基于 0– 1 背包问题的讨论

林 鑫
(同济大学 计算机系, 上海 200433)

摘 要: 简单介绍了贪婪算法、启发式贪婪算法和模拟退火算法(SAA), 并使用这三种算法解决了 0– 1 背包问题, 给出了具体的算法描述和求解过程。对三种方法解决此问题, 进行了仿真模拟和算法分析, 指出了在不同规模下各种方法的优缺点, 最后分析了解的质量和 CPU 时间, 发现模拟退火算法是相对最优的算法。

关键词: 0– 1 背包问题; 贪婪算法; 启发式贪婪算法; 模拟退火算法; CPU 时间
中图分类号: TP301.6 文献标识码: A 文章编号: 1005– 3751(2005)10– 0041– 03

Discussion on the Zero– One Knapsack Problem

LIN Xin
(Dept. of Computer Science, Tongji University, Shanghai 200433, China)

Abstract: Simply introduces greedy algorithm, heuristic greedy algorithm and simulated anneanling algorithm and solves the zero– one knapsack problem through these methods. It provides concrete algorithms and describes the course of solving. To these three kinds of methods, this paper has carried on artificial simulation and algorithm analysis and also points out the pluses and minuses of different metlrods under different scale. Finally it analyses the quality of solution and CPU time. And find that simulated annealing algorithm is comparatively optimal.

Key words: zero– one knapsack problem; greedy algorithm; heuristic greedy algorithm; simulated annealing algorithm(SAA); CPU time

1 问题描述

背包问题是个典型的 N– P 问题, 在实际的工程中有着广泛的应用。

具体描述是: 已知尺寸大小分别为 $W_1, W_2, \cdots, W_n$ 的若干物品和容量为 C 的背包, 物品的价值分别为 $P_1, P_2, \cdots, P_n$。此处, $W_1, W_2, \cdots, W_n$ 和 C 都为整数。要求找出这 n 个物品的一个子集, 使其尽可能使选入背包的物品的价值最大, 即

最大化: $\sum_{i=1}^{n} P_i X_i$ 且满足 $\sum_{i=1}^{n} W_i X_i \leqslant C$

这里 $X_i \in \{0, 1\}$, $1 \leqslant i \leqslant n$, $X_i = 1$ 表示物品 i 被选入背包, $X_i = 0$ 表示未选入。比如当 $n = 3$ 时, $W = [2, 4, 6]$, $P = [3, 10, 12]$, $C = 11$, 即背包中装入物品的最大价值为 \$ 22(如图 1 所示)。

2 各种算法介绍

2.1 贪婪算法

贪婪算法[1]通过一系列的选择得到问题的解, 在每一次总是做出在当前状态下看来是最好的选择, 也就是希望通过局部的最优来达到一个全局的最优。这种启发式的策略并不总能获得最优解, 然而在许多情况下确能达到预期的目的。而且对于很多 N– P 问题来说, 本身就不存在最优解。

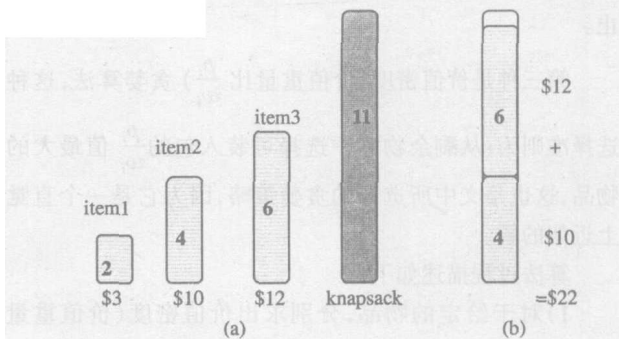

图 1 0– 1 背包示例

2.2 启发式贪婪算法

在贪婪算法的基础上又进行了改进, 详见下面 3.2 算法过程的具体描述。

2.3 模拟退火算法

1983 年, Kirkpatric 等将热力学中的退火思想引入组合优化领域, 提出了一种解大规模组合优化问题, 特别是 N– P 完全组合优化问题的有效近似算法——模拟退火算法[2,3]。它源于对固体退火过程的模拟, 使算法在多项式时间内给出一个近似最优解。

收稿日期: 2004– 12– 27
作者简介: 林 鑫(1983—), 男, 安徽怀远人, 硕士研究生, 研究方向为算法、模式识别与图像处理。

模拟退火算法是模仿固体物质的退火过程。高温物质降温时其内能随之下降, 如果降温过程充分缓慢, 则在降温过程中物质体系始终处于平衡状态, 反之降温太快, 则降到同一低温时会保持内能。模拟退火算法模拟退火过程寻求最优的方法, 过程大致如下:

1) 随机给定初始状 x, 选择合适的退火策略, 给定初始温度足够高。

2) 产生新解: 在 x 的领域选取 x', 并计算 $\Delta E = E(x') - E(x)$, $E(x)$ 为目标函数。

3) 如果 $\Delta E < 0$(或 $\Delta E > 0$), 则接受 x' 为新的状态, 否则以概率 $P = \exp(-\Delta E/t)$ 接受 x' 为新的状态, 以 $1-P$ 仍在原来状态 x。

4) 重复第 2、3 步直至系统达到平衡状态(实际上重复到预先给定的次数就行)。

5) 按第 1 步给定的退火策略下降 t, 重复 2~ 4 步, 直到 $t = 0$ 或某一特定的温度。

3 模型求解

3.1 贪婪算法

对于 0- 1 背包问题来说, 贪婪的策略有常用的 3 种。每种贪婪策略都采用多步过程来完成背包的装入。在每一步过程中利用贪婪准则选择一个物品装入背包。

第一种贪婪准则为: 从剩余的物品中, 选出可以装入背包的价值最大的物品, 利用这种规则, 价值最大的物品首先被装入(假设有足够容量), 然后是下一个价值最大的物品, 如此继续下去。

第二种准则是指从剩下的物品中选择可装入背包的重量最小的物品, 并且如此继续下去直到不能满足条件为止。

第三种是价值密度(价值重量比 $\frac{p_i}{w_i}$) 贪婪算法, 这种选择准则为: 从剩余物品中选择可装入包的 $\frac{p_i}{w_i}$ 值最大的物品, 这也是文中所选择的贪婪策略, 因为它是一个直觉上近似的解。

算法过程描述如下:

1) 对于给定的物品, 分别求出价值密度(价值重量比) $r_i = \frac{p_i}{w_i}$, $i = 1, \cdots, n$。

2) 忽略先前的物品排序, 按照价值密度, 进行非升序排列:

$$\frac{p(1)}{w(1)} \geqslant \frac{p(2)}{w(2)} \geqslant \cdots \geqslant \frac{p(n)}{w(n)}$$

3) 重复以下步骤, 直到不满足条件为止: 对于当前的物品, 如果重量小于包中剩余的容量, 则放入, 并置物品标志为 1(表明被选中), 否则就停止。

算法主要时间是在于将各种物品按价值密度大小排序, 文中利用 MATLAB 软件中的 SORTROWS 函数进行排序, 此函数是基于快速排序(quicksort) 实现的, 因此, 算法的时间复杂度为 $O(n\log n)$。

3.2 启发式贪婪算法

算法描述过程如下[4]:

1)、2) 两步和贪婪算法的过程是一样的, 这里就不再重复。

3) 在 $\{1, 2, \cdots, n\}$ 里取一个子集 S, 其中的元素个数小于等于 k(文中 $k = 2$), 计算其重量和价值, 如果重量小于背包中的容量, 则在剩下的物品中, 按价值密度从大到小装入包中, 直到背包不能装下物品为止, 算出背包中的总价值和对应的解。否则转 4)。

4) 如果子集还没有取完, 转 3)。

5) 在所有的解中, 找出最优的解, 即是算法的最终解。

由于这里 k 取的是 2, 所以此算法的时间复杂度是 $O(n^3)$, 是一个多项式时间。

3.3 模拟退火算法求解

采用模拟退火算法求解的描述如下:

1) 解空间。

$$S = \left\{ (x(1), \cdots, x(n)) \mid \sum_{i=1}^{n} w(i)x(i) \leqslant C,\ x(i) \in \{0, 1\} \right\}$$

由于模拟退火算法已经证明[5]: 最后的最优解并不太依赖初始解的选取, 所以在这里可任选一初始解, 文中初始解为 $(0, 0, \cdots, 0)_{1\times n}$。

2) 目标函数。

最大价值的目标函数为: $\max f = \sum_{i=1}^{n} p(i)x(i)$

$$\text{s.t.} \sum_{i=1}^{n} w(i)x(i) \leqslant C,\ x(i) \in \{0, 1\},\ i = 1, \cdots, n$$

3) 新解的产生。

随机选取物品, 若 i 不在背包中, 则将其直接放入背包中, 或同时从背包中随机取出另一物品 j; 若 i 已在背包中, 则将其取出, 并同时随机装入另一物品 j。

4) 背包的价值差和重量差。

根据上述新解产生的三种可能, 相应的背包价值差为

$$\Delta f = \begin{cases} c(i) & \text{将物品 } i \text{ 直接装入} \\ c(i) - c(j) & \text{将物品 } i \text{ 装入且} j \text{ 取出} \\ c(j) - c(i) & \text{将物品 } i \text{ 取出且} j \text{ 装入} \end{cases}$$

相应的背包重量差为

$$\Delta m = \begin{cases} w(i) & \text{将物品 } i \text{ 直接装入} \\ w(i) - w(j) & \text{将物品 } i \text{ 装入且} j \text{ 取出} \\ w(j) - w(i) & \text{将物品 } i \text{ 取出且} j \text{ 装入} \end{cases}$$

其中 Δm 为当前状态下背包重量 m 的增量。

5) 接受准则。

由于 0- 1 背包问题是个有约束的最优化问题, 所以文中采用的是扩充了的 Metropolis 准则。

$$P=\begin{cases}0 & m+\Delta m>M\\ 1 & m+\Delta m\leqslant M \text{ 且 } \Delta f>0\\ \exp(\Delta f/t) & \text{其它情况}\end{cases}$$

其中 t 为温度控制参数。

3.4 仿真模拟

模拟退火算法的控制参数的设置:

a) 控制参数 t 的初值 t_0 为 500。

b) 控制参数 t 的衰减函数为 $t_{k+1}=\alpha * t_k$, $k=0,1,\cdots$, 取 $\alpha=0.95$。

c) 控制参数 t 的终止准则: 让 tf 的值小于某个充分小的正数 δ, 取 $\delta=0.00001$。

d) Mapkob 链的长度 L_k, 取等长为 100* n, n 为问题的规模。

运行环境: PC 系列机, 操作系统为 WINDOWS2000, CPU 为赛扬 800, 内存为 kingstone128M, 编程软件为 MATLAB6.5。试验中所采用的数据均为随机产生, 其中 P_i 和 W_i 在 1 ~ 100 之间的整数内随机产生, 背包容量 $C=\left[0.8*\sum_{i=1}^{n}W_i\right]$。

模拟结果如表 1 所示。

表 1　三种方法的仿真模拟

问题规模(n)	贪婪算法	启发式贪婪算法	模拟退火算法
	(f_max, time)		
10	(290, 0.0200)	(295, 0.0100)	(295, 0.0500)
20	(1018, 0.1300)	(1024, 0.0900)	(1024, 0.2710)
100	(4506, 0.3100)	(4520, 9.1200)	(4515, 1.2920)
300	(14427, 0.0300)	(14536, 223.2600)	(14427, 3.6850)
1000	(45526, 0.3100)	(48926, 7406.3500)	(47804, 13.5190)

其中, f_max 为目标函数的值, time 为算法运行的时间, 单位为秒(s), 模拟退火算法中的 CPU 时间为模拟 5 次的平均时间。

4　算法分析评价

贪婪算法在运算时间上要比模拟退火算法少, 但是所得解的质量就相对差。不过贪婪算法在时间 $O(n\log n)$ 内获得不错的性能, 在随机模拟 100 个 0- 1 背包问题中, 得到 39 个最优解, 97 个解与最优解相差 9%, 所有 100 个问题的解与最优解相差均在 24% 之内。

启发式贪婪算法在时间 $O(n^3)$ 内, 获得解的质量和模拟退火算法相差不大, 但是随着问题规模 n 的增大, 这个多项式时间算法丧失了可行性。从上面仿真模拟的结果可以看到, 在问题规模 n 达到 1000 时, 启发式贪婪算的 CPU 时间是模拟退火算法的 549 倍。

模拟退火算法进行求解比用最优化方法求解时有更高的可靠性。在问题规模 n 增大过程中, SAA 的解和 CPU 时间却越来越稳定, 且不受初始解影响。而且, SAA 不容易陷入局部最优解上。模拟退火算法试验性能的关键是冷却进度表参数的选择, 往往不容易选出最优的参数配置, 要根据实际的问题, 尽可能地选择好控制参数的配置, 以达到解的质量和 CPU 时间相对最优, 在文中参数的设置控制下, 对文献[6] 中的例 2, 用模拟退火算法所得的最优解就优于其用蚂蚁优化算法得到的最优解, CPU 时间也比其少。

参考文献:

[1] Cormen T H, Leiserson C E. Introduction to Algorithms[M]. Massachusetts: The MIT Press, 2002.

[2] 陈华根, 吴键生. 模拟退火算法机理研究[J]. 同济大学学报(自然科学版), 2004, 32(6): 802- 805.

[3] Grosan, Crina. Improving the performance of evolutionary algorithms for the multiobjective 0/1 knapsack problem using ε- dominance[M]. London: Institute of Electrical and Electronics Engineers Inc, 2004.

[4] 王晓东. 算法设计与分析[M]. 北京: 清华大学出版社, 2003.

[5] Sachs L. Applied statistics: A handbook of techniques[M]. Berlin: Springer Verlag, 1984.

[6] 马　良. 背包问题的蚂蚁优化算法[J]. 计算机应用, 2001, 21(8): 4- 5.

(上接第 40 页)

把文本流转化为一个内部结构可能高度复杂的数据对象。XML 化模式可以提高应用程序的灵活性、健壮性和可扩展性, XML 化过程只改变源数据的样式或格式, 而不改变源数据的意义, 对包括电子商务应用在内的许多 Internet 应用来说, XML 是理想的信息表示和交换格式。HTML 深刻地影响了 Internet 的使用模式, 把 Internet 从一个信息传输媒体变成了全球最大的信息库; 而 XML 将进一步把 Internet 变成一个全球性的无缝的应用平台。

参考文献:

[1] Pitts N. XML 技术内幕[M]. 徐晓梅, 龚志翔, 晓　云等, 译. 北京: 机械工业出版社, 2002.

[2] 蔡翠平. 从 HTML 到 XML[M]. 北京: 北京交通大学出版社, 2001.

[3] Deitel H M, Deitel P J. XML 编程技术大全[M]. 北京: 清华大学出版社, 2002.

[4] 沈兆阳. Java 之 XML 与资料库[M]. 北京: 清华大学出版社, 2002.

[5] Mclaughlin B. Java and Xml[M]. 孙照林, 汪　东, 王　鹏译. 北京: 中国电力出版社, 2001.