示成搜索空间中的一个元素. 但这种编码方式的搜索空间规模为 $|D_1|\times|D_2|\times\cdots\times|D_n|$，会降低搜索效率. 因此，有些算法采用了排列的编码方法并加上一个解码器. 例如文献[4]中每个个体表达成问题变量的一个排列，而解码器用贪婪的方法将一个排列转化成搜索空间中的一个点，即按照排列中变量的顺序，给每个变量赋一个最小的不会引起冲突的值，如果对某个变量这种值不存在，则不给该变量赋值. 下面我们将这种解码方式称为Decoder1. 这种编码方式的搜索空间为

$$\boldsymbol{S}_x^P=\{\boldsymbol{P}_1^x,\boldsymbol{P}_2^x,\cdots,\boldsymbol{P}_{n!}^x\}\text{和}$$
$$\begin{cases} \boldsymbol{P}_i^x=\langle x_{p_i^1},x_{p_i^2},\cdots,x_{p_i^n}\rangle,1\leqslant i\leqslant n! \\ P_i^j\in\{1,2,\cdots,n\},1\leqslant j\leqslant n \\ \forall k,l\in\{1,2,\cdots,n\},(k\neq l)\Rightarrow(P_i^k\neq P_i^l) \end{cases}, \tag{4}$$

由于Decoder1采用了贪婪的方法，我们发现它存在一个很大的缺点，即对于某些问题，在这种解码方式下，任何一个排列都不能转化成解，这就导致基于这种解码方法的算法无论如何也找不到问题的解. 下面是一个简单的例子.

例2：所处理的问题见例1，它的搜索空间 $\boldsymbol{S}_x^P$ 为

$$\boldsymbol{S}_x^P=\{\langle x_1,x_2,x_3\rangle,\langle x_1,x_3,x_2\rangle,\langle x_2,x_1,x_3\rangle,\langle x_2,x_3,x_1\rangle,\langle x_3,x_1,x_2\rangle,\langle x_3,x_2,x_1\rangle\}, \tag{5}$$

根据Decoder1，$\boldsymbol{S}_x^P$ 中的每个元素可转化成如下的部分解：

$$\begin{aligned}&\langle x_1,x_2,x_3\rangle\to\langle 1,1,*\rangle,\langle x_1,x_3,x_2\rangle\to\langle 1,1,*\rangle,\\&\langle x_2,x_1,x_3\rangle\to\langle 1,1,*\rangle,\langle x_2,x_3,x_1\rangle\to\langle 1,*,1\rangle,\\&\langle x_3,x_1,x_2\rangle\to\langle 1,1,*\rangle,\langle x_3,x_2,x_1\rangle\to\langle 1,*,1\rangle,\end{aligned} \tag{6}$$

其中“*”表示相应的变量未赋值. 由(6)式可见，$\boldsymbol{S}_x^P$ 中没有元素能够转化成解.

因此，在分析了已有的各种编码方式后，我们为智能体设计了最小冲突编码方法. 在这种编码方法中，每个智能体不但表达成变量排列的形式，并且记录每个变量的取值. 所以每个智能体必须记录一些信息，它表达成如下的结构：

Agent=Record

$\boldsymbol{P}$：变量的排列，$\boldsymbol{P}\in\boldsymbol{S}_x^P$

$\boldsymbol{V}$：每个变量的取值，$\boldsymbol{V}\in\boldsymbol{S}$；

E：智能体的能量，$E=Energy(\boldsymbol{V})$；

SL：自学习行为的标志；如果 SL 为 $True$，则自学习行为可以作用在该智能体上，否则不能；

End.

在下文中，***Agent***(·)表示一个智能体上述结构中相应的成分. 在最小冲突编码方法中，相应的解码方法为最小冲突解码.

算法 1 最小冲突解码 (**Decoder2**)

输入：***Agent***：需要解码的智能体；Pos：开始解码的位置；输出：***Agent***($\boldsymbol{V}$)；

令 ***Agent***($\boldsymbol{P}$) $=\langle x_{p_1}, x_{p_2}, \cdots, x_{p_n}\rangle$，***Agent***($\boldsymbol{V}$)$=\langle v_1, v_2, \cdots, v_n\rangle$，

$$Conflicts(v_i)=\sum_{j=1}^{m}\chi(v_i,C_j),$$

$$\text{其中 } \chi(v_i,C_j)=\begin{cases}1 & v_i\text{违反约束 } C_j\\ 0 & \text{否则}\end{cases}, \quad (7)$$

$Conflicts(v_i)$只涉及已经赋值的变量.

Step 1：如果($Pos=1$)，令 $v_{p_1}\leftarrow 1$，$i\leftarrow 2$；否则令 $i\leftarrow Pos$；

Step 2：如果 $i>n$，停止；否则令 $v_{p_i}\leftarrow 1$，$Min_C\leftarrow Conflicts(v_{p_i})$，$Min_V\leftarrow 1$；令 $j\leftarrow 2$；

Step 3：令 $v_{p_i}\leftarrow j$；如果 $Conflicts(v_{P_i})<Min_C$，则令 $Min_C\leftarrow Conflicts(v_{p_i})$，$Min_V\leftarrow j$；

Step 4：令 $j\leftarrow j+1$；如果 $j\leqslant |D_{P_i}|$，则转 Step 3；否则转 Step 5；

Step 5：令 $v_{p_i}\leftarrow Min_V$，$i\leftarrow i+1$，并转 Step 2.

下面为智能体设计了两种行为：作用在 $\boldsymbol{P}$ 上的行为($\boldsymbol{P}$-behavior)和作用在 $\boldsymbol{V}$ 上的行为($\boldsymbol{V}$-behavior). 当 $\boldsymbol{P}$-behavior 作用于一个智能体后，必须先用 Decoder2 进行解码. 如果 Decoder2 从排列中的第一个变量开始解码，$\boldsymbol{V}$-behavior 产生的信息就会丢失. 因此，我们设置了参数 Pos，它是 $\boldsymbol{P}$-behavior 所改变的第一个变量的位置. 这样，Pos 之前的变量的值就不会改变，以便保留一些 $\boldsymbol{V}$-behavior 产生的信息. 对于一个完全新产生的智能体，Pos 设为 1. 为了方便起见，当用 Decoder2 对 ***Agent*** 进行解码的时候，我们记为 Decoder2(***Agent***, Pos).

1.3 环境

智能体的生存环境被组织成如下的网格结构：

定义 2 智能体生存在网格 $\boldsymbol{L}$ 上，称为智能体网格，其规模为 $L_{\text{size}}\times L_{\text{size}}$. 每个智能体固定在一个格点位置上，且它只能与其邻域进行相互作用. 将处在第 i 行、第 j 列的智能体记为 $\boldsymbol{L}_{i,j}$，$i, j=1, 2, \cdots, L_{\text{size}}$，则 $\boldsymbol{L}_{i,j}$ 的邻域 $\boldsymbol{Neighbors}_{i,j}$ 为

$$\boldsymbol{Neighbors}_{i,j}=\{\boldsymbol{L}_{i',j}, \boldsymbol{L}_{i,j'}, \boldsymbol{L}_{i'',j}, \boldsymbol{L}_{i,j''}\}, \quad (8)$$

$$\text{其中 } i'=\begin{cases}i-1 & i\neq 1,\\ L_{\text{size}} & i=1\end{cases}, \quad j'=\begin{cases}j-1 & j\neq 1,\\ L_{\text{size}} & j=1\end{cases},$$

$$i''=\begin{cases}i+1 & i\neq L_{\text{size}},\\ 1 & i=L_{\text{size}}\end{cases}, \quad j''=\begin{cases}j+1 & j\neq L_{\text{size}},\\ 1 & j=L_{\text{size}}\end{cases}.$$

因此，智能体网格可以表达成图 1 的形式. 每个圆代表一个智能体，圆中的数字代表该智能体所处的位置，只有有连线的两个智能体才能发生相互作用.

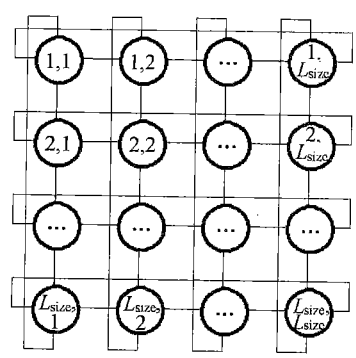

图 1 智能体网络

1.4 行为

本节设计了 3 种行为来实现智能体的目的，其中竞争行为和自学习行为属于 $\boldsymbol{P}$-behavior，变异行为属于 $\boldsymbol{V}$-behavior.

竞争行为：智能体将与其邻域比较能量——若它的能量大于邻域中任一智能体的能量，则它可以生存下去；否则它必须死亡，而它的位置则由其邻域中能量最大的智能体的子代所占据. 具体描述如下：

设竞争行为作用在智能体 $\boldsymbol{L}_{i,j}$ 上，$\boldsymbol{Max}_{i,j}$ 是其邻域中能量最大的智能体. 如果 $\boldsymbol{L}_{i,j}(E)\leqslant\boldsymbol{Max}_{i,j}(E)$，则 $\boldsymbol{Max}_{i,j}$ 产生一个子代来代替 $\boldsymbol{L}_{i,j}$，产生的方法在算法 2 中给出，否则 $\boldsymbol{L}_{i,j}$ 继续存活在网格上.

算法 2 竞争行为

输入：$\boldsymbol{Max}_{i,j}$：$\boldsymbol{Max}_{i,j}(\boldsymbol{P})=\langle x_{m_1}, x_{m_2}, \cdots, x_{m_n}\rangle$；$p_c$：取值为 0 到 1 的参数；

输出：$\boldsymbol{Child}$：$\boldsymbol{Child}_{i,j}(\boldsymbol{P})=\langle x_{c_1}, x_{c_2}, \cdots, x_{c_n}\rangle$；

令 $Swap(x, y)$表示交换 x 和 y 的取值；$U(0, 1)$为 0 到 1 间均匀分布的随机数；$Random(n, i)$是 1，2，…，n 中的随机整数且不等于 i；$Min(i, j)$为 i 和 j 中较小的值.

Step 1：令 $\boldsymbol{Max}_{i,j}(\boldsymbol{P})\leftarrow\boldsymbol{Child}_{i,j}(\boldsymbol{P})$，$i\leftarrow 1$，$Pos\leftarrow n+1$；

Step 2：如果 $U(0, 1)<p_c$，则令 $l\leftarrow Random(n, i)$，$Swap(x_{c_i}, x_{c_l})$；

Step 3：如果 $\mathrm{Min}(l, i)<Pos$，则令 $Pos\leftarrow\mathrm{Min}(l, i)$；

Step 4：令 $i\leftarrow i+1$；如果 $i\leqslant n$，则转 Step 2；

Step 5：执行 Decoder2($\boldsymbol{Child}$, Pos)，并令 $\boldsymbol{Child}(SL)\leftarrow True$.

自学习行为：该行为中，智能体利用自己的知识来提高自身的能量，以增强竞争能力. 设自学习行为作用在 $\boldsymbol{L}_{i,j}$ 上，则具体方式见算法 3.

算法 3 自学习行为

输入：$\boldsymbol{L}_{i,j}$：$\boldsymbol{L}_{i,j}(\boldsymbol{P})=\langle x_{m_1}, x_{m_2}, \cdots, x_{m_n}\rangle$，$\boldsymbol{L}_{i,j}(V)=\langle v_1, v_2, \cdots, v_n\rangle$；输出 $\boldsymbol{L}_{ij}$；

Step 1：令 $Repeat\leftarrow False$，$k\leftarrow 1$，$Iter\leftarrow 1$；

Step 2：如果$(Conflicts)(v_{m_k})=0$），则转 Step 7；

Step 3：令 $Energy_{old}\leftarrow\boldsymbol{L}_{i,j}(E)$，$l\leftarrow Random(n, k)$；

Step 4：执行 $Swap(x_{m_k}, x_{m_1})$，Decoder2($\boldsymbol{L}_{i,j}$, Min(k, l))，令 $Energy_{new}\leftarrow\boldsymbol{L}_{i,j}(E)$；

Step 5：如果 $Energy_{new}>Energy_{old}$，则令 $Repeat\leftarrow True$；否则执行. $Swap(x_{m_k}, x_{m_l})$，Decoder2($\boldsymbol{L}_{i,j}$, Min(k, l))；

Step 6：若 $Iter<n-1$，则令 $Iter\leftarrow Iter+1$，并转 Step 2；否则令 $Iter\leftarrow 1$，并转 Step 7；

Step 7：令 $k\leftarrow k+1$；如果 $k\leqslant n$，则转 Step 2；

Step 8：如果 $Repeat=True$，则转 Step 1；否则令 $\boldsymbol{L}_{i,j}(SL)\leftarrow False$ 并停止.

由于当自学习行为作用在一个智能体上以后，对于同一个智能体利用自学习行为能使其能量再升高的可能性已很小，因此在最后令 $\boldsymbol{L}_{i,j}(SL)$ 为 $False$.

变异行为：该行为类似于传统进化算法中的变异算子. 它可以扩大以上两种行为的搜索取域，以弥补解码方式造成的缺陷. 设该算子作用在 $\boldsymbol{L}_{i,j}$ 上，且 $\boldsymbol{L}_{i,j}(\boldsymbol{V})=\langle v_1, v_2, \cdots, v_n\rangle$，则下述操作作用在 $\boldsymbol{L}_{i,j}$ 上：

$$\text{如果 } U_k(0, 1)<p_m\text{，则令 } v_k\leftarrow Random(|D_k|)\text{，} \tag{9}$$

其中 $k=1, 2, \cdots, n$，$U_k(0, 1)$对每个 k 产生一个随机数. p_m 是预先设定的参数，它的取值范围是 0 到 1. $Random(|D_k|)$表示 1，2，…，$|D_k|$ 中的一个随机整数.

2 求解二元约束满足问题的多智能体进化算法

2.1 算法实现

智能体必须有序的采取上述 3 种行为才能达到求解问题的目的，这里我们用进化的方式来控制智能体与智能体间的相互作用，具体描述见算法 4：

算法 4 求解二元约束满足问题的多智能体进化算法

输入：$Eval_{\mathrm{Max}}$：最大计算量；L_{size}：智能体网格的规模；p_c：竞争行为中的参数；p_m：变异行为中的参数；

输出：$\boldsymbol{s}$：所求解问题的解或近似解；

$\boldsymbol{L}^t$ 为第 t 代智能体网格，$\boldsymbol{Agent}^t_{\mathrm{Best}}$ 是 $\boldsymbol{L}^0$，$\boldsymbol{L}^1$，…，$\boldsymbol{L}^t$ 中最好的智能体，$\boldsymbol{Agent}^t_{i\mathrm{Best}}$ 为 $\boldsymbol{L}^t$ 中最好的智能体.

Step 1：令 $Eval\leftarrow 0$；初始化智能体网格 $\boldsymbol{L}^0$：随机产生一个排列并赋给 $\boldsymbol{L}^0_{i,j}(\boldsymbol{P})$，令 $\boldsymbol{L}^0_{i,j}(SL)\leftarrow True$，执行 Decoder2 ($\boldsymbol{L}^0_{i,j}$, 1)，计算 $\boldsymbol{L}^0_{i,j}(E)$，令 $Eval\leftarrow Eval+1$，其中 i，$j=1$，2，…，L_{size} 更新 $\boldsymbol{Agent}^0_{\mathrm{Best}}$，令 $t\leftarrow 0$；

Step 2：将竞争行为作用在 $\boldsymbol{L}^t$ 上：若 $\boldsymbol{L}^t_{i,j}$ 胜利，令 $\boldsymbol{L}^{t+1}_{i,j}\leftarrow\boldsymbol{L}^t_{i,j}$；否则令 $\boldsymbol{L}^{t+1}_{i,j}\leftarrow\boldsymbol{Child}$，计算 $\boldsymbol{L}^{t+1}_{i,j}(E)$，令 $Eval\leftarrow Eval+1$，其中 $\boldsymbol{Child}$ 是根据算法 2 产生的；

Step 3：更新 $\boldsymbol{Agent}^{t+1}_{(t+1)\mathrm{Best}}$；若($\boldsymbol{Agent}^{t+1}_{(t+1)\mathrm{Best}}(SL)=True$)，则将自学习行为作用在 $\boldsymbol{Agent}^{t+1}_{(t+1)\mathrm{Best}}$ 上，其中 $Eval$ 在算法 3 中会更新，转 Step 5；否则转 Step 4；

Step 4：将变异行为作用在 $\boldsymbol{Agent}^{t+1}_{(t+1)\mathrm{Best}}$，计算 $\boldsymbol{Agent}^{t+1}_{(t+1)\mathrm{Best}}(E)$，令 $Eval\leftarrow Eval+1$；

Step 5：如果 $Agent^{t+1}_{(t+1)\mathrm{Best}}(E)\geqslant Agent^{t}_{\mathrm{Best}}(E)$，则令 $Agent^{t+1}_{\mathrm{Best}}\leftarrow Agent^{t+1}_{(t+1)\mathrm{Best}}$；否则令 $Agent^{t+1}_{\mathrm{Best}}\leftarrow Agent^{t}_{\mathrm{Best}}$，$Agent^{t+1}_{\mathrm{Random}}\leftarrow Agent^{t}_{\mathrm{Best}}$，其中 $Agent^{t+1}_{\mathrm{Random}}$ 从 L^t 中随机选择的一个智能体并与 $Agent^{t}_{(t+1)\mathrm{Best}}$ 不同；

Step 6：如果($Agent^{t+1}_{\mathrm{Best}}(E)=0$)或($Eval\geqslant Eval_{\mathrm{Max}}$)，则令 $s\leftarrow Agent^{t+1}_{\mathrm{Best}}(V)$并停止；否则令 $t\leftarrow t+1$，并转 Step 2.

2.2 收敛性分析

二元约束满足问题的搜索空间是一个离散空间，由 $|D_1|\times|D_2|\times\cdots\times|D_n|$ 个元素构成. 令 $E=\{Energy(a)\mid a\in S\}$. 由于智能体的能量是关于未满足的约束的个数，而约束的总数为 m，则有 $|E|$ 等于 m 且 E 可表达成 $E=\{0,1,2,\cdots,m\}$. 这样，就可将 S 划分成若干非空子集$\{S^i\mid i=0,1,2,\cdots,m\}$，其中 S^i 为

$$S^i=\{a\mid a\in S\text{ 和 }Energy(a)=i\}\,,\quad(10)$$

$$\sum_{i=0}^{m}|S^i|=|S|;S^i\neq\varnothing;$$

$$S^i\cap S^j=\varnothing,\forall_{i\neq j};\bigcup_{i=0}^{m}S^i=S\,,\quad(11)$$

显然，S^0 中的任何智能体均是问题的解.

为了衡量智能体网格的优劣，定义智能体网格 L 的能量为：

$$Energy(L)=\max\{L_{i,j}(E)\mid i,j=1,2,\cdots,L_{\mathrm{size}}\}\,,\quad(12)$$

令 $\mathscr{L}$ 表示所有智能体网格的集合，这样对 $\forall L\in\mathscr{L}$，均有 $0\leqslant Energy(L)\leqslant m$. 因此可以把集合 $\mathscr{L}$ 划分为非空子集$\{\mathscr{L}^i\mid i=0,1,2,\cdots,m\}$，这里 $\mathscr{L}^i$ 为

$$\mathscr{L}^i=\{L\mid L\in\mathscr{L}\text{ 和 }Energy(L)=i\}\,,\quad(13)$$

$$\sum_{i=0}^{m}|\mathscr{L}^i|=|\mathscr{L}|;\mathscr{L}^i\neq\varnothing;$$

$$\mathscr{L}^i\cap\mathscr{L}^j=\varnothing,\forall_{i\neq j};\bigcup_{i=0}^{m}\mathscr{L}^i=\mathscr{L}\,,\quad(14)$$

$\mathscr{L}^0$ 由所有能量为 0 的智能体网格构成.

令 L^{ij}，$i=0,1,2,\cdots,m$，$j=1,2,\cdots,|\mathscr{L}^i|$ 表示 $\mathscr{L}^i$ 中的第 j 个智能体网格. 在任何一代，3 种行为将智能体网格从 L^{ij} 转化成 L^{kl}. 令 $p_{ij.kl}$ 表示从 L^{ij} 到 L^{kl} 的转移概率，$p_{ij.k}$ 表示从 L^{ij} 到 $\mathscr{L}^k$ 中任一网格的转移概率，而 $p_{i.k}$ 表示从 $\mathscr{L}^i$ 中任一网格到 $\mathscr{L}^k$ 中任一网格的转移概率. 显然有：

$$p_{ij.k}=\sum_{l=1}^{|\mathscr{L}^k|}p_{ij.kl},\sum_{k=0}^{m}p_{ij.k}=1,p_{i.k}\geqslant p_{ij.k}\,,\quad(15)$$

明确了上述概念后，基于文献[9]下面给出本文算法全局收敛性的证明.

定理 1[10] 令 P'：$n'\times n'$是一个可归约的随机矩阵，即通过相同的行变换和列变换后可以得到 $P'\begin{pmatrix}C&0\\R&T\end{pmatrix}$，其中 C：$m'\times m'$是一个本原随机矩阵且 R，$T\neq 0$. 则

$$P'^{\infty}=\lim_{k\to\infty}P'^{k}=\lim_{k\to\infty}\begin{pmatrix}C^k&0\\\sum_{i=0}^{k-1}T^iRC^{k-i}&T^k\end{pmatrix}=\begin{pmatrix}C^{\infty}&0\\R^{\infty}&0\end{pmatrix}\,,\quad(16)$$

是一个稳定的随机矩阵且 $P'^{\infty}=1'P^{\infty}$ 这里，$p^{\infty}=p^0P'^{\infty}$ 唯一确定且与初始分布无关，p^{∞} 满足 $p_i^{\infty}>0$，$1\leqslant i\leqslant m'$ 且 $p_i^{\infty}=0$，$m'<i\leqslant n'$.

定理 2 在求解二元约束满足问题的多智能体进化算法中，$\forall i,k\in E$，$p_{i,k}=\begin{cases}>0, & k\leqslant i\\ =0, & k>i\end{cases}$.

证明：令 L^{ij}，$i\in E$，$j=1,2,\cdots,|\mathscr{L}^i|$ 为第 t 代的智能体网格，记为 L^t. L^t 中最优的智能体为 $Agent^{t}_{t\mathrm{Best}}$，且有 $Agent^{t}_{t\mathrm{Best}}(E)=i$. 根据算法 4 Step 5 可得(17)式：

$$Energy(L^{t+1})\geqslant Energy(L^t)\Rightarrow\forall k>i,\ p_{ij.kl}=0\Rightarrow\forall k>i,\ p_{ij.k}=\sum_{l=1}^{|S^k|}p_{ij.kl}=0\Rightarrow\forall k>i,\ p_{i.k}=0\,,\quad(17)$$

设 $\exists Agent'$，$Agent'(E)=k>i$. 自学习行为或变异行为将作用在 $Agent^{t}_{t\mathrm{Best}}$ 上. 显然，若自学习行为作用在 $Agent^{t}_{t\mathrm{Best}}$，则 $Agent^{t}_{t\mathrm{Best}}$ 转化为 $Agent'$ 的概率 $p_{Agent^{t}_{t\mathrm{Best}}\to Agent'}>0$. 若变异行为作用在 $Agent^{t}_{t\mathrm{Best}}$ 上，设 $Agent^{t}_{t\mathrm{Best}}(V)$ 有 n_1 个变量的取值与 $Agent'(V)$相应的变量不一样，则 $Agent^{t}_{t\mathrm{Best}}$ 转化为 $Agent'$ 的概率为

$$p_{Agent^t_{tBest}\to Agent'} = p_m^{n_1}\cdot(1-p_m)^{n-n_1} > 0\ , \quad (18)$$

因此，从 $\boldsymbol{L}^i$ 转移到 $\mathcal{L}^k$ 中任一智能体网格的概率为：

$$p_{ij,k} \geqslant p_{Agent^t_{tBest}\to Agent'} > 0\ , \quad (19)$$

所以有，$\forall k\leqslant i$，$p_{i,k}\geqslant p_{ij,k}>0$.

定理3 求解二元约束满足问题的多智能体进化算法具有全局收敛性.

证明：每个 $\mathcal{L}^i$，$i\in\boldsymbol{E}$ 均可看成时齐有限 Markov 链上的一个状态，根据定理2，该 Markov 链的转移矩阵可表示成如下的形式：

$$\boldsymbol{P}' = \begin{Bmatrix} p_{0,0} & 0 & \cdots & 0 \\ p_{1,0} & p_{1,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,0} & p_{m,1} & \cdots & p_{m,m} \end{Bmatrix} = \begin{pmatrix} \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{R} & \boldsymbol{T} \end{pmatrix}, \quad (20)$$

显然，$\boldsymbol{R}=(p_{1,0}\ p_{2,0}\cdots p_{m,0})^{\mathrm{T}}>0$，$\boldsymbol{T}\neq 0$，$\boldsymbol{C}=(p_{0,0})=(1)\neq 0$.

根据定理1可得：

$$\boldsymbol{P}'^{\infty} = \lim_{k\to\infty}\boldsymbol{P}'^{k} = \lim_{k\to\infty}\begin{Bmatrix} \boldsymbol{C}^k & 0 \\ \sum_{i=0}^{k-1}\boldsymbol{T}^i\boldsymbol{R}\boldsymbol{C}^{k-i} & \boldsymbol{T}^k \end{Bmatrix} = \begin{pmatrix} \boldsymbol{C}^{\infty} & \boldsymbol{0} \\ \boldsymbol{R}^{\infty} & \boldsymbol{0} \end{pmatrix}, \quad (21)$$

其中 $\boldsymbol{C}^{\infty}=(1)$，$\boldsymbol{R}^{\infty}=(1\ 1\cdots 1)^{\mathrm{T}}$. $\boldsymbol{P}'^{\infty}$ 是一个稳定的随机矩阵且 $\boldsymbol{P}'^{\infty}=\begin{Bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{Bmatrix}$.

因此 $$\lim_{t\to\infty} Pr\{Energy(\boldsymbol{L}^t)=0\}=1\ , \quad (22)$$

其中 P_r 为概率. 因此，定理得证.

3 实验

文献[11]利用一个标准的测试问题集合对11个已有的算法进行了比较，因此下面我们用同样的测试集合[1]本文算法的性能进行评价. 该集合包括250个有解的二元约束满足问题，每个问题有20个变量，每个变量的定义域为 $\boldsymbol{D}=\{\boldsymbol{D}_i \mid \boldsymbol{D}_i=\{1, 2, \cdots, 20\}$ 和 $1\leqslant i\leqslant 20\}$. 根据问题的难度 p，可将这250个问题分成 $p=\{0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.30, 0.31, 0.32, 0.33\}$ 10组，每组25个问题.

以下实验从三方面来评价算法的性能——成功率(success rate, SR)、平均误差(mean error, ME)和平均评价次数(average number of evaluations to solution, AES). 成功率是指找到解的运行次数占总运行次数的百分比. 误差指运行结束时，所得到的最优的智能体所不满足的约束个数，即其能量的绝对值. 平均评价次数指成功的运行中所用评价次数的平均值.

本文算法有4个参数，$L_{size}\times L_{size}$ 相当于传统进化算法中的种群，因此 L_{size} 一般从3到10中选取，本文取为5. p_c 和 p_m 分别设为0.2和0.05. $Eval_{Max}$ 设为100 000. 对每个问题独立运行10次，对一组问题的实验结果进行平均.

3.1 本文算法与已有算法的性能比较

在文献[11]所比较的11种算法中，性能最好的4种算法是 H-GA.1[3]，H-GA.3[3]，SAW[4] 和 Glass-Box[2]，因此我们按照文献[11]所给的流程和参数自行实现了这4种方法，其结果与文献[11]一致. 本节将本文算法与这4种算法进行了比较. 图2给出了比较结果. 为了方便其他学者与本文算法进行比较，表1还给出了本文算法的结果.

从比较结果可以看出，本文算法的成功率在5个算法中是最高的，且对于 p 为0.24—0.26的三组问题成功率均达到了100%. 由于算法 SAW 的平均误差远远大于其他算法，其平均误差与本文算法的比较在图2(b)的左上角给出. 图2(b)表明本文算法的平均误差仍是5个算法中最小的. 由于在成功率非常低时，用平均评价次数来比较是没有意义的，因此图2(c)只比较了成功率大于10%的问题的平均评价次数. 5种算法对 p 为0.30—0.33的四

1) htpp://www.xs4all.nl—bcraenen/resources/csps_modelE_v20_d20.tar.gz

组问题的成功率均非常低，但本文算法的平均误差是非常小的，最终求得的解只有1—3个约束为满足．总的来说，本文算法求得的解的质量是非常高的，其性能优于其他4种算法．

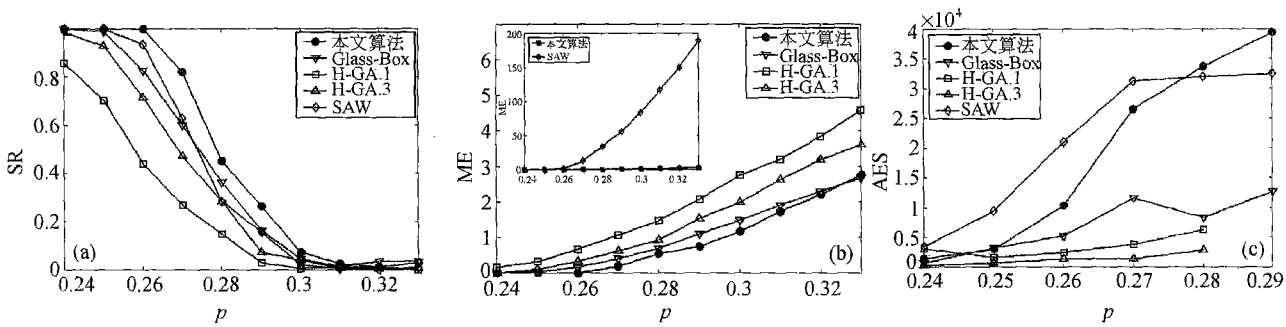

图2 本文算法与已有算法的性能比较

表1 本文算法的实验结果

p	0.24	0.25	0.26	0.27	0.28	0.29	0.30	0.31	0.32	0.33
SR	1	1	1	0.820	0.452	0.264	0.072	0.024	0.012	0.028
ME	0	0	0	0.180	0.548	0.744	1.164	1.736	2.204	2.752
AES	1400	3055	10384	26574	33666	39486	39564	29324	57428	26281

3.2 参数分析

本节 p_c 和 p_m 从0.05以步长0.05增加到1．根据本文算法的成功率，可将10组问题分成3类．第一类包括难度较低的三组问题，即 p 为0.24—0.26．由于对于大多数参数这一类的成功率均高于90%、平均误差非常小且三组问题的结果类似，图3给出了 p=0.26时的平均误差和平均评价次数．第二类包括 p 为0.27—0.29的三组问题．由于该类的成功率较低且三组问题的结果类似，图4给出了 p=0.29时的成功率和平均误差．第三类包括 p 为0.30—0.33的四组问题．由于该类问题难度较高且四组问题的结果类似，图5给出了 p=0.31和0.33时的平均误差．

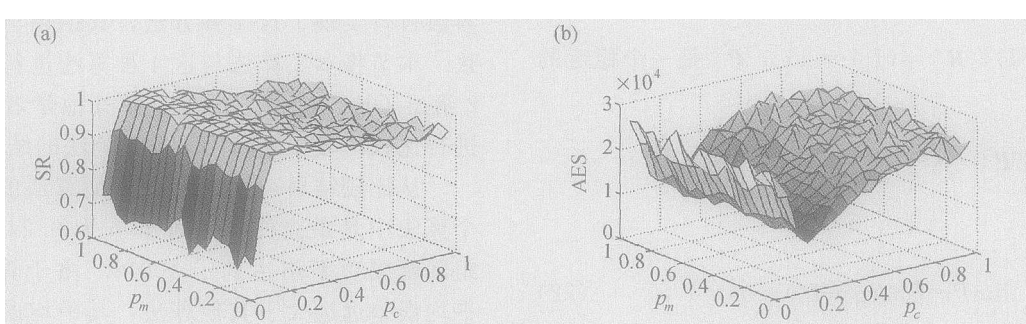

图3 本文算法对问题 p=0.26 所得的成功率和平均评价次数

由结果可见，参数 p_c 对算法性能影响较大．对于第一类，当 p_c 大于0.1时成功率均高于90%．虽然当 p_c 大于0.2时平均评价次数随着 p_c 的增长而增长，但当 p_c 在0.1—0.3之间时，平均评价次数较小．对于第二类，结果类似，仍然是当 p_c 在0.1—0.3之间时，成功率较高，平均误差较小．虽然 p_m 对算法的性能没有明显的影响，但图4和5表明当 p_m 较小时成功率较高而平均误差较小．

因此，一般情况下应从0.1—0.3间选择 p_c，从0.05—0.3间选择 p_m．另外，虽然在这个参数区

间内，本文算法获得了良好的性能，但当 p_c 大于0.2时本文算法的性能也是相当稳定的. 这表明本文算法的性能对参数不敏感，具有良好的鲁棒性，且参数较少，易于使用.

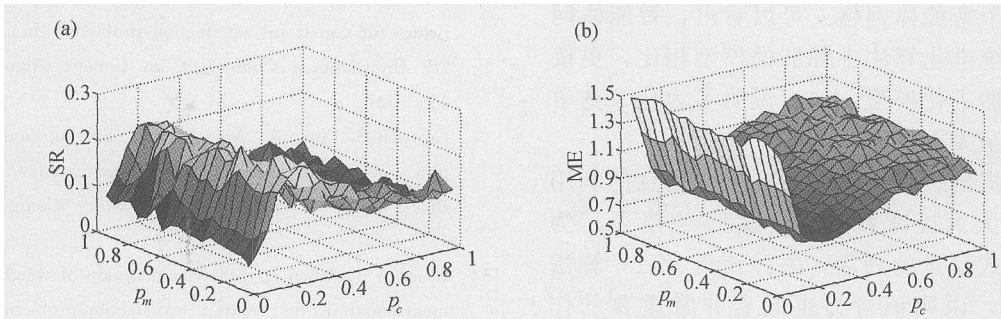

图4 本文算法对问题 $p=0.29$ 所得的成功率和平均误差

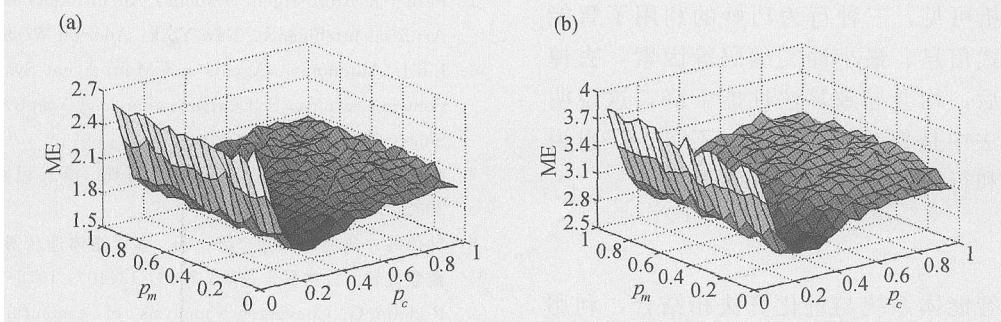

图5 本文算法对问题 $p=0.31$ 和 0.33 所得的平均误差

3.3 3种行为对算法性能的影响

竞争行为、自学习行为与变异行为是本文算法的核心，它们是智能体概念的集中体现，为了进一步分析算法的机理，本节研究这三种行为对算法性能的影响. 由上节的实验可知，p 为0.27—0.29的三组问题的难度适中，因此本节选这三组问题进行实验，内容如下：分别去掉三种行为，用剩余的两种行为构成算法来求解这三组问题. 为了保证与本文算法的公平比较，两种行为构成的算法流程与算法4一致，只是去掉对应行为的步骤. 实验结果在表2中给出.

表2 两种行为构成的算法与本文算法的性能比较

p	0.27			0.28			0.29		
	SR	ME	AES	SR	ME	AES	SR	ME	AES
本文算法	0.820	0.180	26 574	0.452	0.548	33 666	33 666	0.744	39 486
无竞争行为	0.224	1.124	26 140	0.072	1.680	20 483	0.032	2.212	24 645
无自学习行为	0.768	0.232	28 121	0.356	0.668	39 633	0.184	0.976	32 203
无变异行为	0.792	0.204	26 853	0.428	0.576	38 563	0.164	0.896	37 520

由表2可见，去掉竞争行为后，算法性能下降非常严重. 竞争行为一方面体现了智能体间的竞争，另一方面由于它是在智能体网格上进行的，因此也体现了智能体网格的作用. 在传统进化算法中，产生子代的个体通常是根据适应度从整个种群中选择出来的，因此必须先确定整个种群的适应度分布. 但自然界并不存在全局选择，也不可能确定全局适应度分布. 真正的自然选择只发生在一个局部环境中，每个个体只能与周围的个体发生作用. 在智能体网格中，竞争行为只发生在智能体与其邻

域间，不需要全局选择. 一个智能体与其邻域进行作用，以将它的信息传给他们. 在这种方式下，信息将扩散到整个智能体网格. 可以看出，智能体网格的模型与传统进化算法中种群的模型相比，更接近于真正的自然进化模型，这也说明了为什么竞争行为对算法是至关重要的.

文献[11]提出“适当的平衡启发式信息的利用和随机搜索将大大提高算法的性能”. 自学习行为实际上是启发式信息的利用，去掉该行为后，算法性能明显下降，说明该行为也起着非常重要的作用. 变异行为是最小冲突编码方法的体现，因为在此种编码方式下才能进行该行为.

由上述分析可见，三种行为巧妙的利用了智能体网格、启发式信息、最小冲突编码等因素，去掉任意一种行为后，均会导致算法性能下降，这说明三种行为在算法中是相辅相成、缺一不可的，也说明了各种因素和行为设计的有效性.

4 结论

本文将多智能体系统与进化算法相结合，利用进化的方式来控制智能体的行为以达到求解二元约束满足问题的目的. 与4种已有算法的比较结果表明本文算法的性能是最优的. 对于约束非常强的问题($p \geqslant 0.30$)，本文方法与所比较的4种方法的结果都不是很理想，这是因为相对整个空间，可行解数目极小，个体间的相关性很小，因而可相互利用的信息也非常少. 因此，如何有效处理强约束问题将是我们下一步的研究工作.

参 考 文 献

1 Rossi F, Petrie C, Dhar V. On the equivalence of constraint satisfaction problems. In: Proceedings of 9th European Conference on Artificial Intelligence, 1990, 550—556

2 Marchiori E. Combining constraint processing and genetic algorithms for constraint satisfaction problems. In: Proceedings of 7th International Conference on Genetic Algorithms, 1997, 330—337

3 Craenen B, Eiben A, Monchiori E. Solving constraint satisfaction problems with heuristic-based evolutionary algorithms. In: Procceedings of Congress Evolutionary Computation, 2000, 1571—1577

4 Craenen B, Eiben A. Stepwise adaption of weights with refinement and decay on constraint satisfaction problems. In: Proceedings of Genetic and Evolutionary Computation Conference, 2001, 291—298

5 Ferber J. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. New York: Addison-Wesley, 1999

6 Liu J. Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization, and Adaptive Computation. Singapore: World Scientific. 2001

7 韩 靖，等. AER模型中的智能涌现. 模式识别与人工智能，2002, 15(2): 134

8 钟伟才，薛明志，刘 静，等. 多智能体遗传算法用于超高维函数优化. 自然科学进展，2003, 13(10): 1078—1083

9 Rudolph G. Convergence analysis of canonical genetic algorithms. IEEE Trans. Neural Networks, 1994, 5(1): 96—101

10 Iosifescu M. Finite Markov Processes and Their Applications. Wiley, Chichester, 1980

11 Craenen B, et al. Comparing evolutionary algorithms on binary constraint satisfaction problems. IEEE Trans Evolutionary Computation, 2003, 7(5): 424—444