运筹学与最优化

老师 张岩

姓名 傅豪

学号 160400423

一、题目：劳动力安排

戴维斯仪器公司再佐治亚州的亚特兰大有两家制造厂。每月的产品需求变化很大，使戴维斯公司很难排定劳动力计划表。最近，戴维斯公司开始雇佣由劳工无限公司提供的临时工。该公司专长于为亚特兰大地区的公司提供临时工。劳工无限公司提供三种不同合同的临时工，合同规定的雇佣时间和费用各不相同。三种合同选择如下：

选择	雇佣时间	费用（美元）
1	1 个月	2000
2	2 个月	4800
3	3 个月	7500

合同期越长，费用越高。这是因为找到愿意长时间工作的临时工对劳工无限公司更困难。

在接下来的 6 个月中，戴维斯公司计划需要的额外员工数如下：

月份	1	2	3	4	5	6
所需员工数	10	23	19	26	20	14

每个月戴维斯公司都可以根据自己的雇佣需要签署每种合同的员工。例如，如戴维斯公司 1 月份雇佣了 5 名符合第二项选择的员工，劳工无限公司将为戴维斯提供 5 名员工，均在 1、2 月份工作。这种情况下，戴维斯公司将支付 5*4800=24000 美元。由于进行中的某些合并谈判，戴维斯公司不希望任何临时工的合同签到 6 月份以后。

戴维斯公司有一个质量控制项目，并需要每名临时工在受雇佣的同时接受培训。即使以前曾在戴维斯公司工作过，该临时工也要接受培训。戴维斯公司估计每雇佣一名临时工，培训费为 875 美元。因此，如一名临时工被雇佣一个月，戴维斯公司将支付 875 美元的培训费用，但如果该员工签了 2 个或 3 个月，则不需要支付更多的培训费。

试确定一个雇佣计划，使达到目标计划的总花费最少。

二、模型建立

通过对问题的分析，我们可以发现，一月份雇佣临时工的计划会影响二月份雇佣临时工的计划，依次类推，上一个月的雇佣计划都会影响下一个月的雇佣计划。所以可以利用动态规划解决这个问题。

由于每一个月份需要的临时工人数是固定的，所以一月份的雇佣计划的各种情况我们都可以列举出来，但是六月份的雇佣计划我们却不能列举出来，因为它还取决于五月份的雇佣计划，只有一月份的雇佣计划不受其他情况影响，所以这个动态规划只能采用顺推的方式解决。

利用顺推的方式：我们得先确定阶段、状态、决策变量、状态转移方程和指标函数以及最优值函数。

阶段：这很好确定，一个月份就是一个阶段，六个月份共有六个阶段。

状态：可以将本月份的一类临时工人数、二类临时工人数、三类临时工人数作为一个向量，当成这个阶段的状态，根据每个月份需要的临时工总人数，我们可以轻松的枚举出各个阶段的所有状态。

决策变量：决策变量决定了本阶段的雇佣计划，即在上一阶段的已经存在的临时工的基础上，还需要雇佣多少临时工，这也是一个三维向量。

状态转移方程：在确定本阶段的某个状态的情况下，执行某一项决策，可以反向求出上一阶段的状态，这就是状态转移方程。在进行状态转移的时候，我将上一阶段的三类临时工全部当作下一阶段的二类临时工。比如：一月份雇佣计划是(5, 3, 2)，其中 3 个人是签约两个月，但是到下一个月的时候他只需要再工作一个月就行了，所以在下一个月的时候，我把他当作一类临时工处理，同样的有 2 个人是签约了三个月的，但是下一个月的时候他只需要再工作两个月，所以下一个月时，我把他当作二类临时工处理。

指标函数：指标函数取决于决策变量，通过决策变量确定的新的雇佣关系，计算出执行这项决策需要花费多少钱。

最优值函数：通过状态转移方程可以计算出某一状态下执行某项决策的上一个状态，通过计算上一个状态的最优值与执行这项决策花费的指标函数值的和，从中选出最小值作为这个阶段这个状态下的最优值。

三、程序及分析

确定好动态规划中的各个量之后，接下来就是编写程序（用 Java）。首先得编写一个保存三类临时工的向量类（Vector3d.java），如下：

Vector3d.java

```
package fuhao.design;
```

```java
/**
 * 三维向量.
 * 三个分量分别表示一类临时工个数、二类临时工个数和三类临时工个数
 * @author fuhao
 */
public class Vector3d {
    public Integer w1;
    public Integer w2;
    public Integer w3;

    public Vector3d() {
        this(0, 0, 0);
    }

    public Vector3d(int w1, int w2, int w3) {
        this.w1 = w1;
        this.w2 = w2;
        this.w3 = w3;
    }

    /**
     * 求三种临时工的总数量
     * @return
     */
    public int sum() {
        return this.w1 + this.w2 + this.w3;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((w1 == null) ? 0 : w1.hashCode());
        result = prime * result + ((w2 == null) ? 0 : w2.hashCode());
        result = prime * result + ((w3 == null) ? 0 : w3.hashCode());
        return result;
    }

    public String toString() {
        return "(" + this.w1 + ", " + this.w2 + ", " + this.w3 + ")";
    }
}
```

接下来的所有规划过程都放在 Dynamic.java 中完成：包括状态的定义及初始化、决策变

量的计算和最优值函数的计算以及最后的输出。

Dynamic.java

```
package fuhao.design;

public class Dynamic {
    public static final int M1 = 1000000;
    public static final int M2 = Integer.MAX_VALUE/10;

    /**
     * 每个月需要的临时工数量
     */
    private int requirement[] = {10, 23, 19, 26, 20, 14};

    /**
     * 每一阶段的初始状态，行索引表示阶段，列索引是顺序增长的
     */
    private Vector3d s[][] = new Vector3d[6][400];

    /**
     * 每一阶段状态的个数
     */
    private int[] sSize = new int[6];

    /**
     * 决策序列，是初始状态的函数，行索引表示阶段，列索引是顺序增长的
     */
    private Vector3d d[] = new Vector3d[10000];

    /**
     * 决策序列的长度
     */
    private int dSize;

    /**
     * 最优值函数，行索引表示阶段，列索引是这个阶段状态的 hash 值，同一阶段不同的状态会有不同的最优值
     */
    private int f[][] = new int[6][60000];

    /**
     * 指标函数，索引值是决策变量的 hash 值，不同的决策会产生不同的消耗
     */
    private int v[] = new int[60000];
```

```
public Dynamic() {
    init();
}

/**
 * 开始动态规划
 */
public void run() {
    // 第一阶段已经计算完毕，不需要进行规划，直接为后一阶段服务
    for(int i=1;i<6;i++) {
        for(int j=0;j<this.sSize[i];j++) {     // 遍历每一个状态
            Vector3d state = this.s[i][j];

            this.dSize = 0;
            // 进行决策
            for(int m=state.w1;m>=0;m--) {         // 对一类临时工进行决策
                for(int n=state.w2;n>=0;n--) {     // 对二类临时工进行决策
                    int k = state.w3;
                    this.d[dSize] = new Vector3d(m, n, k);
                    this.v[d[dSize].hashCode()] = this.spend(d[dSize]);
                    dSize++;
                }
            }

            // 计算最优值函数，从决策里面选一个最优的
            f[i][state.hashCode()] = min(i, state);
        }
    }

    for(int i=0;i<f[0].length;i++) {
        if(f[5][i] != M2) {
            System.out.println(f[5][i]);
        }
    }
}

/**
 * 初始化
 */
protected void init() {
    // 初始化最优值函数
    for(int i=0;i<this.f.length;i++) {
        for(int j=0;j<this.f[0].length;j++) {
```

```
                this.f[i][j] = M2;
            }
        }

        // 实例化决策序列
        for(int i=0;i<this.d.length;i++) {
            this.d[i] = new Vector3d();
        }

        // 实例化状态
        for(int i=0;i<this.s.length;i++) {
            for(int j=0;j<s[0].length;j++) {
                this.s[i][j] = new Vector3d();
            }
        }

        // 计算每一阶段的状态
        for(int i=0;i<6;i++) {
            this.sSize[i] = this.createList(i, this.requirement[i], this.s[i]);

            if(i==0) {            // 初始化第一阶段的最优值函数
                for(int j=0;j<this.sSize[0];j++) {
                    this.f[0][s[0][j].hashCode()] = this.v[s[0][j].hashCode()];
                    System.out.println("min_0 = " + f[0][s[0][j].hashCode()] + ", d
= " + s[0][j]);
                }
            }
        }
    }

    /**
     * 生成状态序列
     * @param i 阶段
     * @param require 该阶段还需要的临时工
     * @param v
     * @return
     */
    private int createList(int i, int require, Vector3d vec[]) {
        // 清空指标函数
        for(int k=0;k<this.v.length;k++) {
            this.v[k] = M1;
        }

        int size = 0;
```

```
            int w1 = require;
            int w2 = 0;
            int w3 = 0;

            vec[size] = new Vector3d(w1, w2, w3);
            this.v[vec[size].hashCode()] = this.spend(vec[size]);
            size++;

            while(w1 != 0 && i < 5) {
                w1--;
                w2 = require-w1;
                w3 = 0;
                vec[size] = new Vector3d(w1, w2, w3);
                this.v[vec[size].hashCode()] = this.spend(vec[size]);
                size++;

                while(w2 != 0 && i < 4) {
                    w2--;
                    w3++;
                    vec[size] = new Vector3d(w1, w2, w3);
                    this.v[vec[size].hashCode()] = this.spend(vec[size]);
                    size++;
                }
            }

            return size;
        }

        /**
         * 根据决策计算花费
         * @param v
         * @return
         */
        private int spend(Vector3d v) {
            int spend1 = v.w1 * 2000;          // 一类临时工
            int spend2 = v.w2 * 4800;          // 二类临时工
            int spend3 = v.w3 * 7500;          // 三类临时工
            int spend4 = v.sum() * 875;   // 培训

            return spend1 + spend2 + spend3 + spend4;
        }

        /**
         * 计算最优值函数
```

```
     * @param i
     * @return
     */
    private int min(int i, Vector3d s) {
        int minCost = Integer.MAX_VALUE;
        Vector3d vec = null;
        int copy = 0;

        for(int m=0;m<dSize;m++) {
            int w2 = s.w1-d[m].w1;
            int w3 = s.w2-d[m].w2;
            int w1 = this.requirement[i-1] - w2 - w3;
            Vector3d last = new Vector3d(w1, w2, w3);

            int cost = f[i-1][last.hashCode()] + v[d[m].hashCode()];
            if(cost < minCost) {
                minCost = cost;
                vec = d[m];
                copy = f[i-1][last.hashCode()];
            }
        }
        System.out.println("min_" + i + " = " + minCost + ", d = " + vec + ", min_" +
(i-1) + " = " + copy);
        return minCost;
    }

    public static void main(String[] args) {
        new Dynamic().run();
    }
}
```

先从变量开始分析：

• requirement[6：保存着每个阶段需要的总的临时工人数；

• s[6][400]是一个二维数组，每一个元素都是一个 Vector3d 对象，保存的是各个阶段的状态，其中行索引表示的是阶段，列索引顺序增长（每个阶段的状态的个数不确定，但是经过程序确认不会超过 400）；

• sSize[6]：用来确定每个阶段状态的个数，为之后遍历状态的时候少执行几次循环操作；

• d[10000]：决策序列，数组的每一个元素都是 Vector3d 对象，它是状态的函数，同一状态可以有很多决策，所有的决策构成决策序列（经程序测试，决策序列的长度不

会超过 10000），各个阶段的各个状态的决策序列都不相同；

• dSize：决策序列的长度；

• v[60000]：指标函数，在平时的练习中，由于状态都是由一个量确定，所以求指标函数的索引比较容易，但这里状态是由三个量（一类临时工人数、二类临时工人数，三类临时工人数）确定的，在这里，我覆写类中的 hashCode()方法，用这个方法的返回值作为指标函数的索引，这也是 v 数组长度为 60000 的原因，实际上数组中大多数的值是没有作用的。

• f[6][60000]：最优值函数，最优值函数是由上一阶段的最优值与当前决策决定的，与指标函数一样，它的列索引也是由一个 Vector3d 对象的 hash 值确定。

在实例化 Dynamic 类对象的时候，会对这些数组进行初始化：

• 最先初始化最优值函数 f，将它的所有的值先初始化成一个很大的数 M2，这个 M2 的作用类似于罚函数法中的惩罚项，因为 f 数组中的大多数值是没作用或者是无效的，所以当这些值被选中的时候，我让它很大，这要在求最小值的时候，这些值都会被淘汰；

• 再实例化决策序列 d 中的每一个元素，让它都为 0；

• 再实例化并计算每个阶段的状态序列（顺便将这些状态当作决策计算指标函数），由 createList()方法完成，这个方法执行的步骤是：最开始让一类临时工人数等于当前总需求人数，然后逐渐递减这个值，再递减一类临时工人数的时候，对二类临时工人数做与一类临时工同样的操作，这样就能枚举出所有的状态，但是需要注意的是公司希望临时工不要签约到 6 月以后，也就是说五月份的时候就不能签约三类临时工了，六月份的时候就不能签约二类临时工了，所以在递减循环中增加了两个判断；

• 初始化第一阶段的最优值函数，就等于这一阶段的指标函数。

初始化完成后就开始进行动态规划了：

动态规划是一个 for 循环，i 从 1 到 5，共 5 各阶段，因为第一阶段我们是通过初始化完成的。

在每一个阶段中，我们取出它的所有的状态，对每一个状态进行决策。假设是在 i=1 的时候，这个阶段总共需要 23 人，那么就会有一个状态是(20, 3, 0)，那么它的决策就是，20 个一类临时工有多少个是这一阶段雇佣的（其余的一类临时工都是来自于上一

阶段的二类临时工），3 个二类临时工有多少个是这一阶段雇佣的（其余的二类临时工都是来自于上一阶段的三类临时工），由于三类临时工只能是在这一阶段雇佣，所以这个状态有三类临时工，必然是这一阶段雇佣的。所以用两个循环就能列举完所有的决策，然后对每一个决策计算指标函数（用一个 spend()方法），当这个状态的所有决策都完成后，需要从其中挑出最优的决策，也就是最小花费（用一个 min()方法）。

最后一个阶段只有一个状态，所以最后那个状态的最优值函数就是整个劳工分配问题的最小花费。

四、程序运行结果以及最终的雇佣计划

这个题目是从百度文库中摘取的，他利用的是 Lingo 软件，用线性规划的方式解决的，看起来很麻烦也没怎么看懂。

（链接：https://wenku.baidu.com/view/cb3a676c1ed9ad51f01df212.html）

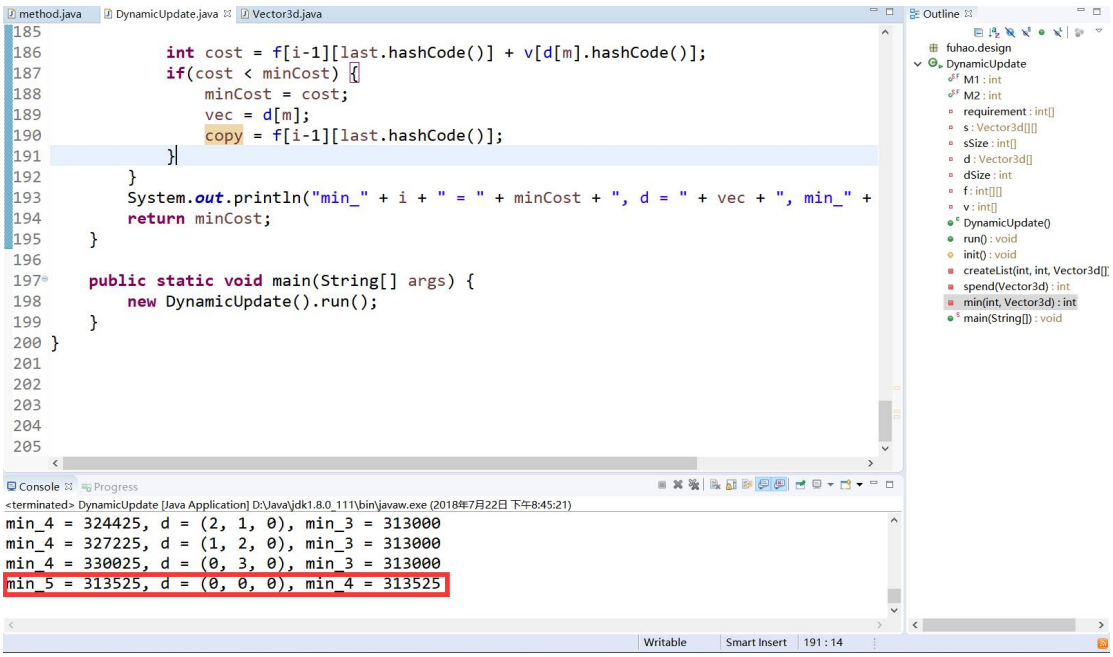
我的程序运行后得到最优值和他的最优值是相同的，也就是说程序没有错误。

合同二	3	0	0	0	0	0	3
合同三	7	12	0	14	0	0	33
新雇佣临时工总数	10	13	0	14	6	0	43

戴维斯公司每月应雇佣签署各种合同的临时工应如下安排：
1 月份雇佣合同二的临时工 3 人，合同三的临时工 7 人，共 10 人；
2 月份雇佣合同一的临时工 1 人，合同三的临时工 12 人，共 13 人；
4 月份雇佣合同三的临时工 14 人；
5 月份雇佣合同一的临时工 6 人；
3 月份和 6 月份没雇佣临时工。
如此可得到总花费最小为 313525 美元。

2．总结表如下：

合同选择	雇佣人数	合同费用	培训费用
合同一	7	14000	6125
合同二	3	14400	2625
合同三	33	247500	28875
合计	43	275900	37625



现在最重要的是获取每一阶段的最佳决策，来计算最佳的雇佣计划（每个月雇佣多少人，每类临时工雇佣几个）。

如果用程序来解决的话，必须保存每一项决策，然后从最后一阶段进行倒推，比较麻烦。我采用的是人为方式进行倒推，将每个阶段的所有状态的最优值函数和取最优值时的决策（包括进行这个决策时上一阶段的最优值）都进行输出（接近 1000 条数据），然后从最后一阶段开始进行倒推对比，找出每一阶段的最优决策。

例如：第 6 阶段（i=5）时，min_5 只有一个就是我们要求的最小值，d 表示这种情况下的决策，后面的 min_4 = 313525 表示执行最优决策时，上一阶段的最优值函数值为 313525，所以从上面找到 min_4 等于这个值的那一行，然后就再可以得到一个 min_3 的值，依此倒推，就可以推出每个阶段应该进行样的决策。

最终得到最优的安排计划为

月份	1	2	3	4	5	6
一类临时工	0	1	0	0	6	0
二类临时工	0	3	0	0	0	0
三类临时工	10	9	0	14	0	0

- 一月份雇佣三类临时工 10 人；
- 二月份雇佣一类临时工 1 人，二类临时工 3 人，三类临时工 9 人；
- 三月份不雇佣人；
- 四月份雇佣三类临时工 14 人；

• 五月份雇佣一类临时工 6 人；

• 六月份不雇佣人；

最小的花费为 313525 美元

可以看出，这个计划安排与文库给出的不一样，但是经过验证之后，这个计划安排的消耗也是 313525。文库中的计划安排经验证后消耗也是 313525。说明这个劳工分配问题的最优解并不只有一个，只是不同的解法得到的最优解不一样。

五、程序的不足

由于这个程序是对特定问题编写的，所以它并不适用于所有的安排情况。主要的局限性有两点：

• 阶段个数的限制（本题是 6 个月 6 个阶段），要是月份数不是 6 个月，本程序都无法解决；

• 每个月需要临时工的总人数，如果需要的总人数过大，在计算它的 hashCode 值时会超出 60000，引起数组索引超出范围异常。