网络出版时间：2018-11-28 09:48:32
网络出版地址：http://kns.cnki.net/kcms/detail/11.1826.TP.20181127.1348.002.html

基于离散差分演化的 KPC 问题降维建模与求解*

贺毅朝 1) 王熙照 2) 张新禄 3) 李焕哲 1)

1)(河北地质大学 信息工程学院，石家庄 050031)
2)(深圳大学 计算机与软件学院，广东 深圳 518060)
3)(河北师范大学 数学与信息科学学院，石家庄 050024)

摘 要 具有单连续变量的背包问题(Knapsack Problem with a single Continuous variable, KPC)是标准 0-1 背包问题的一个新颖扩展形式，它既是一个 NP 完全问题，又是一个带有连续变量 *S* 的新颖组合优化问题，求解难度非常大. 为了快速高效地求解 KPC 问题，本文提出了利用演化算法求解 KPC 的新思路，并给出了基于离散差分演化算法求解 KPC 的两个有效方法. 首先，介绍了基本差分演化算法和具有混合编码的二进制差分演化算法(HBDE)的原理，给出了 HBDE 的算法伪代码描述，并分析了 KPC 的基本数学模型 KPCM1 的计算复杂度. 然后，在基于降维法消除 KPCM1 中连续变量 *S* 的基础上，建立了 KPC 的一个新离散数学模型 KPCM2；随后在基于贪心策略提出处理不可行解的有效算法基础上，基于单种群 HBDE 给出了求解 KPC 的第一个离散演化算法 S-HBDE. 第三，通过把连续变量 *S* 的取值范围划分为两个子区间将 KPC 分解为两个子问题，并基于降维法建立了 KPC 的适于并行求解的第二个数学模型 KPCM3；在利用贪心策略给出处理子问题不可行解的两个有效算法基础上，基于双种群 HBDE 提出了求解 KPC 的第二个离散演化算法 B-HBDE. 最后，在给出四类大规模 KPC 实例的基础上，利用 S-HBDE 和 B-HBDE 分别求解这些实例，并与近似算法 AP-KPC、遗传算法和离散粒子群优化算法的计算结果、耗费时间和稳定性等指标进行比较，比较结果表明 S-HBDE 和 B-HBDE 不仅在求解精度和稳定性方面均优于其它三个算法，而且求解速度很快，非常适于在实际应用中快速高效地求解大规模 KPC 实例.

关键词: 具有单连续变量背包问题；离散差分演化；遗传算法；粒子群优化；降维法；修复与优化法
中图法分类号 TP301

Modeling and Solving by Dimensionality Reduction of KPC Problem Based on Discrete Differential Evolution*

HE Yi-Chao[1)] WANG Xi-Zhao[2)] ZHANG Xin-Lu[3)] LI Huan-Zhe[1)]

1)(College of Information and Engineering, Shijiazhuang Hebei GEO University, Shijiazhuang 050031)
2)(College of Computer Science and Software Engineering, Shenzhen University, Shenzhen Guangdong518060)
3)(College of Mathematics and Information Science, Hebei Normal University, Shijiazhuang 050024)

本课题得到国家自然科学基金(No.71371063，No.11471097)、河北省高等学校科学研究计划项目(No.ZD2016005)和河北省自然科学基金项目(No.F2016403055)资助. 贺毅朝，男，1969 年生，硕士，教授，计算机学会(CCF)会员(15468S)，主要研究领域为演化算法及其应用、计算复杂性和群测试理论，E-mail:heyichao119@163.com，Tel:13623115445. 王熙照，男，1963 年生，博士，教授，IEEE Fellow，主要研究领域为机器学习、进化计算和大数据分析，E-mail:xizhaowang@ieeee.org. 张新禄，男，1968 年生，硕士，副教授，主要研究领域为智能计算和群测试理论，E-mail:xinluzhang@163.com.李焕哲，男，1975 年生，博士，副教授，主要研究领域为演化算法与机器学习，E-mail:lihuanzhe@hgu.edu.cn.

Abstract Knapsack problem with a single continuous variable (KPC) is a new extension of the standard 0-1 knapsack problem. It is not only an NP-complete problem in computer sciences, but also a novel combinatorial optimization problem with continuous variable *S* in practical applications. Because of the range of continuous variable *S* is a closed interval of real numbers, KPC is more difficult to solve than the standard 0-1 knapsack problem. In order to solve KPC problem quickly and efficiently, this paper presents a new idea of using evolutionary algorithm to solve KPC problem, and proposes two effective methods for solving KPC problem based on discrete differential evolutionary algorithm. In the paper, the general principle of standard differential evolution algorithm is firstly introduced. The discretization method in the binary differential evolution with hybrid encoding (HBDE) is introduced based on an encoding transforming function, and the pseudo-code of HBDE is described in more detail. Moreover, the computational complexity of the original mathematical model KPCM1 of KPC problem is analyzed by using a scaling technique. Secondly, On the basis of eliminating the continuous variable *S* in the mathematical model KPCM1 by dimension reduction method, a new discrete mathematical model KPCM2 of KPC problem is established, which is suitable to solve by using binary evolutionary algorithms. Moreover, an effective algorithm M2-GROA for handling the infeasible solutions of KPC problem is given, which used a special greedy repair and optimization strategy. The first discrete evolutionary algorithm for solving KPC problem, named S-HBDE, is proposed based on the single-population HBDE and M2-GROA. The pseudo-code of S-HBDE is described in more detail, and the algorithm time complexity of S-HBDE is analyzed. Thirdly, by dividing the range of the continuous variable *S* into two closed intervals of real numbers, the KPC is decomposed into two sub-problems established in two different intervals respectively. And the second discrete mathematical model KPCM3 of KPC is established by using the dimension reduction method, which is consist of two sub-models KPCM3.1 and KPCM3.2 and is suitable for parallel solving by binary evolutionary algorithms. At the same time, by using the greedy repair and optimization strategy, two effective algorithms M3.1-GROA and M3.2-GROA for dealing with the infeasible solutions of KPCM3.1 and KPCM3.2 is proposed, respectively. Combining with KPCM3.1 and KPCM3.2, the second discrete evolutionary algorithm B-HBDE for solving KPC problem is proposed based on the bi-population HBDE. The pseudo-code of B-HBDE is described in more detail, and the algorithm time complexity of B-HBDE is analyzed. Finally, the four kinds of large-scale KPC instances are first generated by using the existing generation method. For validating the performance of S-HBDE and B-HBDE, they are used to solve the four kinds of large-scale KPC instances respectively, and their average calculation results, average time-consuming and stability compare with that of approximate algorithm AP-KPC, genetic algorithm and discrete particle swarm optimization algorithm. The comparison results show that S-HBDE and B-HBDE are superior to the other three algorithms in solving accuracy and stability, and have very fast computing speed. So it is very suitable for solving large-scale KPC instances quickly and efficiently in practical applications.

Key words knapsack problem with a single continuous variable; discrete differential evolution; genetic algorithm; particle swarm optimization; dimension reduction method; repair and optimization method

1 引言

具有单连续变量的背包问题(knapsack problem with a single continuous variable, KPC)[1]是标准 0-1 背包问题(0-1 knapsack problem,0-1KP)的一个自然扩展形式, 它是由 Marchand 和 Wolsey 于 1999 年提出的一个带有连续变量的新颖组合优化问题. KPC 的一般描述为：给定 n 个物品的集合 $N=\{1,2,\ldots,n\}$ 和一个基本载重为 C 的背包，其中物品 $j\in N$ 具有价值 p_j 和重量 w_j，背包的可变载重 S 是在区间$[l,u]$上连续变化的一个变量，p_j，w_j 和 C 为正有理数，S, l 和 u 为有理数, 且 $l<0<u$. 给定一个正常数 c 作为惩罚系数，确定在 S 取何值以及此时如何选择物品装入背包, 使得装入物品的重量之和在不超过背包载重 $C+S$ 的前提下价值之和减去 cS 最大.

KPC 的基本数学模型[1,2](记为 **KPCM1**)为：

$$\max \quad f(X,S)=\sum_{j=1}^{n} x_j p_j - cS \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_j w_j \le C+S \tag{2}$$

$$x_j\in\{0,1\},\quad j=1,2,\ldots,n,\quad S\in[l,u] \tag{3}$$

其中，$X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$，$x_j=1(j=1,2,\cdots,n)$当且仅当物品 j 被装入了背包. 在 KPC 中, 背包载重不再固定不变，而是由变量 S 进行连续调整, 当 $S>0$ 时背包载重增加，当 $S<0$ 时背包载重减少; 同时，目标函数也不只是装入背包物品的价值之和，而是要加上了一个关于 S 的增量(-cS)。由于 0-1KP 是 KPC 在 $S=0$ 时的特例，因此 KPC 也是一个 NP 完全问题，不存在多项式时间精确算法.

Lin[2]等人利用变量代换将 KPC 转化为一个伪背包问题(pseudo-knapsack problem,PKP)和标准 0-1KP 的组合形式，并分别调用 COMBO[3]和 EXPKNAP[4]进行求解，提出了求解 KPC 的一个精确算法. 由于要对 PKP 进行可行性检查且涉及到动态规划法，故而该算法的时间复杂度较高. Buther 和 Briskorn[5]将 KPC 的物品集划分为三个子集，并根据启发式策略以及对变量的上下界变形，提出了将 KPC 转化为标准 0-1 KP 形式求解的方法；该方法只能求出 KPC 的一个近似结果，而且时间复杂度较高. Zhao 和 Li[6]将单连续变量 S 的取值区间划分为两部分，从而将 KPC 分解为两个具有标准 0-1KP 形式的子问题，进而提出了一个时间复杂度为 $O(n^2)$的 2-近似算法(记为 AP-KPC). 然而，AP-KPC 的近似比较大,计算结果不能令人满意. 贺毅朝[7]等人利用放缩法首先对 KPC 进行等价变换，然后基于动态规划法提出了求解 KPC 的一个精确算法 DP-KPC. 与其它精确算法一样, DP-KPC 也存在时间复杂度较高的缺点。由于 KPC 是一个 NP 完全问题，其精确算法至少具有伪多项式时间复杂度，难以实现对大规模 KPC 实例的快速求解；已有的非精确算法(如 AP-KPC)虽然求解速度快，但是求得解的精度欠佳，不足以满足实际应用的要求. 由此, 设计 KPC 的求解速度快且求解精度高的非精确算法是一个值得探讨的问题.

演化算法(evolutionary algorithms,EAs)[8-16]是一类群体智能算法, 其本质是一种随机近似算法，具有不需要计算目标函数的导数与梯度、不要求目标函数具有连续性的优点, 并且具有内在隐含并行性和极强的全局寻优能力，已被成功用于求解 0-1KP、随机时变背包问题(RTVKP), 多维背包问题(MDKP)、二次背包问题(QKP)、集合联盟背包问题(SUKP)、折扣{0-1}背包问题(D{0-1}KP)、经济调度问题(EDP)、集合覆盖问题(SCP)、设施定位问题(FLP)、旅行商问题(TSP)、流水车间调度问题(FSSP)和可满足性问题(SAT)[17-30]等组合优化问题(combinatorial optimization problems, COPs). 正是由于 EAs 在 COPs 中的成功应用，激发了人们对它的研究兴趣. 一方面, 为了高效求解各种 COPs, 人们不断尝试应用各种理论构造新的优秀演化算法；另一方面，为了拓展已有 EAs 的应用范畴，人们不断探索利用已有 EAs 求解 COPs 的新方法. 本文利用差分演化(differential evolution, DE)[12,31]探索求解 KPC 的新方法, 在利用降维法建立了 KPC 的两个适于离散 EAs 求解的数学模型基础上,基于混合编码二进制差分演化算法(HBDE)[32]给出了快速求解 KPC 的两个高效离散演化算法.

本文其余内容组织如下：在第 2 节介绍了 DE 与 HBDE 的原理, 并给出了 HBDE 的伪代码描述. 在第 3 节中, 分析了 KPCM1 的解空间规模及其对求解算法的影响. 在第 4 节基于降维法建立了 KPC 的离散数学模型 KPCM2, 给出了处理不可行解的有效算法 M2-GROA, 将其与单种群 HBDE 相结合提出了求解 KPC 的第一个离散演化算法 S-HBDE. 在第 5 节中, 基于分解与降维法建立了 KPC 的离散数学模型 KPCM3, 给出了处理不可行解的有效算法 M3.1-GROA 和 M3.2-GROA, 并与双种群 HBDE 相结合提出了求解 KPC 的第二个离散演化算法 B-HBDE. 在第 6 节，首先给出了四类大规模 KPC 实例与算法的参数设置，然后利用 S-HBDE 和 B-HBDE 分别求解四类大规模 KPC 实例, 通过与近似算法 AP-KPC[6]、遗传算法(GA)[33]和离散粒子群优化算法(BPSO)[34]的计算结果比较指出: S-HBDE 和 B-HBDE 不仅求解速度快，而且比其它三个算法的计算精度更佳，非常适于快速高效求解实际应用中的大规模 KPC 实例. 最后, 总结全文并展望今后进一步的研究思路.

2 差分演化与离散差分演化

2.1 差分演化

差分演化(differential evolution, DE)[12,31]是一个具有极强全局搜索能力的演化算法，因其在第一届 *IEEE* 演化大赛中的超群表现引起了国内外学者的极大关注,已被广泛应用于求解众多领域中的数值

优化问题[31,35-38]. 为了介绍 HBDE 的需要,下面首先基于 DE/rand /1/bin 模式简述 DE 的算法原理.

不失一般性，设 $max\ f(Y)$, $Y=[y_1,y_2,\ldots,y_n]\in\Omega$, 是一个数值优化问题, $\Omega=\prod_{j=1}^{n}[L_j,U_j]$是其解空间, 其中 $L_j<U_j$ 且为实数.

在 DE 中，一次迭代进化是由变异操作、交叉操作和选择操作共同完成的，它们的实现方法如下:

1) 变异操作: 设 $Y_i=[y_{i1},y_{i2},\ldots,y_{in}]$是当前种群 **P** 中的第 i 个个体, $Y_1=[y_{11},y_{12},\ldots,y_{1n}]$, $Y_2=[y_{21},y_{22},\ldots,y_{2n}]$ 和 $Y_3=[y_{31},y_{32},\ldots,y_{3n}]$ 是 **P** 中不同于 Y_i 的任意三个不同个体, 则 DE 基于 DE/rand /1/bin 模式进行变异操作的公式为：

$$z_{ij}=y_{1j}+F*(y_{2j}-y_{3j}) \tag{4}$$

其中，$Z_i=[z_{i1},z_{i2},\ldots,z_{in}]$是变异操作产生的临时个体, $z_{ij}\in[L_j,U_j]$, $j=1,2,\cdots,n$; F 为缩放因子且 $F\in(0,1)$.

2) 交叉操作: 对于给定的概率值 CR，DE 利用(5)式对 Z_i 与 Y_i 进行交叉操作，产生对应于 Y_i 的中间个体仍记为 $Z_i=[z_{i1},z_{i2},\ldots,z_{in}]$.

$$z_{ij}=\begin{cases}z_{ij}, & \text{如果}\quad r<CR\quad\text{或}\quad j=R(i);\\ y_{ij}, & \text{否则.}\end{cases} \tag{5}$$

其中，$CR\in(0,1)$称为交叉因子(或交叉概率)，$r\sim(0,1)$是一个随机数, $R(i)$是$[1, n]$上的一个随机正整数.

3) 选择操作: 当**P**中每一个体 Y_i 均产生中间个体 Z_i 之后, DE 利用(6)式从 Y_i 和 Z_i 中选择适应度最大的作为新一代种群中的第 i 个个体(仍记为 Y_i).

$$Y_i=\begin{cases}Z_i, & \text{如果}\quad f(Z_i)>f(Y_i);\\ Y_i, & \text{否则.}\end{cases} \tag{6}$$

DE 通过反复执行 1) ~3)实现进化,直到满足终止条件为止. 最后,输出当前种群中适应度最大个体对应的可行解, 即为 DE 求得 $max\ f(Y)$的最好结果.

2.2 离散差分演化

在 DE 中, 由于变异操作是实数运算, 不适合直接用于求解 COPs. 为了能够利用 DE 求解 COPs(例如 0-1KP, RTVKP 和 D{0-1}KP 等)，文献[32]基于编码转换机制提出了一个离散差分演化算法:具有混合编码的二进制差分演化算法 HBDE. HBDE 在保持 DE 原有进化模式和寻优特性的基础上，利用满射将实数编码转换为 0-1 编码, 非常适于求解以 0-1 向量为可行解的 COPs[27,32,39,40].

下面对于COPs问题**Π**: $max\ f(X)$, $X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$, 基于 DE/rand/1/bin 进化模式给出 HBDE 的算法原理[32,39]和伪代码描述.

记 $Y_i=[y_{i1},y_{i2},\ldots,y_{in}]$为 HBDE 的当前种群 **P** 中第 i 个个体, 并限定 $y_{ij}\in[-A,A]$, $j=1,2,\cdots,n$, n 是问题 **Π** 的维数, A 是一个给定的正实数, $i=1,2,\ldots, N$, N 为种群规模. 记 $X_i=[x_{i1},x_{i2},\ldots,x_{in}]\in\{0,1\}^n$ 为个体 Y_i 按照(7)式求得问题 **Π** 的可行解(或潜在解).

$$x_{ij}=\begin{cases}1, & \text{如果}\quad y_{ij}\ge 0;\\ 0, & \text{否则.}\end{cases} \tag{7}$$

在 HBDE 的一次迭代进化中, 对于 **P** 中每一个体 Y_i ($1\le i\le N$)，首先利用(4)式进行变异操作，利用 (5) 式进行交叉操作，得到中间个体 $Z_i=[z_{i1},z_{i2},\ldots,z_{in}]$, $z_{ij}\in[-A,A]$, $j=1,2,\cdots,n$. 然后, 利用(8)式求得 Z_i 对应问题 **Π** 的可行解 $U_i=[u_{i1},u_{i2},\ldots,u_{in}]\in\{0,1\}^n$，并利用(9)式选择 Y_i 和 Z_i 中适应度最佳的替换个体 Y_i. 重复这一过程，直到 **P** 中所有个体均完成以上操作为止.

$$u_{ij}=\begin{cases}1, & \text{如果}\quad z_{ij}\ge 0;\\ 0, & \text{否则.}\end{cases} \tag{8}$$

$$Y_i=\begin{cases}Z_i, & \text{如果}\quad f(U_i)>f(X_i);\\ Y_i, & \text{否则.}\end{cases} \tag{9}$$

令 $X_b=[x_{b1},x_{b2},\ldots,x_{bn}]\in\{0,1\}^n$为种群**P**中个体对应的最好解，$MIT$ 为 HBDE 的迭代次数, 则 HBDE 的伪代码描述如下:

算法 1. HBDE[32,39]

输入: 问题 **Π** 的实例, 参数 A, N,CR, F 和 MIT.

输出: **Π** 的实例的近似解(或最优解)X_b 与 $f(X_b)$.

```
1  随机产生初始种群 P={Yi=[yi1,…,yin]|yij ∈ [−A, A], 1≤i≤N, 1≤j≤n};
2  利用(7)式计算 Yi(1≤i≤N)对应的可行解 Xi,并根据 f(Xi)确定 P 中最好解 Xb;
3  FOR  t←1 TO MIT  DO
4    FOR  i←1 TO N  DO
5      FOR  j←1 TO n  DO
6        IF (r<CR∨j=R(i))  THEN
                 zij←yp1,j+F*(yp2,j − yp3,j);
7        ELSE  zij←yij;
8        IF zij≥0 THEN uij←1 ELSE uij←0;
9      END FOR
10     IF f(Ui)>f(Xi) THEN Yi←Zi; Xi←Ui;
11   END FOR
12   根据 f(Xi) (1≤i≤N)确定 P 中最好解 Xb;
13 END FOR
```

14 RETURN(X_b, $f(X_b)$).

在 HBDE 中, Y_{p1},Y_{p2} 和 Y_{p3} 是不同于 Y_i 且互不相同的三个个体. 令 O(*f*)表示计算 $f(X_i)$的时间复杂度. 则 HBDE 的时间复杂度为 O(*N***n*)+*N**O(*f*)+*MIT***N**(O(*n*)+O(*f*)). 注意到 *N* 和 *MIT* 是 *n* 的常数倍，O(*f*)是关于 *n* 的多项式, 所以 HBDE 是一个具有多项式时间复杂度的随机近似算法.

3 KPCM1 解空间的规模

在模型 KPCM1 中，KPC 的可行解可视为有序对<*X*,*S*>，其中 $X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$ 是一个 *n* 维 0-1 向量，$S\in[l,u]$是一个有理数。因此, KPC 的解空间 **SAPCE**=$\{0,1\}^n\times[l,u]$是一个 *n*+1 维卡氏积[7,41].

虽然 KPCM1 中 $[l,u]$是有理数构成的一个无限可数集, 但是根据 KPC 的特性不难计算出 **SAPCE** 的基数, 从而确定 KPCM1 的解空间规模.

令 $K=min\{k|k\in\mathbf{Z}^+\wedge kw_j\in\mathbf{Z}\wedge kC\in\mathbf{Z}^+\}$, 其中 $\mathbf{Z}^+$是正整数集, 则有

$$\sum_{j=1}^{n}x_jw_j\le C+S\Leftrightarrow\sum_{j=1}^{n}x_j(Kw_j)\le KC+KS.$$

由于 x_j, Kw_j ($1\le j\le n$)和 *KC* 均为正整数, 从而必然有

$$\sum_{j=1}^{n}x_j(Kw_j)\le KC+KS\Leftrightarrow\sum_{j=1}^{n}x_j(Kw_j)\le KC+\lfloor KS\rfloor$$

其中$\lfloor a\rfloor$为底函数, 表示不超过 *a* 的正整数. 于是, KPCM1 中的(2)式等价于下面的(10)式:

$$s.t.\quad\sum_{j=1}^{n}x_j(Kw_j)\le KC+\lfloor KS\rfloor. \qquad (10)$$

注意到$\lfloor Kl\rfloor\le\lfloor KS\rfloor\le\lfloor Ku\rfloor$, 则$\lfloor KS\rfloor$的取值依次为$\lfloor Kl\rfloor,\lfloor Kl\rfloor+1,\lfloor Kl\rfloor+2,\ldots,\lfloor Ku\rfloor$,令$m=\lfloor Ku\rfloor-\lfloor Kl\rfloor+1$, 于是 **SAPCE** 实质上是一个基数为 $m2^n$ 的 *n*+1 维卡氏积$\{0,1\}^n\times\{\lfloor kl\rfloor,\lfloor Kl\rfloor+1,..,\lfloor Ku\rfloor\}$，即 KPCM1 的解空间规模为 $m2^n$，$m=\lfloor Ku\rfloor-\lfloor Kl\rfloor+1$.

如果利用 EAs 基于 KPCM1 模型求解 KPC，采用混合编码形式表示个体是一种容易想到的方法, 但是针对此编码方法如何高效实施 EAs 的进化操作却是一个有待探讨的问题. 此外, 在利用 EAs 求解 COPs 时，解空间越大 EAs 需要搜索的范围就越大, 越不利于问题的求解. 为此本文另辟蹊径，首先基于降维法建立 KPC 的适于 EAs 求解的离散数学模型，然后利用 HBDE 给出求解 KPC 的两个高效算法.

4 利用单种群 HBDE 求解 KPC

在本节中，首先基于降维法建立 KPC 的离散数学模型 KPCM2，然后利用贪心策略提出一个处理 KPC 不可行解的有效算法 M2-GROA, 最后将单种群 HBDE 与 M2-GROA 相结合给出求解 KPC 的第一个离散演化算法 S-HBDE.

不妨记$W(X)=\sum_{j=1}^{n}x_jw_j$,其中 $X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$ 是一个 *n* 维 0-1 向量.

(a) 当 $W(X)<C+l$ 时. 对任意 $S\in[l,u]$ 均有 $W(X)<C+S$，故 *X* 是 KPC 的一个可行解，并且满足

$$f(X,S)=\sum_{j=1}^{n}x_jp_j+c(-S)\le\sum_{j=1}^{n}x_jp_j+c(-l).$$

所以有 $\max\ f(X,S)=\sum_{j=1}^{n}x_jp_j+c(-l)$.

(b) 当 $C+l\le W(X)\le C+u$ 时. 不妨设 $S_0=W(X)-C$，则$-u\le-S_0\le-l$. 对任意 $S\in[S_0,u]$ 均有 $W(X)\le C+S$, 故 *X* 是 KPC 的一个可行解，并且满足

$$\begin{aligned}f(X,S)&=\sum_{j=1}^{n}x_jp_j+c(-S)\\&\le\sum_{j=1}^{n}x_jp_j+c(-S_0)\\&=\sum_{j=1}^{n}x_jp_j+c(C-W(X)),\end{aligned}$$

所以有 $\max\ f(X,S)=\sum_{j=1}^{n}x_jp_j+c\left(C-\sum_{j=1}^{n}x_jw_j\right)$.

注意到$c\left(C-\sum_{j=1}^{n}x_jw_j\right)=c(-S_0)\le c(-l)$，于是根据(a)和(b)的分析易知: 通过消去连续变量 *S* 使解空间的维数由 *n*+1 降低为 *n*, 可以如下建立 KPC 的一个类似于 0-1KP 的离散数学模型 **KPCM2**：

$$\max f_1(X)=\sum_{j=1}^{n}x_jp_j+\min\left\{c(-l),c\left(C-\sum_{j=1}^{n}x_jw_j\right)\right\} \qquad (11)$$

$$\text{s.t}\qquad\sum_{j=1}^{n}x_jw_j\le C+u \qquad (12)$$

$$x_j\in\{0,1\},\quad j=1,2,\ldots,n \qquad (13)$$

其中, $X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$, $x_j=1(j=1,2,..,n)$当且仅当物品 *j* 被装入了背包中.

显然, 模型 KPCM2 的解空间 **SAPCE**=$\{0,1\}^n$,

其基数 $2^n<m2^n$. 在利用 EAs 基于模型 KPCM2 求解 KPC 时，由于不涉及变量 S，大大降低了求解难度. 此外，当求得可行解 $X=[x_1,\ldots,x_n]\in\{0,1\}^n$ 以后，变量 S 可根据等式 $S=[\sum_{j=1}^{n}x_jp_j-f_1(X)]/c$ 确定.

在基于模型 KPCM2 求解 KPC 时,存在 n 维 0-1 向量 $X=[x_1,x_2\ldots,x_n]\in\{0,1\}^n$ 是 KPC 的不可行解的情况. 不可行解的存在既影响算法求解效率，还不能直接利用目标函数值对个体进行评价. 为此，借鉴求解 KP 问题时处理不可行解的成功经验[18,24,39,40]，基于“价值密度比大而且能够使目标函数值增大的物品优先装入”的贪心策略，给出一个将不可行解修复为可行解、同时还能够对可行解做进一步优化的算法 **M2-GROA**.

在 M2-GROA 中，当输入 $X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$ 是一个不可行解时,反复将背包中价值密度最小的物品从中取出，直到 X 成为一个可行解为止；当 X 是可行解时，则反复对还没有装入背包、且具有最大价值密度、并能够使目标函数值增大的物品进行尝试装入，如果能被装入背包则装入之，否则放弃它继续尝试其它还未装入的物品,直到所有未被装入的物品均被尝试为止.

将 n 个物品的下标按照价值密度 p_j/w_j 由大到小的次序依次存入数组 $H[1\ldots n]$ 中，即 H 满足 $p_{H[1]}/w_{H[1]}\geq p_{H[2]}/w_{H[2]}\geq\ldots\geq p_{H[n]}/w_{H[n]}$. 令 $Y=[y_1,y_2,\ldots,y_n]\in\{0,1\}^n$ 是一个临时向量，$X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$ 是 KPC 的一个潜在解，则 M2-GROA 的算法伪代码描述如下：

算法 2. M2-GROA

输入:潜在解 $X=[x_1,x_2,\ldots,x_n]$和 $H[1\ldots n]$;

输出:可行解 $X=[x_1,x_2,\ldots,x_n]$和 $f_1(X)$.

```
1  j←n;
2  WHILE W(X)>C+u  DO
3      IF x_H[j]=1 THEN x_H[j]←0;
4      j←j−1;
5  END WHILE
6  FOR  j←1 TO n  DO
7      IF (x_H[j]=0∧W(X)+w_H[j]≤C+u ) THEN
8          Y←X;  y_H[j]←1;
9          IF  f1(Y)> f1(X)  THEN  x_H[j]←1;
10     END IF
11 END FOR
12 RETURN(X, f1(X))
```

在算法 2 中，Step2-5 将一个不可行解修复为可行解, Step6-11 对可行解进行优化处理, Step12 输出优化后的可行解 X 与其函数值 $f_1(X)$. 易知，算法 2 的时间复杂度是 $O(n^2)$.

在求解 KPC 时，利用 M2-GROA 对 HBDE 中每一个体对应的潜在解进行修复与优化处理，利用所得可行解的目标函数值作为该个体的适应度对其进行评价，这样既可消除不可行解，又能对个体进行优劣评价. 由于基于模型 KPCM2 求解 KPC 时 HBDE 仅有一个种群，为了区别后面的算法，我们将在模型 KPCM2 下求解 KPC 的单种群 HBDE 记为 **S-HBDE**，其算法伪代码描述如下：

算法 3. S-HBDE

输入: KPC 实例 **Π**，参数 A, N, CR, F 和 MIT.

输出: **Π** 的近似解(或最优解)X_b 与 $f_1(X_b)$.

```
1  按照 p_j/w_j(1≤j≤n)由大到小的顺序依次将 n 个物品的下标存入数组 H[1...n]中;
2  随机产生初始种群 P={Y_i=[y_i1,y_i2,…,y_in]|y_ij ∈[−A,A] ,1≤i≤N, 1≤j≤n };
3  FOR  i←1 TO N  DO
4      利用(7)式计算 Y_i 对应的潜在解 X_i ;
5      (X_i, f1(X_i)) ←M2-GROA(X_i, H[1...n]);
6  END FOR
7  根据 f1(X_i) (1≤i≤N)确定 P 中最好解 X_b;
8  FOR t ←1 TO MIT  DO
9    FOR i←1 TO N  DO
10     FOR  j←1 TO n  DO
11       IF (r<CR∨j=R(i)) THEN
12         z_ij←y_p1,j+F*(y_p2,j − y_p3,j)
13       ELSE  z_ij←y_ij;
14       IFz_ij≥0 THEN u_ij←1ELSE u_ij←0;
15     END FOR
16     (U_i,f1(U_i)) ←M2-GROA(U_i,H[1...n]);
17     IF f1(U_i)>f1(X_i) THEN Y_i←Z_i, X_i←U_i ;
18   END FOR
19   根据 f1(X_i) (1≤i≤N)确定 P 中最好解 X_b;
20 END FOR
21 RETURN(X_b, f1(X_b)).
```

在 S-HBDE 中，Step1 由快速排序算法[42]实现，其时间复杂度为 $O(n\log n)$; Step2 的时间复杂度为 $O(n*N)$, Step3~Step6 的时间复杂度为 $O(n^2*N)$, Step7 的时间复杂度为 $O(N)$, Step8~Step20 的时间复杂度为 $O(MIT*n^2*N)$, 因此 S-HBDE 的时间复杂度为 $O(n\log n)+O(n*N)+O(n^2*N)+O(N)+O(MIT*n^2*N)$

= O(*MIT** n^2**N*).

5 利用双种群 HBDE 求解 KPC

如果将单连续变量 *S* 的取值区间[*l*, *u*]划分为两个子区间[*l*,0]和[0,*u*]，则 KPC 可分解为下面两个子问题. 记 *OPT*1 为子问题 1 的最优值, *OPT*2 为子问题 2 的最优值，则 KPC 的最优值为 *OPT*= *max*{*OPT*1, *OPT*2}.

KPC 子问题 1:

$$\max \quad f(X,S)=\sum_{j=1}^{n} x_j p_j - cS$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_j w_j \le C+S$$

$$x_j\in\{0,1\},\ j=1,2,\ldots,n,\ S\in[l,\ 0].$$

KPC 子问题 2:

$$\max \quad f(X,S)=\sum_{j=1}^{n} x_j p_j - cS$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_j w_j \le C+S$$

$$x_j\in\{0,1\},\ j=1,2,\ldots,n,\ S\in[0,\ u].$$

类似于第 4 节中的分析，基于降维法如下建立 KPC 子问题 1 的一个类似于 0-1KP 的离散数学模型 **KPCM3.1**，建立 KPC 子问题 2 的一个类似于 0-1KP 的离散数学模型 **KPCM3.2**:

KPCM3.1：

$$\max g_1(X)=\sum_{j=1}^{n} x_j p_j + \min\left\{c(-l), c\left(C-\sum_{j=1}^{n} w_j x_j\right)\right\} \quad (14)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_j w_j \le C \quad (15)$$

$$x_j\in\{0,1\},\ j=1,2,\ldots,n\ . \quad (16)$$

KPCM3.2：

$$\max g_2(X)=\sum_{j=1}^{n} x_j p_j + \min\left\{0, c\left(C-\sum_{j=1}^{n} w_j x_j\right)\right\} \quad (17)$$

$$\text{s.t} \quad \sum_{j=1}^{n} x_j w_j \le C+u \quad (18)$$

$$x_j\in\{0,1\},\ \ j=1,2,\ldots,n\ . \quad (19)$$

显然，KPC 的最优值为 *max*{*max* $g_1(X)$, *max* $g_2(X)$}，因此 KPCM3.1 与 KPCM3.2 构成了 KPC 的一个新的离散数学模型,记为 **KPCM3**.易知 KPCM3 的解空间 **SAPCE**=$\{0,1\}^n$, 其基数 $2^n<m2^n$, 而且也不涉及连续变量 *S*, 因此非常适用于 EAs 求解.

下面根据 KPCM3 由 KPCM3.1 与 KPCM3.2 两个子问题构成的事实, 基于双种群 HBDE 提出一个求解 KPC 的离散演化算法(简记 **B-HBDE**).在 B-HBDE 中,存在两个同等规模的种群 $\mathbf{P}_1$ 和 $\mathbf{P}_2$, 其中 $\mathbf{P}_1$ 用于对 KPCM3.1 的求解, $\mathbf{P}_2$ 用于对 KPCM3.2 的求解. 于是, B-HBDE 通过并行计算 KPCM3.1 和 KPCM3.2 实现对 KPC 的求解.

在利用 B-HBDE 求解 KPCM3.1 和 KPCM3.2 时，也会出现个体对应的潜在解是不可行解的情况. 为此, 基于前述贪心策略分别给出处理不可行解的两个有效算法 M3.1-GROA 和 M3.2-GROA, 其中 M3.1-GROA 用于处理 KPCM3.1 的不可行解, M3.2- GROA 用于处理 KPCM3.2 的不可行解.

设数组 $H[1...n]$已按照物品的价值密度 p_j/w_j 由大到小的顺序依次存放了 n 个物品的下标, 即 H 满足 $p_{H[1]}/w_{H[1]} \ge p_{H[2]}/w_{H[2]} \ge ... \ge p_{H[n]}/w_{H[n]}$.令 $Y=[y_1,y_2,\ldots,y_n]\in\{0,1\}^n$ 为一个辅助向量, $X=[x_1,x_2,\ldots,x_n]\in\{0,1\}^n$ 是 KPCM3.1 的一个潜在解，则算法 M3.1-GROA 的伪代码描述如下：

算法 4. M3.1-GROA

输入:潜在解 $X=[x_1,\ldots,x_n]$和 $H[1...n]$;

输出:优化后的可行解 $X=[x_1, x_2,\ldots,x_n]$和 $g_1(X)$.

```
1   j←n;
2   WHILE W(X)>C DO
3       IF x_H[j]=1 THEN x_H[j]←0;
4       j←j−1;
5   END WHILE
6   FOR j←1 TO n  DO
7       IF (x_H[j]=0 ∧ W(X) +w_H[j]≤C) THEN
8           Y←X;  y_H[j]←1;
9           IF g1(Y)> g1(X) THEN x_H[j]←1;
10      END IF
11  END FOR
12  RETURN(X, g1(X))
```

在 M3.1-GROA 中，函数 $g_1(X)$按照(14)式进行计算，Step2-Step5 将不可行解修复为可行解，Step6-Step11 对可行解进行优化处理, Step12 输出优化后的可行解 X 与其函数值 $g_1(X)$. 不难看出，M3.1-GROA 的时间复杂度是 O(n^2).

由于除了目标函数的计算方法不相同之外，**M3.2-GROA** 的算法流程和时间复杂度均与 M2-GROA 相同，即只需将 M2-GROA 中所有的 $f_1(Y)$与 $f_1(X)$分别替换为 $g_2(Y)$与 $g_2(X)$, 并且按照(17)式计算 $g_2(Y)$与 $g_2(X)$即可. 由此, 限于篇幅不再赘述 M3.2-GROA 的算法伪代码.

在利用 M3.1-GROA 和 M3.2-GROA 处理不可行解的基础上，算法 B-HBDE 的伪代码描述如下：

算法 5. B-HBDE

输入: KPC 实例 **Π**, 参数 *A, N,CR, F* 和 *MIT*.

输出: **Π** 的近似解(或最优解)与其目标函数值.

1 按照 p_j/w_j ($1\le j\le n$)由大到小的次序依次将 n 个物品的下标存入数组 $H[1...n]$中;

2 随机产生初始种群 $\mathbf{P}_1=\{Y_{1i}=[y_{1i1},y_{1i2},\ldots,y_{1in}]\mid y_{1ij}\in[-A,A],\ 1\le i\le N,\ 1\le j\le n$ 和 $\mathbf{P}_2=\{Y_{2i}=[y_{2i1},y_{2i2},\ldots,y_{2in}]\mid y_{2ij}\in[-A,A],\ 1\le i\le N,\ 1\le j\le n\}$;

3 FOR $i\leftarrow$1 TO N DO

4 利用(7)式分别计算 Y_{1i} 对应于 KPCM3.1 的潜在解 X_{1i} 和 Y_{2i} 对应于 KPCM3.2 的潜在解 X_{2i};

5 $(X_{1i},g_1(X_{1i}))\leftarrow$**M3.1-GROA**($X_{1i}$,$H[1...n]$);

6 $(X_{2i},g_2(X_{2i}))\leftarrow$**M3.2-GROA**($X_{2i}$,$H[1...n]$);

7 END FOR

8 根据 $g_1(X_{1i})$和 $g_2(X_{2i})$($1\le i\le N$)分别确定 $\mathbf{P}_1$ 和 $\mathbf{P}_2$ 中最好解 X_{1b} 和 X_{2b};

9 FOR $t\leftarrow$1 TO *MIT* DO

10 FOR $i\leftarrow$1 TO N DO

11 FOR $j\leftarrow$1 TO n DO

12 IF ($r_1<CR\vee j=R_1$(i)) THEN

13 $z_{1ij}\leftarrow y_{1,p1,j}+F*(y_{1,p2,j}-y_{1,p3,j})$;

14 ELSE $z_{1ij}\leftarrow y_{1ij}$;

15 IF$z_{1ij}\ge$0 THEN $u_{1ij}\leftarrow$1ELSE $u_{1ij}\leftarrow$0;

16 IF ($r_2<CR\ \vee j=R_2$(i)) THEN

17 $z_{2ij}\leftarrow y_{2,p1,j}+F*(y_{2,p2,j}-y_{2,p3,j})$;

18 ELSE $z_{2ij}\leftarrow y_{2ij}$;

19 IF$z_{2ij}\ge$0 THEN $u_{2ij}\leftarrow$1ELSE $u_{2ij}\leftarrow$0;

20 END FOR

21 $(U_{1i},g_1(U_{1i}))$ $\leftarrow$**M3.1-GROA**(U_{1i},$H[1...n]$);

22 IF $g_1(U_{1i})>g_1(X_{1i})$ THEN $Y_{1i}\leftarrow Z_{1i}$; $X_{1i}\leftarrow U_{1i}$;

23 $(U_{2i},g_2(U_{2i}))$ $\leftarrow$**M3.2-GROA**(U_{2i}, $H[1...n]$);

24 IF $g_2(U_{2i})>g_2(X_{2i})$ THEN $Y_{2i}\leftarrow Z_{2i}$; $X_{2i}\leftarrow U_{2i}$;

25 END FOR

26 根据 $g_1(X_{1i})$和 $g_2(X_{2i})$($1\le i\le N$)确定 $\mathbf{P}_1$ 和 $\mathbf{P}_2$ 中最好解 X_{1b} 和 X_{2b};

27 END FOR

28 IF $g_1(X_{1b})\ge g_2(X_{2b})$ THEN RETURN(X_{1b},$g_1(X_{1b})$);

29 EISE RETURN(X_{2b}, $g_2(X_{2b})$).

在 B-HBDE 中, r_1 和 r_2 为(0,1)中的随机数，R_1(i)和 R_2(i)表示[1,n]上的随机整数.易知: B-HBDE 的时间复杂度也为 O($MIT*n^2*N$). 注意: 当 B-HBDE 的两个种群的规模为$|\mathbf{P}_1|=|\mathbf{P}_2|=N$, S-HBDE 的种群规模为$|\mathbf{P}|=N$ 时, B-HBDE 一次进化过程耗费的时间是 S-HBDE 的 2 倍.

6 实例计算与比较

由于 GA 和 BPSO 在求解组合优化问题中的成功应用，已成为衡量其它 EAs 在求解组合优化问题时算法性能优劣的标准。为了验证 S-HBDE 和 B-HBDE 求解 KPC 的性能，下面分别利用 S-HBDE、B-HBDE、AP-KPC、GA 和 BPSO 对四类大规模 KPC 实例进行仿真计算，通过比较它们的计算效果说明 S-HBDE 和 B-HBDE 的优异性.

所有仿真计算均使用 Acer Aspire E1-570G 笔记本，硬件配置为 Intel(R) Core(TM)i5-3337u CPU-1.8 GHz，4GB DDR3 内存(3.82GB 可用)，操作系统为 Microsoft Windows 8. 所有算法均利用 C++编程实现，编译环境为 Visual C++6.0.

6.1 KPC实例生成方法

根据文献[2,4]中的方法，生成的四类大规模 KPC实例分别是：不相关KPC实例 (记为ukpc), 其编号为ukpc100-ukpc1000; 弱相关KPC实例(记为wkpc), 其编号为wkpc100-wkpc1000; 强相关KPC实例(记为skpc), 其编号为skpc100-skpc1000; 逆强相关KPC实例(记为ikpc), 其编号为ikpc100-ikpc1000.

在 ukpc 实例中，w_j 和 p_j 在区间[1.0, R]上随机均匀取值. 在 wkpc 实例中，w_j 在区间[1.0, R]上随机均匀取值, p_j 在区间[$w_j-R/10$, $w_j+R/10$]上随机均匀取值，并且 $p_j\ge 1.0$. 在 skpc 实例中, w_j 在区间[1.0, R]上随机均匀取值,并且 $p_j=w_j+R/10$. 在 ikpc 实例中, p_j 在区间[1.0, R]上随机均匀取值, 并且 $w_j=p_j+R/10$.

在所有 KPC 实例中, 取 R=100.1, C=0.55W, 其中 $W=\sum_{j=1}^{n}w_j$. l 在区间[$-W/12$, $-W/30$]上随机均匀取值, u 在区间[$W/30$,$W/12$]上随机均匀取值; c 在区间[$3E/10$, $23E/10$] 上随机均匀取值，其中

$E=\frac{1}{n}\sum_{j=1}^{n}p_j$.在网址 https://www.researchgate.net/project/KPC-problem-and-Its-algorithms 中给出了所有 KPC 实例的完整数据.

6.2 计算结果比较

在 S-HBDE、S-HBDE、GA 和 BPSO 中, 各算法的种群规模均为 N=20, 其中 B-HBDE 的迭代次数为 MIT=3*n，其它 3 个算法的迭代次数均为 MIT=6*n, n 为 KPC 实例中物品的个数. 此外，在 S-HBDE 和 B-HBDE 中，交叉因子 CR=0.3，缩放因子 F=0.5，[-A, A]=[-5.0,5.0]. 在 GA 中，采用单点交叉算子、基本变异算子和赌轮选择算子，交叉概率 P_c=0.8,变异概率为 P_m=0.003. 在 BPSO 中,惯性权重 W=1.5,加速常数 C_1=C_2=2.0,粒子速度向量中各维分量的取值范围均为[-3.0,3.0].

记 *OPT* 是利用文献[7]中方法求得 KPC 实例的最优值，*APP* 为 AP-KPC 求得实例的近似值；*Best* 为 S-HBDE、B-HBDE、GA 和 BPSO 独立计算实例 50 次所得计算结果中的最好值,*Mean* 和 *StD* 分别是 50 次计算结果的平均值和标准差.对于 AP-KPC，*Time* 是算法求解实例一次的耗费时间；对于 S-HBDE、B-HBDE、GA 和 BPSO,*Time* 是算法求解实例一次的平均耗费时间. 在表 1~表 4 中给出了各算法求解四类 KPC 实例的计算结果.

记 *AR* 表示各算法求解 KPC 实例的计算结果与最优值之间的绝对误差，对于算法 AP-KPC, 有 *AR*=|*OPT*-*APP*|；对于算法 S-HBDE、B-HBDE、GA 和 BPSO，则有 *AR*=|*OPT*- *Mean*|. 根据表 1~表 4 中的计算结果，在图 1~图 4 绘出了五个算法的 *AR* 拟合曲线，并利用它们比较五个算法的计算结果优劣.

由图1可以看出：GA仅求解实例ukpc400和ukpc600的计算结果很好，对于其它实例其计算结果较差； BPSO求解实例ukpc100~ukpc400和ukpc600的结果较好，但是求解其它实例的计算结果较差；AP-KPC的计算结果虽然比GA和BPSO的好，但是均不如S-HBDE和B-HBDE的计算结果更优，S-HBDE和B-HBDE求解所有实例的结果几乎达到最优值，比其它算法的均优.

从图2中不难看出：GA和BPSO求解所有实例的结果明显较差； AP-KPC仅对实例wkpc600~wkpc1000的求解结果较好；S-HBDE和B-HBDE对所有实例的计算结果均很好，几乎均能达到最优值，比其它算法的结果均优.

由图3可以看出：GA和AP-KPC求解所有实例的结果均较差；虽然BPSO对实例skpc100~skpc600的计算结果很好，但仍然比S-HBDE和B-HBDE的计算结果相差许多；S-HBDE和B-HBDE求解所有实例的结果几乎达到最优值，明显优于其它算法.

表 1 S-HBDE, B-HBDE, AP-KPC, DP-KPC, GA 和 BPSO 求解 ukpc 类实例的计算结果

Instances:		ukpc100	ukpc200	ukpc300	ukpc400	ukpc500	ukpc600	ukpc700	ukpc800	ukpc900	ukpc1000
DP-KPC	*OPT*	39052.39	80494.30	118625.60	95519.51	189079.18	160094.72	311350.87	325528.60	388174.95	403401.30
AP-KPC	*APP*	38944.59	80398.72	118534.83	95437.02	189048.00	159759.82	311115.29	325415.48	387695.25	403391.31
	Time	0.000	0.000	0.000	0.002	0.003	0.003	0.015	0.015	0.015	0.016
S-HBDE	*Best*	**39052.39**	**80494.30**	118597.77	**95519.51**	**189079.18**	**160094.72**	**311350.87**	325524.54	**388174.95**	**403401.30**
	Mean	39048.65	**80494.30**	118592.40	**95519.51**	189073.09	**160094.72**	311329.45	325498.77	388143.23	403389.06
	StD	4.31	0.00	12.40	0.00	18.17	0.00	52.96	59.89	59.39	54.19
	Time	0.073	0.264	0.587	1.080	1.623	2.391	3.104	4.071	5.139	6.224
B-HBDE	*Best*	**39052.39**	**80494.30**	**118625.60**	**95519.51**	**189079.18**	**160094.72**	**311350.87**	325521.44	388169.85	**403401.3**0
	Mean	39049.74	80493.75	118596.22	**95519.51**	189066.28	**160094.72**	311319.04	325491.30	388089.04	403383.51
	StD	3.68	1.87	7.82	0.00	44.27	0.00	42.07	51.22	82.43	52.23
	Time	0.070	0.270	0.556	1.121	1.567	2.390	3.069	3.956	5.047	6.003
GA	*Best*	39004.43	80125.30	118054.82	**95519.51**	186502.70	**160094.72**	308880.86	322244.98	385437.30	402954.94
	Mean	38734.40	79550.42	115869.08	95519.45	185810.83	**160094.72**	307694.74	321468.61	384991.64	402616.24
	StD	242.97	41.17	99.68	0.08	191.51	0.00	947.11	773.96	401.01	162.45
	Time	0.060	0.115	0.380	0.616	0.783	1.308	1.333	1.806	2.384	2.322

BPSO	*Best*	39051.19	**80494.30**	118588.30	**95519.51**	188766.04	**160094.72**	310610.57	323813.72	385724.35	402219.59
	Mean	39047.26	80481.80	118528.45	**95519.51**	188323.68	**160094.72**	309549.43	322751.17	385413.24	401330.45
	StD	3.60	8.73	38.957	5.850	187.41	0.00	359.66	478.07	153.91	382.32
	Time	0.164	0.638	1.481	3.334	4.120	7.033	7.676	11.601	13.709	15.351

表 2 S-HBDE, B-HBDE, AP-KPC, DP-KPC, GA 和 BPSO 求解 wkpc 类实例的计算结果

Instances:		wkpc100	wkpc200	wkpc300	wkpc400	wkpc500	wkpc600	wkpc700	wkpc800	wkpc900	wkpc1000
DP-KPC	*OPT*	27645.54	53590.29	81911.91	111347.92	135034.66	176171.08	203897.99	212996.08	271861.55	278301.95
AP-KPC	*APP*	27566.71	53548.69	81728.55	111243.46	134581.44	176115.28	203885.41	212824.10	271840.45	278060.4
	Time	0.000	0.000	0.000	0.000	0.001	0.004	0.006	0.005	0.011	0.009
S-HBDE	*Best*	**27645.54**	**53590.29**	**81911.91**	111345.06	135031.16	176170.20	**203897.99**	212994.72	271860.50	278268.02
	Mean	27642.59	53589.17	81907.73	111342.17	135006.63	176158.83	203896.46	212986.97	271853.52	278188.82
	StD	2.40	1.46	5.13	3.94	21.23	7.85	4.30	17.75	11.24	45.37
	Time	0.071	0.263	0.583	1.038	1.519	2.327	3.018	3.935	4.866	6.295
B-HBDE	*Best*	27643.68	53590.07	**81911.91**	111345.06	135034.24	176161.34	**203897.99**	212994.72	271860.30	278205.46
	Mean	27641.64	53588.25	81907.54	111339.42	135029.55	176131.97	203892.50	212983.45	271850.46	278086.08
	StD	10.02	2.69	5.44	7.10	9.82	18.44	10.88	8.45	9.77	65.71
	Time	0.068	0.249	0.551	0.990	1.460	2.178	2.843	3.681	4.728	6.451
GA	*Best*	27534.36	52180.49	80117.39	111140.74	135032.66	175241.26	203853.67	212933.18	271588.35	276003.67
	Mean	27530.07	51884.29	79640.16	110237.15	135013.13	174532.7	203797.14	212859.04	271519.31	275368.01
	StD	3.96	211.02	219.77	404.43	10.4	113.58	30.71	33.72	32.68	376.16
	Time	0.034	0.141	0.269	0.572	0.619	1.144	1.174	1.716	2.067	3.039
BPSO	*Best*	**27645.54**	53586.00	81882.56	111264.98	135032.66	175761.18	203772.00	212894.89	271515.15	276196.74
	Mean	27640.91	53560.04	81812.00	111184.93	135018.87	175590.61	203697.42	212800.82	271397.83	275704.15
	StD	2.57	14.26	32.21	38.08	8.52	88.24	37.11	49.57	66.17	235.42
	Time	0.185	0.836	1.518	2.7709	3.915	7.249	8.203	11.553	14.196	16.084

表 3 S-HBDE, B-HBDE, AP-KPC, DP-KPC, GA 和 BPSO 求解 skpc 类实例的计算结果

Instances:		skpc100	skpc200	skpc300	skpc400	skpc500	skpc600	skpc700	skpc800	skpc900	skpc1000
DP-KPC	*OPT*	31496.98	61521.86	95438.79	130815.65	171113.64	181381.21	214926.22	274760.24	282503.87	316111.04
AP-KPC	*APP*	**31496.98**	61505.45	95419.81	130496.2	170731.23	181325.99	214879.32	274482.44	282342.77	315828.80
	Time	0.000	0.000	0.000	0.001	0.015	0.015	0.010	0.016	0.016	0.015
S-HBDE	*Best*	**31496.98**	**61521.86**	**95438.79**	**130815.65**	**171113.64**	**181381.21**	**214926.22**	**274760.24**	**282503.87**	**316111.04**
	Mean	**31496.98**	**61521.86**	95438.78	130815.64	171113.63	181381.20	214926.21	274757.54	282503.85	316111.00
	StD	0.00	0.00	0.01	0.01	0.05	0.01	0.02	13.95	0.08	0.04
	Time	0.064	0.265	0.585	1.027	1.679	2.290	3.217	4.189	5.143	6.393
B-HBDE	*Best*	**31496.98**	**61521.86**	**95438.79**	**130815.65**	**171113.64**	**181381.21**	**214926.22**	**274760.24**	**282503.87**	**316111.04**
	Mean	**31496.98**	61521.54	95438.73	130815.63	171113.57	181381.20	214926.19	274732.10	282503.62	316110.87
	StD	0.00	1.08	0.08	0.04	0.10	0.01	0.04	41.11	0.30	0.28
	Time	0.068	0.257	0.562	0.985	1.587	2.201	3.014	4.063	5.003	6.475
GA	*Best*	**31496.98**	61179.89	**95438.79**	**130815.65**	171108.56	**181381.21**	214834.2	274125.64	**282503.87**	**316111.04**
	Mean	**31496.98**	60998.28	**95438.79**	130815.64	170541.07	**181381.21**	214133.6	273864.42	**282503.87**	316111.00
	StD	0.00	122.70	0.00	0.01	51.00	0. 00	94.81	108.32	0.00	0.02

	Time	0.028	0.133	0.211	0.370	0.718	0.947	1.503	2.223	1.899	2.590
BPSO	*Best*	**31496.98**	**61521.86**	**95438.79**	**130815.65**	171113.44	181381.19	214913.20	274658.74	282500.96	316085.32
	Mean	**31496.98**	61521.78	95438.76	130815.52	171078.98	181378.95	214839.08	274542.79	282352.35	315931.85
	StD	0.00	0.56	0.02	0.15	42.65	13.98	28.80	48.52	55.79	57.10
	Time	0.184	0.658	1.473	2.880	4.346	5.969	7.596	11.607	12.683	16.863

表 4　S-HBDE, B-HBDE, AP-KPC, DP-KPC, GA 和 BPSO 求解 ikpc 类实例的计算结果

Instances:		ikpc100	ikpc200	ikpc300	ikpc400	ikpc500	ikpc600	ikpc700	ikpc800	ikpc900	ikpc1000
DP-KPC	*OPT*	25652.04	55631.38	81573.48	110248.04	134053.13	149641.10	186713.21	222120.57	230704.90	265768.46
AP-KPC	*APP*	25424.78	55406.60	81346.58	109723.95	133990.51	149558.21	186439.91	221978.06	230473.05	265485.12
	Time	0.000	0.000	0.000	0.002	0.002	0.003	0.005	0.005	0.006	0.006
S-HBDE	*Best*	**25652.04**	**55631.38**	**81573.48**	**110248.04**	**134053.13**	**149641.10**	186704.60	**222120.57**	230702.65	**265768.46**
	Mean	**25652.04**	**55631.38**	81573.45	110239.81	134050.78	149639.15	186598.92	222100.77	230607.47	265689.17
	StD	0.00	0.00	0.13	21.72	14.18	9.58	50.92	44.43	35.84	40.69
	Time	0.064	0.232	0.510	0.886	1.408	2.024	3.181	3.609	4.630	5.537
B-HBDE	*Best*	**25652.04**	**55631.38**	**81573.48**	**110248.04**	**134053.13**	**149641.10**	186610.24	**222120.57**	230604.80	265668.36
	Mean	**25652.04**	55631.36	81573.47	110221.79	134044.21	149635.82	186437.21	222040.27	230542.56	265615.38
	StD	0.00	0.04	0.01	38.53	27.52	17.89	80.01	46.24	52.14	48.28
	Time	0.062	0.244	0.522	0.910	1.398	2.042	2.881	3.512	4.492	5.441
GA	*Best*	**25652.04**	**55631.38**	**81573.48**	110247.86	**134053.13**	149640.71	185763.41	222020.38	230404.50	265267.88
	Mean	25651.77	55631.31	**81573.48**	110218.99	**134053.13**	149542.94	185612.65	221886.54	230288.55	265126.68
	StD	0.13	0.03	0.00	30.36	0.00	18.62	128.69	51.34	46.13	70.89
	Time	0.0262	0.125	0.241	0.474	0.649	0.907	2.042	2.209	2.209	2.773
BPSO	*Best*	**25652.04**	**55631.38**	**81573.48**	**110248.04**	**134053.13**	**149641.10**	186072.56	**222120.57**	230349.45	265722.30
	Mean	25651.68	**55631.38**	**81573.48**	110247.56	**134053.13**	149620.83	185998.24	222105.46	230246.08	265631.70
	StD	0.19	0.00	0.00	0.34	0.00	31.45	101.34	29.36	69.71	45.71
	Time	0.162	0.789	1.597	2.825	3.957	6.581	7.696	11.536	12.725	16.688

从图4中不难看出：AP-KPC对所有实例的计算结果都非常差；虽然S-HBDE、B-HBDE与GA、BPSO一样求解实例ikpc100~ikpc600的计算结果很好，求解实例ikpc700~ikpc1000的结果不佳,但是S-HBDE和B-HBDE的计算结果明显比GA和BPSO的更好.

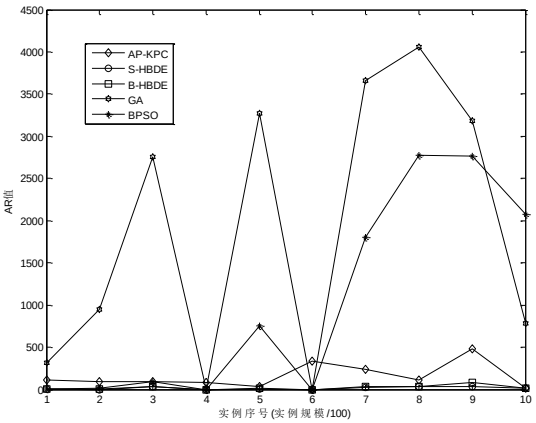

图 1 求解实例 ukpc100~ukpc1000 的 *AR* 拟合曲线

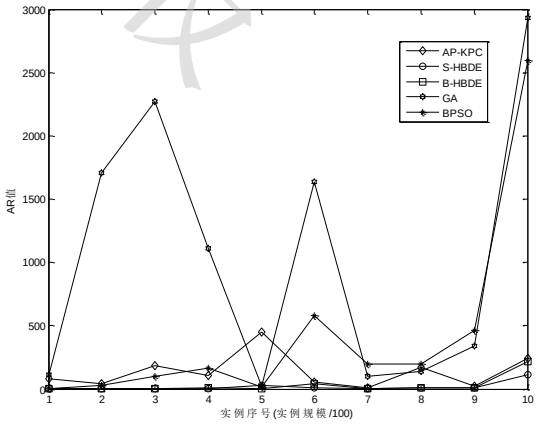

图 2 求解实例 wkpc100~wkpc1000 的 *AR* 拟合曲线

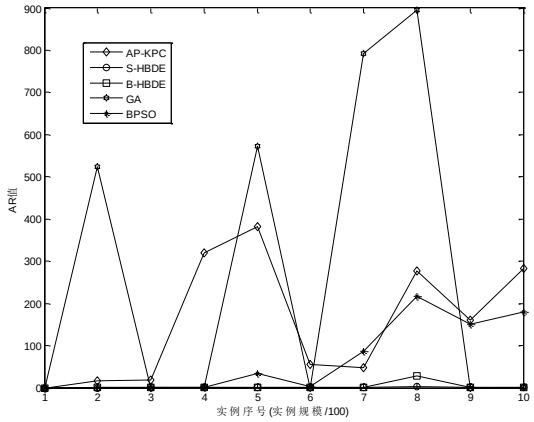

图 3 求解实例 skpc100~skpc1000 的 *AR* 拟合曲线

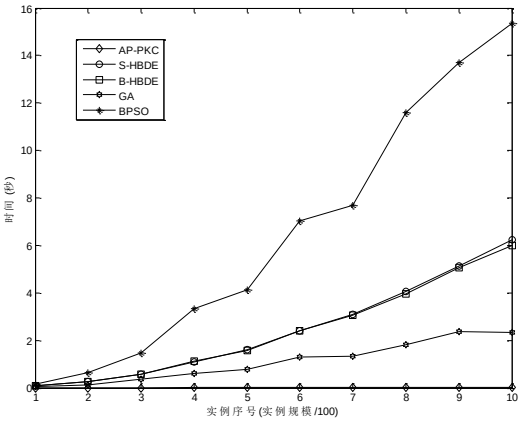

图 5 求解实例 ukpc100~ukpc1000 的时间变化曲线

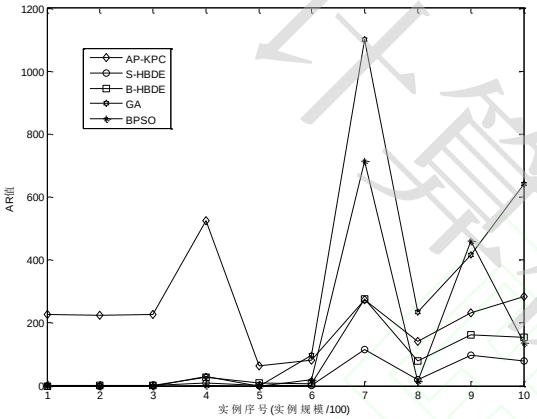

图 4 求解实例 ikpc100~ikpc1000 的 *AR* 拟合曲线

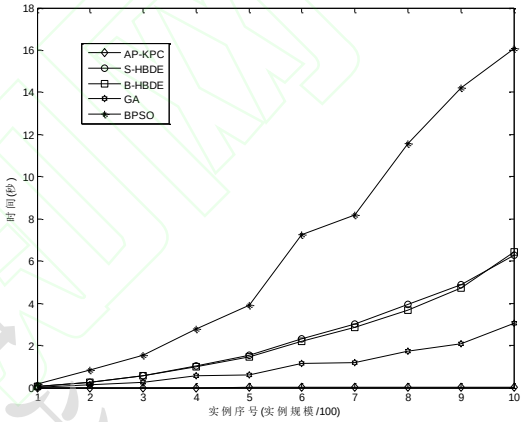

图 6 求解实例 wkpc100~wkpc1000 的时间变化曲线

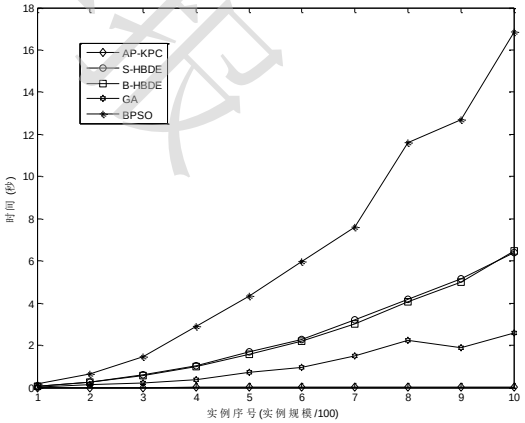

图 7 求解实例 skpc100~skpc1000 的时间变化曲线

通过以上比较可以看出: S-HBDE和B-HBDE求解四类KPC实例的计算结果不仅比AP-KPC、GA和BPSO的均优, 而且非常接近实例的最优值.

在图5~图8中给出了各算法求解实例一次的耗费时间(或平均耗费时间)变化曲线. 从中不难看出: 各算法求解实例的耗费时间变化曲线的趋势基本保持不变,并且根据它们的变化趋势可知 AP-KPC 的速度最快, 它求解所有实例的耗费时间几乎为 0; 其次是 GA、S-HBDE 和 B-HBDE,它们的时间变化曲线均可视为是线性的, 其中 GA 的速度最快, BPSO 的速度最慢,并且随着实例规模的增大,BPSO 耗费的时间呈现出急剧上升的态势.

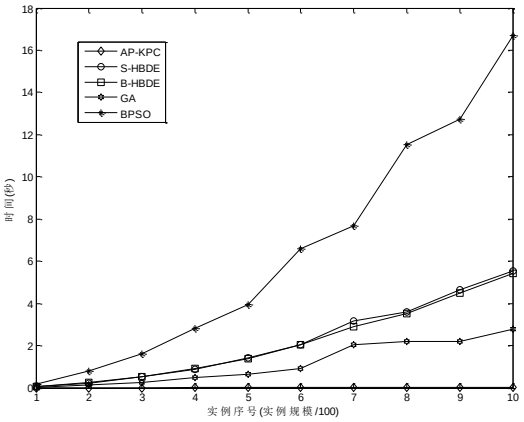

图 8 求解实例 ikpc100~ikpc1000 的时间变化曲线

在图 9~图 12 中是 S-HBDE、B-HBDE、GA 和 BPSO 的 *StD* 对应的直方图，从中不难看出，对于 ukpc、wkpc 和 skpc 类实例，S-HBDE 和 B-HBDE 的算法稳定性不仅很好,而且比 GA 和 BPSO 的均优; 对于 ikpc 类实例，虽然 S-HBDE 和 B-HBDE 的稳定性略差，但仍然比 GA 和 BPSO 的优.

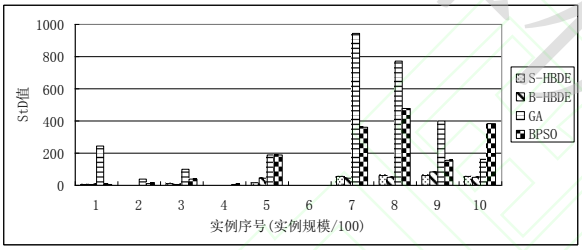

图 9 求解实例 ukpc100~ukpc1000 的 *StD* 直方图

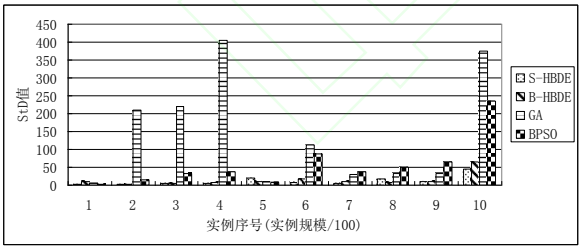

图 10 求解实例 wkpc100~wkpc1000 的 *StD* 直方图

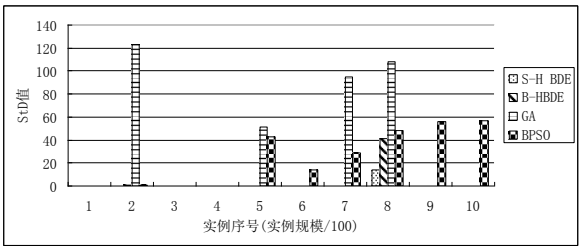

图 11 求解实例 skpc100~skpc1000 的 *StD* 直方图

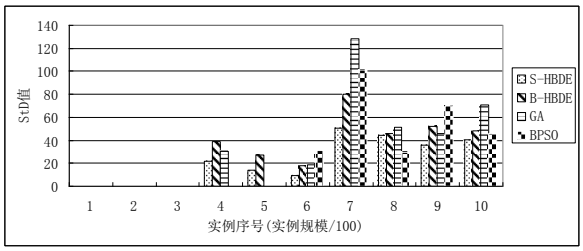

图 12 求解实例 ikpc100~ikpc1000 的 *StD* 直方图

综上所述，S-HBDE和B-HBDE的计算结果与算法稳定性均比AP-KPC、GA和BPSO的优，虽然它们的求解速度比不上AP-KPC和GA的快，但足以满足实际应用中的快速求解要求.因此，S-HBDE和B-HBDE的求解效果最佳，最适于快速高效求解实际应用中的大规模KPC实例.

7 结束语

本文提出了利用差分演化算法快速高效求解 KPC 的新方法. 首先，基于降维法建立了 KPC 的两个离散数学模型，给出了处理不可行解的有效算法；然后，基于 HBDE 提出了求解 KPC 的两个离散演化算法 S-HBDE 和 B-HBDE. 对于四类大规模的 KPC 实例，通过与 AP-KPC、GA 和 BPSO 的计算结果比较指出: S-HBDE 和 B-HBDE 不仅求解速度快，而且求解效果好，非常适于求解 KPC 问题.

按照 Pisinger[4]的观点, skpc 类实例应该是一类最难求解的实例，但是从 S-HBDE 和 B-HBDE 的求解结果来看，skpc 类实例的计算结果却是最佳. 由此我们猜测: Pisinger 所提出的难实例分类方法对于精确算法具有一定的适用性，但是对演化算法等非精确算法而言不一定适用.为进一步证实这一猜测，我们将继续基于演化算法求解其它 KP 问题，以期寻找到充分的证据.

根据 KPCM3.1 和 KPCM3.2 的相互独立性易知: 算法 B-HBDE 完全可以在并行环境下(如 MPI)对 KPCM3.1 和 KPCM3.2 并行编程实现，这样 B-HBDE 的计算速度可提高一倍，从而更具有实用性. 此外，探讨利用其它演化算法(例如 AFS[13]、ABC[15]和 GWO[16]等)求解 KPC 的性能优劣也是一个值得今后研究的问题.

致 谢 感谢审稿人提出的评审意见，这些评审意见对提高论文水平有很大的帮助. 感谢本刊编辑的

辛勤工作.

参 考 文 献

[1] Hugues Marchand, Laurence A. Wolsey. The 0-1 Knapsack problem with a single continuous variable. Math Program, 1999, 85 (1):15-33.

[2] Geng Lin, Wenxing Zhu, M.M.Ali. An exact algorithm for the 0-1 knapsack problem with a single continuous variable. Journal of global optimization. 2011, 50(4): 657-673.

[3] Martello S, Pisinger D, Toth P. Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science. 1999, 45(3): 414–424.

[4] Pisinger D. An expanding-core algorithm for the exact 0–1 knapsack Problem. European Journal of Operational Research. 1995, 87(1): 175-187.

[5] Buther M, Briskorn D. Reducing the 0–1 knapsack problem with a single continuous variable to the standard 0–1 knapsack problem. International Journal of Operations Research and Information Systems, 2012, 3(1): 1-12.

[6] Chenxia Zhao, Xianyue Li. Approximation algorithms on 0-1 linear knapsack problem with a single continuous variable. Journal of Combinatorial optimization. 2014, 28(4): 910-916.

[7] Yichao He, Xinlu Zhang, Wenlong Qu, Ning Li. Exact Algorithm for Solving Knapsack Problem with a Single Continuous Variable. Journal of Mathematics in Practice and Theory, 2018,48(13): 193-198.
(贺毅朝, 张新禄, 曲文龙, 李宁. 求解具有单连续变量背包问题的精确算法. 数学的实践与认识, 2018,48(13): 193-198.)

[8]Ashlock D. Evolutionary Computation for Modeling and Optimization, Evolutionary computation for modeling and optimization, Springer, 2006.

[9] Cotta C, van Hemert J. Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science 4446 (EvoCOP 2007), Valencia, Spain:Springer-Verlag, 2007.

[10] Mitchell M. An Introduction to Genetic Algorithms. Cambridge, UK: MIT Press, 1998.

[11] Kennedy, J, and Eberhart, R C. Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 1995, 1942–1948.

[12]Storn R, Price K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization. 1997, 11: 341-359.

[13]LI Xiao-Lei. A new intelligent optimization method--artificial fish swarm algorithm [Ph. D. theses]. Zhejiang University, 2003.
(李晓磊. 一种新的智能优化方法：人工鱼群算法[博士论文]. 浙江大学, 2003)

[14] Dorigo M, and Stützle T. Ant Colony Optimization. Cambridge,UK: MIT Press,2004.

[15] Dervis Karaboga, Barbaros Basturk. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. Journal of Global Optimization. 2007, 39(3): 459-471.

[16] Seyedali Mirjalili, Seyed Mohammad Mirjalili, Andrew Lewis. Grey Wolf Optimizer. Advances in Engineering Software. 2014, 69(1): 46-61.

[17]Dexuan Zou, Liqun Gao,Steven Li ,Jianhua Wu, Solving 0-1 knapsack problem by a novel global harmony search algorithm. Applied Soft Computing, 2011,11 :1556–1564

[18]Yichao He, Xinlu Zhang, Wenbin Li，et al. Algorithms for randomized time-varying knapsack problems. Journal of Combinatorial Optimization. 2016, 31(1): 95-117.

[19]Chu P, Beasley J. A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics , 1998 ,4 (1): 63-86.

[20]Azad M A K, Rocha A M A C, Fernandes E M G P. Improved binary artificial fish swarm algorithm for the 0–1 multidimensional knapsack problems. Swarm & Evolutionary Computation, 2014, 14: 66-75.

[21] Haddar B, Khemakhem M, Hanafi S, et al. A hybrid quantum particle swarm optimization for the Multidimensional Knapsack Problem. Engineering Applications of Artificial Intelligence, 2016, 55(C): 1-13.

[22]Julstrom, Bryant A. Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. Proceedings of the Conference on Genetic and Evolutionary Computation , Washington, USA, 2005: 607-614.

[23]Yichao He, Haoran Xie, Tak-Lam Wong, Xizhao Wang. A novel binary artificial bee colony algorithm for the set-union knapsack problem, Future Generation Computer Systems, 2018, 78(1): 77-86.

[24]Yi-chao He, Xi-zhao,Wang Yu-lin He, Shu-liang Zhao, Wen-bin Li. Exact and approximate algorithms for discounted {0-1} knapsack problem. Information Sciences. 2016, 369(11): 634–647.

[25] Jayabarathi T, Raghunathan T, Adarsh BR, Ponnuthurai Nagaratnam Suganthan. Economic dispatch using hybrid grey wolf optimizer. Energy, 2016, 111(9): 630-641.

[26] Yang Yu, Xin Yao, Zhi-Hua Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. Artificial Intelligence, 2012, 180–181 (2): 20–33.

[27]Mina Husseinzadeh Kashan,Ali Husseinzadeh Kashan,Nasim Nahavandi. A novel differential evolution algorithm for binary optimization. Computational Optimization and Applications, 2013, 55(2): 481–513

[28]Mavrovouniotis M, Yang S. A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. Soft Computing, 2011, 15(7): 1405-1425.

[29]Pan Quan-Ke, Wang Ling, Li Jun-qing. A novel discrete artificial bee

colony algorithm for the hybrid flowshop scheduling problem with makespan minimization. OMEGA-International Journal of Management Science, 2014, 45:42-56.

[30]Jens Gottlieb, Elena Marchiori, Claudio Rossi. Evolutionary Algorithms for the Satisifiability problem. Evolutionary Computation, 2002, 10(1): 35-50.

[31] Qin A K, Huang V L, Suganthan P N. Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. IEEE Transactions on Evolutionary Computation, 2009, 13(2):398-417.

[32] HE Yichao, WANG Xizhao, KOU Yingzhan. A binary differential evolution algorithm with hybrid encoding. Journal of Computer Research and Development, 2007, 44(9): 1476-1484.
(贺毅朝, 王熙照, 寇应展. 一种具有混合编码的二进制差分演化算法.计算机研究与发展, 2007,44(9):1476-1484)

[33] Chen GL, Wang XF, Zhuang ZD, Wang DS. Genetic algorithms and its applications. Beijing: POSTS&TELECOM Press, 2003.
(陈国良,王熙法,庄镇泉,王东生.遗传算法及其应用.北京:人民邮电出版社, 2003)

[34] Kennedy J, Eberhart R.C. A discrete binary version of the particle swarm optimization. Proceedings of the IEEE International Conference on Computational Cybernetics and Simulation. Orlando, USA, 1997, 5: 4104–4108.

[35]Das S, Suganthan P N. Differential Evolution: A Survey of the State-of-the-Art. IEEE Transactions on Evolutionary Computation, 2011, 15(1):4-31.

[36] Wang Y, Cai Z, Zhang Q. Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters. IEEE Transactions on Evolutionary Computation, 2011, 15(1):55-66.

[37]Swagatam Das, Sankha Subhra Mullick, P.N.Suganthan. Recent advances in differential evolution – An updated survey. Swarm and Evolutionary Computation, 2016, 27(2): 1-30.

[38]Qu BY, Suganthan PN, Liang JJ. Differential Evolution With Neighborhood Mutation for Multimodal Optimization. IEEE Transactions on Evolutionary Computation, 2012, 16(5):601-614.

[39]Hong Zhu, Yichao He, Eric Tsang, Xizhao Wang. Discrete differential evolution for the discounted {0-1} knapsack problem. International Journal of Bio-inspired Computation, 2017, 10(4): 219-237.

[40] Wang L, Wang S Y, Xu Y. An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem. Expert Systems with Applications, 2012, 39(5): 5593-5599.

[41]GENG Suyun, QU Wanling, WANG Hanpin. A course in discrete mathematics. Beijing: Peking University press, 2007.
(耿素云，屈婉玲，王捍贫. 离散数学教程.北京:北京大学出版社, 2007)

[42] Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. Cambridge: the MIT Press, England, 2001.

HE Yi-Chao, born in 1969, master, professor. His main research areas are the theory and applications of evolutionary algorithm and its applications, computational complexity theory and group testing theory.

WANG Xi-Zhao, born in 1963, Ph.D., professor, IEEE Fellow. His main research areas are machine learning, evolutionary algorithm and big data analysis.

ZHANG Xin-Lu, born in 1968, master, associate professor. His main research areas are intelligent computation and group testing theory.

LI Huan-Zhe, born in 1975, Ph. D., associate professor. His main research areas is evolutionary algorithm and machine learning.

Background

The knapsack problem with a single continuous variable, (KPC) is a new extension problem of the classical knapsack problem; besides, it is also a dynamic combinatorial optimization problem. KPC can be applied in commerce, investment decision, resource allocation, computational complexity theory, cryptography and applied mathematics etc, and it will become a hot problem in evolutionary computation. Since the exact solution of KPC is not really necessary in practice, an efficient approximation algorithms for finding approximate solution is of practical importance. So, it's more significant to design efficient evolutionary algorithm for solving the large scale and hard instances of KPC.

The authors of this paper have conducted the research in evolutionary algorithms and NP-hard problems direction from 2004, and have been supported by Natural Science Foundation of China (71371063，11471097), Natural Science Foundation of Hebei Province (F2016403055), Scientific Research Project Program of Colleges and Universities in Hebei Province (ZD2016005). The research team has done some creative work and published some papers on international or domestic

journals and conference proceedings.

The second author WANG Xi-Zhao is the leader in our research team. He takes charge of the project "Data Mining for Imbalanced Big Data in Electronic Health Records" (NSF CNS-71371063) at present. He concentrated on the research of evolutionary algorithms for combinational optimization problem and cooperatively published several papers on differential evolution and genetic algorithm etc.

Because KPC is a NPC problem, the complexity of exact algorithm based on dynamic programming is pseudo polynomial time. It is not practical for the large scale and hard instances of KPC with huge value coefficients and weight coefficients. In this paper, we establish two new mathematic models of KPC, which provide new theory foundational for designing algorithm. Furthermore, we propose two algorithms, S-HBDE and B-HBDE, based on differential evolution under different mathematic models. The extensive simulations show that the S-HBDE and B-HBDE can achieve excellent approximation solutions for large scale and hard instances of KPC .