改进的基于分解的子图同构算法

张志祥[1,2]　李庆华[2]　罗建明[3]

(海军工程大学计算机系　武汉 430033)[1]　(华中科技大学计算机学院　武汉 430070)[2]

(海军驻海南地区检修室　三亚 572000)[3]

摘　要　分析了 Messmer 提出的基于分解的子图同构算法，指出了该算法存在的问题。从分解和组合两个方面对该算法进行了改进。改进的算法不仅解决了原有的问题，而且其性能有所提高。实验结果证明了算法的有效性。

关键词　子图同构，分治法，分解

A Modified Algorithm of Subgraph Isomorphism Based on Decomposition

ZHANG Zhi-Xiang[1,2]　LI Qing-Hua[2]　LUO Jian-Ming[3]

(Department of Computer, Naval University of Engineering, Wuhan 430033)

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430070)[2]

Abstract　The algorithm of subgraph isomorphism based on decomposition proposed by Messmer et. is analysed, and its problems are pointed out. A revised algorithm is given, which adapted the decomposition and combination procedure of the original algorithm, and resolve the problems of the original algorithm with higher effecience. The experimental result shows that this algorithm is correct.

Keywords　Subgraph isomorphism, Divide and conquer, Decomposition

1　前言

子图同构是指求一个图(称为输入图)的子图，该子图与另一个图(模式图)同构。子图同构在计算机许多领域得到广泛的应用。同构子图是 NP-完全问题，求同构子图的最通用的方法是基于搜索树的回溯。为了防止搜索树变得过大，已提出了很多不同的优化方法，如 Ullman 方法[1]和基于深度优先搜索的 VR 方法[2]，还有利用网格分割方法、神经网络方法和遗传算法[3]等。也有人采用同构子图。

Messmer 提出了一种基于"分治法"思想的同构子图方法[4]。该方法将模式图分解成两个子图和子图之间的边，接着分别进一步分解两个子图，直到子图是单个节点的图。然后从单个节点的图开始，求每个子图到输入图的子图同构。在此基础上再求由这两个子图组合而成的(子)图到输入图的子图同构，直至求出模式图到输入图的子图同构。该算法适合于以下两种情况：(1)存在多个模式图；(2)模式图之间存在着相同的子图。其优点是在相同或不同的模式图中出现多次的子图只需要与输入图匹配一次。在存在多个模式图，且所有的模式图高度相似的极端情况下，这种方法的时间复杂度并不和模式图的数量呈线性关系。但是 Messmer 算法有如下不足：对模式图的分解过程中，对分解的子图要求过于严格。它对子图的定义为：给定图 $G=(V,E)$，$S=(V_S,E_S)$ 是 G 的子图，则 $V_s\subseteq V$ 且 $E_s=E\cap(V_s\times V_s)$。在很多情况下，我们需要的条件为：$V_s\subseteq V$ 且 $E_s\subseteq E\cap(V_s\times V_s)$。

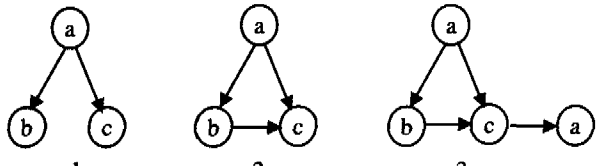

图 1　子图同构示例

例如，在图 1 中，当分别以 g_1，g_2 为模式图，而 g_3 为输入图时，使用 Messmer 算法可以得到 g_2 到 g_3 的子图同构，而无法得到 g_1 到 g_3 子图同构。本文在 Messmer 算法的基础上，针对上述问题，给出改进的基于分解的子图同构算法。下面先给出相关定义，然后分别介绍模式图的分解过程和子图同构求解过程。

2　相关定义

定义 1　图 G 是一个四元组，$G=(V,E,u,v)$，其中：V 是顶点集，$E\subseteq V\times V$ 是边集；u 和 v 分别是顶点和边到相应属性值的函数。

定义 2　给定图 $G=(V,E,u,v)$，$S=(V_S,E_S,u_S,v_S)$ 是 G 的子图，则：1)$V_s\subseteq V$；2)$E_s\subseteq E\cap(V_s\times V_s)$；3)$u_S$，$v_S$ 分别是 u，v 在 V_S，E_S 上的约束：

$$u_s(v)=\begin{cases}u(v) & \text{如果 } v\in V_s\\ \text{未定义} & \text{其它情况}\end{cases}$$

$$v_s(e)=\begin{cases}v(e) & \text{如果 } e\in E_s\\ \text{未定义} & \text{其他情况}\end{cases}$$

定义 3　给定图 $G=(V,E,u,v)$ 和 G 的子图 $S=(V_S,E_S,u_S,v_S)$，(1)如果 $V_S=V$，两者的差 $G-S$ 是边集 $E_c=E-E_s$；(2)如果 $v_s\subset V$，两者的差 $G-S$ 是 G 的子图 $G_c=(V_c,E_c,u_c,v_c)$，其中：

- $V_c=V-V_S$；$E_c=E\cap(V_c\times V_c)$
- u_c，v_c 分别是 u，v 在 V_c，E_c 上的约束：

$$u_c(v)=\begin{cases}u(v) & \text{如果 } v\in V_c\\ \text{未定义} & \text{其它情况}\end{cases}$$

$$v_c(e)=\begin{cases}v(e) & \text{如果 } e\in E_c\\ \text{未定义} & \text{其它情况}\end{cases}$$

张志祥　副教授，研究方向为程序设计方法。

如果两个图顶点之间存在保持相邻关系的一一映射，则称这两个图同构，这样的映射称为同构映射。

定义 4 给定两个图 $G=(V,E,u,v)$ 和 $G'=(V',E',u',v')$，S 是 G' 的子图，如果存在一个函数 $f:V\to V'$，且 f 是从 G 到 S 的同构，那么，称 f 是从 G 到 G' 的子图同构。

3 模式图的分解

首先将一组模式图 $B=\{G_1,G_2,\cdots,G_n\}$ 分解，得到一组由四元组 $\langle G,G',G'',E\rangle$ 构成的有限集 DB，要求：

(1) G,G',G'' 是图，且 $G',G''\in G$(是 G 的子图)

(2) 如果 $G'<>$NULL 且 $G''<>$NULL，则 $G=G'\cup_E G''$

(3) 如果 $G'<>$NULL 且 $G''=$NULL，则 $G=G'\cup_E$；

(4) 对于每个 $G_i\in B$，一定存在一个四元组 $\langle G,G',G'',E\rangle\in DB$，其中 $G_i=G$；

(5) 对于每个四元组 $\langle G,G',G'',E\rangle\in DB$，一定不存在一个四元组 $\langle G_i,G'_i,G''_i,E_i\rangle\in DB$，其中 $G_i=G$；

(6) 对于每个四元组 $\langle G,G',G'',E\rangle\in DB$，如果 G' 含有多个节点，则一定存在一个四元组 $\langle G_i,G'_i,G''_i,E_i\rangle\in DB$，其中 $G_i=G'$；

(7) 对于每个四元组 $\langle G,G',G'',E\rangle\in DB$，如果 G'' 含有多个节点，则一定存在一个四元组 $\langle G_i,G'_i,G''_i,E_i\rangle\in DB$，其中 $G_i=G''$；

DB 表示将图分割成更小的子图，直至单个顶点的图的分解结果。四元组 (G,G',G'',E) 表示对图 G 的分解，G' 和 G'' 是分解的两个部分，它们都是 G 的子图，G' 的节点数加上 G'' 的节点数等于 G 的节点数。这里又分为两种情况：(a) G'' 为空。E 在属于图 G 而不属于 G' 的边的集合(2)。(b) G'' 不为空，G' 的节点数小于 G 的节点数。E 在子图 G 中 G' 和 G'' 之间边(3)。(4)确保在 B 中每个模式图都被分解，(5)表示每个分解都是唯一的。(6)和(7)保证分解是完全的。

对于一个模式图或者是一个给定的模式图集，显然存在很多种不同的分解。本文采用如下分解方法：对于一个图 G_i，如果 $S_{max}=\max\{G\mid \exists\langle G,G',G'',E\rangle\in DB\wedge G$ 是 G_i 的子图$\}$存在，则将 G_i 分解成 S_{max} 和 G_i-S_{max}，否则将 G_i 随机(按照节点个数)分为两部分。这种分解方法的特点是它的计算量很小。

如果几个模式图 $G_i,G_j,\cdots$ 有公共子图 G，或者在模式图 G_i 中出现多次子图 G，在 $D(B)$ 中用四元组 (G,G',G'',E) 就足够了。

```
Decomposition (B)
{
  Let B={G1, G2,…,GN}, D=ϕ
    按照节点个数从小到大的顺序对 B 进行排序
for i=1 to N
  Decompose (Gi)
}
Decompose (G){
    设 D 是已有的分解结果，Smax=ϕ
    if G 中只有一个节点 then exit.
```

调用 matchGraph(G)在已分解的 $tuple\in D$ 中求每个 parent 到 graph 的子图同构，并找出其中的最大子图 S_{max}，即

$S_{max}=\max\{$parent$\mid\exists\langle$parent, sub1, sub2, E$\rangle\in$ D $\wedge$ parent 是 graph 的子图$\}$

```
if Smax 与 G 同构 then
   exit // Smax 与 graph 同构 - 无需分解，返回 Smax，
else if Smax 为空 then {
  随机选择 G 的一个子图 Smax；
  Decompose(Smax)
  Decompose (G−Smax)
  求 G 中 Smax 与 G-Smax 之间的边集 E；
  将(G, Smax, G−Smax, E)加入到 D 中
  exit
 }
else if (Smax 的节点数= = graph 的节点数 &&
      Smax 的边数() < graph 的边数) {
   求属于 G 而不属于 Smax 的边集 E，从 Smax 中删除这些边，
   并分解 graph，增加 tuple1 〈graph, Smax, NULL, E〉
else {
  Decompose (G-Smax)
  求 G 中 Smax 与 G-Smax 之间的边集 E；
  将(G, Smax, G-Smax, E)加入到 D 中
  exit
  }
}
```

Decomposition(B)的输入是要被分解的模式图集合 B，分别表示模式图集合；分解后得到元组集，用 D 表示，其初值为空。Decomposition 对 B 中的每个模式图 G 调用 Decompose (G)进行分解。对一个图的分解可以使用先前其它模式图的分解结果。

D 是一个全局变量，它保存调用所有 Decompose()产生的结果。

这种分解方法是递增的，例如给定一个用分解 $D(B)$ 表示的模式图集 B，通过简单的调用 decompose(G_{N+1}) 就会在数据库中加入新的模式图 G_{N+1}。因此，不用完全重新计算 D。这种方法尤其适用于待匹配的模式图数量很大或者需要在运行时向数据库中新增新模式图的情形。

4 同构子图的组合

Decomposition 得到一个分解 $D(B)$。如果对于其中一个四元组 $tuple=(parent,sub_1,sub_2,E)\in D(B)$ (说明 G 是由子图 sub_1、子图 sub_2 以及 sub_1 和 sub_2 之间的边集 E 组成)，已经找到了所有从 sub_1 和 sub_2 到输入图的子图同构，那么需要将它们组合成从 $parent$ 到输入图的子图同构，这一工作由 mergeTuple(Tuple tuple, Graph target)完成。图 2 是子图同构组合的示意图：如果 f_1 和 f_2 分别是 sub_1 和 sub_2 到输入图 target 的子图同构。需要判断由 $f1$ 和 $f2$ 能否组合成 parent 到 target 的子图同构 f。

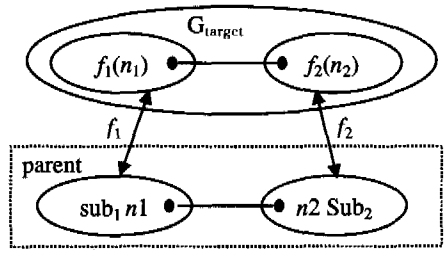

图 2 子图同构组合示意图

```
mergeTuple(Tuple tuple, Graph target) {
//假设 tuple 中的两个子图 sub1、sub2 均被标记为 ALIVE，F1，F2 是
  sub1，sub2 到 target 的子图同构映射，算法寻找从 tuple 的 parent
  到 target 的子图同构 F 并返回。
如果sub2 为空，对于子图 sub1 到 target 的每种可能的映射组合 f1∈
  F1，进行如下检查：
    对每条边 r∈Eparent，且 r∉Esub1，其中 v_parent^from(r)=v1，v_parent^to(r)=
    v2，存在一条边 r1∈Etarget，v_target^from(r1)=f1(v1)，v_target^to(r1)=f1
    (v2)，且 v_target^type(r1)=v_parent^type(r)。
    对于满足条件的 f1∈F1，得到新的映射 f：Vsub1→Vtarget，且 f
    (n)=f1(n)；并加入到 parent 的同构映射表 F 中，即 F=F∪{f}
    返回 F
否则，对于每个 f1∈F1，f2∈F2，进行如下检查：
    (1) 检查两个子图到 target 中的映射是不相交的，即 f1(Vsub1)
  ∩f2(Vsub2)=∅；
    (2) 对于两个子图到 target 的每种可能的映射组合，判断边的
  约束
      · 对每条边 r∈Eparent，其中 v_parent^from(r)=v1∈Vsub1，v_parent^to(r)=v2
        ∈Vsub2，存在一条边 r1∈Etarget，v_target^from(r1)=f1(v1)，v_target^to
        (r1)=f2(v2)，且 v_target^type(r1)=v_parent^type(r)。
```

· 对每条边 $r\in E_{parent}$，其中 $v^{to}_{parent}(r)=v_1\in Vsub_1$，$v^{from}_{parent}(r)=v_2\in V_{sub2}$，存在一条边 $r_1\in E target$，$v^{to}_{target}(r_1)=f_1(v_1)$，$v^{from}_{target}(r_1)=f_2(v_2)$，且 $v^{type}_{target}(r_1)=v^{type}_{parent}(r)$.

对于满足(1)和(2)的每种组合，合成新的映射 $f: V_{sub1}\cup V_{sub2}\rightarrow V_{target}$：

$$f(n)=\begin{cases}f_1(n) & \text{if } n\in V_{sub1}\\ f_2(n) & \text{if } n\in V_{sub2}\end{cases}$$

并加入到 *parent* 的同构映射表 F 中，即 $F=F\cup\{f\}$

返回 F

}

在 *decompose*() 和 *mergeTuple*() 的基础上，我们可以定义新的同构子图算法 *matchGraph*(*Graph GI*)，这个算法的输入是输入图 *GI*，当然算法还要使用 *decompose*() 得到的分解 $D(B)$。这个算法首先寻找分解 $D(B)$ 中最小的图到输入图 *GI* 的同构子图，然后逐渐将它们组合成大的同构子图。每一个 $D(B)$ 中的四元组(*parent*，sub_1，sub_2，*edgeset*)用三种不同的标志来标记：

· DORMANT 表示尚未求解 *parent* 到 *GI* 的子图同构；

· ALIVE 表示已经求得 *parent* 到 *GI* 的子图同构；

· DEAD 表示 *parent* 到 *GI* 的子图同构不存在。

在开始时，所有分解中的子图用 DORMANT 来标记，当子图与输入图作了同构子图测试后，它被标记为 ALIVE 或者 DEAD：如果存在同构子图，则该子图标记为 ALIVE，并记录所有已经求得的同构子图；否则，该子图被标记为 DEAD，说明它没有对应的同构子图。当所有的图要么被标记为 ALIVE，要么被标记为 DEAD 时，算法结束。最后输出所有用 $D(B)$ 表示的模式图中找到的同构子图。容易看出，这种新算法找到了所有从模式图 $G_1G_2\cdots G_N$ 到输入图 *GI* 的同构子图，而且，如果 S 是模式图的一部分，且存在四元组(S，S_1，S_2，E)，则所有从 S 到输入图 *GI* 同构子图的计算量只有一次。

matchGraph(Graph GI) {

1. 对于所有的 tuple=〈parent，sub1，sub2，edgeset〉
   如果 parents 中节点数大于 1，将 parents 标记为 DORMANT；
   否则，parents 是单节点的图，求该节点到 GI 的所有子图同构，也就是求 GI 的节点中与 parent 中的节点属性相同的节点。如果存在子图同构，将它们标记为 ACTIVE，否则标记为 DEAD；
2. 对于所有的 tuple=〈parent，sub_1，sub_2，edgeset〉：
   (2.1)如果 sub1 和 sub2 均被标记为 ALIVE，调用 *mergeTuple*(tuple，GI)求 parent 到 GI 的子图同构，如果找到这样的子图同构，将 parent 标记为 ALIVE，否则标记为 DEAD；
   (2.2)如果 sub1 和 sub2 中有一个被标记为 DEAD，则 parent 标记为 DEAD；
   (2.3)否则将 parent 标记为 DORMANT；
3. 对于所有的 tuple=(parent，sub1，sub2，edgeset)，如果存在有标记为 DORMANT 的 parent 时转 2。

}

5 例子

为了阐明这个新算法，下面给出了一个具体的例子。

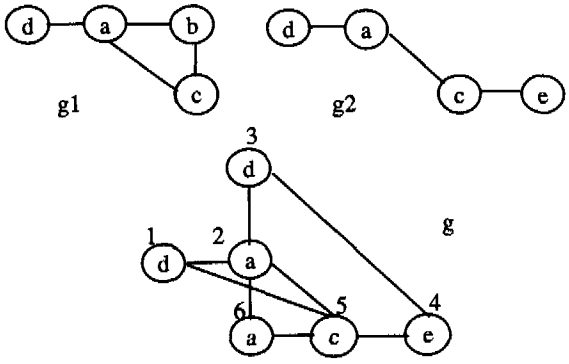

图 3 两个模式图 g_1，g_2 和一个输入图 g

图 3 给出了两个模式图 g_1，g_2 和一个输入图 g。在图 4 中，用网络图表示对 g_1 和 g_2 的分解和匹配过程。分解步骤如下：

(1)将 g_1 分解为 g_6 和 g_3；

(2)将 g_2 分解为 g_4 和 g_{10}；

(3)将 g_3 分解为 g_4 和一条边($d-c$)；

(4)将 g_4 分解为 g_5 和 g_9；

(5)将 g_5 分解为 g_7 和 g_8；

最后如下分解结果：

$DB=\{\langle g_1,g_6,g_3\rangle,\langle g_2,g_4,g_{10}\rangle,\langle g_3,g_4,\text{NULL}\rangle,\langle g_4,g_5,g_9\rangle,\langle g_5,g_7,g_8\rangle,\langle g_6,\text{NULL},\text{NULL}\rangle,\langle g_7,\text{NULL},\text{NULL}\rangle,\langle g_8,\text{NULL},\text{NULL}\rangle,\langle g_9,\text{NULL},\text{NULL}\rangle,\langle g_{10},\text{NULL},\text{NULL}\rangle\}$

其中 g_6、g_7、g_8、g_9 和 g_{10} 等 5 个图只含有一个节点。

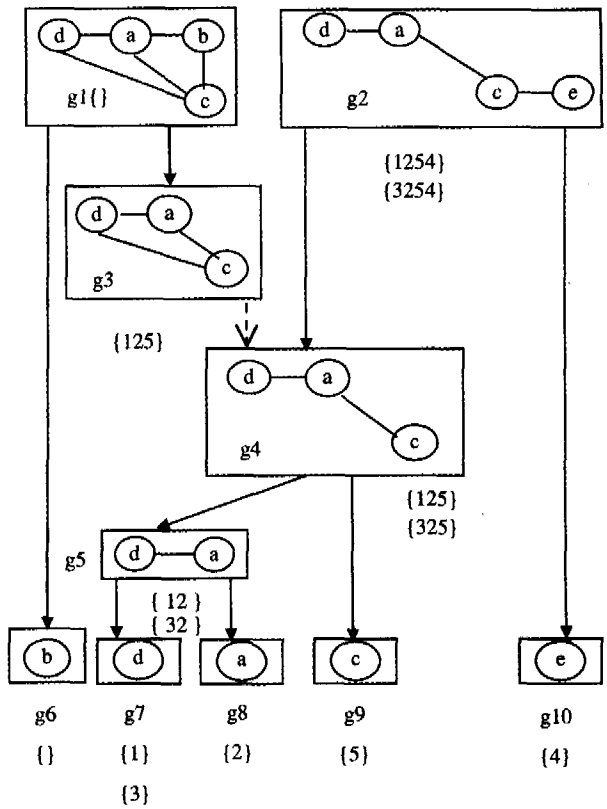

图 4 分解结果和匹配过程

首先将所有的图标记为 DORMANT。然后分别求 g_6、g_7、g_8、g_9 和 g_{10} 的顶点到 g 的同构子图，得到的结果在图 4 的每个图的下方表示，如 g_6 没有与 g 对应的节点(被标记为 DEAD)；g_7 中的顶点可以与 g 中的编号为 1 和 3 的顶点匹配，g_8 中的顶点只能与 g 中的编号为 2 的顶点匹配，它们被标记为 ALIVE；其它被标记为 DORMANT。

由于 g_5 被分解为 g_7 和 g_8，而且 g_7 和 g_8 的映射均已被求出，我们就可以根据这些信息求出 g_5 的同构映射，得到两组同构映射：{12}和{32}，同时将 g_5 标记为 ALIVE。同样可以求出 g_4 的同构映射：{125}和{325}，并将 g_4 标记为 ALIVE。当根据 g_4 的同构映射求 g_3 的同构映射时，由于图 g 中编号为 3 的顶点与编号为 5 的顶点之间没有边，所以从 g_4 的同构映射中去掉{325}，得到{125}，将 g_4 标记为 ALIVE。

同样我们可以求出 g_1 和 g_2 的同构映射。由于 g_6 的同构映射为空，所以 g_1 的同构映射也为空。将 g_1 标记为 DEAD；而 g_2 被标记为 ALIVE。并得到 g_2 到 g 的两个子图同构{1254}和{3254}。

6 算法分析

本算法的时间复杂度和空间复杂度与文[4]中的算法一

样。算法用四元组集记录分解得到的所有子图。一般来说这些子图的个数越少、子图的平均节点个数越多，那么算法的效率越高。而子图的个数与子图的平均节点个数是相关的。如果子图的个数增加，那么子图的平均节点个数就会减少。反之，子图的平均节点个数就会增加。为了保证子图的平均节点个数增加，算法进行了如下的处理：

对待处理的模式图按节点数进行排序，并按照节点从小到大的顺序进行分解。这样能保证当模式图中存在"包含"关系时，一定能体现这种包含关系。

在分解一个模式图 G 时，首先在已经得到的子图中查找最大的子图 S_{max}。然后将图 G 分解为 S_{max} 和 $G-S_{max}$ 两部分。只有当找不到这样的 S_{max} 时，才将 G 随机分解为两部分。

和 Messmer 算法相比，分解算法增加了这样一种情况：S_{max} 与 *graph* 的节点数相同，但是 S_{max} 的边数()小于 *graph* 的边数。此时，求属于 G 而不属于 S_{max} 的边集 E，从 S_{max} 中删除这些边，并分解 *graph*，增加四元组(*graph*，S_{max}，NULL，E)。这样既能保证子图节点个数的最大化，又能解决引言中提到的问题。实验结果也反映了这一优点。

参 考 文 献

1 Ullman J R. An Algorithm for Subgraph Isomorphism. J. of the Assoc. for Computing Machinery, 1976,23(1):31～42

2 Cordella L P, Foggia P, Sansone C, et al. Evaluating Performance of the VF Graph Matching Algorithm. In: Proc. of the 10th Intl. Conf. on Image Analysis and Processing, IEEE Computer Society Press, 1999. 1172～1177

3 Kuner P, Ueberreiter B. Pattern Recognition by Graph Matching Combinatorial versus Continous Optimization. Int'l J. Pattern Recognition and Artificial Intelligence, 1988,2(3):527～542

4 Messmer B T, Bunke H. Efficient Subgraph Isomorphism Detection: A Decomposition Approach. IEEE Tranactions on Knowledge and Data Engineering, 2000,12(2):307～323

(上接第 215 页)

表 7 测试函数的收敛成功率

函数	IADPSO	CPSO	SPSO	RIW	LDW	CFM
Easom	1	1	0.85	0.975	1	1
$H_{6,4}$	0.6	0.525	0.135	0.445	0.55	0.55
$S_{4,10}$	0.585	0.425	0.265	0.47	0.405	0.58
Shubert	0.03	0.03	0.015	0.01	0.025	0.04
Sphere	1	1	0.99	1	1	1

表 8 测试函数的总的 CPU 时间

函数	IADPSO	CPSO	SPSO	RIW	LDW	CFM
Easom	10.328	10.344	19.594	32.844	10.313	11.438
$H_{6,4}$	38.906	48.656	73.875	71.656	45.141	51
$S_{4,10}$	153.94	190.75	222.19	229.73	188.25	157.83
Shubert	66.625	71.531	68.375	75.094	70.578	80.484
Sphere	12.266	13.031	18.688	38.047	12.875	16.047

小结与未来工作 在本文中，我们提出了一种用于 PSO 算法的新的多样性策略，该策略的有效性在大量著名测试函数的优化上得到了验证。充分的模拟实验结果显示，基于多样性策略的混合算法在困难的多峰函数数优化问题上相比其他已有方法具有很强的竞争力。未来的工作将集中于分析集中性例子比例参数 p 的影响，对混合方法在高维函数上的性能进行研究，开展与其他多样性策略比较实验，并考虑将算法扩展至有约束多目标优化问题。

参 考 文 献

1 Kennedy J, Eberhart R C. Particle swarm optimization. In: Proc. of IEEE Intl. Conf. on Neural Networks, Piscataway, NJ, IEEE Press, 1995. 1942～1948

2 Clerc M, Kennedy J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation, 2002,6(1):58～73

3 Hu X, Eberhart R C, Shi Y H. Engineering optimization with particle swarm. In: Proc. of the IEEE Swarm Intelligence Symposium, Indianapolis, Indiana, USA, 2003. 53～57

4 Fan S-K S, Liang Y-C, Zahara E. Hybrid Simplex Search and Particle Swarm Optimization for the Global Optimization of Multimodal Functions, Engineering Optimization, 2004,36(4):401～418

5 Parsopoulos K E, Vrahatis M N. Recent approaches to global optimization problems through particle swarm optimization. Natural Computing, 2002,1:235～306

6 Lei Kaiyou, Wang Kaiyou, et al. A Novel Forward Search Strategy to Automatically Harmonize Intensification and Diversification in Tabu Search. In: Proc. of Intl. Symposium on Autonomous Decentralized Systems, Chengdu, China, 2005. 513～519

7 Kennedy J, Eberhart R C. Swarm Intelligence. Morgan: Kaufmann Publishers, 2001

8 Shi Y H, Eberhart R C. A modified particle swarm optimizer. In: Proc. of the IEEE Congress on Evolutionary Computation, Piscataway, N J, IEEE Press, 1998. 69～73

9 Shi Y H, Eberhart R C. Parameter selection in particle swarm optimization. Evolutionary Programming VII. In: Proceedings of the Seventh Annual Conference on Evolutionary Programming, New York, 1998. 591～600

10 Shi Y H, Eberhart R C. Empirical study of particle swarm optimization. In: Proc. of the IEEE Congress on Evolutionary Computation, Piscataway, NJ, IEEE Press, 1999. 1945～1950

11 Clerc M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: Proc. of the IEEE Congress on Evolutionary Computation, 1999. 1951～1957

12 Carlisle A, Dozier G. An off-the-shelf PSO. In: Proc. of the Workshop on Particle Swarm Optimization, Indianapolis, USA, 2001. 1～6

13 张丽平，俞欢军，陈德钊，胡上序. 粒子群优化算法的分析与改进. 信息与控制，2004,33(5):513～517

14 Levy A, Montalvo A, Gomez S, et al. Topics in Global Optimization. New York: Springer-Verlag, 1981