背包问题的二分网格算法*)

李庆华 潘 军 李肯立

(国家高性能计算中心(武汉) 华中科技大学计算机科学与技术学院 武汉430074)

摘 要 本文介绍了属于NP难问题的无界背包问题的一种新的精确算法,基于问题的几何结构通过二分搜索方法不断减小解空间,最终直接求出问题的最优效益值和最佳装包方案。当待装入包中的物品数量固定时,算法的时间复杂性为线性时间,初步解决了求解当前呈指数增长背包实例时存在的困难,实验中各种数据实例证明与常用的MTU2和EDU相比,该新算法在理论上是可行的。

关键词 背包问题,计算复杂性,NP-难,二分网格

A Dimidiate Grid Algorithm for the Unbounded Knapsack Problem

LI Qing-Hua PAN Jun LI Ken-Li

(National High Performance Computing Center (Wuhan), School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract This paper presents a new exact algorithm for the unbounded knapsack problem, which is a famous NP-hard problem. It is based primarily on the basic geometric structure of the problem, by the tool of two-partition, the solution space of the problem is increasingly reduced until the optimal allocation of objects and optimal profits is directly obtained. As the complexity of the algorithm is linear to the length of the input data when the number of items is fixed, it can preparatorily overcome the present hardness in solution of knapsack problems with exponentially growing coefficients. The computational experiments with various data instances, comparing our new algorithm with the well-known MTU2 and EDUK are presented, and they demonstrate the theoretical performance of the proposed algorithm.

Keywords Knapsack problem, Computational complexity, NP-hard, Dimidiate grid

1 引言

背包问题(Knapsack problem)属于组合优化理论中最重要的问题之一,在网络设计、网络路由、任务调度和资金分配等问题中具有重要的应用[1,2]。无界背包问题属于NP-难问题[3],该问题可描述如下:给定一容量为 c 的背包和 n 种物品,其中每种物品的数量是无限的,物品 i 的容量和效益分别为 w_i 和 p_i,问采用何种装包方法可使装入物品的总效益值最大。其形式化描述为:

$$\text{maximize } Z=\sum_{i=1}^{n} p_i x_i$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i \leqslant c, x_i \in Z^+, i=1,2,\cdots,n \qquad (1)$$

目前,求解此问题的算法主要有动态规划和分枝限界法,分枝限界算法对某些问题实例表现了较好的求解性能,但由于它的求解过程是一棵最坏情况下具有指数节点数的二元树,因而很难求解问题规模稍大的难解背包问题[4];动态规划方法能以 $O(nc)$ 的时间求解背包问题,因此能够在现实时间求解待装入背包的物品数很多但背包容量不太大的背包实例,而对那些系数呈指数增长的实例,则同样很难在合理的时间内求得问题的精确解[4],因为就算法复杂性而言,这些算法的运算时间都囿界于问题输入规模 L 的指数函数,这里问题规模定义为 $L=n\log(c+2)$[5]。

特别地,当待装入背包的物品数 n 为一固定值时,上述算法仍非多项式算法(尽管动态规划的时间复杂性对背包容量 c 是线性的)。不过Lenstra在他的著名文[5]中提出了该问题的一个多项式算法,但遗憾的是,该算法使用了较复杂的数学工具,使得算法只具有理论意义[2]。枚举法对物品数固定的0-1背包是多项式算法,但对本文所讨论的一般无界背包问题,该算法同样是指数时间算法[2]。

本文利用二分网格这一几何工具,提出一种求解无界背包问题的新算法。本文的结论主要体现在以下方面:

·当待装入背包的物品数固定时,新算法的时间复杂性为 $O(L)$,这一结论虽不比不差于Lenstra[5]算法的理论复杂性更低,但新算法没有使用繁难的数学工具——椭球法。

·通过数值实验证明了算法的优越性。实验结果表明:对物品数 n 不太多而背包容量 c 十分庞大的背包实例,本文算法在引入一些处理技术后,其求解效率和稳定性明显优于目前为止性能最高的分枝限界法MTU2和动态规划算法EDUK。

·新算法具有较好的并行性。由于问题的难解性,算法的内在并行性对精确求解大规模背包实例十分重要。但目前无论是并行分枝限界算法还是并行动态规划算法,都存在一些瓶颈问题没有解决[6~8],因此,新算法的并行化可望为实际应用中大规模背包问题的精确求解提供新的途径。

本文第2节提出新算法并分析算法的理论性能;第3节引入两种提高算法效率的启发式方法;第4节是数值实验,最后是本文的总结和今后工作的研究方向。

记号:n 维向量 W、P 分别表示给定背包实例的容量向量和效益向量;$WX=\sum w_i x_i$、$PX=\sum p_i x_i$ 分别表示 n 维实空间中向量 X 所产生的容量和效益,若 X 表示集合,则 $|X|$ 表

*)本文受到国家自然科学基金(60273075)和国家863高科技发展计划(863-306ZD-11-01-06)资助。李庆华 教授,博士生导师,研究领域为组合优化、并行处理。潘 军 副教授,研究兴趣为组合优化。李肯立 博士,研究兴趣为组合优化、并行计算。

示该集合的基数；物品的利润率是指其效益与容量的比值 p_i/w_i。

2 新算法

考虑(1)式中的容量约束 $\sum_{i=1}^{n} w_i x_i \leqslant c, x_i \in Z^+, i=1,2,\cdots,n$，假定所有物品的容量 w_i 都满足 $w_i \leqslant c$，否则，若某 $w_j > c$，则可将物品 j 直接舍弃，因为背包实例的任何最优解都不可能包含物品 j。实际上，变量 x_i 只能取范围 $0 \leqslant x_i \leqslant [c/w_i]$ 中的整数值。

2.1 二分网格

令 $a=\max\{[c/w_i], i=1,2,\cdots,n\}=[c/w_k], k\in\{1,2,\cdots,n\}$。引入一各边界面分别与 n 维坐标平面平行、其中一个顶点为坐标原点而边长为 a 的 n 维超立方体 D，显然，立方体区域 D 包含问题(1)的全部可行解。

定义1 R^n 中任意点 X 到容量约束超平面 $WX=c$ 的离差 d 定义为：

$$d=(WX-c)/|W|, \text{其中} |W|=(\sum_{i=1}^{n} w_i^2)^{\frac{1}{2}}, \tag{2}$$

显然，点 X 的分量是背包的一种可行装入方法的必要条件是 $d\leqslant 0$。今将超立方体 D 用平行于 D 边界面的超平面(即平行于 n 个坐标平面的超平面)等分成 2^n 个子超立方体，记每个子超立方体为 $D_i^1(i=1,2,\cdots,2^n)$。由于 D 包含问题的全部可行解，因此：问题的最优解一定位于某些子超立方体内。这 2^n 个子超立方体中的绝大多数将根据删除规则删除。再用同样的办法将保留下来的每个子超立方体分成 2^n 个更小的子超立方体 $D_i^2(i=1,\cdots,2^n,\cdots)$，这样继续下去，随着子超立方体边长不断减少，通过必要的计算，最后必能直接找到问题的最优解和相应的效益值。注意到 D 的边长为 a，在经过 m 次分割后得到的子超立方体边长为 $a_m=a/2^m$，外接球半径 r_m 为：

$$r_m=\frac{1}{2}a_m\sqrt{n}=\frac{a}{2^{m+1}}\sqrt{n}, m=1,2,\cdots \tag{3}$$

由于背包问题的任意可行解的各分量必须为正整数，在使用分枝限界算法或动态规划方法求解背包实例时常会出现因频繁取整而导致累积误差过大的现象，这种累积误差可使最终解偏离实际的精确最优解[9]。考虑到计算机使用二进制编码，若将初始超立方体的边长 a 取成2的幂，并设 D_i^m 的中心为 X_i^m，D 的中心为 X_0。则只要 $a_m\geqslant 2$，子超立方体 D_i^{m-1} 被分割后得到的各子超立方体 D_j^m 的中心 X_j^m 可由 D_i^{m-1} 的中心 X_i^{m-1} 通过简单的坐标移位得到，同时 X_j^m 的各分量坐标依然保持为整数。为此，将初始超立方体 D 的边长修正成：

$$a=2^{\lceil \log \frac{c}{w_k} \rceil} \tag{4}$$

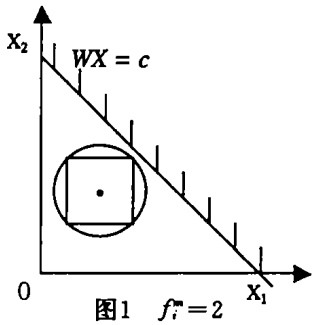

图1 $f_i^m=2$

由(2)式，X_i^m 可行，$i=1,2,\cdots 2^n,\cdots,m=1,2,\cdots$，当且仅当 $d_i^m=WX_i^m-c\leqslant 0$。不过，$X_i^m$ 仅被视为整个子超立方体 D_i^m 区域的代表，而我们真正感兴趣的则是子超立方体 D_i^m 区域内是否含有满足背包问题容量约束的整数解。因此，定义如下分段函数 f_i^m：

$$f_i^m(D_i^m)=\begin{cases} 2 & d_i^m\leqslant -r_m, \\ 1 & -r_m<d_i^m\leqslant 0, \\ 0 & 0<d_i^m\leqslant r_m, \\ -1 & d_i^m>r_m \end{cases} \tag{5}$$

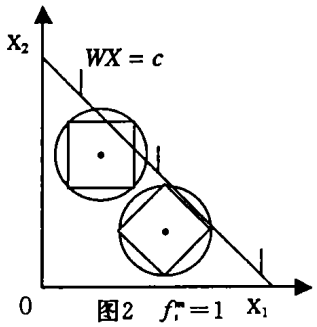

图2 $f_i^m=1$

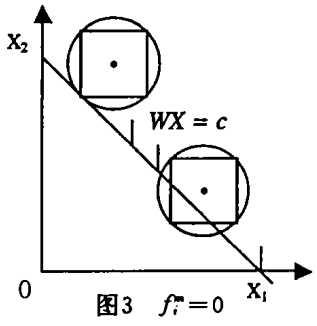

图3 $f_i^m=0$

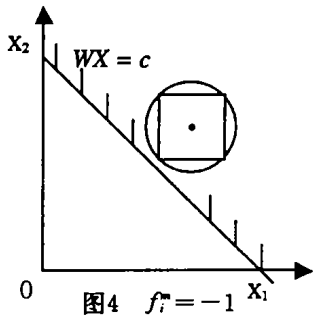

图4 $f_i^m=-1$

从几何上考虑，函数 f_i^m 的意义是显然的。如图1示，若 $f_i^m=2$，则整个立方体 D_i^m 区域内的整数点都是可行的。当 $a_m\geqslant 1$ 时，子超立方体 D_i^m 所有 2^n 个顶点中的某一顶点(同时也是整数点)的效益值是 D_i^m 内各整点中效益值最大的点。因为所有物品的效益值均非负，这个效益最大值顶点 $U_i^m=(u_{i1}^m,\cdots,u_{ik}^m,\cdots,u_{in}^m)$ 的各坐标分量 u_{ik}^m 可从 D_i^m 的中心 $X_i^m=(x_{i1}^m,\cdots,x_{ik}^m,\cdots,x_{in}^m)$ 按下式

$$u_{ik}^m=x_{ik}^m+\frac{a_m}{2}, m=1,2,\cdots \tag{6}$$

求得。此时，在 U_i^m 用于效益值比较后，可将子立方体 D_i^m 直接删除。

如图2，$f_i^m=1$ 表明：一方面，如果 $a_m>1$，则 D_i^m 的中心 X_i^m 对应一种可行的装包方案，但不能肯定 D_i^m 内是否还有效益值比 X_i^m 的效益值更高的其它整数点；另一方面，如果 $a_m=1$，X_i^m 已不再满足整数性要求，但其整数部分 $[X_i^m]$ 则依然可行。因此，此时的子超立方体 D_i^m 除了用于效益值的比较外，还应将 D_i^m 保留，以用于下一步计算。

如图3示，若 $f_i^m=0$，则子超立方体 D_i^m 虽可能包含有可行的背包装入方案，但其中心 X_i^m 一定不可行，此时，Z_i^i 不能用于效益值的比较，但 D_i^m 应当保留。

若 $f_i^m=-1$，显然，由于整个子超立方体 D_i^m 区域都在问题的容量约束之外，D_i^m 内的任意点都不可行，此时应将 D_i^m 直接删除，见图4。

2.2 新算法

记 $I=\{1,2,\cdots,2^n\}$。将边长为 a 的初始立方体 D 等分成 2^n 个子超立方体 $D_i^1(i\in I)$。D 的中心为 $X_0=(a/2, a/2, \cdots, a/2)$，由于背包问题的最优解包含在 D 所表示的有限区域内，这样，在对 D 进行多次类二分搜索后，最后必能直接求得问题的最优解。新算法的主要步骤如下：

背包问题的新算法(The new algorithm for the UKP)

Step 0 begin

Step 1 while $a \geqslant 2$ do {

$a=a/2; r=\sqrt{n}\,a/2; Z=0$; //$Z$ 相当于装包效益值能够达到的一个下界，算法开始执行时被赋初值0//

每个子超立方体被分割成2^n个更小的子超立方体，生成子超立方体的下标集 I;

Step 2 下标集 $I=\{1,2,\cdots\}$根据 $f_i(i\in I)$的值分裂成下述三个指标集 E_1、E_2、E_3，其中：

保留集：$E_1=\{i\in I \mid f_i\geqslant 0\}$ //下标在 E_1内的子超立方体可能包含可行解//

比较集：$E_2=\{i\in I \mid f_i=1\}$//下标在 E_2内的子超立方体将用于效益值比较中//

内点集：$E_3=\{i\in I \mid f_i=2\}$ //下标在 E_3内的子超立方体在用于效益值比较后，将被删除//

Step 3 if ($E_2\cup E_3=\Phi$) o to step 1 //对每个子超立方体继续分割//

else {if ($E_2\neq\Phi$){

if($a>1$) compute $Z_1=\max\{PX_i \mid i\in E_2\}$

else compute $Z_1=\max\{P[X_i] \mid i\in E_2\}$//边长 $a=1$时，D_i 的中心 X_i 不再可行//

}

elseif ($E_3\neq\Phi$) compute $Z_2=\max\{PU_i \mid i\in E_3\}$where U_i is given by formula (6)

compute $Z=\max\{Z_1, Z_2, Z\}$

}

Step 4 $E_1=E_1-E_3$

if($PU_i<Z, i\in E_1$)where U_i is given by formula (6)

将子超立方体 D_i 的下标从保留集 E_1中删除; //删除子超立方体内最大效益值小于下界 Z 的子超立方体//

}

Step 5 {

计算保留集 E_1中子超立方体的所有顶点；

删除不可行的顶点；

计算可行顶点的效益值；

比较效益值，得到最优效益值和相应的装包方案；

} //若 $a=1$，不再继续迭代，直接计算最优效益值//

Step 6 end

说明：由于上一次迭代中得到的下界 Z 可能是背包问题的最优效益值，而 Z 值可能由内点集得到(即 $Z=Z_2$)，但内点集 E_3与步骤4中保留集 E_1的交为空，因此，为防止最优效益值和相应的最优装包方案丢失，在后续迭代的步骤3中，下界 Z 取成上次迭代的界 Z、本次迭代得到的 Z_1和 Z_2三个值中的最大值。

2.3 算法性能分析与比较

算法的正确性是显然的，因为与 Lenstra[5]的结果一样，新算法也是对背包问题可行的装包方案或解空间进行隐枚举，不同的是，文[5]中使用复杂的椭球规约方法，本文使用相对简单的类二分搜索方法，当待装入背包的物品数固定时，算法的复杂性虽然都是多项式的，但本文算法的复杂性更低，仅为问题输入规模的线性函数。以下详细分析新算法的复杂性。

如引言所述，问题的输入规模为 $L=n\log(c+2)$[5]，显然，使用本文算法最多迭代(while 循环)$\log a$ 次即可使子超立方体的边长 a 缩减成1，在可能进行的 $\log a$ 次迭代中，每一次迭代的运算量主要来自对子超立方体的众多下标进行划分，因为这需要计算函数 f_i 的多个函数值。首先考虑第一次迭代的运算量：步骤1生成2^n个子超立方体的下标集及每个子超立方体的坐标需要 $O(n2^n)$次算术或移位运算；步骤2中任意子超立方体被归入某一下标集时，最坏情况下需对每一个子超立方体计算函数 $f_i(i\in I)$的值，由于计算 f_i 的一个函数值需 $O(n)$次运算，因此，完成步骤2的运算量为 $O(n2^n)$。在初始指标集划分完后，大量不可能包含可行装包方案的子超立方体已被删除；算法的步骤3对保留的子超立方体内整数点的效益值进行比较，以求得问题的一个下界，其运算量为比较集 E_2 与内点集 E_3中元素个数之和的2n 倍，即 $O(n(|E_2|+|E_3|))$；步骤4中利用效益值下界 Z 对步骤2保留的子超立方体作进一步删除，其运算量不超过 $O(n(|E_1|-|E_3|))$。因此，第一次迭代或循环的总运算量为 $O(n2^n)$。后续各次迭代中，待分割的子超立方体个数为上次迭代步骤4中的$|E_1|$，虽然一般情况下$|E_1|>1$，但此时$|E_1|$的值仅与 n 有关，不妨令$|E_1|=g(n)$，显然 $g(n)$将远小于2^n。而且随着子超立方体边长的缩小，保留集 E_1的基数$|E_1|$将持续减少。而当 $a=1$时，步骤5直接求问题的最优效益值和相应的装包方案，其运算量不会超过一次迭代的运算 $O(n2^n)$。综合以上分析，本文算法的总运算量最多为：$O(n2^n(\log a+1)\times g(n))$。于是，有下述算法复杂性定理成立：

定理1 物品数目固定的背包问题能在线性时间 $O(L)$ 内求解。

证明：当待装入背包的物品数 n 固定，从而与问题的输入规模 L 无关时，注意到此时 $g(n)$将被视为一个常数，由(6)式及 L 的定义，任意背包问题实例用本文算法求解的时间复杂性为：

$$O(n2^n(\log a+1)\times g(n))=O(n2^n\log\frac{c}{w_k})=O(n2^n(\log c-\log w_k))\leqslant O(n2^n\log c)=O(n2^n\log(2^{\frac{L}{n}}-2))=O(2^nL)=O(L)$$

□

对于物品数为一固定值的背包问题，考虑求解背包问题常用的两种算法。动态规划以 $O(nc)$的时间(实际上就是 $O(c)$)求解背包问题，尽管一般认为分枝限界算法较动态规划更为有效，但和动态规划算法一样，其最坏情形下的计算时间是问题输入规模 L 的指数函数[2,5]。因此，当问题的规模尤其是背包容量 c 较大时，用上述方法很难在合理的时间内求得问题的精确解。Lenstra 根据 Hieschberg 和 Wong 在求解二维背包问题时提出的猜测[10]，利用 Khachiyan 关于线性规划的多项式可解性和规约方法，提出了该问题的一个多项式算法[5]，目前尚不见关于该算法复杂性的详细分析，但根据椭球方法的复杂性可知：Lenstra 算法的复杂性将至少为 $O(L)$，而且遗憾的是：由于该算法使用了复杂的数学工具，使得它仅具理论意义[2]。显然，即使和 Lenstra 算法相比：由于新算法中使用二分网格的几何工具，使得算法相对简单，易于实现，而且其理论时间复杂性也不高于该算法。当然，新算法并不改变问题在求解上的困难性，因为算法的复杂性对问题维数 n 还是指数阶的，这一点和包括分枝限界算法在内的其它算法没有区别。

正如二分查找算法对于选择问题是最优算法一样，由于算法中使用的基本方法是二分搜索方法，因此，新算法对物品数固定的背包问题在相差一个常数的范围内是最优的。

另外必须指出的是算法的内在可并行性。由于背包问题属于 NP-难问题，对问题的并行求解是寻求实际应用中大规模问题精确解的途径之一，而且，目前高性能计算在体系结构和并行处理软件上的日趋成熟为实现这一途径提供了可能，但是，商业上普遍使用的并行分枝限界算法在并行化时尚存在一些瓶颈问题，如由于问题结点树在深度优先搜索时本质上的不可并行性所导致的并行粒度过小、难于实现并行任务的有效调度、分配和管理、分枝点的保存池过于庞大[6]等。这些问题直接导致并行背包求解软件效率较低的局面。显然，如果要将新算法并行化，不难发现：算法的一次迭代中，只需在步骤4删除子超立方体前对各并行进程同步，而且各并行处理机间只需对下界 Z 的值进行通讯。此外，算法中使用的二分网格工具也非常适合超级计算机的体系结构。算法的这些特

点可望使并行新算法在粗粒度并行计算机上运行时取得较高的并行效率，为一般大规模背包问题的求解提供新的可选方法。

3 数值实验

不相关、弱相关和强相关三类规范的随机背包实例是背包问题常用的测试实例[1,11]，我们用这三类实例将本文算法和其它两种常用算法进行了数值对比实验。

表1 新算法与EDUK、MTU2算法用于求解规范随机背包实例时的性能对比

物品数 n	容量 c	不相关实例			弱相关实例			强相关实例		
($\times 10^3$)	($\times 10^7$)	EDUK	MTU2	NEW	EDUK	MTU2	NEW	EDUK	MTU2	NEW
1	4	0.15	0.05	1.20	1.25	0.84	3.43	1.96	1.07	2.89
	64	2.21	3.46	5.14	10.4	7.83	11.4	18.1	12.7	12.5
	1024	36.2	30.0	20.7	89.6	125	40.6	172	209	144.7
4	4	0.12	0.95	2.84	1.42	2.14	14.2	7.65	3.77	15.6
	64	1.25	4.98	9.64	18.2	16.4	28.6	37.8	46.5	55.1
	1024	29.2	121	34.4	305	294	91.3	1450	912	278
16	4	2.17	1.89	3.45	5.24	18.6	94.4	27.5	42.1	162
	64	14.5	12.2	14.5	96.1	82.4	296	504	393	514
	1024	641	245	62.5	—	—	954	—	—	827
64	4	3.25	3.24	12.4	25.3	248	685	183	835	—
	64	52.3	30.1	42.5	357	359	—	—	—	—
	1024	652	415	186	—	—	—	—	—	—

动态规划和分枝限界算法分别使用目前为止效率最高的EDUK算法[11]和MTU2算法[1]。而新算法的代码使用Fortran 77实现，实验平台为Intel 1.1G CPU和256M RAM。在使用新算法计算之前，采用了其它算法常用的文[12]中的预处理方法。表1是三种算法对180(60×3)个随机实例的实验结果，表中时间单位为秒，各背包实例中物品的最小容量均为15。为消除偶然因素对实验结果的影响，每个实验时间都是5次重复运算的平均时间，当某一实例的运行时间超过1000秒时，即认为该实例无法在合理的时间界限内完成，从而中止程序运行，在表1中相应位置用"—"表示。

从表1可见：对于物品数不多而背包容量特别大的实例，本文算法的效率和稳定性明显优于动态规划算法EDUK和分枝限界算法MTU2。目前文献结果表明[1,11]，对于物品效益和容量强相关的背包实例，使用动态规划和分枝限界算法都难以求得其精确解，但使用本文算法则可在较短的时间内得到这些实例的最优效益值，这一结果较好地验证了其它两种算法较本文算法对背包容量更敏感的理论分析；尽管最坏情况下，新算法和分枝限界法的时间复杂性都是随着问题中物品数增大而以指数形式增加，但是：对那些物品数较多而容量较小的背包实例，新算法的性能却明显不及分枝限界算法。使用一些其它方法，如完善程序代码、利用更有效的预处理技术等，新算法的实际性能可能得到进一步提高，但这尚需进一步的实验支持。

结论 背包问题属于NP-难问题，寻求一般背包问题的多项式算法是极为困难的。动态规划与分枝限界算法能够成功求解小型背包实例，但由于它们的求解时间都囿界于背包容量的一个线性函数，而背包容量恰是问题输入规模的指数形式，因此，对于容量呈指数方式增长的背包实例的求解，目前尚存在实质困难。本文提出一种求解背包问题的新算法，算法通过对问题的可行装包方案进行二分搜索，不断缩小其解空间，最终直接求出问题的最优效益值和最佳装包方案。当待装入背包的物品数固定时，算法的时间复杂性为线性时间O(L)。因此，本文算法可初步解决当前求解系数呈指数增长背包实例时存在的困难，而且由于算法适于并行处理，算法的并行化可望为一般大规模背包问题的精确求解提供一种新的可选方法。此外，如果只需求具有某种精度的近优方案，则算法可在中途某次迭代后停止，其中的下界Z就是近优效益值。

另一方面，实验结果表明：如果待装入背包的物品数较多，新算法不仅不如动态规划，而且也不及理论性能相同的分枝限界算法。因此，在今后工作中，我们拟在基于Cluster结构的并行平台上研究新算法的并行算法，并将它和目前广为使用的并行分枝限界算法和并行动态规划做更深入的比较研究，以求更好地解决一般大规模背包问题的精确求解问题。

参考文献

1 Martello S, Toth P. Knapsack Problems: Algorithms and Computer Implementation. New York: Wiley Press, 1990

2 Nemhause G L, Wolsey L A. Integer and Combinatorial Optimization. New York: Wiley Press, 1988

3 Garey M R, Johnson D S. Computers and Intractability, A Guide to the Theory of NP-Completeness. San Francisco: Freeman &Co, 1979

4 Martello S, Pinsinger D, Toth P. New trends in exact algorithms for the 0-1 knapsack problem. European Journal of Operational Research, 2000, 123: 325～332

5 Lenstra H W. Integer programming with a fixed number of variables. Mathematics of Operation Research, 1983, 8: 538～548

6 Teng S. Adaptive parallel algorithm for integral knapsack problems. Journal of Parallel and Distributed Computing, 1990, 8: 400～406

7 Andonov R, Rajopadhye S. Knapsack on VLSI: from algorithm to optimal circuit. IEEE Transactions on Parallel and Distributed Systems, 1997, 8(6): 545～562

8 Johnson E L, Nemhauser G L, Savelsbergh M P. Progress in linear programming-based algorithms for integer programming: An Exposition. INFORMS Journal on Computing, 2000, 12: 1201～1222

9 Pinsiger D. An expanding-core algorithm for the exact 0-1knapsack problem. European Journal of Operational Research, 1995, 87: 175～187

10 Hirschberg D S, Wong C K. A polynomial-time algorithm for the knapsack problem with two variables. Journal of ACM, 1976, 23: 147～154

11 Andonov R, Poirriez V, Rajopadhye S. Unbounded knapsack problem: Dynamic programming revisited. European Journal of Operational Research, 2000, 123: 394～407

12 Zhu N, Broughan K. A note on the reducing the number of variables in integer programming problems. Computational Optimization and Applications, 1997, 8: 263～272