一种求解 0-1 背包问题的算法

王秋芬[1],梁道雷[2,3]
(1. 南阳理工学院 计算机与信息工程学院,河南 南阳 473004;
2. 华东师范大学 计算机系,上海 200062;
3. 浙江理工大学 理学院,浙江 杭州 310018)

摘 要:文中针对各种智能搜索算法可能找不到问题的最优解、出现局部收敛,而动态规划、回溯法、分支限界法时间复杂度又比较高的缺点,分析了 0-1 背包问题的数学模型,刻划了最优解的结构特征,建立了求最优值的递归关系式。进一步分析递归关系式的函数特征,提出了一种求解 0-1 背包问题的确定性算法,并用 C++程序设计语言编码实现,该算法的时间复杂度为 $O(\min\{nW,2^n\})$。对三组不同规模的数据进行实验,算法运行的结果表明,该算法实际效率较高且总能得到该问题的最优解。

关键词:数学模型;最优子结构;递归关系;跳跃点;算法
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2013)01-0123-05
doi:10.3969/j.issn.1673-629X.2013.01.031

An Algorithm of Solving 0-1 Knapsack Problem

WANG Qiu-fen[1], LIANG Dao-lei[2,3]
(1. School of Computer and Information Engineering, Nanyang Institute of Technology, Nanyang 473004, China;
2. Computer Department, East China Normal University, Shanghai 200062, China;
3. School of Science, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: Intelligent search algorithm may not find the optimal solution of the problem and may exit the phenomenon of the local convergence. The time complexity of dynamic programming, backtracking, branch and bound method is relatively high. In order to solve these problems, analyze the mathematical model of the 0-1 knapsack problem, and discuss the structure characteristics of the optimal solution. At the same time, it establishes the recurrence relation which is used to solve the optimal value of the 0-1 knapsack problem. According to the recurrence relation, put forward an algorithm to solve the 0-1 knapsack problem and code the algorithm by using C++. Its time complexity is $O(\min\{nW,2^n\})$. Three groups of different size data are inputted in the algorithm. Their output results show that the algorithm is high efficiency and can always get the optimal solution of the problem.

Key words: mathematical model; optimal substructure; recursive relation; jump points; algorithm

0 引 言

0 - 1 背包问题是指给定 n 个物品,每个物品均有自己的价值 vi 和重量 wi ($i = 1,2,\cdots,n$),再给定一个背包,其容量为 W。要求从 n 个物品中选出一部分物品装入背包,这部分物品的重量之和不超过背包的容量,且价值之和最大。单个物品要么装入,要么不装入。很多问题都可以抽象成该问题模型,如配载问题、物资调运[1] 问题等,因此研究该问题具有较高的实际应用价值。

目前,解决 0 - 1 背包问题的方法有很多,主要有动态规划法[2]、回溯法、分支限界法、遗传算法[3~5]、粒子群算法[6]、人工鱼群算法[7]、蚁群算法[8]、模拟退火算法[9]、蜂群算法[10]、禁忌搜索算法[11] 等。其中动态规划、回溯法、分支限界法时间复杂性比较高,计算智能算法可能出现局部收敛,不一定能找出问题的最优解。文中在动态规划法的基础上进行了改进,提出一种求解 0 - 1 背包问题的算法,该算法每一次执行总能得到问题的最优解,是确定性算法,算法的时间复杂性最坏可能为 $O(2^n)$。

1 0-1 背包问题的数学模型

根据问题描述,每个物品只有两种状态——装或

收稿日期:2012-05-03;**修回日期**:2012-08-10
基金项目:国家自然科学基金资助项目(90818013);华东师范大学 211 重点项目(521B0108);浙江理工大学基金项目(yb07002)
作者简介:王秋芬(1978-),女,河南濮阳人,硕士,讲师,CCF 会员,研究方向为计算机软件与理论、算法理论。

不装。故用 n 维 0 - 1 向量$(x_1, x_2, \cdots, x_n)$表示 n 个物品的状态[12,13],其中,第 i 个物品不装入背包用“0”表示,即:$x_i = 0$;第 i 个物品装入背包用“1”表示,即:$x_i = 1$。由此,设计出如下的约束条件和目标函数。

$$\text{约束条件:}\begin{cases}\sum_{i=1}^{n} w_i x_i \leqslant W \\ x_i \in \{0,1\},(1 \leqslant i \leqslant n)\end{cases}$$

$$\text{目标函数:}\max \sum_{i=1}^{n} v_i x_i \text{。}$$

2 分析最优解的性质,刻划最优解的结构

设 n 维 0 - 1 向量$(x_1, x_2, \cdots, x_n)$是该问题的一个最优解,则$(x_2, \cdots, x_n)$是式(1)的一个最优解。

$$\text{约束条件:}\begin{cases}\sum_{i=2}^{n} w_i x_i \leqslant W - w_1 x_1 \\ x_i \in \{0,1\},(2 \leqslant i \leqslant n)\end{cases}$$

$$\text{目标函数:}\max \sum_{i=2}^{n} v_i x_i \tag{1}$$

进而$(x_3, \cdots, x_n)$是式(2)的一个最优解。

$$\text{约束条件:}\begin{cases}\sum_{i=3}^{n} w_i x_i \leqslant W - w_1 x_1 - w_2 x_2 \\ x_i \in \{0,1\},(3 \leqslant i \leqslant n)\end{cases}$$

$$\text{目标函数:}\max \sum_{i=3}^{n} v_i x_i \tag{2}$$

依此类推,假设$(x_k, \cdots, x_n)$是式(3)的一个最优解。

约束条件:

$$\begin{cases}\sum_{i=k}^{n} w_i x_i \leqslant W - w_1 x_1 - w_2 x_2 - \cdots\cdots - w_{k-1} x_{k-1} \\ x_i \in \{0,1\},(k \leqslant i \leqslant n)\end{cases}$$

$$\text{目标函数:}\max \sum_{i=k}^{n} v_i x_i \tag{3}$$

其中,$k = 1,2,3,\cdots,n$。

故:各级子问题及其最优解可以一般性地描述为:

$$\text{约束条件:}\begin{cases}\sum_{i=m}^{n} w_i x_i \leqslant j \\ x_i \in \{0,1\},(m \leqslant i \leqslant n)\end{cases}$$

$$\text{目标函数:}\max \sum_{i=m}^{n} v_i x_i \tag{4}$$

$(x_m, x_{m+1}, \cdots, x_n), 1 \leqslant m \leqslant n, 0 \leqslant j \leqslant W$。

假设$(x_m, x_{m+1}, \cdots, x_n)$是式(4)的一个最优解,则$(x_{m+1}, \cdots, x_n)$是式(5)的一个最优解。

$$\text{约束条件:}\begin{cases}\sum_{i=m+1}^{n} w_i x_i \leqslant j - w_m x_m \\ x_i \in \{0,1\},(m+1 \leqslant i \leqslant n)\end{cases}$$

$$\text{目标函数:}\max \sum_{i=m+1}^{n} v_i x_i \tag{5}$$

证明:(反证法)设$(x_{m+1}, \cdots, x_n)$不是子问题(5)的一个最优解,而$(z_{m+1}, \cdots, z_n)$是子问题(5)的一个最优解,则最优解向量$(z_{m+1}, \cdots, z_n)$装入背包的价值大于向量$(x_{m+1}, \cdots, x_n)$装入背包的价值,即:

$$\sum_{k=m+1}^{n} v_k z_k > \sum_{i=m+1}^{n} v_i x_i \tag{6}$$

又因为最优解向量$(z_{m+1}, \cdots, z_n)$满足约束条件:$\sum_{k=m+1}^{n} w_k z_k \leqslant j - w_m x_m$,即 $w_m x_m + \sum_{k=m+1}^{n} w_k z_k \leqslant j$,这说明$(x_m, z_{m+1}, \cdots, z_n)$是问题(5)的一个可行解。此时,在式(6)的两边同时加上 $v_m x_m$,得不等式 $v_m x_m + \sum_{k=m+1}^{n} v_k z_k > v_m x_m + \sum_{i=m+1}^{n} v_i x_i = \sum_{i=m}^{n} v_i x_i$,这说明$(x_m, x_{m+1}, \cdots, x_n)$不是(4)的最优解,与前提矛盾。故$(x_{m+1}, \cdots, x_n)$是子问题(5)的一个最优解,该问题的最优解具有最优子结构性质,证毕。

3 最优值递归关系式的建立

令 $M[m][j]$ 表示式(4)的最优值,$M[m][j] = \max \sum_{i=m}^{n} v_i x_i$,则 $M[m+1][j - w_m x_m] = \max \sum_{i=m+1}^{n} v_i x_i$。

如果 $j < w_m$,第 m 个物品肯定不能装入背包,$x_m = 0$,此时,问题(4)及其子问题(5)的最优值相同,即:$M[m][j] = M[m+1][j]$;如果 $j \geqslant w_m$,第 m 个物品能够装入背包。如果第 m 个物品不装入背包,即 $x_m = 0$,则 $M[m][j] = M[m+1][j]$;如果第 m 个物品装入背包,$x_m = 1$,则 $M[m][j] = M[m+1][j - w_m x_m] + v_m = M[m+1][j - w_m] + v_m$。可见当 $j \geqslant w_m$ 时,$M[m][j]$应取二者的最大值。最小子问题的最优值是递归的边界条件,当 $m = n+1$ 时表示最小规模的子问题,问题规模为 0。

由此可得最优值的递归定义式为:

$$M[m][j] = \begin{cases} 0 & m = n+1 \\ M[m+1][j] & m \leqslant n, j < w_m \\ \max\{M[m+1][j], M[m+1][j - w_m] + v_m\} & m \leqslant n, j \geqslant w_m \end{cases}$$

由式 $M[m][j]$ 的递归关系式可证明,对每一个确定的 $m(1 \leqslant m \leqslant n)$,$M[m][j]$ 是关于变量 j 的阶梯状单调不减函数,这一类函数的特征可用跳跃点来描述,在直角坐标系中,跳跃点的横坐标为重量,纵坐标为价值,如图 1 所示。

该类函数只要确定了跳跃点,则函数在直角坐标系中的图像便唯一确定。根据函数这一特征,该问题的算法设计如下:

(a)设计点类型的数据结构 p 存储跳跃点。对 $\forall m(1 \leqslant m \leqslant n$,且 m 是整数$)$,$p[m]$ 用来存储

$M[m][j]$ 的全部跳跃点,并将 $p[m]$ 中的跳跃点按横坐标(重量) 升序排列,则其对应的纵坐标(价值) 也是升序排列。

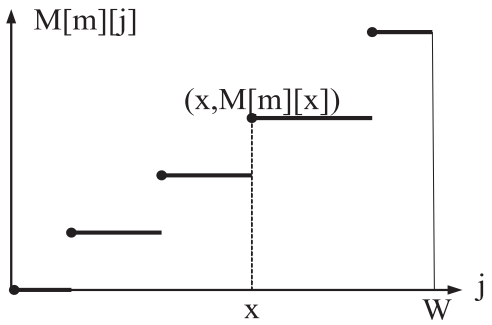

图 1　阶梯状单调不减函数特征示意图

(b)$p[m](1 \leqslant m \leqslant n)$ 可由子问题的跳跃点 $p[m+1]$ 得到。由 $M[m][j]$ 的递归关系式可知,问题(4) 的跳跃点要么由其子问题不装入物品 m 得到,要么由其子问题装入物品 m 得到。故:

$$p[m] = p[m+1] \cup (p[m+1] \oplus (w_m, v_m)) \quad (7)$$

另外,式(7) 得到的 $p[m]$ 中可能存在一些重量增加、价值反而下降的跳跃点,或重量超过背包容量的跳跃点,根据此类函数的特征和 0 - 1 背包问题的约束条件,将此类跳跃点从 $p[m]$ 中删除。初始时 $p[n+1] = \{(0,0)\}$,令 $q[m+1] = \{(s,t) \mid$ 当且仅当 $w_m \leqslant s \leqslant W$ 且 $(s - w_m, t - v_m) \in p[m+1]\}$,则 $p[m] = p[m+1] \cup q[m+1]$ 并清除重量增加、价值反而下降的跳跃点。

(c) 在 $p[1]$ 的跳跃点中,重量最大的跳跃点描述了问题的最优值及与该最优值相对应的装入背包的物品总重量。

(d) 构造最优解。

首先,令 m 初始值为 1,j 初始值为 $p[1]$ 中最大的横坐标,k 初始值为 $p[1]$ 中最大的纵坐标;然后检查 $p[m+1]$ 中的所有点跳跃点 (w,v),如果 $w + w_m$ 等于 j 并且 $v + v_m$ 等于 k,则 x_m 赋值为 1,j 赋值为 w,k 赋值为 v;否则 x_m 赋值为 0。m ++。重复检查,直到 m 等于 $n+1$ 为止。

4　算法设计

用一维数组 $x[]$ 存放 0 - 1 背包问题的最优解。求解步骤如下:

步骤 1:令 $p[n+1] = \{(0,0)\}$;

步骤 2:令 $m = n$;

步骤 3:$p[m] = p[m+1] \cup q[m+1]$,然后从 $p[m]$ 中清除重量增加、价值反而下降的跳跃点;

步骤 4:如果 $m > 1$,则 m --,转步骤 3;否则继续一下步骤;

步骤 5:令 $m = 1$,$j = p[1]$ 中的最大重量,$k = p[1]$ 中的最大价值;

步骤 6:检查 $p[m+1]$ 中的所有点跳跃点 (w,v),如果 $w + w_m$ 等于 j 并且 $v + v_m$ 等于 k,则 $x_m = 1$;$j = w$;$k = v$;否则 $x_m = 0$;

步骤 7:如果 $m < n$,则 m ++,转步骤 6;

步骤 8:输出数组 $x[]$ 的值即为最优解。

5　算法实现

按照算法设计的步骤,用 C++程序设计语言编程实现,具体代码如下:

```
floatGKnapSack(int n,float W,float * v,float * w,float *p
[],int * x)
{
int head[n+2];
head[n+1]=0;
p[0][0]=0;
p[0][1]=0;
int left=0,right=0;
int next=1; head[n]=1;//初始化完毕
for(int i=n;i>=1;i--)
{
    int k=left;
    for(int j=left;j<=right;j++)
    {
        f(p[j][0]+w[i]>W)  break;
        float y=p[j][0]+w[i];
        float m=p[j][1]+v[i];
while(k<=right&&(p[k][0]<y))
{
p[next][0]=p[k][0]; p[next++][1]=p[k++][1];
  }
//两个 if 判断新产生的点能否加入 p 中
        if(k<=right && p[k][0]==y)
        {
if(m<p[k][1]) m=p[k][1];k++;
}
        if(m>p[next-1][1])
       {
p[next][0]=y;p[next++][1]=m;
        }
//取出新产生的点可以支配的点
    while(k<=right&&p[k][1]<=p[next-1][1])k++;
}
while(k<=right)
{
p[next][0]=p[k][0];p[next++][1]=p[k++][1];
     }
     left=right+1;
     right=next-1;
```

```
        head[i-1]=next;
    }
    Traceback (n,w,v,p,head,x);
    cout<<"最优解为:x=(";
    for(i=1;i<=n+1;i++)
    cout<<x[i]<<",";
    cout<<")";
    return p[next-1][1];
    }
    //回溯求最优解
    void Traceback(int n,float * w,float * v,float * p[],int *
head,int * x)
    {
        float j=p[head[0]-1][0],m=p[head[0]-1][1];
        for(int i=1;i<=n;i++)
      {  x[i]=0;
       for(int k=head[i+1];k<=head[i]-1;k++)
          if(p[k][0]+w[i]==j && p[k][1]+v[i]==m)
        {
                x[i]=1;
              j=p[k][0];
              m=p[k][1];
             break;
    }
      }
    }
```

6 仿真实验

实例 1(参考文献 4 中的实例)

n = 10;

w = {95,4,60,32,23,72,80,62,65,46};

v = {55,10,47,5,4,50,8,61,85,87};

W = 269。

算法的运行结果如图 2 所示。

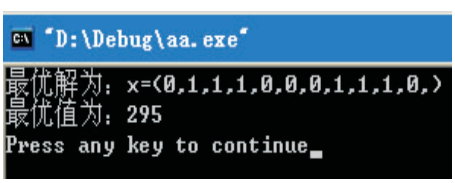

图 2 实例 1 的结果

实例 2(参考文献 2 中的实例)

n = 50;

w = {80,82,85,70,72,70,66,50,55,25,50,55,40,48,50,32,22,60,30,32,40,38,35,32,25,28,30,22,50,30,45,30,60,50,20,65,20,25,30,10,20,25,15,10,10,10,4,4,2,1};

v = {220,208,198,192,180,180,165,162,160,158,155,130,125,122,120,118,115,110,105,101,100,100,98,96,95,90,88,82,80,77,75,73,72,70,69,66,65,63,60,58,56,50,30,20,15,10,8,5,3,1};

W = 1000。

算法运行结果如图 3 所示。

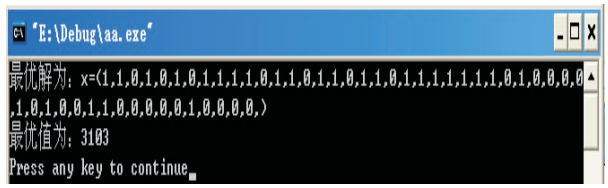

图 3 实例 2 的结果

实例 3(参考文献 3 中的实例)

n = 99;

w = {54,183,106,82,30,58,71,166,117,190,90,191,205,128,110,89,63,6,140,86,30,91,156,31,70,199,142,98,178,16,140,31,24,197,101,73,169,73,92,159,71,102,144,151,27,131,209,164,177,177,129,146,17,53,164,146,43,170,180,171,130,183,5,113,207,57,13,163,20,63,12,24,9,42,6,109,170,108,46,69,43,175,81,5,34,146,148,114,160,174,156,82,47,126,102,83,58,34,21};

v = {597,596,593,586,581,568,567,560,549,548,547,529,529,527,520,491,482,478,475,475,466,462,459,458,454,451,449,443,442,421,410,409,395,394,390,377,375,366,361,347,334,322,315,313,311,309,296,295,294,289,285,279,277,276,272,248,246,245,238,237,232,231,230,225,192,184,183,176,171,169,165,165,154,153,150,149,147,143,140,138,134,132,127,124,123,114,111,104,89,74,63,62,58,55,48,27,22,12,6};

W = 6718。

算法运行结果如图 4 所示。

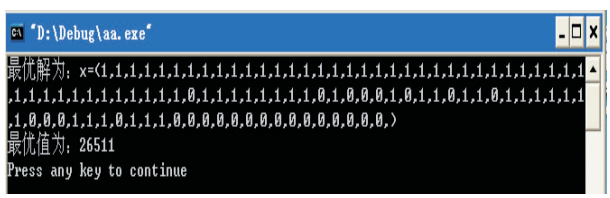

图 4 实例 3 的结果

7 算法分析

该算法的主要计算量在于计算跳跃点集 $p[m]$ ($1 \leqslant m \leqslant n$)。根据 $q[m+1]=p[m+1]\oplus(w_m,v_m)$ 和 $p[m]=p[m+1]\cup q[m+1]$ 并清除重量增加、价值反而下降的跳跃点可知：$q[m+1]$ 中的跳跃点个数等于 $p[m+1]$ 中的跳跃点个数;$p[m]$ 中的跳跃点个数最多是 $p[m+1]$ 中跳跃点个数的 2 倍。由此,$p[n+1]$ 中的跳跃点有 1 个,$p[n]$ 中的跳跃点个数不超过 2 个,$p[n-1]$ 中的跳跃点不超过 4 个,以此类推,$p[1]$ 中的

跳跃点不超过2^n个。故该算法计算跳跃点集$p[m]$ ($1 \leqslant m \leqslant n$) 所花费的计算时间最多为:$1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1$,即:$O(2^n)$。但是,如果物品的重量为正整数,则装入背包的物品总重量的增幅至少是 1,又知跳跃点的横坐标(重量) 不能超过背包的容量 W,因此$p[m]$ ($1 \leqslant m \leqslant n$) 中的跳跃点个数最多不超过 W 个,此时算法的时间复杂度为 $O(nW)$。综合两种情况,该算法的时间复杂度为 $O(\min\{nW, 2^n\})$。

8　结束语

文中在详细讨论 0-1 背包问题数学模型的基础上,根据该问题呈现的特殊性,提出了一种改进的确定性算法,算法的求解结果不会受任何不确定因素的影响,其时间复杂度在最坏情况下为 2^n,解决了各种智能计算方法存在的近似求解的问题。

是否能够融入其他的启发策略(如贪心策略),设计更为有效的确定性算法是我们进一步思考、努力的方向。

参考文献:

[1] Choi S, Park S, Kim H M. The Application of the 0-1 Knapsack Problem to the Load-shedding Problem in Microgrid Operation[J]. Communications in Computer and Information Science: Control and Automation, and Energy System Engineering, 2011, 256(1): 227-234.

[2] 王　乐,王世卿,张静乐. 基于 Matlab 的 0-1 背包问题的动态规划方法求解[J]. 计算机技术与发展,2006,16(4): 88-89.

[3] 田建立,晁学鹏. 求解 0-1 背包问题的混沌遗传算法[J]. 计算机应用研究,2011,28(8):2838-2839.

[4] 单小军,吴素萍. 基于贪心策略的遗传算法求解 0-1 背包问题[J]. 计算机应用与软件,2010,27(12):238-239.

[5] 赵新超,韩　宇,艾文宝. 求解背包问题的一种改进遗传算法[J]. 计算机工程与应用,2011,47(24):34-36.

[6] 柳　寅,马　良. 0-1 背包问题的模糊粒子群算法求解[J]. 计算机应用研究,2011,28(11):4026-4027.

[7] 厍向阳,朱命昊,赵亚敏. 求解 0/1 背包问题的改进人工鱼群算法研究[J]. 计算机工程与应用,2011,47(21):43-46.

[8] 王会颖,贾瑞玉,章义刚,等. 一种求解 0-1 背包问题的快速蚁群算法[J]. 计算机技术与发展,2007,17(1):104-107.

[9] 张盛意,蔡之华,占志刚. 基于改进模拟退火的遗传算法求解 0-1 背包问题[J]. 微电子学与计算机,2011,28(2):61-64.

[10] 樊小毛,马　良. 0-1 背包问题的蜂群优化算法[J]. 数学的实践与认识,2010,40(6):155-160.

[11] 廖飞雄,马　良,王　攀. 一种改进的禁忌搜索算法求解背包问题[J]. 计算机应用与软件,2009,26(3):131-133.

[12] Kakimura N, Makino K, Seimi K. Computing Knapsack Solutions with Cardinality Robustness[J]. Lecture Notes in Computer Science: Algorithms and Computation, 2011, 7074: 693-702.

[13] Escudero L F, Martello S, Toth P. On tightening 0-1 programs based on extensions of pure 0-1 knapsack and subset-sum problems[J]. Annals of Operations Research, 1998, 81: 379-404.

(上接第 122 页)

[2] Ye W, Heidemann J, Estrin D. An energy-efficient MAC protocol for wireless sensor networks[C]//Proceeding - IEEE INFOCOM. New York, NY, United States: [s. n.], 2002: 1567-1576.

[3] Thaskani S, Kumar K V, Murthy G R. Mobility tolerant TDMA based MAC protocol for WSN[C]//2011 IEEE Symposium on Computers & Informatics (ISCI). [s. l.]: [s. n.], 2011: 515-519.

[4] 刘子京,裴文江. 基于 ZigBee 协议的无线传感器网络研究[J]. 计算机技术与发展,2009,19(5):192-194.

[5] 熊　磊,董奎勇,钱　炜,等. 基于 ZigBee 的无线网络系统的设计与实现[J]. 计算机技术与发展,2009,19(4):242-245.

[6] 徐敬东,赖锡盛. TinyOS 2.0 在 CC2430 上的移植[J]. 计算机工程,2011,37(2):256-257.

[7] Gay D, Levis P, Culler D. Software design patterns for TinyOS[J]. ACM SIGPLAN Notices, 2005, 40(7): 40-49.

[8] 郭文生,刘奎安,桑　楠. TinyOS 集成开发环境的设计与实现[J]. 计算机应用,2008,28(5):1283-1286.

[9] 刘信新,邵明凯. 无线传感器网络操作系统 TinyOS 研究[J]. 计算机与数字工程,2007,35(7):66-68.

[10] 程　龙,杨　波. 无线传感器网络操作系统 TinyOS 的移植[J]. 计算机科学,2011,38(10):323-325.

[11] Elson J, Girod L, Estrin D. Fine-grained network time synchronization using reference broadcasts[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 147-163.

[12] Ganeriwal S, Kumar R, Srivastava M B. Timing-sync protocol for sensor networks[C]//Proceedings of the First International Conference on Embedded Networked Sensor Systems. Los Angeles, CA, United States: [s. n.], 2003: 138-149.

[13] Maroti M, Kusy B, Simon G, et al. The flooding time synchronization protocol[C]//Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems. Baltimore, MA, United States: [s. n.], 2004: 39-49.