基于动态规划法求解动态0-1背包问题

贺毅朝[1] 田海燕[2,3] 张新禄[2] 王志威[2] 高锁刚[2]
(石家庄经济学院信息工程学院 石家庄 050031)[1] (河北师范大学数学与信息科学学院 石家庄 050016)[2]
(计算数学与应用河北省重点实验室 石家庄 050016)[3]

摘 要 随机时变背包问题(RTVKP)是一种动态组合优化问题,也是一种典型的NP-hard问题。由于RTVKP问题中物品的价值、重量和背包载重均是动态变化的,导致问题的求解非常困难。在动态规划法基础上,提出了一种求解背包载重随机变化的RTVKP问题的确定性算法,分析了其复杂度和成功求解需要满足的条件。对两个大规模实例的计算表明,该算法是求解RTVKP问题的一种高效算法。

关键词 NP-难问题,0-1背包问题,动态优化,时变背包问题,动态规划法
中图法分类号 TP18 **文献标识码** A

Solving Dynamic 0-1 Knapsack Problems Based on Dynamic Programming Algorithm

HE Yi-chao[1] TIAN Hai-yan[2,3] ZHANG Xin-lu[2] WANG Zhi-wei[2] GAO Suo-gang[2]
(Information Engineering School, Shijiazhuang University of Economics, Shijiazhuang 050031, China)[1]
(College of Mathematics and Information Science, Hebei Normal University, Shijiazhuang 050016, China)[2]
(Hebei Key Laboratory of Computational Mathematics and Application, Shijiazhuang 050016, China)[3]

Abstract Random time-varying knapsack problem (RTVKP) is a dynamic combinatorial optimization problem, is a typical NP-hard problem too. Because the value and size of items and the size of knapsack can change along with the time, it causes that solving this problem is more difficult. We proposed an efficient algorithm for solving RTVKP with dynamic size of knapsack based on dynamic programming method, and analyzed the complexity of new algorithm and the condition of its successful executing. The results of simulation computation show that the exact algorithm is an efficient algorithm for solving RTVKP.

Keywords NP-hard problem, 0-1 knapsack problem, Dynamic optimization, Time-varying knapsack problems, Dynamic programming

时变背包问题(Time-varying Knapsack Problem, TVKP)[1,2]是一种动态0-1背包问题,也是一种动态组合最优化问题,在投资决策、资源分配、装箱问题以及下料问题等研究中具有重要的理论与应用价值。在TVKP问题中,物品的价值、重量和背包载重都是可以动态变化的,从而大大增加了问题的求解难度。由于0-1背包问题(0-1KP)是NP-hard问题,不存在多项式时间算法,而TVKP本质上是多个不同0-1KP的动态组合,因此也是NP-hard问题,不存在多项式时间算法。目前,有关TVKP的研究主要集中于背包载重在两个固定值之间振荡变化的情况,并且它是基于进化算法进行求解的[3-6]。随机时变背包问题(Random Time-varying Knapsack Problems, RTVKP)[3-6]是TVKP的更一般形式,其物品的价值、重量和背包载重均是随着时间随机动态变化的,这也是难度最大的TVKP问题。

本文基于动态规划法求解TVKP问题,提出了一种求解背包载重随机变化的RTVKP问题(以下简记RTVKP3)的有效算法。本文第1节介绍了RTVKP3的数学模型,分析并给出了基于动态规划法快速求解RTVKP3的有效算法;第2节分析了算法的复杂度与保证算法成功求解的条件,并通过对两个大规模RTVKP3问题实例的求解验证了算法的可行性与高效性;最后总结全文并展望下一步的工作。

1 求解RTVKP3的动态规划算法

背包载重随机变化的RTVKP问题一般描述为:有n个重量与价值分别为w_i和$s_i(1\leqslant i\leqslant n)$的物品和一个载重可以随机改变的背包,当背包的载重在区间$[A,B]$上以时间间隔Δt随机变化时,确定各时间间隔对应的最优装载方案及相应物品的价值之和。也就是对每个Δt时间段内的背包载重C,如何选择物品装入背包,使得在不超过当前载重C的前提下装入的物品总价值最大?

1.1 RTVKP3问题的数学模型

已知n个物品的价值集$S=\{s_1,s_2,\cdots,s_n\}$与重量集$W=$

到稿日期:2011-09-08 返修日期:2011-11-25 本文受国家自然科学基金(10971052),河北省高等学校科学技术研究青年基金(2010260)资助。

贺毅朝(1969—),男,硕士,教授,CCF会员,主要研究领域为算法理论与智能计算,E-mail:heyichao@sjzue.edu.cn;田海燕(1972—),女,硕士,讲师,主要研究领域为网络优化算法;张新禄(1968—),男,硕士,副教授,主要研究领域为网络优化与算法理论;王志威(1962—),男,硕士,副教授,主要研究领域为算法理论;高锁刚(1958—),男,博士,教授,博士生导师,主要研究领域为代数组合与近似算法。

$\{w_1, w_2, \cdots, w_n\}$，其中 $s_i, w_i \in Z, 1 \leqslant i \leqslant n$。RTVKP3 问题为：当背包载重在 $[A,B]$ 上以时间间隔 Δt 随机选取 m 个不同值 $C_1, C_2, \cdots, C_m$ 时（其中 $A \leqslant C_k \leqslant B, C_k \in Z, 1 \leqslant k \leqslant m$，$Z$ 为正整数集），分别求出对应 m 个 n 维二元向量 $X_k = (x_{k1}, x_{k2}, \cdots, x_{kn}) \in \{0,1\}^n, 1 \leqslant k \leqslant m$，使得满足：

$$\max f(X_k) = \max \sum_{i=1}^{n} s_i x_{ki}$$
$$\text{s.t.} \sum_{i=1}^{n} w_i x_{ki} \leqslant C_k \tag{1}$$

式中，$C_k \in \{C_1, C_2, \cdots, C_m\}, 1 \leqslant k \leqslant m$。在 $X_k = (x_{k1}, x_{k2}, \cdots, x_{kn})$ 中，当 $x_{ki}=1$ 时，表示在第 k 个时间间隔内第 i 个物品装入载重变为 C_k 的背包；当 $x_{ki}=0$ 时，表示第 i 个物品不装入。显然，0-1KP 为 $m=1$ 时 RTVKP3 的特殊情况；而且当背包载重在某时间间隔 Δt 中保持为 C_k 时，RTVKP3 恰对应一个重量集、价值集与背包载重分别为 $S=\{s_1, s_2, \cdots, s_n\}$，$W=\{w_1, w_2, \cdots, w_n\}$ 和 C_k 的 0-1KP。为了便于论述，在下文中将 RTVKP3 在时间间隔 Δt 中对应背包载重为 C_k 的 0-1KP 记为 0-1KP(n, C_k, S, W)。

由于 RTVKP3 本质上是由 m 个不同的 0-1 背包问题依次组合而成的，因此最直接的解法是分别利用动态规划法顺次求解这 m 个 0-1KP 问题。这种解法虽然可行，但并不是高效的，因为它忽略了这 m 个 0-1KP 问题具有相同价值集与重量集并且是固定不变的这一重要特性，这将导致大量的重复计算。为了充分利用 RTVKP3 的物品价值集与重量集不随背包载重改变而改变这一特性，下面通过讨论 RTVKP3 在不同时间间隔 Δt 所对应 0-1KP 之间的内在联系，基于动态规划法给出了一种求解 RTVKP3 的高效确定性算法。

1.2 求解 0-1KP(n,C,S,W)问题的动态规划算法

设 $V[0..n, 0..C]$ 是一个 $(n+1)\times(C+1)$ 的二维数组，其中 $V[i,j]$ $(1 \leqslant i \leqslant n$ 且 $1 \leqslant j \leqslant C)$ 表示从物品 $1,2,\cdots,i$ 中选择装入背包载重为 j 的 0-1KP$(i, j, S[1..i], W[1..i])$ 问题的最优解（最优方案）所对应的最优值，这里的 $S[1..i]$ 与 $W[1..i]$ 分别用于存放相应物品的价值与重量。令 $V[i,0]=V[0,j]=0 (0 \leqslant i \leqslant n$ 且 $0 \leqslant j \leqslant C)$，于是有递归方程[7]：

$$V[i,j] = \begin{cases} V[i-1,j], & \text{若 } i \geqslant 1 \text{ 且 } 1 \leqslant j \leqslant w_i \\ \max\{V[i-1,j], V[i-1, j-S[i]] + W[i]\}, & \text{若 } i \geqslant 1 \text{ 且 } j \geqslant w_i \end{cases} \tag{2}$$

显然，0-1KP(n, C, S, W) 问题的最优值为 $V[n, C]$。

以便描述求解 RTVKP3 问题的算法，下面先基于递归方程式(2)分别给出计算 $V[0..n, C_1+1..C_2]$ $(C_1 < C_2)$ 和求解 0-1KP$(n, C, S[1..n], W[1..n])$ 最优解的算法伪代码描述。设 $0 \leqslant C_1 < C_2$ 且 C_1, C_2 均为正整数，并设 $V[0..n, 0..C_1]$ 已知，则有：

算法 1 DPKPV(n,C_1,C_2,S[1..n],W[1..n])

```
1. for j= C1+1 to C2 do V[0,j]←0;
2.   for j= C1+1 to C2 do
3.     for i=1 to n do
4.       V[i,j]←V[i-1,j];
5.     if S[i]≤j then V[i,j]←max{V[i,j],V[i-1,j-S[i]]+W[i]};
6.   endfor
7. endfor
8. return V[1..n,1..C2]
```

注意到 $V[i,j]$ 的计算仅与 $V[i-1,j]$ 和 $V[i-1, j-S[i]]$ 有关，因此算法 1 在计算 $V[0..n, C_1+1..C_2]$ 时，可以按照列优先的顺序进行，这样在求解 RTVKP3 问题时便于理解与实现。接下来，利用一维数组 $X[1\cdots n]$ 存放问题的最优解，在算法 1 的基础上，求解 0-1KP$(n, C, S[1..n], W[1..n])$ 问题最优解的算法描述如下。

算法 2 DPKPS(n,C,S[1..n],V[1..n,1..C])

```
1. for i=1 to n do X[i]←0;
2. i←n and j←C;
3. while i>0 do
4.   if V[i,j]≠V[i-1,j] then X[i]←1 and j←j-S[i];
5.   i←i-1;
6. endwhile
7. return X[1…n]
```

注意到算法 1 的时间复杂性为 $O(nC_2)$，算法 2 的时间复杂度为 $O(n)$，利用算法 1 和算法 2 求解 0-1KP$(n, C, S[1..n], W[1..n])$ 的时间复杂度为 $O(nC+n)=O(nC)$，因此时间复杂度主要取决于算法 1。但是，由于 $\log C$ 为 C 的输入规模，因此算法 1 是一个伪多项式时间算法。尽管如此，在实际应用中利用上述算法求解 0-1 背包问题仍然是首选方法之一。

1.3 RTVKP3 问题的数学模型

对于 RTVKP3 问题，注意到其物品价值与重量并不随背包载重的变化而改变，因此在不同时间间隔 Δt 内 RTVKP3 对应于具有相同价值集与重量集的 0-1KP 问题，为此，下面基于动态规划法给出一种求解 RTVKP3 的高效算法。设背包初始载重为 C_0，令背包载重变化过程中当前所达到的最大值为 C_1。又设在当前时间间隔 Δt 中背包载重为 C_2，则 RTVKP3 问题在当前时间间隔 Δt 中对应于 0-1KP$(n, C_2, S[1..n], W[1..n])$ 问题。此时，对 RTVKP3 问题的求解必为以下两种情形之一。

情形(1) 当 $C_1 \geqslant C_2$ 时，由于 0-1KP$(n, C_2, W[1..n], S[1..n])$ 问题的最优值 $V[n, C_2]$ 在二维数组 $V[0..n, 0..C_1]$ 中已经求出，因此只要直接调用算法 DPKPS$(n, C_2, S[1..n], V[1..n, 1..C_2])$ 即可求得 0-1KP$(n, C_2, W[1..n], S[1..n])$ 的最优解，即 RTVKP3 问题在此时间间隔的最优解。

情形(2) 当 $C_1 < C_2$ 时，由于 0-1KP$(n, C_2, S[1..n], W[1..n])$ 问题的最优值 $V[n, C_2]$ 是未知的，因此应首先求之。注意到二维数组 $V[0..n, 0..C_1]$ 为 $V[0..n, 0..C_2]$ 的一部分，而数组 $V[0..n, 0..C_1]$ 已知，因此只需调用算法 DPKPV$(n, C_1, C_2, S[1..n], W[1..n])$ 来计算 $V[0..n, C_1+1..C_2]$ 部分即可，如此 $V[n, C_2]$ 也就求得。当数组 $V[0..n, 0..C_2]$ 确定后，调用算法 DPKPS$(n, C_2, S[1..n], V[1..n, 1..C_2])$ 即求得 0-1KP$(n, C_2, S[1..n], W[1..n])$ 问题的最优解，即 RTVKP3 在此时间间隔的最优解。

基于以上两种情形的分析，令 *Threshold* 表示时间间隔 Δt 的上界，C_1 为背包已达到的最大载重，C_2 为背包当前载重，m 为背包载重的振荡次数且振荡间隔为 Δt。令 $S[1..n]$ 与 $W[1..n]$ 分别用于存放相应物品的价值与重量，$X[1\cdots n]$ 用于存放 RTVKP3 在各时间间隔所对应 0-1KP 问题的解，背包载重的变化区间为 $[A,B]$ $(A<B$ 且均为正整数$)$。又设函数 *GetSystemTime*() 用于获取系统的当前时间，函数 *Random*(A,B) 用于随机产生 $[A,B]$ 上的整数。则求解 *RTVKP3* 的动态规划算法描述如下。

算法 3 DPforRTVKP (n,W[1..n],S[1..n],Threshold,A,B)

1. C_1←0; C_2←Random(A,B); Iter←1;
2. for i=0 to n do V[i,0]←0;
3. for j=0 to C_2do V[0,j]←0;
4. while Iter≤m do
5. T←GetSystemTime( );
6. if C_1<C_2 then
7. V[1..n,1..C_2]←DPKPV(n,C_1,C_2,S[1..n],W[1..n]);
8. X[1..n]←DPKPS(n,C_2,S[1..n],V[1..n,1..C_2]);
9. C_1←C_2;
10. else
11. X[1…n]←DPKPS(n,C_2,S[1..n],V[1..n,1..C_2]);
12. Δt←GetSystemTime( )−T;
13. if Δt<Threshold then output(X[1…n])
14. else return failed;
15. while Δt<Threshold do Δt←GetSystemTime( )−T;
16. C_2←Random(A,B); Iter←Iter+1;
17. endwhile
18. return success

显然,如果算法 3 返回 failed,则说明在某时间间隔的求解失败导致算法结束,即算法不能够求出此时间间隔内 RTVKP3 所对应的 0-1 背包问题的最优解,但之前的求解还是成功的。如果返回 success,则说明算法求解成功,并且所输出的 m 个解向量即为 RTVKP3 问题在各时间间隔中对应 0-1 背包问题的最优解。需要说明的是:由于算法 1 中 $V[0..n, C_1+1..C_2]$是按列计算的,因此当算法 3 在某时间间隔的计算失败时,也可以不用返回 failed 且不结束算法,而是输出此时间间隔已经求得的部分解并且更新 C_1,这样并不影响接下来对其后时间间隔的计算。此外,如果区间$[A,B]$预知,则可将算法 3 的 Step15 改为利用 Threshold−Δt 的时间继续计算 $V[0..n,C_1+1..B]$,从而起到加快算法求解速度的作用。

2 算法分析与实例计算

下面首先分析算法 3 的时间复杂度和保证其成功求解的充分条件,然后利用 2 个大规模的 RTVKP3 问题实例,通过仿真计算进一步阐明算法的可行性与高效性。

不妨设算法 3 依次随机产生的 m 个背包载重构成的集合为 $P=\{C_1,C_2,\cdots,C_m\}$,其中 C_k 为$[A,B]$上的随机正整数$(1\leqslant k\leqslant m)$,并且 C_k 先于 C_{k+1}产生。令 $\Delta C=\max\{C_{k+1}-C_k \mid C_k\in P\wedge 1\leqslant k\leqslant m-1\}$。注意到算法 3 初始相当于求解 0-1KP($n$,$C_1$,$S$,$W$),时间复杂度为 $O(nC_1)$;此外,各时间间隔计算的时间复杂度为 $O(n\Delta C)$,故而算法 3 的时间复杂度为 $O(nC_1)+(m-1)O(n\Delta C)=O(\max\{C_1,\Delta C\}nm)$。为了保证算法 3 求解成功,必须保证各时间间隔的计算能够成功,为此只需满足 *Threshold* 的值大于等于 $\max\{O(nC_1),O(n\Delta C)\}$。若假设算法 3 中的基本运算的最大耗时为 T,则只要 *Threshold* 满足 $Threshold\geqslant nT\max\{C_1+1,\Delta C+1\}$,就能够保证算法 3 成功求解 RTVKP3 问题。

为了进一步说明利用本文算法求解 RTVKP3 问题的可行性与高效性,下面利用它求解两个大规模的 RTVKP3 问题实例。其中,两实例的时间间隔 Δt 的上界 *Threshold* 设置为 3*s*。仿真实验使用 lenovo X201i 笔记本,其硬件环境为 Intel (R) Pentium(R) CPU:U5400−1.20GHz,2.00GB 内存,操作系统为 Windows 7,并利用 VC++6.0 进行编程实现。

TVKP3 实例 1　有 300 个物品的 RTVKP3 问题实例。

$S[1..300]$={383,519,420,272,166,125,354,374,44,540,9,108,13,4,403,376,599,432,184,439,114,45,333,238,95,10,195,542,231,476,129,582,223,210,442,250,116,211,342,461,300,368,327,524,460,158,171,261,24,89,174,214,455,87,222,588,25,453,256,458,375,129,104,428,344,165,556,166,359,440,373,210,576,14,548,105,396,116,243,196,583,307,141,345,544,500,250,280,449,388,107,135,182,235,521,480,48,272,17,190,122,6,380,226,243,567,513,444,469,567,86,520,573,125,494,123,30,276,288,219,191,91,531,382,508,541,574,568,111,581,452,351,74,411,239,513,39,43,213,484,189,314,240,25,253,430,239,494,71,296,568,359,460,242,307,186,366,215,347,240,386,178,510,118,487,468,116,376,136,593,500,514,294,508,514,322,164,544,20,224,408,436,418,234,102,558,452,362,527,240,288,179,544,174,498,370,325,521,543,248,341,516,49,440,319,346,551,454,587,374,29,511,424,419,127,471,596,385,578,148,28,421,542,358,108,538,143,405,59,267,300,458,140,383,364,445,424,488,42,65,179,303,435,370,304,584,277,82,33,77,382,434,438,232,169,160,390,24,340,332,541,91,574,318,317,577,356,332,237,172,415,489,444,102,46,406,122,269,18,296,516,42,490,107,109,294,391,164,162,438,518,122,290,504,448,408,205,266,390,470}

$W[1..300]$={653,11,543,649,278,173,879,796,710,840,238,280,844,886,522,30,982,754,182,163,155,969,766,433,710,888,802,295,386,985,8,152,483,828,488,685,373,44,117,599,369,619,543,902,177,655,842,257,945,684,238,512,570,507,516,557,27,839,566,613,612,524,456,82,485,810,492,889,729,636,263,645,191,45,109,937,688,42,634,890,431,34,291,916,478,173,258,977,443,920,643,87,91,565,822,374,438,421,759,246,791,420,714,546,134,238,173,874,904,71,624,150,778,378,607,576,686,547,249,120,483,563,733,217,108,645,898,861,646,751,422,165,528,288,590,342,683,147,495,32,676,192,464,480,853,322,978,914,126,637,673,634,194,29,659,735,477,726,996,201,336,515,533,483,434,956,139,95,448,140,362,150,777,480,731,549,49,492,324,977,252,72,837,198,746,600,770,195,736,197,956,74,464,853,273,659,926,571,527,495,563,216,784,396,510,35,926,253,877,740,85,839,447,108,575,912,639,985,738,774,948,66,544,789,905,331,347,980,951,699,653,854,488,594,99,161,698,579,476,712,782,545,29,996,818,225,44,501,93,319,565,80,101,173,846,279,264,338,784,356,976,733,536,911,607,722,167,862,93,263,334,471,727,808,648,973,396,730,927,118,455,559,771,538,306,378,478,698,469,490,140,121,396,292,722,431,830,472,174,541}

背包载量变化区间$[A,B]$=[81750,117564],依次随机产生 10 个不同的背包载重。

TVKP3 实例 2　有 500 个物品的 RTVKP3 问题实例。

$S[1..500]$={461,583,490,253,305,153,70,52,463,370,348,27,119,284,532,156,37,564,540,589,191,590,34,515,27,128,214,439,347,254,515,396,116,358,572,499,273,97,319,378,96,6,118,545,595,480,478,416,386,

545,391,89,523,524,124,88,285,569,380,146,299,114,424,299,583,58,549,331,431,200,223,163,537,188,410,182,252,166,315,353,282,491,26,79,348,395,403,523,431,201,204,159,106,338,292,245,104,155,150,18,59,105,528,363,280,95,30,599,254,203,438,30,251,249,483,2,192,472,55,308,13,436,15,502,174,227,44,370,48,117,170,324,509,527,228,417,523,237,473,112,313,308,371,79,354,109,166,500,506,476,562,365,507,416,255,357,121,444,181,427,119,389,570,220,62,370,547,511,206,255,356,37,65,361,323,572,120,136,348,246,219,291,320,211,111,571,533,596,21,154,330,205,581,303,409,299,162,569,467,512,21,345,454,313,442,119,119,238,299,484,90,555,49,103,339,420,568,82,152,230,468,305,558,128,564,586,479,557,135,490,318,81,125,543,219,357,150,346,378,599,443,58,122,351,148,81,24,581,98,263,152,384,428,280,374,236,528,14,318,287,72,330,512,113,214,307,237,115,565,63,562,568,549,73,161,3,303,27,20,254,469,160,492,565,342,289,323,186,118,375,353,512,166,95,240,518,360,557,84,28,417,75,216,363,259,320,394,481,157,573,33,545,412,283,471,105,398,354,325,266,240,61,355,531,109,22,147,468,104,194,160,364,576,217,587,594,370,189,529,187,474,540,521,67,345,383,106,560,528,550,205,31,561,302,210,67,184,157,79,127,140,355,273,195,391,381,257,42,194,176,278,342,567,246,227,110,396,167,129,316,166,163,118,552,458,374,169,288,529,424,141,290,276,224,106,392,578,445,178,410,373,311,358,443,92,350,87,380,229,447,368,402,467,259,546,257,219,172,528,324,253,308,235,352,454,380,299,102,578,338,360,132,486,531,117,342,4,10,276,23,487,17,482,231,533,410,80,294,89,387,396,398,183,372,100,407,506,224,169,270,110,578,371,141,2,130,398,141,338,170,588,194,80,328,110,486,166,348,35,118,26,355,550,313,472,565,51,98,464,22,415,294,80,359,103,565,563,461,115,365}

$W[1..500]$={122,158,190,628,444,596,335,167,854,861,364,276,507,842,334,34,620,358,977,455,922,99,8,263,848,330,84,301,9,619,947,817,825,383,668,844,905,504,702,295,436,223,319,12,158,934,490,179,411,332,391,731,411,690,18,591,856,948,469,535,598,787,992,549,104,946,538,461,432,201,341,112,847,811,646,257,998,695,723,244,984,718,270,485,240,353,949,953,753,511,379,721,753,997,692,561,780,380,407,262,478,931,109,842,616,42,708,327,486,553,149,684,176,213,134,160,271,893,175,838,283,193,489,35,944,641,277,974,903,541,902,555,119,667,490,633,697,451,92,460,903,62,106,480,648,434,525,928,595,251,90,654,10,295,897,779,665,433,998,429,944,609,882,373,853,109,195,959,468,302,207,621,158,98,560,137,883,526,15,285,124,680,916,939,526,242,529,978,236,216,733,894,403,92,626,274,410,874,954,619,135,981,892,900,381,466,102,482,594,358,591,52,396,569,699,953,449,983,734,589,401,153,585,261,401,220,42,141,738,313,530,135,724,4,410,509,930,296,346,873,116,323,875,771,574,226,511,884,220,795,486,483,330,336,282,346,285,551,857,182,15,895,703,757,94,606,845,189,588,38,332,589,8,939,511,916,49,684,49,642,106,78,983,665,20,981,174,294,30,766,29,930,441,421,484,659,519,151,113,405,162,671,725,835,454,653,366,483,432,107,632,662,797,556,253,461,287,430,440,771,32,71,139,643,441,809,306,539,905,125,746,344,884,101,416,941,268,33,530,389,763,374,407,308,411,80,742,122,177,747,883,370,716,198,190,704,468,910,627,20,783,941,909,122,411,753,518,162,283,371,327,282,105,499,547,63,90,425,327,426,561,324,731,207,128,602,815,22,197,94,533,451,505,539,589,576,425,584,573,277,760,287,96,732,945,814,46,139,400,491,705,673,932,276,236,525,342,267,270,828,328,855,731,507,451,580,397,873,35,868,745,191,185,127,406,235,999,980,327,273,333,903,801,292,13,644,673,493,654,97,235,882,390,421,756,185,623,452,569,302,571,411,825,933,666,356,651,11,604,981,435,126,904,812,367,433,943,794,349,1,989,724,615,266,130,535,965,209,454,867,171,76,790,741,29,861,742,371,448,604}

背包载量变化区间$[A,B]$=[137598,179582],依次随机产生 10 个不同的背包载重。

实例 1 和 2 的仿真实验结果如表 1 所列,其中的"最优结果"采用"最优值/最优载重"的形式给出[1],"求解用时"为各时间间隔所对应 0-1 背包问题的求解耗时,其时间单位为秒。

表 1　利用算法 3 求解 RTVKP3 的实例 1 和 2 的计算结果

TVKP3 实例 1	背包载重随机取值	84340	108991	109546	86214	117084
	求解用时	0.685	0.199	0.005	0.000	0.069
	最优结果	78116/84337	87130/108982	87280/109541	78899/86205	89174/117083
TVKP3 实例 1	背包载重随机取值	96073	117490	88360	88808	113907
	求解用时	0.000	0.004	0.000	0.000	0.000
	最优结果	82790/96073	89280/117487	79790/88346	79965/88791	88406/113903
TVKP3 实例 2	背包载重随机取值	142367	161067	157625	143779	150350
	求解用时	2.094	0.208	0.000	0.000	0.000
	最优结果	129440/142361	135999/161062	134892/157617	129976/143776	132381/150333
TVKP3 实例 2	背包载重随机取值	138072	151095	146019	171569	146895
	求解用时	0.000	0.000	0.000	0.155	0.000
	最优结果	127769/138072	132649/151078	130811/146017	139050/171550	131134/146893

从表 1 中可以看出,在 TVKP3 实例 1 中,对于在区间[81750,117564]上依次随机选取的背包载重 84340,108991,109546,86214,117084,96073,117490,88360,88808,113907,只要 $Threshold \geqslant 1.0$(s),算法 3 就能够成功地对其进行求解;在 TVKP3 实例 2 中,对于在区间[137598,179582]上依次随机选取的背包载重 142367,161067,157625,143779,150350,138072,151095,146019,171569,146895,只要 $Threshold \geqslant 2.1$(s),算法 3 就能够成功地对其进行求解。此外,将上述求解结果与利用文献[5]中进化算法所得的求解结

[1] 注:由于篇幅所限,没有给出最优解对应的二进制向量,若需要请发邮件索取

果相比较，容易发现进化算法通常不易求得 RTVKP3 问题的最优解，而且所耗费的计算时间也非常多。特别是当 TVKP3 实例的规模≥300，并且背包载重变化周期不超过 3s 时，进化算法的求解效果与速度往往较差。

结束语 本文利用 RTVKP3 问题的物品价值与重量不随背包载重的变化而改变这一特性，给出了一种基于动态规划算法求解 RTVKP3 问题的有效方法。通过仿真计算容易看出：动态规划法不仅是目前求解背包问题的最有效算法之一，而且也是求解背包载重动态变化的动态背包问题的一种有效算法。通过以上讨论可知，利用动态规划法求解 RTVKP3 问题，其效果和速度均优于进化算法，并且便于编程实现。此外，注意到本文方法的通用性，今后将进一步分析如何将本文方法推广，以用于求解物品价值、重量与背包载重均随时间变化的 RTVKP 问题。

参 考 文 献

[1] Jin Y, Branke J. Evolutionary optimization in uncertain environments[J]. IEEE Transactions on Evolutionary Computation, 2005, 9(2): 303-317

[2] 陈国良，王熙法，庄镇泉，等. 遗传算法及其应用[M]. 北京：人民邮电出版社，2003

[3] Goldberg D E, Smith R E. Nonstationary function optimization using genetic algorithms with dominance and diploidy[C]// International Conference on Genetic Algorithms. Hillsdale: L. Erlbaum Associates Inc., 1987: 59-68

[4] 汪定伟，王俊伟，王洪峰，等. 智能优化方法[M]. 北京：高等教育出版社，2007

[5] Yang S. Non-stationary problem optimization using the primal-dual genetic algorithm[C]// Proceeding of the 2003 congress on Evolutionary Computation. 2003, 4: 2246-2253

[6] 周传华，谢安世. 一种基于动态小生境的自组织学习算法[J]. 软件学报，2011(08)

[7] Sedgewick R, Flajolet P. An introduction to the analysis of algorithms[M]. Boston: Addison-Wesley Publishing Company Inc., 1999

(上接第 221 页)

表 1 不同相似度量比较

	u	v	M_*[7]	M_{**}[7]	M_{**}[10]	M[13]	M[15]	M[16]	MM_1	MM_2
1	[0.4,0.8]	[0.3,0.7]	0.9	0.9	0.89	0.9	0.94	0.675	0.81	0.644
2	[0.4,0.8]	[0.3,0.8]	0.9	0.9167	0.9463	0.9	0.965	0.7125	0.8325	0.6815
3	[0.4,0.8]	[0.3,0.9]	0.8	0.8667	0.945	0.9	0.95	0.675	0.7767	0.5872
4	[0.4,0.8]	[0.5,0.7]	0.8	0.8667	0.945	0.9	0.93	0.765	0.8267	0.6775
5	[0.3,0.7]	[0.4,0.6]	0.8	0.8667	0.95	0.9	0.93	0.765	0.8317	0.685
6	[0.3,0.6]	[0.1,0.7]	0.9	0.9	0.8925	0.9	0.93	0.72	0.8325	0.6863
7	[0.3,0.8]	[0.1,0.7]	0.8	0.8667	0.9475	0.9	0.94	0.72	0.8042	0.6359
8	[0.5,0.5]	[0.5,0.5]	1	1	1	1	1	1	1	1
9	[0.3,0.7]	[0.3,0.7]	1	1	1	1	1	0.8	0.9	0.8
10	[0.2,0.8]	[0.2,0.8]	1	1	1	1	1	0.7	0.85	0.7
11	[0,1]	[0,1]	1	1	1	1	1	0.5	0.75	0.5

结束语 本文方法为提出 Vague 集多测度相似度量提供了一种新的思路。与文献[3-16]中的 Vague 集相似度量相比，本文提出的多测度 Vague 集相似度量主要有以下优点：(1)在空间上体现出“点-线-面”多特征相似性度量格局，当其中的某个特征失效时，其它特征还能继续发挥度量作用，因而具有更高和更好的分辨力；(2)多测度的相似度量将不同测度体现出的不同区分能力有效地结合起来，进一步增强了分辨力，从而避免了文献[8,10]人为给定参数权重来提高区分力度的情况，使度量相似性结果更客观。

参 考 文 献

[1] Gau W L, Daniel J. Vague sets[J]. IEEE Transactions on Systems, Man, and Cybemetics, 1993, 23(2): 610-614

[2] 王昌. Vague 集的模糊熵、相似度量和距离测度的关系[J]. 计算机科学，2010，37(10)：221-224

[3] Chen S M. Measures of similarity between vague sets[J]. Fuzzy Sets and Systems, 1995, 74(2): 217-223

[4] Chen S M. Similarity measures between vague sets and between elements[J]. IEEE Transactions on Systems, Man, and Cybemetics, 1997, 27(1): 153-158

[5] Hong D H, Kim C. A note on similarity measures between vague sets and between elements [J]. Information Sciences, 1999 (115): 83-96

[6] 李凡，徐章艳. Vague 集之间的相似度量[J]. 软件学报，2001，12(6)：922-926

[7] 范九伦. Vague 值与 Vague 集上的贴近度[J]. 系统工程理论与实践，2006，26(8)：95-100

[8] 兰蓉，范九伦. Vague 值和三参数 Vague 值上的贴近度[J]. 模式识别与人工智能，2010，23(3)：341-348

[9] 刘华文. 模糊模式识别的基础—相似度量[J]. 模式识别与人工智能，2004，17(2)：141-145

[10] 张清华. Vague 值(集)相似度量的研究[J]. 电子与信息学报，2007，29(8)：1855-1859

[11] 李艳红，迟忠先，阎德勤. Vague 相似度量与 Vague 熵[J]. 计算机科学，2002，29(12)：129-132

[12] 闫德勤，迟忠先，李艳红. 关于 Vague 集的相似度量[J]. 模式识别与人工智能，2004，17(1)：22-26

[13] 闫德勤. Vague 集的相似度量[J]. 计算机科学，2006，33(5)：195-196

[14] 朱振国，王匡胤. Vague 集相似度量[J]. 计算机科学，2008，35(9)：220-225

[15] 黄国顺，刘云生. 基于 Vague 熵的 Vague 集相似度量[J]. 小型微型计算机系统，2008，29(1)：139-144

[16] 贾保华. 基于未知度的 Vague 集相似度量新方法[J]. 昆明理工大学学报：理工版，2010，35(5)：112-117

[17] Eulalia S, Janusz K. Entropy for intuitionistic fuzzy sets[J]. Fuzzy Sets and Systems, 2001(118): 467-477

[18] 耿修瑞，张兵，张霞，等. 一种基于高维空间凸面单形体体积的高光谱图像解混算法[J]. 自然科学进展，2004，14(7)：810-814