MOEA/D：基于分解的多目标遗传算法

摘 要

基于分解的方法是多目标优化的一种基本策略。然而，它们并没有被广泛的应用在多目标优化遗传算法中。这篇论文提出了一个基于分解的多目标遗传算法框架（MOEA/D）。它将多目标优化问题瓦解成多个单目标优化子问题，然后同时优化这些子问题。每个子问题只利用了邻居子问题的信息去优化，这也使得 MOEA/D 能够有比起 MOGLS 和 NSGA-II 更低的计算复杂度。基于朴素分解策略的 MOEA/D 在 0-1 背包问题，连续多目标优化问题上的实验表现与 MOGLS 和 NSGA-II 结果相近或者超过了它们。本文也证实了基于目标函数正则化的 MOEA/D，能够很好的解决由目标函数范围不同导致优化效率低的问题。并且，在一个 3 目标函数的测试样例上，基于更先进分解策略的 MOEA/D 能够生成一个均匀分散的非支配解集。种群小情况下 MOEA/D 的性能，MOEA/D 的可扩展性，参数敏感程度等问题也在本篇论文中进行了讨论。

关键词：计算复杂度；分解方法；遗传算法；多目标优化；帕累托最优

1.介绍

MOP，即多目标优化问题，其数学表示如下：

$$maximize\ F(x) = \left(f_1(x), \dots f_m(x)\right)^T$$
$$subject\ to\ x \in \Omega \qquad (1)$$

其中Ω是决策变量空间，$F{:}\,\Omega \to R^m$包括了m个实目标函数，R^m被称之为目标函数空间。目标函数集合被定义为$\{F(x)|x \in \Omega\}$。

若$x \in R^n$，目标函数皆为连续目标函数，并且Ω被定义为

$$\Omega = \{x \in R^n | h_j(x) \le 0, j = 1, \dots, m\}$$

其中$h_j(x)$是一个连续的函数。此时，我们称（1）是一个连续的多目标问题。

有一种经常出现的情况：因为（1）中的目标函数常互相冲突，在Ω空间中没有一个解，可以同时使得所有目标函数都得到优化。此时，一个解需要在多个目标函数中作出选择。最后折中得到的最优解，被称之为帕累托最优解。

令$u, v \in R^m$，当且仅当对于每一个$i \in \{1, \dots, m\}$有$u_i \ge v_i$并且至少有一个$j \in \{1, \dots, m\}$使得$u_j > v_j$时，u支配v。（1）式中，我们令解$x^* \in \Omega$是帕累托最优解，仅当

没有其他解$x \in \Omega$，能够使得$F(x)$支配$F(x^*)$。$F(x^*)$被称之为帕累托最优向量。换一句话说，在帕累托最优解集中，任何一个目标函数的优化都会带来至少一个目标函数的弱化。所有的帕累托最优解被称之为*帕累托解集*（PS）。帕累托最优向量的集合称之为*帕累托前沿*（PF）。

现实生活中有许多 MOPs，决策者们需要一个近似 PF 的解集然后进一步做选择。几乎所有的 MOPs 都可能产生许多甚至无数的帕累托最优向量。即使可以得到一个整的 PF，运算过程也是非常耗时的。另一方面说，一个决策者也并不希望得到过多的帕累托最优向量，因为信息泛滥并不是件好事。所以，许多的多目标优化算法的目标是找到一个易于处理的一个帕累托最优向量解集，并且，它们是均匀的分布在 PF 上的（这意味着，它们是整个 PF 的一个不错的代表[1]-[4]）。一些研究者们通过建立数学模型[5]-[8]来估计 PF。

很多文章指出对于 MOP 的一个帕累托最优解，在合适的情况下，也会是一个单目标函数的最优解（该函数中，目标函数是所有f_i的聚合）。因此，对于 PF 的估计，可以被分解为对一系列单目标函数子问题的估计。这是许多估计 PF 的传统数学规划方法的基本动机。可以在文献中查到许多基于分解的方法（比如文献[1]）。其中，最出名的是 weighted sum（以下称之为 WS 方法）方法和 Tchebycheff（以下称之为切比雪夫方法）方法。最近，boundary intersection methods（以下称之为边界相交法）也获得学界的许多关注（[9]-[11]）。

然而当下的 state-of-the-art 的多目标遗传优化方法，大多不是基于分解策略的[2]-[4], [12]-[19]。这些算法，并不分解 MOP 问题。每一个单独的解，并不会对应一个单目标优化问题。对于一个单目标问题来说，一切都较为简单，所有解的好坏，通过目标函数值来判断，而单目标优化的遗传算法（EA）就是去寻找唯一的一个最优解。然而，在 MOPs 中，支配这种偏序关系，并不能指明目标函数空间中所有解的排序关系。退而求其次，MOEA（多目标优化）希望能够得到一系列的帕累托最优解，越多样越好（更能够代表整个 PF）。因此，传统的针对单目标优化的选择算子（适应度计算方式），不能直接应用在非分解的 MOEAs 中。值的思考的是，如果能够设计一个适应度计算方式：使得一个解能够匹配合理的适应度值，去反映该解的优劣（是否值得选择）。那么单目标 EAs（遗传算法）就可以很轻松的应用在 MOPs 问题上。虽然，其他的一些策略，如繁殖约束，多样性评估，MOPs 的其他属性和外部种群也需要被加进来，用以提高算法的性能。因为这个原因，适应度计算是当下多目标优化遗传算法的一个重要问题。目前，流行的适应度计算方法包括，基于交替的适应度计算算法，如向量评估遗传算法（VEGA）[24]，和基于支配的适应度计算的算法，如 PAES[14]，加强帕累托遗传算法 2（SPEA-II），和非支配排序遗传算法（NSGA-II）[16]。

已经有一些基于分解策略的元启发式算法被应用在 MOPs 中[25]-[29]。比如说，

二阶段局部搜索算法(TPLS)[25]，它通过考虑一个单目标（MOP 中多个目标的聚合函数）优化子问题的集合，然后，一个单目标优化函数根据系数依次用来解决这些单目标优化问题。解决之前问题得到的解被当作在目标空间上的起始点去解决下一个子问题，因为两个相邻的问题之间，聚合目标函数是差别不大的。MOGLS，多目标遗传局部搜索算法，旨在通过切比雪夫和 WS 算法来解决每个子问题。每次迭代中，它都会优化随机生成的聚合目标函数。

在这篇文章中，我们提出了一个新的基于分解的多目标优化遗传算法（MOEA/D）。MOEA/D 将 MOP（1）分解成 N 个单目标优化子问题。通过进化种群，该算法同时对这些子问题进行优化。在每次迭代中，种群里保存每个子问题目前的最优解。子问题间的邻居关系是由它们的目标函数聚合系数来决定的。两个相邻（之后会定义什么是“相似”）子问题的最优解答一定是相似的。每个子问题（即单目标函数）都只用相邻子问题的信息来进行优化。MOEA/D 有如下几个特点：

1. MOEA/D 使得基于分解的思想可以简单但是有效的利用到多目标遗传算法里去。基于分解的许多方式，往往是在数学规划领域提出来的，可以被轻松的应用到 EAs，帮助 MOEA/D 解决 MOPs。
2. 因为 MOEA/D 旨在解决 N 个单目标优化问题，而不是解决一个多目标优化问题，诸如适应度匹配，多样性保护（会给不是基于分解的算法）带来的问题，在 MOEA/D 中会更好的处理。
3. MOEA/D 比起 NSGA-II，MOGLS 来说，每次遗传迭代的时间复杂度更小。总的来说，在分解方式相同时，同样测试 0-1 多目标背包问题，MOEA/D 的表现比 MOGLS 更好。在连续的多目标优化测试上，基于切比雪夫分解方式的 MOEA/D 表现和 NSGA-II 相近。在有三个目标函数的多优化实例上，基于更加先进分解方式的 MOEA/D 比 NSGA-II 结果好很多。MOEA/D 在种群小的情况下，也能生成均匀分布的解。
4. 如果你要解决目标函数取值范围迥异的问题，目标函数的正则化可以被轻易的整合到 MOEA/D 中。
5. MOEA/D，使得单目标优化的那套方法自然的应用到 MOP 里，因为每个解都与一个单目标优化问题相联系。相反，对于不是基于分解的 MOEAs，它们的一大缺点就是，并没有一种简单的方法能够利用单目标优化的好处。

这篇文章的结构如下：第二部分介绍解决 MOPs 的三种分解方法。第三部分正式引出 MOEA/D。第四部分，第五部分是 MOEA/D 和 MOGLS 以及 NSGA-II 的对比。第六部分是实验部分。第七部分是最后的总结。

2.三种基于分解的方式

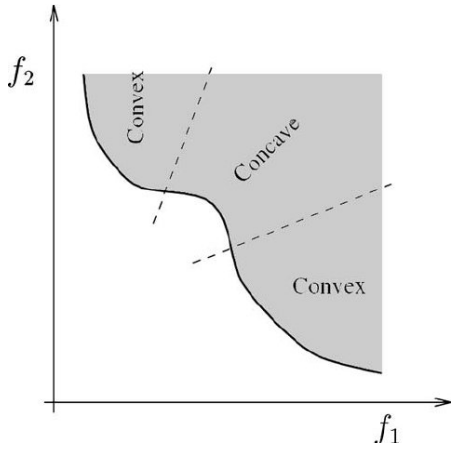
目前有不少方式可以将近似优化 PF 这个问题转变为一系列的单目标优化问题。以下部分，我们介绍三种基于分解的方式。同时，我们也实验了这三种分解方式的效果。

A.加权方式[1]

这种方式将不同的目标函数进行凸组合。令$\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_m)^T$为一个权重向量（凸组合满足：$\lambda_i \geq 0, \sum_{i=1}^{m} \lambda_i = 1$）。求解下面单目标优化函数，就能得到关于（1）式的帕累托最优点。

$$maximize\ g^{ws}(x|\boldsymbol{\lambda}) = \sum_{i=1}^{m} \lambda_i f_i(x)$$
$$subject\ to\ x \in \Omega \qquad (2)$$

其中， λ是系数向量，x是需要优化的自变量。为了得到一系列的帕累托最优向量，我们可以利用不同的系数矩阵。如果该 PF 是凹曲线，该方法的效果会不错（对于最小化问题，PF 对应的则是是凸曲线的情况）。然而，对于非凹的 PF 来说，不是每一个帕累托最优向量都可以得到。为了解决这样的问题，一些技巧如ε-constraint，可以加在这种分解方式中。更多的资料可以在这里找到[1]。



（译者注）

B.切比雪夫方式[1]

在这种方式中，单目标优化问题可以如下表示：

$$minimize\ g^{te}(x|\lambda, z^*) = \max_{1 \leq i \leq m} \{\lambda_i |f_i(x) - z_i^*|\}$$
$$subject\ to\ x \in \Omega \qquad (3)$$

其中，$z^* = (z_1^*, \dots, z_m^*)^T$是参考坐标。对所有的$i = 1, \dots m$而言，定义$z_i^* = \max\{f_i(x)|x \in \Omega\}$。对于每一个帕累托最优点$x^*$都会存在一个权重向量$\lambda$使得$x^*$是（3）式的帕累托最优解，并且每个（3）式的最优解都是一个（1）式的帕累托最优点。因

此，我们通过改变权重向量就可以得到不同的帕累托最优解。这种方式的一个缺点就是对于连续的多目标优化，它的聚合目标函数并不顺滑。但是，我们在本次实验中，并不用求该聚合目标函数的导数。因此，该方法依然可以用用到 MOEA/D 框架中。

C.边界相交（BI）法

在最近被提出的一些 MOP 分解方式，比如 Normal-Boundary Intersection Method [9]，和 Normalized Normal Constraint Method [10]都可以被归类为边界相交分解法。它们是为了连续 MOP 所提出的。在某些常见条件下，MOP 的 PF 是可以取到的目标函数集合区域的最右上方边界（对于最大化而言）。几何上来讲，这些 BI 方法旨在找到最右上方边界和一个直线集合的交点。如果这些直线是均匀分布的，那么我们可以期待，得到的交点，因为其均匀分布性，可以是整个 PF 不错的近似。

在这篇文章中，直线从参考点出发。数学定义上讲，我们考虑下述的单目标优化子问题：

$$\begin{aligned} &minimize\ g^{bi}(x|\lambda, z^*) = d \\ &subject\ to\ z^* - F(x) = d\lambda, \\ &x \in \Omega \end{aligned} \tag{4}$$

其中，λ, z^*分别是权重向量以及参考点。如图 1 所示，约束条件$z^* - F(x) = d\lambda$是为了确保$F(x)$永远在直线 L 上（方向由权重向量决定，穿过z^*）。想法就是将$F(x)$值推到右上方的边界处。

该方式的缺点就是它需要去处理上述的等式约束。在我们的实现中，我们用了惩罚版本的 BI 分解方法。数学定义如下：

$$\begin{aligned} &minimize\ g^{bip}(x|\lambda, z^*) = d_1 + \theta d_2 \\ &subject\ to\ x \in \Omega \end{aligned} \tag{5}$$

其中，

$$d_1 = \frac{\| \left(z^* - F(x)\right)^T \lambda \|}{\| \lambda \|}$$

$$and\ d_2 = \| F(x) - (z^* - d_1\lambda) \|$$

$\theta > 0$是预设好的惩罚系数。令y为$F(x)$在线 L 的投影。如图 2 所示，d_1是$F(x)$和L的直线距离。若θ合适，对于式子（4），（5）的解应该会非常接近。因此，我们称该方法为惩罚边界相交法（PBI）。和切比雪夫分解方法比较，BI 方法的优点有

1. 对于相同的一些权重向量，PBI 生成的解会比切比雪夫生成的解更加均匀，特别是当这些权重向量的数量较小的时候。
2. 即使x 支配 y，仍然可能有$g^{te}(x|\lambda, z^*) = g^{te}(y|\lambda, z^*)$。在 PBI 或者 BI 方法中，这种可能性很低。

但是，PBI 方法也有其麻烦的地方。我们需要确定惩罚系数的值。该值太大或者太小都会影响算法的性能。

上述的方式都能将近似求解 PF，将 MOP 转变为一系列的子问题。通过生成一些合理大小的，均匀分布的权重向量，通常能够得到一些的帕累托最优向量，虽说不一定均匀分布，但也能够不错的估计整个 PF。

当然，文献中还有不少的基于分解的方法可以应用在 MOEA/D 中。因为我们的主要目的是研究 MOEA/D 的可行性和效率，我们只讨论这三种分解方式。也只利用了这三种方式做实验。

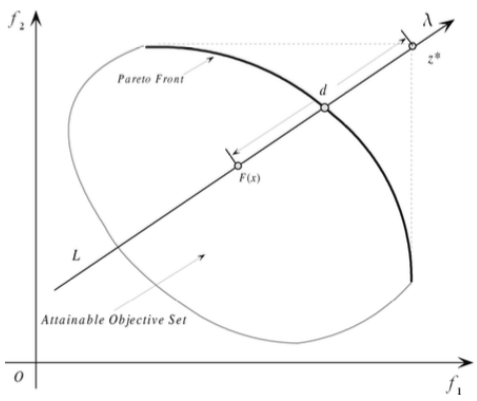


图 1 边界相交法的介绍

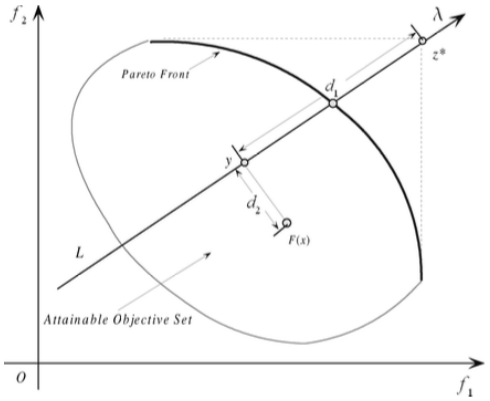


图 2 惩罚边界相交法的介绍

3.基于分解的多目标优化的框架

A.整体框架

本篇文章中提出的 MOEA/D 算法，需要在合适的条件转化 MOP。任何一种基于分解的方式都能达到这个目的。在下面的叙述中，我们假设切比雪夫的方法被应用在 MOEA/D 中。修改 MOEA/D 采取的分解方式很容易，并不是我们关注的重点。

令$\lambda^1,\dots,\lambda^N$为一组均匀分布的权重向量，$z^*$是参照点。就如在第二部分中说的那样，通过切比雪夫方式，可以将估计 PF 的问题（1）降解成 N 个单目标优化问题。第j个子问题的目标函数为

$$g^{te}\left(x\middle|\lambda^j,z^*\right)=\max_{1\le i\le m}\left\{\lambda_i^j\middle|f_i(x)-z_i^*\middle|\right\} \tag{6}$$

其中$\lambda^j=(\lambda_1^j,\dots,\lambda_m^j)$。在一轮迭代中，MOEA/D 同时最小化所有的 N 个目标函数。

注意到g^{te}是关于λ连续的，如果权重向量λ^j，λ^i相似的话，关于$g^{te}\left(x\middle|\lambda^i,z^*\right)$和$g^{te}(x|\lambda^j,z^*)$的最优解理应也是相似的。因此，所有和权重向量$\lambda^i$相邻的子问题都能有效的优化$g^{te}\left(x\middle|\lambda^i,z^*\right)$。这是 MOEA/D 背后的主要动机。

在 MOEA/D 中，λ^i的邻居被定义为一些离它最近的一些权重向量构成的集合。那么，第i个子问题的邻居就包括了该权重向量的那些邻居向量对应的子问题。种群是由

每个子问题中的最优解构成的。只有其邻居子问题的解才能用来优化该问题。

在第t次迭代，基于切比雪夫方法的 MOEA/D 保留以下的信息：

（1） 一个 N 大小的种群，$x^1,\dots,x^N \in \Omega$。其中，x^i是第i个子问题当前找到的最优解

（2） $FV^1,\dots,FV^N$，其中FV^i是x^i的 F 函数值。即，$FV^i = F(x^i)$，对于所有$i = 1,\dots,N$。

（3） $z = (z_1,\dots,z_m)^T$，其中z^i是对于目标函数f_i找到的最优的值。

（4） 一个外部种群（EP）。在整个搜索过程中，它用来存储非支配解。

该算法的流程图如下

算法：MOEA/D
Input：MOP（1），终止条件，N（种群的大小），均匀分布的 N 个权重向量，T：邻居空间的大小 Output：EP 步骤 1） 初始化： 步骤 1.1）令$EP = \phi$ 步骤 1.2）计算任意两个权重向量的欧式距离，对于每个权重向量得到 T 个最近的权重向量。对于每个$i = 1,\dots,N$，令$B(i) = \{i_1,\dots,i_T\}$，令$\lambda^{i_1},\dots,\lambda^{i_T}$是$\lambda_i$最近的 T 个权重向量。 步骤 1.3）采取特定的算法或随机的生成一个初始种群$x^1,\dots,x^N$。初始化$FV^i = F(x^i)$ 步骤 1.4）根据应对的问题来初始化参考点$z = (z_1,\dots,z_m)^T$。 步骤 2）更新： 对于$i = 1,\dots,N$，依次执行以下操作 步骤 2.1）繁殖：从$B(i)$随机选取两元素l,k。之后，通过基因算子使得父解x^k,x^l生成新解y。 步骤 2.2）改进：利用专门的修改方法或者启发式算法将解y改进为y'。 步骤 2.3）更新z：对于每个$j = 1,\dots,m$。如果$z_j < f_j(y')$，那么令$z_j = f_j(y')$。 步骤 2.4）更新邻居解：对每个元素$j \in B(i)$，如果$g^{te}\left(y'\middle|\lambda^j,z\right) \le g^{te}(x^j|\lambda^j,z)$，那么令$x^j = y'$并且$FV^i = F(y')$。 步骤 2.5）更新 EP：移除 EP 中所有被$F(y')$支配的解。如果 EP 中没有解支配$F(y')$将$F(y')$加进去。 步骤 3） 停止条件：如果满足停止条件，终止算法并返回 EP。否则，返回步骤 2）

在初始化中，$B(i)$包括了λ^i最近的 T 个权重向量的索引。我们用欧式距离计算任

意两个权重向量的相近程度。因此，λ^i最近的权重向量是它自己，并且$i \in B(i)$。第 j 个子问题可以被看作是第 i 个子问题的邻居，当且仅当$j \in B(i)$。

在第 i 轮迭代中，T 个相邻的子问题也被考虑。在步骤 2.1 中，因为x^k, x^l是其子问题的当前最优解，那么生成的解 y 也很有希望是一个很不错的解。在步骤 2.2 中，一个解决某种特定问题的启发式算法被应用于改进解y，以防生成的解不满足问题的约束限制，还可以优化g^{te}。因此，得到的解y'是可行的，而且很有可能它的g^{te}值比第i个子问题的邻居解低。在步骤 2.4 中，所有的邻居解都被考虑了，用以更新相邻子问题。因为得到一个准确的参照点z^*往往是一件非常耗时的任务，我们用在步骤 1.4 中初始化的，在步骤 2.3 被更新的z，作为z^*的替代。外部种群 EP，在步骤 1.1 中初始化，在步骤 2.5 中会被解y'更新。

如果（1）式的目标是求最小值，需要将步骤 1.3 中的不等式方向调整。

B.讨论

为什么 MOEA/D 仅考虑部分的子问题作优化： MOEA/D 提前选定了 N 个权重向量，花费在每个聚合目标函数优化的时间接近，相反 MOGLS 在每次迭代中都会随机的生成一个权重向量，旨在去优化所有的子问题。因为决策者实际上只需要有限的均匀的帕累托解，去优化一部分子问题不仅是现实的而且也是合理的。因为算力资源总是受限制的，优化所有的子问题是不现实的，并且很耗时。

MOEA/D 中的多样性保护是怎么做到的：在第一部分提到过，为了生成优质解，MOEA 需要保存解的多样性。基本上所有的 MOEAs，比如 NSGA-II 和 SPEA-II 应用了嘈杂距离来控制解的多样性。然而，在这些算法中，得到一个均匀分布的帕累托最优目标向量并不总是轻松的。在 MOEA/D 中，一个 MOP 被分解成了多个单目标优化问题。种群中的不同的解维护着不同子问题的集合。子问题的多样性自然地引导着种群的多样性。如果基于分解的方法和权重向量选的恰到好处，那么所有子问题最优解就会均匀的分布在 PF 上。如果对于每个子问题都优化好了，MOEA/D 会有较大的可能得到一系列均匀分布的帕累托解。

MOEA/D 中繁殖限制和参数 T 的作用：T 定义了邻居的范围大小。只有邻居子问题才会用来优化对应的解 MOEA/D。因此，两个解进行基因算子仅可能两解互为邻居解。这就是繁殖限制。需要特别注意 T 值的选取。如果 T 太小了，在步骤 2.1 中选取出来的解进行基因操作后，得到的新解可能和原解非常类似，因为两父解的相似性很高。并会导致算法的搜索能力下降。另一方面来说，如果 T 太大了，x^k, x^l有可能不是子问题的“合格”的解。因此，它们生成的解y'也可能是该子问题的劣质解。因此，该算法的搜索能力被弱化了。另外，增加 T 的大小，也会使得算法复杂度提高。

C.MOEA/D 的变种

我们可以在 MOEA/D 中使用其他的基于分解的方法。当 WS 的方法应用时，我们不需要维护 z 。

步骤 2.2 允许 MOEA/D 算法能够很自然的使用单目标优化方法。我们只需要将$g^{te}\left(x\middle|\lambda^i, z\right)$或者$g^{ws}(x|\lambda^j)$当作某些启发式算法的目标函数就好了。虽然 2.2 是该算法的一大特色，但也不是必须的，特别是当步骤 2.1 能够生成有效解的时候。

使用外部种群 EP 也不是必要的，虽然它能够极大的提高算法的性能。一个替代的做法是，返回一个最终的种群作为 PF 的近似。当然，其他用来更新 EP 的复杂的策略[21]可以被轻易的整合到 MOEA/D 框架中。我们可以限制 EP 的大小从而避免内存溢出问题。

Murata[40]等人提出的蜂窝式多目标基因算法（cMOGA）可以被认为是基于分解的遗传算法。从根本而言，它们也用了邻居关系作为繁殖限制。cMOGA 与 MOEA/D 的不同在于解的选取和内部种群的更新。每次轮回，为了解决非凸的 PF，它需要从外部种群中取出解插入到内部种群中。其原因在与，它主要使用了 WS 方法，并且没有机制存储内部种群中每个子问题对应的最优的解。

4.和 MOGLS 算法的对比

以下内容中，我们首先介绍 MOGLS 算法，之后分析该算法和 MOEA/D 的时间复杂度。我们还比较了该算法在 0-1 背包问题的测试例子的实验结果。用 MOGLS 做比较是因为该算法是基于分解算法的，并且它在 0-1 背包问题上的效果比许多算法要好[29]。

A. MOGLS

MOGLS 最先由 Ishibuchi，和 Murata[28]提出，之后由 Jaszkiewicz 改进[29]。基本的想法就是将 MOP（1）重新表达为同时优化所有的带权切比雪夫，WS 目标函数的问题。在以下内容中，我们给出 Jaszkiewicz 版本的 MOGLS 的简要说明。

每次迭代中，MOGLS 维护以下信息：

（1） 当前解集（CS），和这些解集的 F 函数值。

（2） 一个外部种群（EP），是用来存储非支配解的。

如果 MOGLS 使用切比雪夫分解方式，它们还会维护以下信息：

（1） $z = (z_1, \dots, z_m)^T$，其中z_i是搜索过程发现的最大的f_i值。

MOGLS 中有两个控制参数 K 和 S。K 是临时精英种群的大小，S 是 CS 的初始大小。

算法流程图如下：

算法：MOGLS
Input：MOP（1），终止条件，K（临时精英种群的大小），S（初始种群的大小） Output：EP 步骤 1） 初始化： 步骤 1.1）用问题特定的方法或者随机生成S个初始解答$x^1, \dots x^S$。CS 表示为$\{x^1, \dots x^S\}$。 步骤 1.2）根据应对的问题来初始化参考点$z = (z_1, \dots, z_m)^T$。 步骤 1.3）EP 被初始化为来自 CS 中非支配解的 F 函数值。 步骤 2）更新： 步骤 2.1）繁殖：随机均匀的生成权重向量λ。在该权重向量下，通过比较切比

雪夫聚合目标函数g^{te}，在 CS 中选出 K 个最优解，形成临时精英解集（TEP）。在 TEP 中随机选择两个解，进行遗传操作后形成新解y。

步骤 2.2）改进：利用专门的修改方法或者启发式算法将解y改进为y'。

步骤 2.3）更新z：对于每个$j = 1, \ldots, m$。如果$z_j < f_j(y')$，那么令$z_j = f_j(y')$。

步骤 2.4）更新 TEP 中的解：比较g^{te}，如果y'比 TEP 中最差的解好，并且和所有在 TEP 中解的 F 函数值都不同，将其加入到 CS 中。

步骤 2.5）更新 EP：移除 EP 中所有被$F(y')$支配的解。如果 EP 中没有解支配$F(y')$将$F(y')$加进去。

步骤 3） 停止条件：如果满足停止条件，终止算法并返回 EP。否则，返回步骤 2）

如同 MOEA/D 中应用切比雪夫方法，在 MOGLS 中z也被g^{te}中z^*的替代。如果使用 WS 方法，g^{te}需要被替换成g^{ws}，并且不会存储z。因此，步骤 2.3 可以被去掉。MOGLS 需要保存所有解的 F 值，因为步骤 2.4 中会需要用来计算g^{te}。

B. 比较 MOEA/D 和 MOGLS 的复杂度

（1） 空间复杂度：在算法执行过程中，MOEA/D 需要保存 N 个解（内部种群），外部种群 EP；MOGLS 需要保存种群 CS 和外部种群 EP。CS 的大小会不断上升直到达到上限（作者建议设置大小为$K \times S$[29]）。因此，如果最大上限远大于 MOEA/D 使用的 N 并且两个算法生成一样数量的非支配解，MOEA/D 的空间复杂度会比 MOGLS 低。

（2） 时间复杂度：MOEA/D 和 MOGLS 的时间复杂度都主要消耗在步骤 2 中。两者每个轮回都会生成一个带审判的解y'。

a) 步骤 2.1 中：MOGLS 需要生成 TEP，因此算法需要计算 CS 中的所有g^{te}值，所需$O(m \times |CS|)$的时间复杂度。如果应用最朴素的算法对 TEP 中的解进行选择，还需要$O(K \times |CS|)$的时间复杂度。相反，在 MOEA/D 的步骤 2.1 中，只需要随机选取两个解并进行基因操作就可以了。注意到$|CS|$，即 CS 的大小，可能是非常大的（被设置为 3000～7000 在[29]中）。因此对于步骤 2.1 来说，MOGLS 的复杂度是远大于 MOEA/D 的。

b) 对于步骤 2.2，2.3，两者的时间复杂度相同。

c) 2.4 步中，MOEA/D 需要$O(T)$的时间复杂度，而 MOGLS 需要$O(K)$的时间复杂度。如果两者参数设置的接近，两步骤的时间复杂度差异不大。

因此，我们的结论是：MOEA/D 步骤 2 的时间复杂度低于 MOGLS 步骤 2 的时间复杂度。

C. 多目标 0-1 背包问题

n 个物体和 m 个背包的多目标优化 0-1 背包问题（MOKP）的数学表达如下

$$maximize\ f_i(x) = \sum_{j=1}^{n} p_{ij}x_j,\ \ i = 1, \dots, m$$

$$subject\ to \sum_{j=1}^{n} \omega_{ij}x_j \le c_i,\ \ i = 1, \dots, m$$

$$x = (x_1, \dots, x_n)^T \in \{0,1\}^n \qquad (7)$$

其中，$p_{ij} \ge 0$，代表物品 j 装到背包 i 的利润。$\omega_{ij} \ge 0$是物品 j 在背包 i 中的重量。c_i 代表背包 i 的容量。$x_j = 1$意味着物品 j 被选中，并且被放入了所有的背包中。

和许多资源调度问题类似，MOKP 是 NP-hard 问题。文献[13]中设计了该问题的 9 个测试样例，并且被广泛利用于测试多目标启发式算法。MOGLS 在这些测试样例上的表现超过了许多 MOEAs。在本篇文章中，我们使用这 9 个样例来比较 MOGLS 和 MOEA/D。

D. 关于 MOKP 问题 MOEA/D 和 MOGLS 的实现

1）修复方式：为了将 EA 算法应用于 MOKP，我们需要一个启发式算法来修改不满足约束条件的解。[13]，[29]提出了一些修复方法。

令$y = (y_1, \dots, y_m)^T \in \{0,1\}^n$是关于（7）式的不合理解。注意到（7）式中的$\omega_{ij} \ge 0$，$p_{ij} \ge 0$，我们可以移除一些物品（即：将某些$y_i$从 1 变成 0），从而使该解可行。最近，Jaszkiewicz 提出了下述的贪心修复方式。

算法：MOKP 贪心修复方式

Input：MOP（7），一个解$y = (y_1, \dots, y_m)^T$，一个需要最大化的目标函数 $g: \{0,1\}^n \to R$

Output：一个可行的解

步骤 1） 如果y可行，返回y。

步骤 2）令$J = \{j | 1 \le j \le n\ and\ y_j = 1\}$，令$I = \{i | 1 \le i \le m\ and\ \sum_{j=1}^{n} \omega_{ij}y_j > c_i\}$。

步骤 3）设置$k \in J$，使得

$$k = \arg\min_{j=J} \frac{g(y) - g(y^{j-})}{\sum_{i \in I} \omega_{ij}}$$

其中y^{j-}是仅不同于y在第 j 个位置上的解。令$y_k = 0$，返回步骤 1。

在本方法中，物品一个个的被移掉，直到y成为一个可行解。一个较重的物品

（即，$\sum_{i\in \mathrm{I}}\omega_{ij}$），并且对$g(x)$没有太大贡献（即，$g(y)-g(y^{j-})$）更可能被移除。

2）MOGLS 的实现：为了比较的公正，我们直接使用了 Jaszkiewicz 最新的 MOGLS 算法应用在 MOKP 问题中。基于切比雪夫的 MOGLS 实现的细节如下。

算法：MOGLS-MOKP

初始化z：令f_i为目标函数，使用修复方式使得解变为可行解。令z_i为该点的f_i值。

初始化 EP 和 CS：设置$EP=\phi$和$CS=\phi$。之后，重复 S 个轮回。

1. 随机生成一个权重向量，通过[29]中给出的样例算法
2. 随机生成一个解$x=(x_1,\ldots,x_n)^T\in\{0,1\}^n$。$x_i=1$的概率为 0.5
3. 令$-g^{te}(x|\lambda,z)$作为目标函数，利用修复方法使得x成为一个可行解
4. 将x'加入到 CS 中。更新 EP：移除 EP 中所有被$F(y')$支配的解。如果 EP 中没有解支配$F(y')$，将$F(y')$加进去。

注：

遗传算子有交叉和变异：交叉是将两个父解生成一个子解，然后再用变异算法改进解，变异算法在解 y 上各个位置上的变异概率为 0.01。

（3） MOEA/D 在 MOKP 问题的实现：我们利用了和 MOGLS 在步骤 2.1 和 2.2 中一样的遗传算子和修复方法。z的初始化和 MOGLS 一样。x^i的初始化（关于第 i 个子问题的初始解）过程如下所示

a) 令$-g^{te}(x|\lambda^i,z)$作为目标函数。利用修复方法使得该解可行

切比雪夫的聚合目标函数被应用在 MOEA/D 和 MOGLS 中。此外，利用 WS 方法进行实验和利用切比雪夫进行实验大体一致。除了前者利用y^{ws}作为修复算法的目标函数，并且不维护z的信息。

E. 参数设置

MOGLS 中参数 S 和 K 的设置（决定了 CS 的大小），是和文献[29]中一样的。对于所有测试样例，K 都被设置为 20。S 参数对于不同测试样例设置见表 1。

对于所有的测试样例，MOEA/D 中的 T 参数被设置为了 10。关于 MOEA/D 中参数 N 的选取以及$\lambda^1,\ldots,\lambda^N$的选取是由参数 H 控制的。具体来说，$\lambda^1,\ldots,\lambda^N$中的每一个权重向量的每个元素（共 m 个）都从下面的集合取值

$$\{\frac{0}{H},\frac{1}{H},\ldots,\frac{H}{H}\}$$

因此可以计算出，权重向量的所有可能排列情况有$N=C_{H+m-1}^{m-1}$种。（译者注：$C_{H+m-1}^{m-1}=\sum_{i=0}^{i=m-1}C_{H-1+i}^{i}$。可以通过 0 出现的情况枚举得到右边的各项）

表 1 列出了所有样例对应的 MOEA/D 的参数 N，H 的取值。对于有两个目标函数的样例，MOEA/D 中的 N 的取值和 MOGLS 中 S 的取值是一样的。对于所有具有三个目标函数的样例来说，$H = 25$，因此$N = 351$。对于所有具有四个目标函数的样例来说，$H = 12$，因此$N = 455$。需要注意的是 MOGLS 中内部种群 CS 的大小是会达到$K \times S$的，所以是比 MOEA/D 的内部种群大很多。

上述在 MOEA/D 中生成权重向量的方法在本次实验中作用很好。但是，当目标函数的个数过大的时候，它们也可能生成非常大的种群。为了解决这种方法，我们可以通过更先进的方法[30],[31]来生成权重向量。

两个算法在调用$500 \times S$次修复方法后就会终止。

在本次实验中，g^{te}, g^{ws}都被用在了修复方法上。在下述内容中，W-MOEA/D（W-MOGLS）代表 MOEA/D（MOGLS）利用了g^{ws}。T-MOEA/D（T-MOGLS）代表 MOEA/D（MOGLS）利用了g^{te}。

Instance		S	N (H)
m: # of objectives	n: # of items	in MOGLS	in MOEA/D
2	250	150	150 (149)
2	500	200	200 (199)
2	750	200	250 (249)
3	250	200	351 (25)
3	500	250	351 (25)
3	750	300	351 (25)
4	250	250	455 (12)
4	500	300	455 (12)
4	750	350	455 (12)

表 1 0-1 背包问题上， MOGLS 和 MOEA/D 的参数设置

Decomposition Method			Tchebycheff		Weighted Sum	
	m	n	MOEA/D	MOGLS	MOEA/D	MOGLS
Instance	2	250	3.93	26.47	3.70	29.37
	2	500	10.40	70.80	9.40	72.97
	2	750	20.13	127.30	17.87	129.60
	3	250	7.00	81.70	6.53	78.17
	3	500	17.50	147.70	15.30	137.13
	3	750	31.90	219.30	26.73	202.47
	4	250	17.33	188.80	19.60	186.17
	4	500	42.60	292.10	45.17	287.60
	4	750	75.17	425.33	70.93	396.20

表 2 MOGLS 和 MOEA/D 的平均 CPU 耗时比对

Decomposition Method			Tchebycheff		Weighted Sum	
	m	n	$C(A,B)$	$C(B,A)$	$C(A,B)$	$C(B,A)$
Instance	2	250	93.17	3.99	45.06	40.81
	2	500	95.45	3.54	55.93	29.47
	2	750	97.22	2.21	73.98	15.85
	3	250	78.83	7.88	45.22	21.76
	3	500	90.97	2.42	61.74	10.09
	3	750	98.03	0.39	78.01	4.33
	4	250	29.01	26.48	19.34	26.11
	4	500	35.98	14.44	25.53	16.41
	4	750	52.70	6.49	41.40	8.52

表 3 两个算法的 C-指标的比对

Decomposition Method			Tchebycheff		Weighted Sum	
	m	n	MOEA/D	MOGLS	MOEA/D	MOGLS
Instance	2	250	54.10 (4.57)	96.38 (6.03)	37.17(2.98)	38.18(3.21)
	2	500	184.85 (11.88)	328.00 (19.85)	79.07(5.41)	98.63 (9.16)
	2	750	437.07 (21.19)	765.66 (44.95)	166.04(13.63)	274.20 (22.70)
	3	250	158.71 (6.58)	217.25 (8.45)	97.75(7.23)	141.21 (12.25)
	3	500	489.27 (15.91)	701.70(27.40)	270.31 (11.92)	419.15 (23.86)
	3	750	960.17 (23.62)	1378.64 (54.63)	446.12(19.12)	768.30 (31.86)
	4	250	253.23 (7.17)	301.32 (6.33)	176.52 (7.25)	266.17 (9.27)
	4	500	763.96 (14.81)	964.41 (24.85)	431.94(11.59)	725.16 (24.57))
	4	750	1546.44 (27.78)	1994.78 (72.84)	761.57 (17.29)	1246.54 (29.16)

表 4 两个算法在 D-指标上的比对

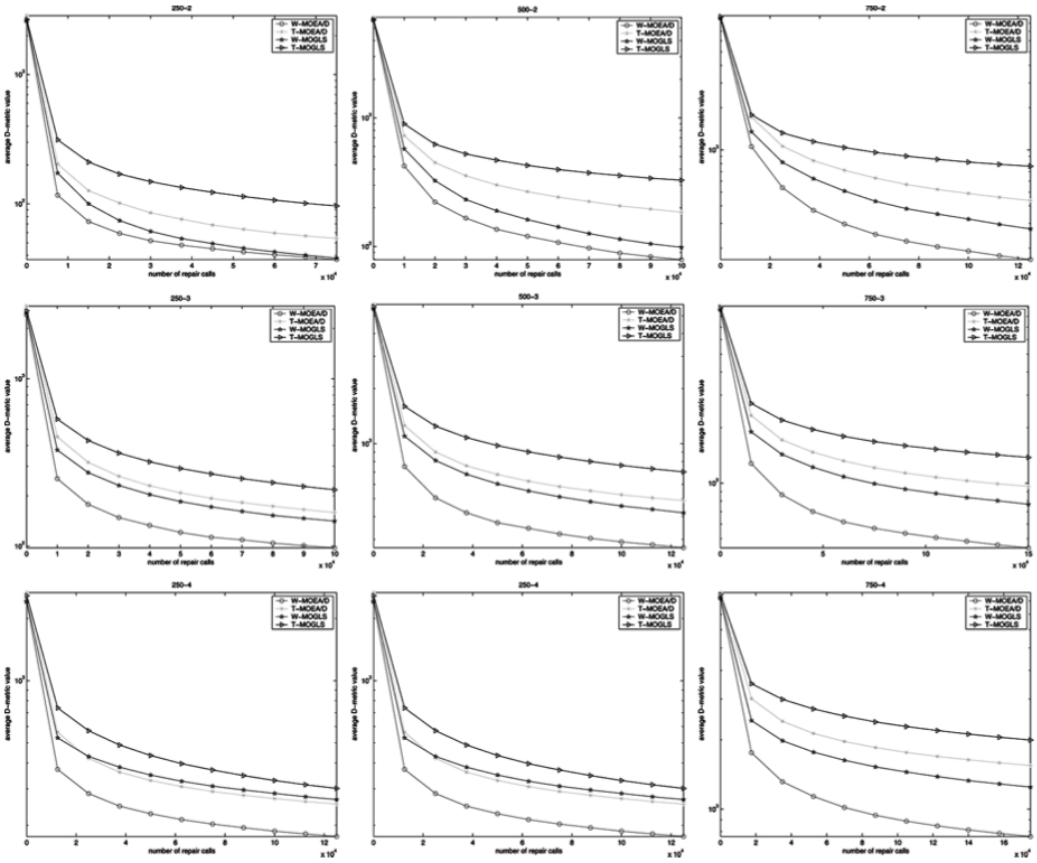


图 3 MOGLS 和 MOEA/D 算法迭代中 D-指标的变化

F. 实验结果

对于每一个测试样例，两个算法都独立的运行30次。测试电脑的配置是Pentium(R) 3.2GHZ，1.00GB。因为MOPs问题的特性，一些相关的指标需要被引入来评比算法的优劣性[2]，[32]。在我们的实验中，下面的几项指标被采纳了进来。

（1） Set Coverage（C-指标）：令 A，B 分别是两个关于MOP问题对PF的近似。$C(A,B)$被定义为 B 中的解被至少一个 A 中的解所支配的比例。即，

$$C(A,B)=\frac{|\{u\in B\}\exists v\in A: v\ dominates\ u|}{|B|}$$

需要注意的是，$C(A,B)$不一定就等于$1-C(B,A)$。$C(A,B)=1$代表 B 的所有解都被至少一个 A 中的解所支配。$C(A,B)=0$。代表 B 中的所有解都不被 A 中的解支配。

（2） Distance from Representatives in the PF （D-指标）：令P^*是一组在PF上均匀分布的点。令 A 是一组关于PF的估计。P^*到A的平均距离定义如下

$$D(A,P^*)=\frac{\sum_{v\in P^*} d(v,A)}{|P^*|}$$

其中$d(v,A)$代表最小的v到A中点的欧几里得距离。如果$|P^*|$足够大，能够代表整个PF，那么$D(A,P^*)$可以衡量解集 A 的多样性和收敛度。$D(A,P^*)$要足够低的话，A 中的解需要与PF非常接近，并且不遗失PF中的任何一部分。

如果对于不能找到整个PF的情况，我们令P^*为PF的上界估计。通过解决式子（3）的线性规划松弛问题（用了一组均匀分布的λ），Jaszkiewicz得到每组0-1背包问题样例的很好的上界估计。对2目标的样例来说，估计的点有202个；对3目标的样例来说，估计的点有1326个，对4目标的样例来说，估计的点有3276个。我们的实验按照Jaszkiewicz的估计设定P^*。

表2给出了两个算法执行不同样例的平均CPU消耗时间。表3给出了关于两种算法，两种修复方式的平均C-指标。表4给出了两个算法在不同样例上的D-指标的平均值和标准差。图3中给出了$D(EP,P^*)$的实验结果。因为D-指标的范围过大，因此在图3中，我们用了对坐标轴进行了指数化。在图4，图5中，对于2目标函数的问题，图中选取了30次实验中D-指标最好的非支配解来做图的。

我们可以得到以下结论：

a) 在一样的迭代次数下，显然易见MOEA/D比MOGLS更快。平均下来MOEA/D仅所需MOGLS消耗的14%的CPU时间。换句话说，MOEA/D的速度比MOGLS快7倍。这和我们之前的分析是一致的。

b) 图 3 清晰的表明了对于所有的测试样例，W-MOEA/D（T-MOEA/D）需要更少的迭代次数去降低 D-指标。这表明了 MOEA/D 是在 MOKP 上更为有效的优化方式

c) 表 3 和表 4 表明 W-MOEA/D（T-MOEA/D）得到的最终 EP 是比 W-MOGLS（T-MOGLS）好的，无论是从 C-指标还是 D-指标而言。在所有的测试样例中，只有样例 250-4 上，对于 C-指标，W-MOEA/D 比 W-MOGLS 稍逊一筹。举个 500-3 的例子，平均来说，T-MOGLS 生成的 90.97%的解都被 T-MOEA/D 支配了。相反而言，T-MOEA/D 生成的 2.42%的解被 T-MOGLS 支配。图 5 给出两个算法在样例 250-2，500-2，750-2 上的可视化展示。图 4 给出了 W-MOEA/D 和 W-MOGLS 在样例 750-2 上得到的帕累托前沿的区别展示。

d) 表 4 可以看出 W-MOEA/D（T-MOEA/D）和 W-MOGLS（T-MOGLS）中前者 D-指标的标准差更低，对于所有样例都是这样。可以看出 MOGLS 的稳定性是比 MOEA/D 更弱的。

e) 表 4 和图 3 展示了 WS 方法在两种算法上都比切比雪夫算法效果好。这可以看出不同的分解方式对于不同的算法，有不同的影响。

总的来说，对于 MOKP 的样例来说，我们能够认为 MOEA/D 在估计 PF 上和在时间复杂度上都比 MOGLS 要好。

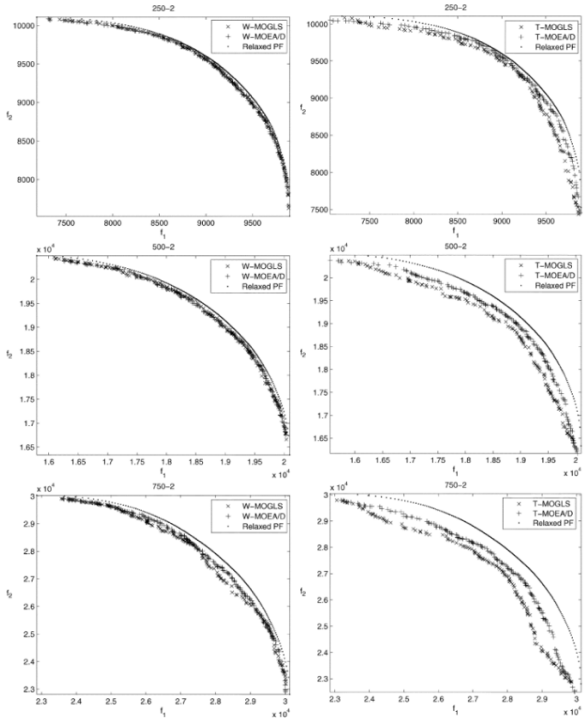


图 4 D-指标的解在解空间里的分布（WS）图 5 D-指标的解在解空间里的分布（切比雪夫）

5.和 NSGA-II 算法的对比

为了比较 NSGA-II 和 MOEA/D，我们用了广泛使用的 5 个 2 目标测试样例 ZDT 测试集[33]和 2 个 3 目标的测试样例[34]。所有的测试样例目标都是最小化所有的目标函数。

(1) ZDT1：

$$f_1(x) = x_1$$

$$f_2(x) = g(x)\left[1 - \sqrt{\frac{f_1(x)}{g(x)}}\right]$$

$$where\ g(x) = 1 + \frac{9(\sum_{i=2}^{n} x_i)}{(n-1)}$$

(2) ZDT2：

$$f_1(x) = x_1$$

$$f_2(x) = g(x)\left[1 - \left(\frac{f_1(x)}{g(x)}\right)^2\right]$$

其中$g(x)$和x的维度和范围都是和 ZDT1 一样的。ZDT2 的 PF 是非凸的。

(3) ZDT3：

$$f_1(x) = x_1$$

$$f_2(x) = g(x)\left[1 - \sqrt{\frac{f_1(x)}{g(x)}} - \frac{f_1(x)}{g(x)}\sin(10\pi x_1)\right]$$

其中$g(x)$和x的维度和范围都是和 ZDT1 一样的。它的 PF 是离散的。两个目标函数范围是不同的。f_1范围从 0～0.852，f_2范围从-0.773 到 1。

(4) ZDT4：

$$f_1(x) = x_1$$

$$f_2(x) = g(x)\left[1 - \sqrt{\frac{f_1(x)}{g(x)}}\right]$$

其中，

$$g(x) = 1 + 10(n+1) + \sum_{i=2}^{n}[x_i^2 - 10\cos(4\pi x_i)]$$

并且，$x = (x_1, \ldots, x_n)^T \in [0,1] \times [-5,5]^{n-1}$。它有许多局部 PFs。在本实验中$n = 10$。

(5) ZDT5：

$$f_1(x) = 1 - \exp(-4x_1)\sin^6(6\pi x_1)$$
$$f_2(x) = g(x)\left[1 - \left(\frac{f_1(x)}{g(x)}\right)^2\right]$$

其中，$g(x) = 1 + 9\left[\frac{(\sum_{i=2}^{n} x_i)}{n-1}\right]^{0.25}$。并且，$x = (x_1, \ldots, x_n)^T \in [0,1]^n$。它的 PF 是非凸的。解在帕累托前沿上的排列是非常不均匀的。即，对于一组在决策空间的帕累托解，它们在帕累托前沿上是聚在中间的。在本实验中，$n = 10$。

(6) DTLZ1：

$$f_1(x) = \big(1 + g(x)\big)x_1x_2$$
$$f_2(x) = \big(1 + g(x)\big)x_1(1 - x_2)$$
$$f_3(x) = (1 + g(x))(1 - x_1)$$

其中，$g(x) = 100(n-2) + 100\sum_{i=3}^{n}\{(x_i - 0.5)^2 - \cos[20\pi(x_i - 0.5)]\}$

并且，$x = (x_1, \ldots, x_n)^T \in [0,1]^n$。它的 PF 是非凸的。帕累托最优解的函数值有 $\sum_{i=3}^{3} f_i = 1$ with $f_i \geq 0, i = 1,2,3.$ 在本实验中，$n = 10$。

(7) DTLZ2：

$$f_1(x) = \big(1 + g(x)\big)\cos\left(\frac{x_1\pi}{2}\right)\cos\left(\frac{x_2\pi}{2}\right)$$

$$f_2(x) = \big(1 + g(x)\big)\cos\left(\frac{x_1\pi}{2}\right)\sin\left(\frac{x_2\pi}{2}\right)$$

$$f_3(x) = \big(1 + g(x)\big)\sin\left(\frac{x_1\pi}{2}\right) \qquad (8)$$

其中，

$$g(x) = \sum_{i=3}^{n} x_i^2$$

并且，$x = (x_1, \ldots, x_n)^T \in [0,1]^2 \times [-1,1]^{n-2}$。它的 PF 是非凸的。帕累托最优解的函数值有$\sum_{i=3}^{3} f_i^2 = 1$ with $f_i \geq 0, i = 1,2,3.$ 在本实验中，$n = 10$。

A. NSGA-II [16]

NSGA-II 并不需要外部种群。在 t 时刻它只维护一个种群大小为 N 的P_t，并生成P_{t+1}。下述内容介绍如何产生新种群。

（1） 直接利用选择，交叉，变异算子得到P_t的后代种群Q_t。

（2） 从$P_t \cup Q_t$中选择 N 个最佳的解形成P_{t+1}。

NSGA-II 的主要特点是它利用了快速非支配排序和嘈杂距离来比较解的质量（步

骤 2）和它的选择方式。它每次迭代的计算复杂度为$O(mN^2)$。其中m是目标函数的个数，N 是种群的大小。

B.更改 MOEA/D 用以比较

为了比较的公正性，我们用了以下版本的 MOEA/D 来进行我们的实验

（1） 我们不维护一个外部种群 EP。作为输出的是最后一轮的内部种群，我们把它当作对 PF 的近似估计。步骤 2.5 便不需要了。

（2） 没有改进手段，没有修复方法。因此步骤 2.2 也不需要了。

（3） 在步骤 2 中，我们使用g^{te}。

因为在某些样例中 WS 方法没法很好的处理非凸的 PFs，我们不采用g^{ws}。因为 NSGA-II 没有外部种群，我们也不维护外部种群。对比 NSGA-II，我们多余的一点要求就是 MOEA/D 需要维护和更新z。z的大小是$O(m)$，比较小，可以被忽略。

C.比较两者的时间复杂度

在上述所给的 MOEA/D 的版本中，主要的时间消耗花费在步骤 2。每次迭代，在步骤 2 中，MOEA/D 会产生 N 个待审判的种群，和 NSGA-II 的每次迭代是一样的。注意到 MOEA/D 的步骤 2.2 和步骤 2.5 已经被移除了，步骤 2.1 是随机的挑选两个解用以基因操作。步骤 2.3 需要$O(m)$次比较和赋值。步骤 2.4 需要$O(mT)$的时间复杂度，因为需要计算 T 个解的g^{te}函数值，每次需要消耗$O(m)$的时间。因此，第 2 步骤的时间复杂度总共是$O(mNT)$，总共 N 轮。如果 MOEA/D 和 NSGA-II 的总群大小一致的话，那么两者的时间复杂度的比值是

$$\frac{O(mNT)}{O(mN^2)} = \frac{O(T)}{O(N)}$$

因为 T 是比 N 小的，因此该改动版本的 MOEA/D 的时间复杂度小于 NSGA-II。

D.实验参数设置

在本次实验中，NSGA-II 的实现和文献[16]一致。对于 2 目标函数的测试样例，两个算法的种群的大小都设置为 100。对于 3 目标函数的测试样例，两个算法的种群的大小都设置为 300。两个算法总迭代次数皆为 250 次。

两个算法的初始种群都是通过在可行的解空间中均匀随机抽样生成而来的。MOEA/D 中的z^i被初始化为初始种群中最小的f_i。MOEA/D 和 NSGA-II 都采用的是 Simulated Binary Crossover（SBX）和 Polynomial Mutation。具体讲，在 MOEA/D 的步骤 1 中，先通过交叉算法生成一个后代解，然后用变异操作对解进行修改。两个算法中的交叉和变异的参数被设置为相同。就像[16]中实现的那样，Simulated Binary

Crossover 和 Polynomial Mutation 的分布索引都被设置为 20。交叉的概率为 100%。变异的概率为$\frac{1}{n}$。其中n是自变量的数量。

在 MOEA/D 中，权重向量的设置同第 4 部分 E 小节，T 被设置为 20。

在所有的测试样例上，两个算法分别独立的进行 30 次后，得到最终的结果评估。

E.实验结果

表 5 给出了每个测试样例上两个算法的平均 CPU 耗时。很明显，在 2 目标函数的测试样例上，MOEA/D 的平均速度是 NSGA-II 的两倍。在 3 目标函数的测试样例上，MOEA/D 的速度是 NSGA-II 的 8 倍。这个结果是和第 5 部分 D 小节的分析一致的。

在 MOKP 中，我们用了 C-指标和 D-指标来比较两个算法的性能。计算 D-指标的过程中，对于 2 目标函数的测试样例上，P^*被设置为了 500 个均匀分布在 PF 上的点的集合。3 目标函数的测试样例上，P^*被设置为了 990 个均匀分布在 PF 上的点的集合。

当目标函数的计算非常耗时的时候，能够降低函数的执行次数会是很重要的。图 6 中给出了，对于所有测试样例，同样的函数执行次数下，两个算法关于现种群和P^*的 D-指标值。该图体现了，在同样的目标函数执行次数下，对于 D 指标的优化，在 ZDT4，ZDT6，DTLZ1，DTLZ2 测试样例上，MOEA/D 的收敛速度是快于 NSGA-II 的。在其他三个测试样例上有一样的收敛速度，或者稍微慢一些。

表 6 给出了这样的结果，就 C-指标而言，除了测试样例 ZDT4 之外，MOEA/D 的结果都比 NSGA-II 的要好。

表 7 给出了，所有测试样例上两个算法关于 D-指标的平均值和标准差。结果显示，就 D-指标而言，在 ZDT4，ZDT6，和其他两个 3 目标函数样例上，MOEA/D 是比 NSGA-II 结果要好的。在其他三个例子上要差一些。

图 7，8 给出了在目标函数空间上，两个算法最好的 D-指标值的解集（从 30 次实验中取得）。很明显可以看出就解的规整性而言，MOEA/D 在 ZDT1，ZDT2，ZDT6 和其他两个 3 目标函数样例上比 NSGA-II 要好。ZDT4 上，两者是一样的。在 ZDT3 上，MOEA/D 要差一点，MOEA/D 在 PF 的左边两段解集上找到的解更少。MOEA/D 在 ZDT3 上的表现不佳可能是由于 ZDT3 的目标函数范围是不同的。

从上述结果中，我们可以得到基于切比雪夫方式的 MOEA/D 耗时更少，在 3 目标函数样例上表现更好，在 2 目标函数样例上表现和 NSGA-II 相近。

表 5 NSGA-II 和 MOEA/D 算法平均 CPU 耗时

		NSGA-II	MOEA/D
Instance	ZDT1	1.03	0.60
	ZDT2	1.00	0.47
	ZDT3	1.03	0.57
	ZDT4	0.77	0.33
	ZDT6	0.73	0.27
	DTLZ1	10.27	1.20
	DTLZ2	8.37	1.10

表 6 NSGA-II 和 MOEA/D 的 C-指标比较

		$C(A,B)$	$C(B,A)$
Instance	ZDT1	15.88	1.64
	ZDT2	15.53	5.37
	ZDT3	14.61	2.92
	ZDT4	10.74	23.09
	ZDT6	99.56	0
	DTLZ1	7.84	0.49
	DTLZ2	9.91	0.00

表 7 NSGA-II 和 MOEA/D 算法 D-指标的平均值和标准差的比较

		NSGA-II	MOEA/D
Instance	ZDT1	0.0050 (0.0002)	0.0055 (0.0039)
	ZDT2	0.0049 (0.0002)	0.0079 (0.0109)
	ZDT3	0.0065 (0.0054)	0.0143 (0.0091)
	ZDT4	0.0182 (0.0237)	0.0076 (0.0023)
	ZDT6	0.0169 (0.0028)	0.0042 (0.0003)
	DTLZ1	0.0648 (0.1015)	0.0317 (0.0005)
	DTLZ2	0.0417 (0.0013)	0.0389 (0.0001)

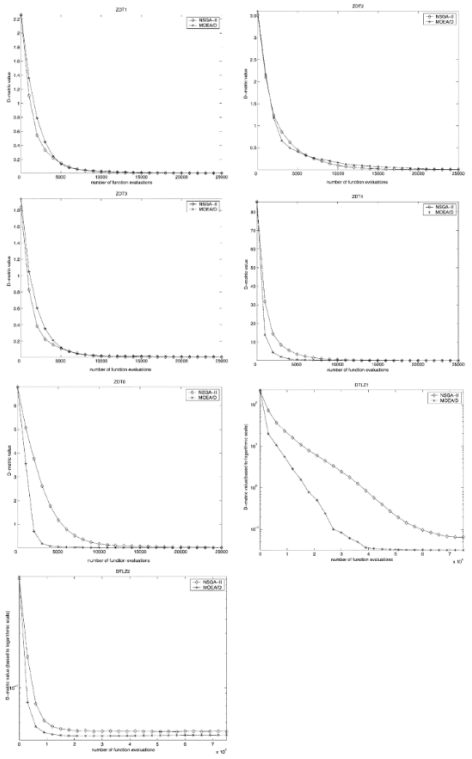


图 6 NSGA-II 和 MOEA/D 中 D-指标随着迭代次数的变化

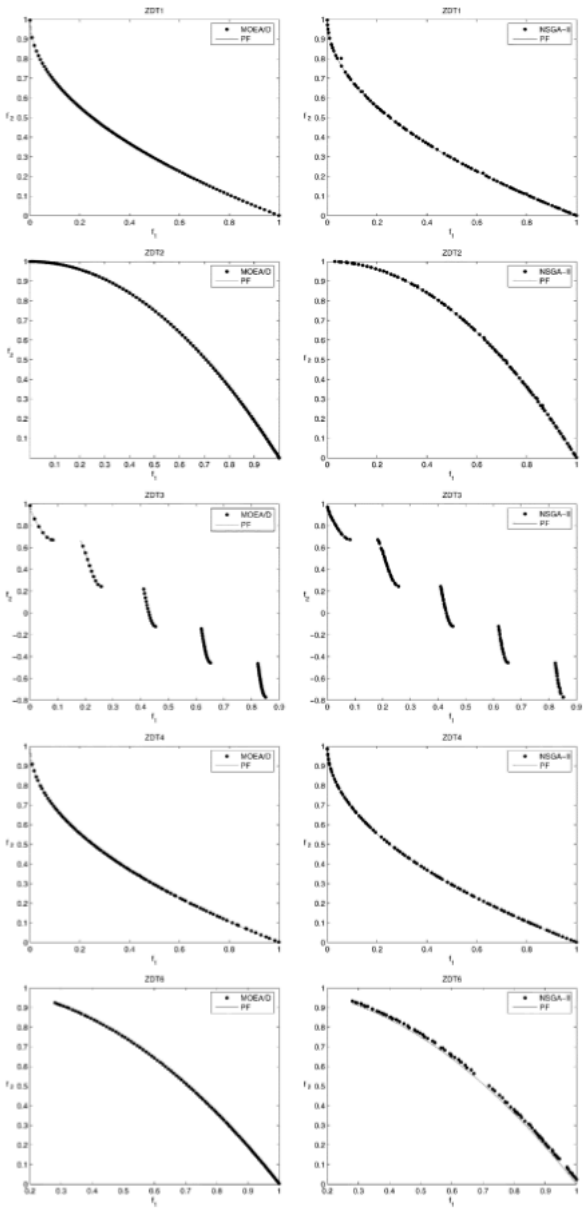


图 7 最优 D 指标解的分布（NSGA-II 和基于切比雪夫的 MOEA/D 算法，分别在右边和左边）

F.改进 MOEA/D

在上述的实验中，我们关于 MOEA/D 的实现非常简单。我们采取的分解方式是经典的切比雪夫方式。我们也没有进行目标函数的正则化。

如同在第二部分第 C 小节中谈到的，就生成解的均匀性而言，切比雪夫的方式可能会比 BI 方式更差，特别是当目标函数的数量多于 2 个。将目标函数正则化是常用的一种提高解的规整性，均匀特点的方法，特别是当各个目标函数范围不同时。

重新审查图 7，8，在 ZDT3，DTLZ1，DTLZ2 上，MOEA/D 解的均匀性并不好。注意到 ZDT3 的两个目标函数规模是有较大差异的，并且 DTLZ1，DTLZ2 有三个目标函数。其自然的引出了以下的两个问题。

（1） 在三目标测试样本上，MOEA/D 基于更先进的分解方式，如 PBI，能否找到更加均匀分布的解？

（2） 如果采用目标函数正则化，MOEA/D 能否在目标函数范围不同的 ZDT3 上表现的更好？

我们围绕这两个问题做了一些实验。下面我们讲一下实验结果。

（1） 基于 PBI 分解方式的 MOEA/D：在两个 3 目标测试样例上，我们测试了基于 PBI 方式的 MOEA/D。在实验中，g^{bip}的惩罚因子被设置为 5，其他的设置是和之前实验中基于切比雪夫的 MOEA/D 一样的。

表 8 给出了 NSGA-II 和两种 MOEA/D 在 D-指标上的比较。图 9 给出了 30 次实验中具有最低的 D 指标值的非支配解的分布（本次基于 PBI 分解方式的实验）。通过这些结果和图 8，可以明显的看出，基于 PBI 方式的 MOEA/D 是比 NSGA-II 和基于切别雪夫方式的 MOEA/D 要好的。这些积极的实验结果表明利用更先进的分解方式去解决多于 2 两个目标函数的 MOP 问题是非常值得的。我们需要指出，用这些更加先进的分解方式，是会需要更多的实验，比如说设置好 PBI 的惩罚参数。

（2） 目标函数的正则化：我们仍然使用切比雪夫方式，并不改变任何的实验参数设置。我们只是加了目标函数的正则化手段去解决目标函数范围差异较大这种情况。

在数学规划，计算遗传学等领域，有很多研究目标函数正则化的的文章[1]，[35]，[36]-[38]。一种简单的更改目标函数f_i的方式

$$\overline{f}^i = \frac{f_i - z_i^*}{z_i^{nad} - z_i^*} \tag{9}$$

其中，$z^* = (z_1^*, \dots, z_m^*)^T$，是参照点。$z^{nad} = \left(z_1^{nad}, \dots, z_m^{nad}\right)^T$，是目标函数空间的最低点，即，$z_i^{nad} = \max\{f_i(x) | x \in PS\}$。换句话说，$z^{nad}$定义了 PF 的上界，帕累托前沿。这样的做法使得，PF 空间上的任意目标函数范围变成了[0,1]。

提前计算z^{nad}和z^*并不是必要的。在我们的实验中，我们用 z 替代了z^*（参考前面章节）。用$\tilde{z}^{nad}$替代z^{nad}，现有种群中最大的f_i值。因此步骤 2.4 中的基于切比雪夫的 MOEA/D 的g^{te}被替换成了

$$\max_{1 \le i \le m} \left\{ \lambda_i \left| \frac{f_i - z_i}{\tilde{z}^{nad} - z^{nad}} \right| \right\} \tag{10}$$

我们在 ZDT3 和 ZDT1 的改动版本（为了使得目标函数范围差异大，我们将原来的f_2改为了$10f_2$）上测试了此 MOEA/D。图 10 给出了一次实验中有无目标正则化的 MOEA/D 得到的非支配解。图 11 给出了目标正则化使用后的 MOEA/D 在一次实验中

得到的非支配解。

图 10，11，一起也给出了 NSGA-II 和没有目标正则化的 MOEA/D 的 ZDT3 上的结果。很明显在目标函数范围不同的样例中，就解的规整性而言，目标正则化极大的提高了 MOEA/D 的算法性能。

总的来说，就之前引出的两个问题，我们得到的实验结果是积极的。利用其他的正则化手段或者是分解手段是之后可以研究的内容。

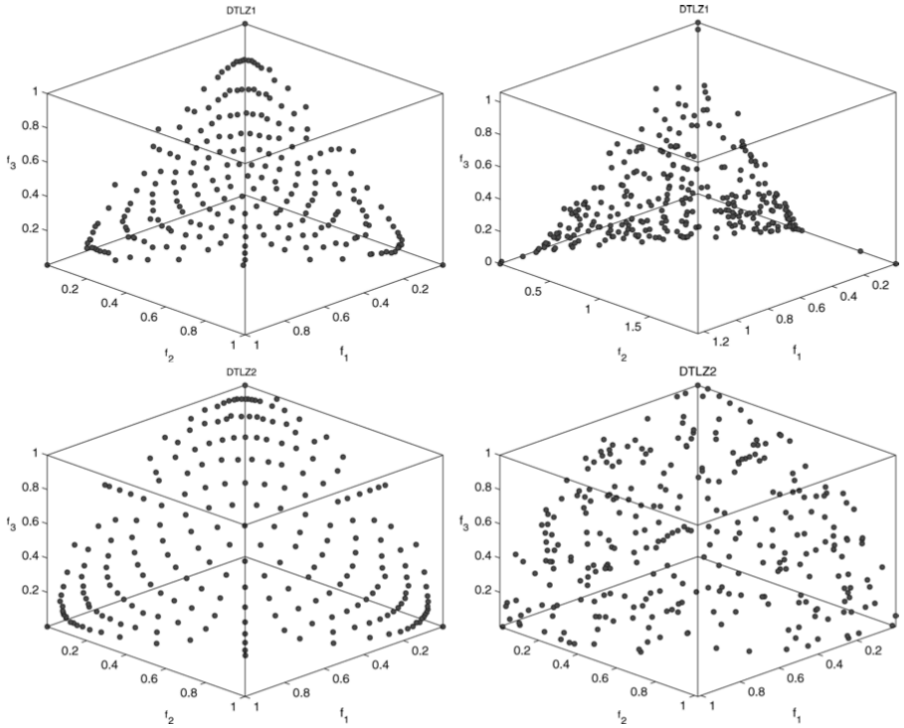


图 8 最优 D 指标解的分布（NSGA-II 和基于切比雪夫的 MOEA/D 算法，对于 3 目标函数样例）

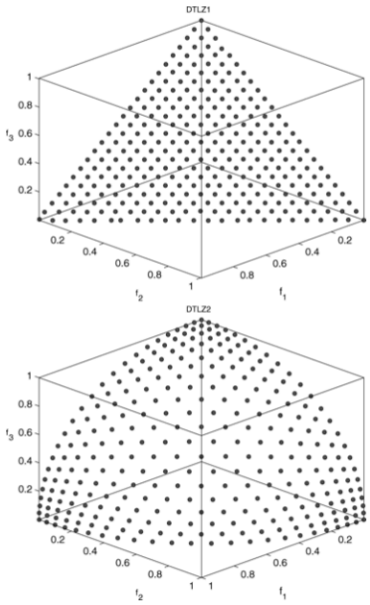


图 9 最优 D 指标解的分布（基于 PBI 的 MOEA/D 算法，对于 3 目标函数）

表 8 三种算法 D-指标的平均值和标准差的对比

Instance	NSGA-II	MOEA/D with Tchebycheff	MOEA/D with PBI
DTLZ1	0.0648 (0.1015)	0.0317 (0.0005)	0.0232 (0.0018)
DTLZ2	0.0417 (0.0013)	0.0389 (0.0001)	0.0280 (4.7034e-006)

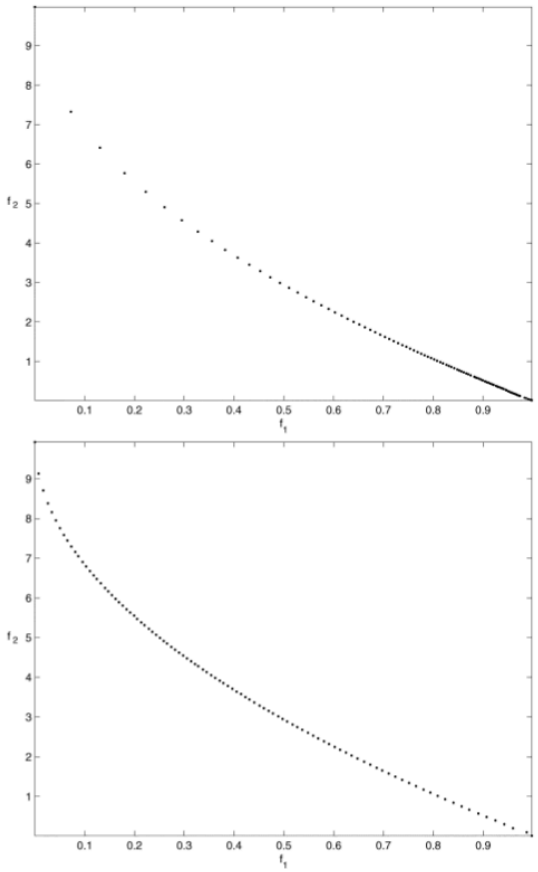


图 10 有无目标函数正则化的 MOEA/D 算法在 ZDT1 上的结果

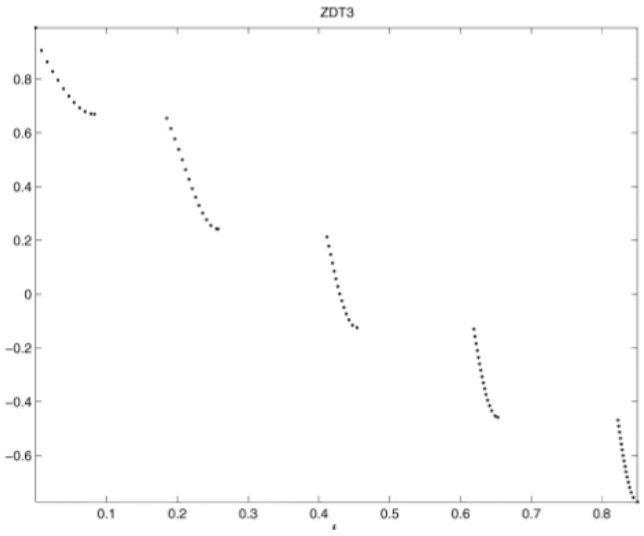


图 11 目标正则化后 MOEA/D 在 ZDT3 测试样例上找到的非支配解

6.MOEA/D 的可扩展性，敏感度，和种群小的情况

A.MOEA/D 中参数 *T* 的敏感度

参数 *T* 是 MOEA/D 非常重要的一个参数。为了研究 *T* 对算法在解决离散或者连续 MOPs 的影响，对于在测试样例 250-2（第 4 部分）上的基于 WS 方法的 MOEA/D 中，我们测试多种 *T* 的设置。基于切比雪夫的 MOEA/D 中，我们在第 5 部分在 ZDT1 上测试了 *T* 的影响。所有的参数设置都和第四部分 E 小节和第 5 部分 E 小节一样，除了 *T* 的选取。图 12 很明显的可以看出在 *T* 很小的时候 MOEA/D 的算法性能在两个测试样例上效果都不好。我们在第三部分-B3 小节讲到过，小的 *T* 会导致搜索能力较差。MOEA/D 在背包问题 250-2 的差劲表现可以通过第三部分-B3 小节解释。

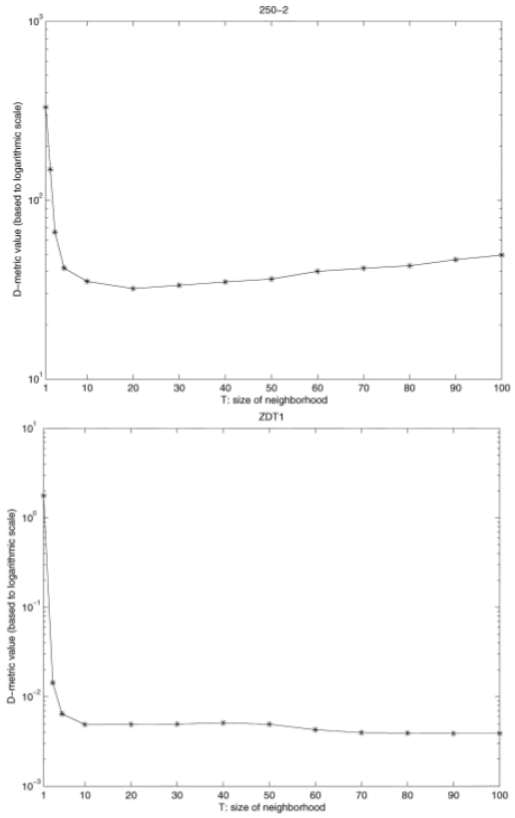


图 12 参数 *T* 对 MOEA/D 在 2-250 和 ZDT1 测试样例上的影响

B.小种群对 MOEA/D 的影响

在第一部分中提到，决策者并不需要花很多时间去找到特别多的帕累托最优解。决策者们感兴趣的是在最短的时间内找到一组小的但是均匀分布在 PF 上的解。下述内容中，我们将展示小种群的 MOEA/D 的能力。我们的样例选择的是 ZDT1，选择基于切比雪夫的 MOEA/D。参数设置和第 5 部分相同，除了将 *N* 改为 20 以外。作为对比，NSGA-II 的 *N* 也改为 20。两个算法执行 250 次后终止。

图 3 给出了得出的最终种群。可以看出 MOEA/D 找到了一组均匀分布的帕累托最优解，而 NSGA-II 没有达到 PF。MOEA/D 的这项优势来自于基于分解的策略。

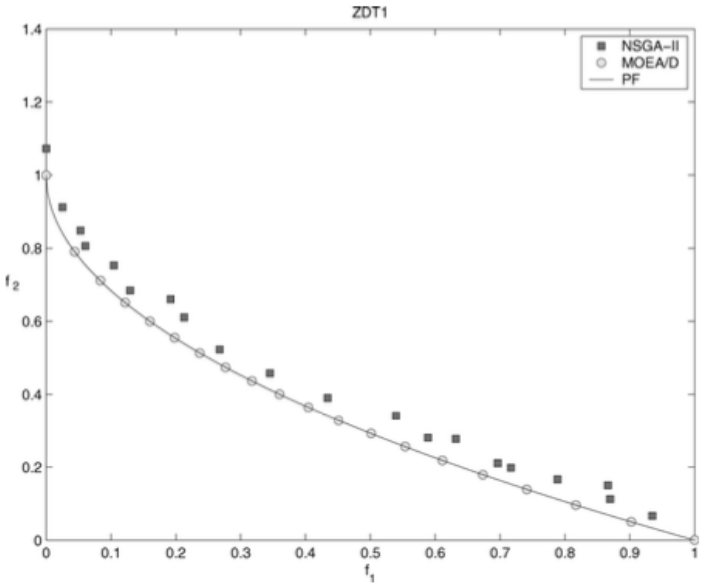


图 13: MOEA/D 和 NSGA-II 在小种群的算法表现情况（$N = 20$）

C.可扩展性

为了研究决策变量的增加对函数执行次数增加的影响，我们在 ZDT1 上进行了不同的测试（测试的算法同第 5 部分 C 小节的算法，参数一致，除了执行次数改为了 1000）。通过更改决策变量的数量，我们做了 30 次独立的实验，每次 MOEA/D 都将 D-指标降到了 0.005。图 14 给出了 ZDT1 上 D-指标低于 0.005 所需要的执行次数。可以看出决策变量的变化是和函数执行次数成线性相关的关系。线性的扩展性可以归于两个原因：1）无论有多少的决策变量，子问题最高是 100 个。2）计算子问题的复杂度也是和决策变量的升高线性相关的。

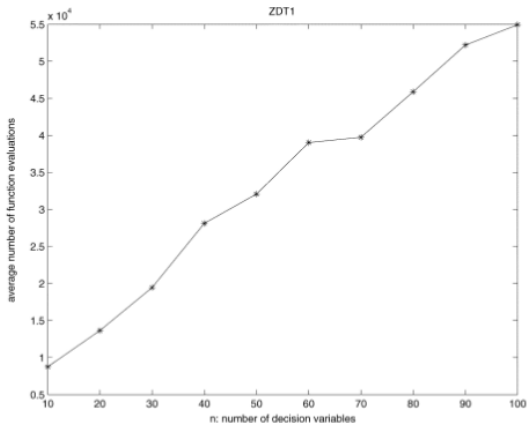


图 14 将 D-指标降到 0.005 所需要的目标函数此行次数和决策变量多少的关系

7.总结

基于分解的方式是数学规划中用以解决 MOPs 问题常用的传统方法。相反，在 MOEAs 算法的研究领域中，多数研究都将 MOP 当作一个不可分割的问题对待，大多都是通过支配的相互关系判断解的质量。这些算法可能在生成一组均匀分布的 PF 的解上的能力有所欠缺。

本篇论文提出了一种简单，实用的基于分解的多目标优化遗传算法框架，即 MOEA/D。它先用分解的方法来将 MOP 问题转变为一系列的单目标优化子问题。之后，再中 EA 算法同时优化这些子问题。种群中的每个解都针对一个子问题。子问题之间的相邻关系是通过它们的权重向量的欧几里得距离得到的。在 MOEA/D 中，对一个子问题的优化也用到了它相邻的子问题的信息，因为两个相邻子问题的解非常可能是相似的。我们比较了在背包问题或者连续多目标问题上 MOEA/D 和 MOGLS，NSGA-II 的性能差异。我们的实验结果是 MOEA/D 的算法复杂度是三者中最低的。基于简单的分解方式的 MOEA/D 的算法性能高过 MOGLS，NSGA-II 在大多测试样例上结果或者与 MOGLS，NSGA-II 在大多测试样例上结果相近。

我们也展示了基于 PBI 分解方式的 MOEA/D 是有能力在 3 目标函数的测试样例上生成一组均匀解的。我们展示了基于简单分解方式的 MOEA/D 加上目标函数的正则化是可以很好的解决目标函数范围相异的这个问题的。这些实验结果表明，改进 MOEA/D 算法可以得到更好的算法性能。

我们的实验证实了 MOEA/D 的算法可扩展性，以及 T 参数对算法的影响。我们发现其计算复杂度是和决策变量的多少成线性增长的关系。并且，MOEA/D 算法性能对 T 非常敏感。我们还得到了即使是只具有小种群的 MOEA/D，也能低时耗的生成一组均匀分配的帕累托最优解。

将 EA 算法和数学规划相整合在一起可以很好的解决单目标优化问题。我们的论文给出了一个非常自然的方法将分解算法（常用在数学规划领域）整合到 EAs 中解决多目标优化问题。其他的分解算法或者是单目标优化技巧可以被轻松的和 EAs 一起整合到 MOEA/D 中。

致谢

作者非常感谢 Yao 和其他本文的匿名审稿人。他们也很感谢 Zhou 和 Lucas 关于本文提出的建议。

参考文献

[1] K. Miettinen, *Nonlinear Multiobjective Optimization*. Norwell, MA: Kluwer, 1999.

[2] K. Deb, *Multi-Objective Optimization using Evolutionary Algo- rithms*. New York: Wiley, 2001.

[3] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Norwell, MA: Kluwer, 2002.

[4] K. Tan, E. Khor, and T. Lee, *Multiobjective Evolutionary Algorithms and Applications*, ser. Advanced Information and Knowledge Pro- cessing. Berlin, Germany: Springer-Verlag, 2005.

[5] E. Polak, "On the approximation of solutions to multiple criteria de- cision making problems," in *Multiple Criteria Decision Making*, M. Zeleny, Ed. Berlin, Germany: Springer-Verlag, 1976, vol. 123, Lec- ture Notes in Economics and Mathematical Systems, pp. 271–282.

[6] S. Helbig, "On a constructive approximation of the efficient outcomes in bicriterion vector optimization," *J. Global Optim.*, vol. 5, pp. 35–48, 1994.

[7] M. Wiecek, W. Chen, and J. Zhang, "Piecewise quadratic approxima- tion of the non-dominated set for bi-criteria programs," *J. Multi-Cri- teria Decision Analysis*, vol. 10, no. 1, pp. 35–47, 2001.

[8] S. Ruzika and M. Wiecek, "Approximation methods in multiobjective programming," *J. Optim. Theory Appl.*, vol. 126, no. 3, pp. 473–501, Sep. 2005.

[9] I. Das and J. E. Dennis, "Normal-bounday intersection: A new method for generating Pareto optimal points in multicriteria optimization prob- lems," *SIAM J. Optim.*, vol. 8, no. 3, pp. 631–657, Aug. 1998.

[10] A.Messac,A.Ismail-Yahaya,andC.Mattson,"Thenormalizednormal constraint method for generating the Pareto frontier," *Struct Multidisc. Optim.*, vol. 25, pp. 86–98, 2003.

[11] C. A. Mattson, A. A. Mullur, and A. Messac, "Smart Pareto filter: Ob- taining a minimal representation of multiobjective design space," *Eng. Optim.*, vol. 36, no. 6, pp. 721–740, 2004.

[12] C. A. C. Coello, "An updated survey of GA-based multiobjective opti- mization techniques," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 109–143, 2000.

[13] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, Nov. 1999.

[14] J. D. Knowles and D. W. Corne, “The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimisation,” in *Proc. Congr. Evol. Comput.*, Washington, D.C., Jul. 1999, pp. 98–105.

[15] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimiza- tion,” in *Proc. Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, K. C. Gian- nakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, Eds., Athens, Greece, pp. 95–100.

[16] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[17] H.LuandG.G.Yen,“Rank-density-basedmultiobjectivegeneticalgo- rithm and benchmark test function study,” *IEEE Trans. Evol. Comput.*, vol. 7, no. 4, pp. 325–343, Aug. 2003.

[18] T. Okabe, Y. Jin, B. Sendhoff, and M. Olhofer, “Voronoi-based esti- mation of distribution algorithm for multi-objective optimization,” in Congress on Evolutionary Computation (CEC), Y. Shi, Ed. Piscat- away, NJ: IEEE Press, 2004, pp. 1594–1602.

[19] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, “Handling mul- tiple objectives with particle swarm optimization,” IEEE Trans. Evol. Comput., vol. 8, no. 3, pp. 256–279, Jun. 2004.

[20] D. A. V. Veldhuizen and G. B. Lamont, “Multiobjective evolutionary algorithms: Analyzing the state-of-the-art,” Evol. Comput., vol. 8, no. 2, pp. 125–147, 2000.

[21] J. D. Knowles and D. Corne, “Properties of an adaptive archiving algo- rithm for storing nondominated vectors,” IEEE Trans. Evol. Comput., vol. 7, no. 2, pp. 100–116, Apr. 2003.

[22] Q. Zhang, A. Zhou, and Y. Jin, “Modelling the regularity in an estima- tion of distribution algorithm for continuous multiobjective optimisa- tion with variable linkages,” Dept. Comput. Sci., Univ. Essex, Colch- ester, U.K., Tech. Rep. CSM-459, 2006, (a revised version of this paper has been accepted for publication by the IEEE Trans. Evol. Comput., 2007).

[23] P. A. N. Bosman and D. Thierens, “The balance between proximity and diversity in multiobjective evolutionary algorithms,” IEEE Trans. Evol. Comput., vol. 7, no. 2, pp. 174–188, Apr. 2003.

[24] J. D. Schaffer, “Multiple objective optimization with vector evaluated genetic algorithms,” in Proc. 1st Int. Conf. Genetic Algorithms, 1985, pp. 93–100.

[25] L.PaqueteandT.Stützle,“Atwo-phaselocalsearchforthebiobjective traveling salesman problem,” in Proc. Evol. Multi-Criterion Optim., 2003, pp. 479–493.

[26] E. J. Hughes, “Multiple single objective Pareto sampling,” in Proc. Congr. Evol. Comput., Canberra, Australia, 2003, pp. 2678–2684.

[27] Y.Jin,T.Okabe,andB.Sendhoff,“Adaptingweightedaggregationfor multiobjective evolutionary strategies,” in Evolutionary Multicriterion Optimization. : Springer, 2001, vol. 1993, LNCS, pp. 96–110.

[28] H. Ishibuchi and T. Murata, “Multi-objective genetic local search al- gorithm and its application to flowshop scheduling,” IEEE Trans. Syst., Man, Cybern., vol. 28, pp. 392–403, Aug. 1998.

[29] A. Jaszkiewicz, “On the performance of multiple-objective genetic local search on the 0/1 knapsack problem – A comparative experi- ment,” IEEE Trans. Evol. Comput., vol. 6, no. 4, pp. 402–412, Aug. 2002.

[30] Q. Zhang and Y. W. Leung, “Orthogonal genetic algorithm for multi- media multicast routing,” IEEE Trans. Evol. Comput., vol. 3, no. 1, pp. 53–62, Apr.l 1999.

[31] N. J. W. T. J. Santers and W. I. Notz, The Design of Analysis of Com- puter Experiments. Berlin, Germany: Springer-Verlag, 2003.

[32] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fon- seca, “Performance assessment of multiobjective optimizers: An anal- ysis and review,” IEEE Trans. Evol. Comput., vol. 7, no. 2, pp. 117–132, Apr. 2003.

[33] E. Zitzler, K. Deb, and L. Thiele, “Comparison of multiobjective evo- lutionary algorithms: Empirical results,” Evol. Comput., vol. 8, no. 2, pp. 173–195, 2000.

[34] K.Deb,L.Thiele,M.Laumanns,andE.Zitzler,“Scalablemultiobjec- tive optimization test problems,” in Proc. Congr. Evol. Comput., May 2002, vol. 1, pp. 825–830.

[35] R. E. Steuer*, Multiple Criteria Optimization: Theory, Computation and Application*. New York: Wiley, 1986.

[36] M. Gen and R. Cheng*, Genetic Algorithms and Engineering Design*. New York: Wiley, 1997.

[37] P. J. Bentley and J. P. Wakefield, “Finding acceptable solutions in the Pareto-optimal range using multiobjective genetic algorithms,” in *Proc. 2nd On-Line World Conf. Soft Comput. Eng. Design Manuf.*, Jun. 1997, pp. 231–240.

[38] I. C. Parmee and D. Cvetkovic, “Preferences and their application in evolutionary multiobjective optimization,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 42–57, Feb. 2002.

[39] M. Ehrgott*, Multicriteria Optimization*, 2nd ed. Berlin, Germany: Springer-Verlag, 2005.

[40] T. Murata, H. Ishibuchi, and M. Gen*,* “Specification of Genetic Search Direction in Cellular Multiobjective Genetic Algorithm,” in *Evolutionary Multicriterion Optimization*. Berlin, Germany: Springer-Verlag, 2001, LNCS1993, pp. 82–95.