第 11 卷第 2 期　　广 西 科 学 院 学 报　　Vol. 11, No. 2
1995 年 6 月　　Journal of the Guangxi Academy of Sciences　　Jun., 1995

人员配置日程安排算法的研究
Study on the Algorithm of Staff Scheduling

兰红星
Lan Hongxing

（广西计算中心　南宁市　530022）
(Computing centre of Guangxi, Nanning, 530022)

摘要　人员配置日程安排问题可以分为两部分来研究，即条件的表示和匹配。在这篇文章中，只讨论如何进行匹配而不论及条件表示。作者给出一个匹配算法，并且证明了根据这个算法可以得到条件最佳解。从而我们就可以把人员配置日程安排问题转化为排序处理问题。换言之，如果人员和部门可以排序处理，那么我们就可以得到最佳匹配结果。

关键词　日程安排　算法

Abstract　Staff scheduling can be thought as a problem of matching staff with division. The expression of conditions of staff may be important. Even so, it is convenient to study by dividing the problem into two parts, matching and expression. In this article we discussed the matching and noninvolvement in expression. We advanced an algorithm of scheduling for staff and proved that the conditional best result of matching could be got in it. Thus the problem of scheduling for staff can be changed into ordinary sorting processing. If staff and divisions can be sorted we can get the best of matching result.

Key words　Scheduling, Algorithm

1　引言

在要求经常调整部门人员组合的单位中，如何综合考虑各个部门的人员需求，以及各人员素质、技能、人员间的关系等因素，合理地调整人员的工作安排，这已经成为这类单位的业务管理中的一个重要问题。简单说来，也就是考虑单位中各部门的要求、个人的希望、人员之间的关系等条件后，如何合理作出人员在各部门轮换的计划的问题。

人员在各部门轮换配置的问题，是带有一定普遍意义的问题。在计算机技术迅猛发展的今天，利用计算机技术来解决这个问题，是很自然的想法。然而，在制作实用程序的时候，却发现解决这个问题并非易事。原因是：配置算法、条件和要求的表示、如何评价配置结果等

1995-03-16 收稿。

问题都未明确。在编制程序的时候，往往受到诸多条件的强烈束缚而无法找到解。为了解决这个问题，不少学者曾经进行过各种各样方法的探讨研究。例如，有利用了称为目标最优化的 goal programming 模型的，有应用了计算机科学理论中称为形式语言的古典理论的[1,2]。也有了关于医院里护理人员的轮值日程安排实用系统的报道[3]。

其实，人员的配置问题，可以作为组合和匹配问题来考虑。我们把配置算法看成是与条件的描述独立的一个问题，专门研究它。并且编制了实际的程序进行了模拟，取得了理想的结果。本文将介绍这个算法并给出它的证明。

2 配置算法

这个算法的基本思路如下。设有 m 个部门，n 个人选。首先，这 m 个部门都根据本部门对人员的需求，对 n 个人选排出一个优先顺序队列，称为“人选队列”（在此不考虑“人选队列”如何编排）。见图 1。

另一方面，全部 n 个人选也根据自己对部门的选择的优先顺序排出一个队列，称为“部门队列”，见图 2。

部门	人选队列
d_1	s_1，s_2……s_i……s_n
⋮	
d_j	s_x……
⋮	
d_m	s_a，s_b……s_x……s_y

图 1　人选队列的组成形式

人员	部门队列
s_1	d_1，d_2……d_j……d_m
⋮	
s_i	d_x……
⋮	
s_n	d_a，d_b……d_x……d_y

图 2　部门队列的组成形式

d_j 表示某个部门，s_i 表示某人选。

然后实施匹配操作。各部门相继取自己的“人选队列”上，排在最前面的人员。这时，如果这个人员没有被别的部门选中的话，这个部门就无条件地选中这个人员。所有部门都同样地，查看能否选中各自“人选队列”中排在前面的人员。如果某个部门所选的人员 s_i 已经被别的部门选中了（即两个部门争夺同一个人员），就按以下的办法处理：比较这两个部门在该人员 s_i 的“部门队列”中位置，出现在“部门队列”中靠前面的那个部门争得这个人员。争不到这个人员的部门，再试选自己的“人选队列”中的下一个人员。当然，这又可能再次发生“争夺”。只要发生“争夺”，就用以上算法处理。这个过程用递归方法是很有效的。全部的部门都轮过一遍后，把被选中的人员从各部门的“人选队列”中都删去，另外，把已经选够了人数的部门从各人员的“部门队列”中也都删去。再做下一轮的匹配操作。

以下是具体的算法：

Step1：D 是全部部门的集合。S 是人员集合。

If (D 是空集) then stop.

Step2：根据各部门的要求形成人选队列，优先级高的放在前面。

Step3：根据各人员的希望和要求，形成各人员的“部门队列”，优先级高的部门放在前面。

Step4：形成包含全部部门的集合 A。

直到 A 空为止，做

```
do choice (d).
  do need (s, d). /s 是部门 d 的“人选队列”中最前的人员. /
    If (s 未被任何部门选中) then
      d 选中 s.
      把 d 从 A 中删去.
    else  /已经有某部门 d' 选中了 s/
      以 s 的“部门队列”为基准，比较 d 和 d'.
      If (d' 在 d 之前) then
        把 d 的“人选队列”中的 s 删去.
        把 s 的“部门队列”中的 d 删去.
        do choice (d).
      else
        d 选中 s,
        把 d' 的“人选队列”中的 s 删去.
        把 s 的“部门队列”中的 d' 删去.
        do choice (d')
  end if
end if
```

Step4：把被选中的人员从 S 删去，把满足人数要求的那些部门从 D 中删去，返回 Step1。下面我们以图 3 为例，实施各步的操作作为说明。

部门	人选队列		
d_1	s_1	s_2	s_3
d_2	s_2	s_1	s_3
d_3	s_1	s_3	s_2

人员	部门队列		
s_1	d_2	d_1	d_3
s_2	d_3	d_2	d_1
s_3	d_1	d_3	d_2

图 3　例子 1

在这里，部门集合 A 的内容是 $[d_1, d_2, d_3]$。假设经 Step1 得到“人选队列”如上图左，经 Step2 得到“部门队列”如上图右。进行到 Step3，最先执行 choice (d_1)。因为部门 d_1 的第一人选是 s_1，所以执行 need (s_1, d_1)，这时 s_1 未曾被任何部门选中过，故部门 d_1 无条件地得到了 s_1。把 d_1 从部门集合 A 中除去。轮到部门 d_2，d_2 的第一人选是 s_2，d_2 也可以选中 s_2，把 d_2 也从部门集合 A 中除去。最后轮到 d_3。执行 choice (d_3)。因为 d_3 的第一人选也是 s_1，而 s_1 已经被部门 d_1 所选中，所以当执行到 need (s_3, d_3)，就发生了两个部门 d_1 和 d_3 争夺同一个人选 s_1 的情况。这时，就必须根据 s_1 的“部门队列”比较 d_1 和 d_3，比较的结果表明 s_1 愿意去部门 d_1 甚于去 d_3，所以 s_1 拒绝了 d_3 的“邀请”。把 s_1 从 d_3 的“人选队列”中删去，把 d_3 也

从 s_1 的“部门队列”中删去。再次执行 choice (d_3)。这时，因为 s_1 已被从 d_3 的“人选队列”中删去了，s_3 变成了 d_3 的第一人选，执行 need (s_1, d_3)，部门 d_3 就选中了 s_3。把 d_3 从集合 A 中删去，A 变为空集。一轮匹配完结，匹配结果是 { (d_1, s_1), (d_2, s_2), (d_3, s_3)}。

这是个简单的例子。我们再看一个较复杂的例子。

假设经过 Step1，Step2 后，“人选队列”和“部门队列”的形式如图 4 所示：

部门	人选队列
d_1	s_3 s_2 s_1
d_2	s_2 s_3 s_1
d_3	s_2 s_3 s_1

人员	部门队列
s_1	d_1 d_2 d_3
s_2	d_1 d_3 d_2
s_3	d_3 d_1 d_2

图 4　例子 2

执行 Step3 进行匹配，部门 d_1、d_2 都顺利地选中了各部门的第一人选 (d_1, s_3) (d_2, s_2)。轮到 d_3 时，执行 choice (d_3)，做 need (s_2, d_3) 时，发生了部门 d_3 和 d_2 争夺 s_2 的情况，根据人员 s_2 的“部门队列”比较 d_3 和 d_2，结果是 d_3 比 d_2 优先，d_3 把 s_2 从 d_2 夺过来了。d_2 失去了 s_2 后，把 s_2 从“人选队列”中删去，执行 choice (d_2)，做 need (s_3, d_2)，这又再次引起 d_2 和 d_1 对 s_3 的争夺，根据 s_3 的“部门队列”比较 d_2 和 d_1，结果是 s_3 拒绝了 d_2，这样 d_2 再次把 s_3 从自己的“人选队列”中删去，再做 need (s_1, d_2)，s_1 未曾被别的部门选中，所以 d_2 得到了 s_1。最后的匹配结果是 { (d_1, s_3), (d_2, s_1), (d_3, s_2)}。

通过例子可见，使用这个算法处理可以得到稳定的匹配结果。

假设部门数为 n，如果每个部门的“人选队列”上的人数都有 n 个以上，一轮匹配后，每个部门都可得到一个人员。然而，如果有某些部门“人选队列”上的人数不足 n 个，就有可能不能完成匹配操作。为了解决这个问题。我们可以用追加假想人员的办法。见图 5。

部门	人选队列
d_1	s_3 s_1
d_2	s_1 s_2 s_3
d_3	s_3

→

部门	人选队列
d_1	s_3 s_1 s_0 s_2
d_2	s_1 s_2 s_3 s_0
d_3	s_3 s_0 s_1 s_2

图 5　在人选队列中追加假想人员 s_0

人员	部门队列
s_1	d_1 d_2 d_3
s_2	d_3
s_3	d_1 d_2

→

人员	部门队列
s_1	d_1 d_2 d_3 d_0
s_2	d_3 d_0 d_2 d_3
s_3	d_1 d_2 d_0 d_3

图 6　　在部门队列中追加假想部门 d_0

同样地，在人员方面，如果有“部门队列”上不足 n 个部门的，也追加假想部门 d_0，见图 6。

在这个追加了假想人员 s_0 和假想部门 d_0 的例子中，根据我们的算法进行匹配，结果是 $\{(d_1, s_3), (d_2, s_1), (d_3, s_0)\}$。部门 d_3 得到 s_0，这意味着，部门 d_3 得不到所希望的人员。或者说，没有既满足 d_3 工作条件要求又满足人员希望的匹配。在实际中，这种情况是常有的。

假想部门 d_0 后面的部门的顺序是没有特别意义的。s_0 后面的人员顺序也无关要紧。不影响匹配结果，也不增加算法的复杂度。

3 证明

命题①　本算法是必定可停机的。

证明：在执行 need (s, d) 的时候，会发生两个部门 d 和 d′ 争夺同一个人员 s 的情况。但是，得不到 s 的部门，从自己的“人选队列”中删去了 s，人员 s 也从自己的“部门队列”中删去了该部门，所以绝对不会返回到相同的匹配尝试操作中去，故不会陷入无限死循环。

命题②　根据本算法是必定取得解的。

证明：设有 n 个部门，为了保证每个部门的“人选队列”都有 n 人以上，在“人选队列”中添加 s_0，在“部门队列”中添加 d_0。人员 s 一旦被某个部门选中，他只有可能往更好的部门移动，但不会出现哪个部门都不选取他的情况。而且，存在暂时选不到人的部门时，肯定存在未被任何部门选中的人员，所以，只要这些部门对剩余的这些人员继续执行 need 操作，最后，全部的部门都可以选到人员。

命题③　本算法的解是稳定的。

所谓稳定，是指任意的两个匹配对子之间，不存在部门和人员双方都同时能找到比当前组合更好的配对。

证明：假设某个匹配结果不稳定，也就是说，还有比已经得到的组合更好的匹配结果。例如，设 (d_1, s_1) (d_2, s_2) 是得到的匹配结果，现在我们假定人员 s_1 和部门 d_1 两方都对这个匹配不满，希望组合成 (d_1, s_2)。

按我们的算法，部门方面，从自己的“人选队列”的先头开始顺序地把各个人员作为候选人，而人员方面，是往各自的“部门队列”的排在前面的部门“移动”。根据以上的假设，因为 d_1 希望取 s_2 甚于取 s_1，所以，在 d_1 的“人员队列”中，s_2 排在 s_1 之前；同样地，s_2 也是希望 d_1 甚于 d_2，所以 s_2 的“部门队列”中 d_2 排在 d_1 之前（见图 7）。

部门	人选队列
⋮	
d_1	$\cdots s_2 \cdots s_1 \cdots$
⋮	

人员	部门队列
⋮	
s_1	$\cdots d_2 \cdots d_1 \cdots$
s_2	$\cdots d_1 \cdots d_1 \cdots$
⋮	

图 7

但是，匹配的结果却是 d_1 选中了 s_1。为什么呢？我们可以说是 d_1 在某个时刻曾向 s_2 发出过“邀请”，即执行过 need (s_2, d_1)，但是 s_2 拒绝了 d_1 的邀请。他拒绝 d_1 的原因肯定有比 d_1 更排在前面的某个部门 d_i 邀请过他（见图 7 右图）。这时，按本算法，s_2 最终无论如何都不会

被 d_1 选中的。也就是说，按上面的“人选队列”和“部门队列”，匹配得的结果为 (d_1, s_1) (d_2, s_2) 的话，d_1 和 s_2 是肯定不会双方同时希望放弃原来的组合而匹配为 (d_1, s_2) 的。故，解是稳定的。

命题④　按本算法取得的解对部门而言是最佳的解。

证明：假定在一轮匹配结果中，有两个对子 (d, s) (d′, s′)，又假设我们削除对子 (d, s)。也就是说，让两个部门 d 和 d′ 争夺同一个人员 s，并假设争夺的结果是 d 放弃了 s。这时，如果在 d′ 的“人选队列”中 s 排在 s′ 前面，(如图 8)，d′ 和 s 互相吸引，组合就变得不稳定。另一方面，如果在 d′ 的“人选队列”中，s′ 排在 s 的前面 (如图 9)。

部门	人选队列
⋮	
d′	…s…s′…
⋮	

图 8

部门	人选队列
⋮	
d′	…s′…s…
⋮	

图 9

为了配成 (d′, s)，就必须要删除对子 (d′, s′)。但是，根据 need 的定义，在删除 (d′, s′) 的操作之前，一定是删除了另一个对子 (d″, s″)。这就意味着，在稳定的解中，删去一个对子，就必定也要删去另一个对子。根据归纳法可知，这样的话被删去的对子就有无限多个。但是，这显然是不可能的。

故本算法取得匹配结果是部门最佳解。

4　结论

如果能够作成“部门队列”和“人选队列”，由命题②可知，肯定能得到解。另外，对部门方面来说，在做“人选队列”时，就可以完全不考虑别的部门的人员需求；同样地，各人员做“部门队列”时，也是不必考虑别人的希望和条件的，只须考虑自身的情况即可。这样，我们就能把制作“人选队列”和“部门队列”所需的条件表达问题分离出来了。也就是说表达问题能解决，能排出队列，就一定能得到匹配结果。并且根据命题③和④，匹配结果是没有矛盾的，匹配结果是部门最佳解。

参考文献

1　岗田美保子. 关于病房护理人员配置计划制作的数理模型. 医疗情报学，(日) 1989，9 (3).

2　Mihoko. Okada, Masahiko. Okada: Prolog - Based System for Nursing Staff Scheduling Implemented on a Persond Computer. Computers and Biomedical Research. 1989, 21.

3　伊藤优利子. 最佳化护理日程安排管理系统. 医疗情报学. (日). 1991, 11 (1).