ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2007/01(03)-0305-09

E-mail:fcst@public2.bta.net.cn
http://www.ceaj.org
Tel:+86-10-51616056

# 最少比较排序问题中 $S(15)$和 $S(19)$的解决 *

成维一,刘晓光+,王 刚,刘 璟
CHENG Weiyi, LIU Xiaoguang+, WANG Gang, LIU Jing

南开大学 信息技术科学学院,天津 300071
College of Information Science, Nankai University, Tianjin 300071, China
+Corresponding author: E-mail: liuxg74@yahoo.com.cn

**CHENG Weiyi, LIU Xiaoguang, WANG Gang, et al. The results of $S(15)$ and $S(19)$ to minimum-comparison sorting problem. Journal of Frontiers of Computer Science and Technology, 2007,1(3):305-313.**

**Abstract**: The minimum-comparison sorting is the problem of finding the minimum number of key comparisons needed to sort $n$ elements in the worst case. Let $S(n)$ be the minimum number of comparisons that will suffice to sort $n$ elements. M.Wells found that $S(12)=30$ by an exhaustive search in 1965. With the help of parallel computer, M.Peczarski improved Wells algorithm and proved that $S(13)=34$, $S(14)=38$ and $S(22)=71$ in 2002 and 2004 respectively. The authors extend both Wells algorithm and Peczarski algorithm. Some new results that $S(15)=42$ and $S(19)=58$ are first proved on a parallel computer named NKStars.

**Key words**: minimum-comparison sorting; optimal sorting; linear extensions counting

**摘 要**:最少比较排序问题就是要研究在最坏情况下,对 $n$ 个元素完成排序所需要的最少比较次数 $S(n)$。1965 年 M.Wells 用穷举法证明了 $S(12)=30$。2002 年和 2004 年,M.Peczarski 通过计算先后得到 $S(13)=34$,$S(14)=38$,$S(22)=71$。文章在 Wells 算法和 Peczarski 算法基础上,设计了一个新的 PS 算法,并改进了线性扩展计数算法,在并行机"南开之星"上计算得到 $S(15)=42$,$S(19)=58$。

**关键词**:最少比较排序;最优排序;线性扩展计数
**文献标识码**:A **中图分类号**:TP301.6

* the Key Project of National Natural Science Foundation of China under Grant No.90612001(国家自然科学基金重大项目); the Tianjin Science and Technology Development Plan Foundation of China under Grant No.043185111-14(天津市科技发展计划);the Science & Technology Innovation Fund of Nankai University(南开大学科技创新基金).
Received 2007-08,Accepted 2007-10.

## 1 引言

最小比较排序问题是研究在最坏情况下，对 $n$ 个元素完成排序所需要的最少比较次数。这一问题是排序算法的一个基础性问题。Steinhaus 在《Mathematical Snapshots》一书中首次提出了最少比较排序问题[1]，Knuth 在《The Art of Computer Programming》第 3 卷中也专门用一节来讨论这一问题[2]。比较次数虽然不是衡量排序算法优劣的唯一标准，但研究最少比较排序问题有助于更好地理解排序过程的本质，从而设计出更加高效的排序算法。

现有研究证明最小比较排序问题具有理论下界：设 $S(n)$是将 $n$ 个元素排序所需要的最少比较次数，根据信息论，它具有如下的理论下界：

$$S(n)\geqslant\lceil\lg n!\rceil=C_n \quad (1)$$

1956 年，Demuth 证明了对 5 个元素的排序最少需要进行 7 次比较，L.Ford 和 S.Johnson 对这一方法进行了推广，设计了合并插入排序算法[1,3]。合并插入排序是现有的在最坏情况下的比较次数最少的排序算法。用 $F_n$ 表示合并插入算法对 $n$ 个元素进行排序所需要的最大比较次数。$F_n$ 和 $C_n$ 的比较如表 1。

**表 1 $F_n$ 和 $C_n$ 的关系比较**

Table 1 The relationship between $F_n$ and $C_n$

| $n$ | $C_n$ | $F_n$ | $n$ | $C_n$ | $F_n$ |
|---|---|---|---|---|---|
| 2 | 1 | 1 | 14 | 37 | 38 |
| 3 | 3 | 3 | 15 | 41 | 42 |
| 4 | 5 | 5 | 16 | 45 | 46 |
| 5 | 7 | 7 | 17 | 49 | 50 |
| 6 | 10 | 10 | 18 | 53 | 54 |
| 7 | 13 | 13 | 19 | 57 | 58 |
| 8 | 16 | 16 | 20 | 62 | 62 |
| 9 | 19 | 19 | 21 | 66 | 66 |
| 10 | 22 | 22 | 22 | 70 | 71 |
| 11 | 26 | 26 | 23 | 75 | 76 |
| 12 | 29 | 30 | 24 | 80 | 81 |
| 13 | 33 | 34 | | | |

不妨设集合：$N1=\{2,3,4,5,6,7,8,9,10,11,20,21\}$，$N2=\{12,13,14,15,16,17,18,19,22,23,24\}$，对 $\forall n\in N1$ 有 $F_n=C_n$，所以 $S(n)=F_n$，即合并插入方法是最优的。对 $\forall n\in N2$ 有 $F_n=C_n+1$。因此，只要能够证明 $S(n)>C_n$，则有 $S(n)=F_n$。此时，合并插入方法也是最优的。文章所要研究的就是集合 $N2$ 中 15 和 19 两个元素的 $S(n)$的证明问题。

## 2 相关的工作

在表 1 中，第一个需要证明的 $S(n)$是 $S(12)$。1965 年，Wells 使用穷举法证明 $S(12)>29$，即 $S(12)=30$，从而证明了当 $n$ 为 12 时，合并插入排序方法是最优的[3]。假设 $R_c$ 是经过 $c$ 次比较得到的偏序的集合，$r\in R_c$，$u_j$ 和 $u_k$ 是 $r$ 中两个不相关的元素，$r_1=r+u_ju_k$，$r_2=r+u_ku_j$。Wells 证明了如果 $r_1$ 和 $r_2$ 都能经过 $C_n-c-1$ 次比较完成排序，则 $r$ 能经过 $C_n-c$ 次比较完成排序；如果所有不相关的元素对$(u_j,u_k)$都不能生成这样的 $r_1$ 和 $r_2$，则 $r$ 不能经过 $C_n-c$ 次比较完成排序。Wells 算法首先测试 $r_1$ 和 $r_2$ 中的任一个，若不能经 $C_n-c-1$ 次完成排序，则另一个不必再考虑。如果 $R_{C_n}=\Phi$，则可以断定 $S(n)>C_n$。用 Wells 算法计算 $S(12)$时，得到 $R_{24}=R_{25}=\cdots=R_{29}=\Phi$，因此有 $S(12)=F_{12}=30$。因为在每次比较以后，Wells 算法只向 $R_c$ 中加入了一个新的偏序 $r_1$ 或 $r_2$，所以另一个能否在 $C_n-c-1$ 次比较后完成排序是不确定的。此时，如果 $R_{C_n}\neq\Phi$，Wells 算法不能断定 $S(n)$与 $C_n$ 是否相等，需要通过其它方法来判断。在计算 $S(13)$、$S(14)$时，Wells 算法就遇到了这一问题。

这以后的几十年中，最少比较排序问题的研究几乎没有什么显著进展。直到 2002 年，Peczarski 计算证明了 $S(13)=34$[4]。2004 年他又并进一步计算证明了 $S(14)=38$ 和 $S(22)=71$[5]。Peczarski 算法是在 Wells 算法的基础上，通过增加一个阶段测试，解决了 $R_{C_n}\neq\Phi$，$S(n)$是否等于 $C_n$ 的判定问题。但是，在计算 $S(15)$和 $S(16)$的时候，Peczarski 算法遇到了问题。它的计算量非常巨大，以至于现有计算机条件根本无法完成。在 $n$ 较大时，Peczarski 算法只能解决一

些特殊形式,例如 $S(22)$。

最少比较排序问题的核心是 $R_c$ 的计算问题,作者在 Wells 和 Peczarski 工作的基础上,给出了一个新的 $R_c$ 计算算法,成功地证明了 $S(15)=42$ 和 $S(19)=58$。据作者所知,这一证明结果是世界上首次发表。

## 3 文章算法

文章给出了一个用于解决最少比较排序问题的算法。该算法的基本步骤可以分为两大部分:第一,偏序的计算,其核心是线性扩展计数的计算,这一部分仍然使用 Wells 算法,作者对其进行了一些改进;第二,$R_c$ 的计算,这里给出了一个新算法——PS(Posets Sorting)算法取代 Peczarski 算法。

### 3.1 线性扩展计数

线性扩展计数的计算是计算 $S(n)$的过程中最重要的步骤之一。在用 Wells 算法计算 $S(13)$的过程中,计算线性扩展计数的时间占程序总运行时间的70%以上。

**定义** 1(线性扩展) 设 $r$ 为集合 $U$ 上的偏序,$r_L$ 是 $U$ 上的一个全序。对于 $U$ 中任意的元素 $x,y$,如果$(x,y)\in r$,有$(x,y)\in r_L$,那么称 $r_L$ 是 $r$ 的一个线性扩展。

求偏序的线性扩展数问题称为线性扩展计数(Counting Linear Extensions)问题。Brightwell 和Winkler 证明了线性扩展计数是 NP 完全问题[6]。Kahn 和 Saks 证明了总存在一次比较把一个偏序的线性扩展分为两组,数量比例在 3/8 和 8/3 之间,但是他们没有给出找出这样比较的方法[7]。Peczaski 证明了若偏序 $r$ 能在 $c$ 次比较以后完成排序,则 $r$ 的线性扩展数 $e(r)\leqslant 2^c$;并且,等价的(指同构或对偶的)偏序完成排序所需要的比较次数也相同[4]。

**定义** 2(可接受的划分) 设$(U,r)$是一个偏序,$D\in U,d\in D$。若满足下列条件,则称$(A,B)$是 $D$ 上关于 $d$ 的可接受的划分:(1)$A\cup B=D-\{d\}$ 且 $A\cap B=\Phi$;(2)若$(a,d)\in r$ 且 $a\neq d$,则 $a\in A$;(3)若$(d,b)\in r$ 且 $b\neq d$,则 $b\in B$;(4)对任意的 $a\in A$ 和 $b\in B$,有$(b,a)\notin r$。

Peczarski 的论文中给出了如下定理。

**定理** 1 设 $r$ 是 $U$ 上的一个偏序:

(1)若 $A,B\subseteq U,A\cap B=\Phi$,且对任意的 $a\in A$ 和 $b\in B$,有$(a,b)\notin r$,则

$$e(r|A\cup B)=e(r|A)\cdot e(r|B)\cdot C_{|A|+|B|}^{|A|} \tag{2}$$

(2)若 $D\subseteq U$ 且 $d\in D$,则

$$e(r|D)=\sum_{A,B} e(r|A)\cdot e(r|B) \tag{3}$$

其中式(3)是对 $D$ 上关于 $d$ 的所有可接受的划分求和。

定理 1 是 Peczarski 给出的,他用实验证明了该算法是计算线性扩展数最快的方法[4,5]。此外,对于 $|U|\leqslant 5$ 的偏序,Peczarski 还给出了一种时间代价为 $O(n)$阶的快速算法。

为叙述方便,文中使用有向图来表示偏序。公式(2)把图划分为连通子图,通过计算各子图的线性扩展数来计算偏序的线性扩展数。公式(3)针对连通图情况,选择一个节点 $d$,把偏序划分为多个关于 $d$ 的可接受划分。通过计算这些小规模偏序的线性扩展数来计算偏序的线性扩展数。

对于计算线性扩展计数的方法,文中的改进工作主要体现在缓存机制的引入。分析公式(3),可以发现 $A$、$B$ 都可能不是连通子图,而它们的连通子图在计算的过程中,需要重复计算式(3)。若这些连通子图的节点数不大于 5,使用 Peczarski 给出的快速算法计算;若节点数大于 5,就调用式(3)计算。为了提高计算效率,作者引入缓存机制,保存已经计算出的节点数不小于 5 的子图的线性扩展数。当再遇到这些子图时,就可以避免重复计算。

以图 1 所示偏序为例,其中$|U|=14$。这里省略有向图边上的箭头,用左右关系表示大小关系,使用公式(3)计算线性扩展数。取 $d=u_5$,则 $A$、$B$ 的所有取值中,节点数大于 5 的子图共 4 个,记为 $D_1=\{u_0,u_1,u_2,u_3,u_4,u_{10},u_{11}\}$,$D_2=\{u_0,u_1,u_2,u_4,u_{10},u_{11}\}$,$D_3=\{u_0,u_2,u_3,u_4,u_{10},u_{11}\}$,$D_4=\{u_0,u_2,u_4,u_{10},u_{11}\}$,且都被计算 2 次。$n$

越大，这种子图的线性扩展数被重复计算的次数越多。如果在第一次计算后保存了结果，则以后可以查询得到 $e(r|D_i)$。

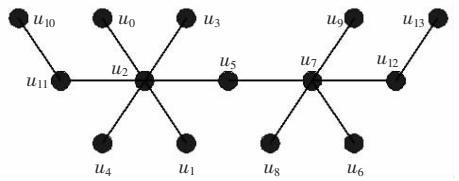


图 1 $U$ 上的一个偏序

Fig.1 A poset in graph $U$

在计算 $e(r|D_i)$ 的过程中，$r$ 始终不变，只需要保存 $D_i$ 和它对应的 $e(r|D_i)$。在实际实现中，引入位图机制来标示元素 $u_k$ 是否在 $D_i$ 中。当 $n$ 较小时，可以使用线性表表示缓存结构；$n$ 较大时，为了减小查找长度，可以把缓存组织成二叉排序树或平衡二叉树。实验证明，这样的改进极大地提高了计算线性扩展计数的效率。

## 3.2 Wells 算法

Wells 算法核心部分的描述如下：

$R_0$:={$r_0$}, where $r_0$={$(u_0,u_0),(u_1,u_1),\cdots,(u_{n-1},u_{n-1})$}

$R_1$ := $R_2$ := ⋯ := $R_{C_n}$ := Φ

if Wells(0,$C_n$)=false then $S_n$ := $F_n$

else $S_n$ := −1 //不能得出结论

BOOL Wells($c_1$,$c_2$) //输入 $R_{C_1}$,$c_1$,$c_2$,计算 $R_{c_1+1}$,$R_{c_1+2}\cdots R_{c_2}$

for $c$ := $c_1$+1 to $c_2$ step 1 do

for each $r\in R_{c-1}$ do

for each $(j,k)$ where $j<k$ and $(u_j,u_k)\notin r$ and $(u_k,u_j)\notin r$ do

$r_1$ := $r$+$(u_j,u_k)$

$r_2$ := $r$+$(u_k,u_j)$

if Search($R_c$,$r_1$)=false and Search($R_c$,$r_2$)=false then

if $e(r_1)<=2^{C_n-c}$ and $e(r_2)<=2^{C_n-c}$ then

if $e(r_1)>=e(r_2)$ then

$R_c$ := $R_c\cup\{r_1\}$

else $R_c$ := $R_c\cup\{r_2\}$

if $R_c$=Φ then return false

return true;

如算法所示，初始状态只包含一个完全无序的偏序 $r_0$ 的集合 $R_0$。在第 $c$ 步，对 $R_{c-1}$ 中每一个偏序 $r$ 中每一个没有关联的元素对 $(u_j,u_k)$ 进行一次比较，得到新的偏序 $r_1$ 和 $r_2$，若 $r_1$（或 $r_2$）的线性扩展数大于 $2^{C_n-c}$，则 $r_1$（或 $r_2$）不能经过 $C_n-c-1$ 次比较完成排序（文献[4]的定理 1）。那么，对 $r$ 来说，在第 $c$ 步比较 $u_j$ 和 $u_k$ 是不合适的。若 $r_1$ 和 $r_2$ 的线性扩展数都不大于 $2^{C_n-c}$，则将它们中线性扩展数较大的一个加入 $R_c$。如果 $r_1$ 或 $r_2$ 在 $R_c$ 中有一个等价的偏序，则它们没有必要加入 $R_c$（文献[4]的定理 2），这样可以有效地减少 $R_c$ 中偏序的数量。根据文献[5]的推论 1，如果 $R_{C_n}$=Φ，则可以断定 $S(n)>C_n$。文中同样要用到这一结论。

## 3.3 对 Wells 算法的改进

### 3.3.1 调整 Wells 算法中的比较顺序

**定理 2** 设 $r\in R_c$ 是 $U$ 上的偏序，$\{u_i,u_j,u_k,u_q\}\subseteq U$，$(u_j,u_k)\notin r$，$(u_k,u_j)\notin r$，$(u_i,u_j)\in r$，$(u_k,u_q)\in r$，$r_1=r+u_ju_k$，$r_2=r+u_iu_q$，如果 $e(r_1)>2^{C_n-c-1}$，则有 $e(r_2)>2^{C_n-c-1}$。

**证明** 根据偏序的传递性，$r_1$ 的线性扩展必然是 $r_2$ 的线性扩展。定理得证。

因此，如果先比较 $u_j$ 和 $u_k$，那么 $u_i$ 和 $u_q$ 就没有必要再进行比较，为此引入一个节点的势的定义。

**定义 3**（元素的势）元素 $u$ 的势 $C(u)=|r[u]|-|r^*[u]|$。其中 $r[u]$ 表示 $r$ 中比 $u$ 大的元素的集合，$r^*[u]$ 表示 $r$ 中比 $u$ 小的元素的集合。

在算法实现中，如果先比较势差较小的元素对，就可能把更多的元素对标记为不需要比较。

以图 2 为例，图 $U$ 上的偏序 $r$，$|U|=10$，$e(r)=252<2^8$。如果使用 Wells 算法，需要比较 25 个元素对，若先比较势差小的元素对，则只需要比较 13 个元素

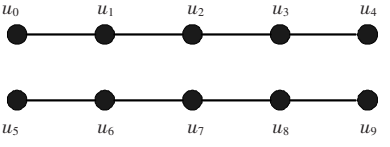


图 2 图 $U$ 上的一个偏序 $r$

Fig.2 Poset $r$ in graph $U$

对,运行时间减少了约一半。

### 3.3.2 需要将偏序加入 $R_c$ 时才查找等价偏序

Wells 算法中,每次进行比较以后,首先会在 $R_c$ 中查找 $r_1$ 和 $r_2$ 的等价偏序, 目的是希望找到等价偏序,从而避免计算 $r_1$ 和 $r_2$ 的线性扩展数。但实验数据的统计表明,找到等价偏序的概率是很小的。以计算 $S(13)$为例,找到等价偏序的概率约为 1/25,而计算 $S(19)$时还不到 1/50。但在计算 $S(13)$时,这个查找过程占用了总运行时间的 20%以上。

作者对 Wells 算法进行了如下改进:在一次比较以后,直接计算 $r_1$ 和 $r_2$ 的线性扩展数,只在需要把 $r_1$ 或 $r_2$ 加入 $R_c$ 时,才检查 $R_c$ 中是否存在等价偏序。实验表明,在计算 $S(13)$时,改进算法使运行时间减少了 20%左右。

## 3.4 计算 $R_c^*$ 的算法 PS 算法

Wells 算法和 Peczarski 算法都是要在所建立的众多比较树中,搜索一个高度为 $C_n$ 的比较树。如果发现第 $c$ 层上的某节点的子树的高度大于 $C_n-c$,则这个子树所在的比较树的高度大于 $C_n$, 停止测试该比较树。这两个算法的不同在于:Wells 算法同时处理所有的比较树,对这些比较树的节点按层访问,可以将其称之为广度优先访问。这样的处理可以避免同一层上等价偏序的重复处理。而 Peczarski 算法对单个的子树进行深度优先的访问, 没有考虑等价偏序,效率较低。例如 $n$ 为 12 时,$C_n$=29,用 Wells 算法计算时间不足 3 s,用 Peczarski 算法需要 6 h。

随着 $n$ 和 $C_n$ 的增大,Peczarski 算法的时间代价迅速增加。Peczarski 在计算 $S(13)$和 $S(14)$时,计算 $R_c^*$ 分别用了 3 min 和 390 h。可以预见,该算法不能用于计算 $S(15)$。

PS 算法中作者将计算 $R_c^*$ 的过程修改为广度优先的访问。生成 $R_c^*$ 时,先对 $R_c$ 进行一次扫描,把需要处理的偏序收集起来,组成新的 $R_{c+1}$;这些偏序对应的 $R_c$ 中的偏序组成 $R_c'$; 再使用 Wells 算法和 PS 算法计算 $R_{c+2},R_{c+3},\cdots,R_{C_n},R_{C_n}^*,R_{C_n-1}^*,\cdots,R_{c+1}^*$;再次扫描 $R_c'$,就可以得到新的 $R_c^*$。

当 $c$ 较大时(取经验值 $c$ 大于 $C_n-13$),Peczarski 算法搜索的深度不深, 而且线性扩展计数的效率也很高,Peczaski 算法总体效率较高。在这个阶段,使用 Peczarski 算法,而 $c$ 值较小时,使用 PS 算法。PS 算法的描述如下:

```
R0:={r0}, where r0={(u0,u0),(u1,u1),…,(un-1,un-1)}
        if Wells(0,Cn)=false then
          Sn :=Fn
          return
        RCn* := RCn
        for c:=Cn-1 to 1 step -1 do
          MaxPecz = c-M      // M is a constant
          if PS(c)=false then
            Sn := Fn
            return
          Sn := Cn
BOOL PS(c)                              // Calculate Rc*
if c>MaxPecz then
  return Pecz2(c) //Call Peczarski algorithm
Rc* := RP := Rc+1 := Φ
EP := Φ
for each r∈Rc do
  if ∃e∈Ec where e.gn=r.gn and Search(Rc+1*,r1,1)
    and Search(Rc+1*,r2,1) then
    r1 := r+e
    r2 := r+(-e)
    Rc* := Rc*∪{r}
  else
    ∀e∈Ec where e.gn=r.gn and Search(Rc+1*,r1,1)
      RP := RP∪{r}
      EP := EP∪{e}          // modify e.gn
      Rc+1 := Rc+1∪{r2}
if | Rc+1|>0 and Wells(c+1,Cn) then
  for c1:=Cn-1 to c+1 step -1
    if PS(c1)=false then
      break
    for each r∈RP do
```

```
    for each e∈E_P where e.gn=r.gn
        r_1 := r+e
        if Search(R_{c+1}*,r_1,1) then
            R_c* := R_c*∪{r}
            break
if R_c*=Φ then return false
return true
```

由于 Wells()函数总是保存 $r_1$ 和 $r_2$ 中线性扩展数较大的一个，新生成的 $R_{c+1}$ 中则保存了线性扩展数较小的一个，经过几次递归调用以后，$R_{c+1}$ 中的偏序经过 $C_n-c-1$ 次比较完成排序的概率增大，递归调用的代价也随之增大，需要通过修改 *MaxPecz* 的值限制递归深度。实验表明，计算 $S(13)$、$S(14)$、$S(15)$ 时，$M$ 取 1 或 2 较好。例如，计算 $S(14)$时，取 $M$ 为 1，计算 $R_c^*$ 的时间进一步降低到 4 h。

## 4 计算结果

实验分为两部分：第一，检验算法的有效性：通过计算 $S(13)$、$S(14)$和 $S(22)$，得到了与 Peczarski 相同的结果，从而验证了算法的有效性；第二，计算证明 $S(15)=42$ 和 $S(19)=58$。实验是在“南开之星”并行计算机上运行完成的。“南开之星”是由南开大学建立的包括 800 个 CPU 和 54 TB 存储空间的集群计算机。它的理论计算峰值达到每秒 4.7 万亿次浮点计算。2004 年建立时，“南开之星”的当年排名世界第 42 位，中国第 3 位。

### 4.1 计算 $S(13)$、$S(14)$和 $S(22)$

计算 $S(13)$时，由于计算量不大，在实验中使用的是串行程序。计算进行了两次，分别使用 Peczarski 算法和 PS 算法。表 2 中 $T_{R_c-1}$ 和 $T_{R_c-2}$ 分别表示两次实验中计算 $R_c$ 的时间(单位为秒)。从表 2 中可以看出，当比较进行到第 15 次时，$R_c^*$ 是空集。根据文献[5]的推论 1，得到结论 $S(13)=F_{13}=34$。

计算 $S(14)$时，先用串行程序计算 $R_1$~$R_{17}$，把 $R_{17}$ 中的偏序排序后划分成 16 块，定义 MAXD 为 26，使用并行化 Wells 算法，计算出 $R_{35}[i]$后，分别计算 $R_{35}^*[i]$~$R_{27}^*[i]$，合并 $R_{27}^*[i]$后使用 PS 算法继续计算 $R_c^*$，算法中的 $M$ 为 1。在程序中，当 $c$ 大于 17 时，是并行计算 $R_c$，且计算结果没有消除全部等价的偏序，因此，表 3

**表 2 计算 $S(13)$的结果**

Table 2 The results of sorting 13 elements

| $c$ | $\|R_c\|$ | $T_{R_c-1}$ | $T_{R_c-2}$ | $\|R_c^*\|$ | $c$ | $\|R_c\|$ | $T_{R_c-1}$ | $T_{R_c-2}$ | $\|R_c^*\|$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | | 17 | 304 242 | 153 | 65 | 6 |
| 2 | 2 | 0 | 0 | | 18 | 390 933 | 199 | 84 | 12 |
| 3 | 4 | 0 | 0 | | 19 | 418 068 | 197 | 84 | 26 |
| 4 | 8 | 0 | 0 | | 20 | 374 590 | 157 | 64 | 46 |
| 5 | 18 | 0 | 0 | | 21 | 276 672 | 98 | 43 | 73 |
| 6 | 31 | 0 | 0 | | 22 | 171 496 | 50 | 23 | 116 |
| 7 | 63 | 0 | 0 | | 23 | 91 011 | 20 | 11 | 194 |
| 8 | 170 | 0 | 0 | | 24 | 41 796 | 8 | 4 | 318 |
| 9 | 463 | 1 | 1 | | 25 | 17 155 | 2 | 1 | 428 |
| 10 | 1 261 | 0 | 0 | | 26 | 6 531 | 1 | 1 | 500 |
| 11 | 3 437 | 0 | 0 | | 27 | 2 336 | 0 | 0 | 441 |
| 12 | 9 100 | 1 | 1 | | 28 | 832 | 1 | 0 | 314 |
| 13 | 22 784 | 4 | 3 | | 29 | 265 | 0 | 0 | 177 |
| 14 | 53 262 | 15 | 9 | | 30 | 81 | 0 | 1 | 78 |
| 15 | 110 429 | 40 | 22 | 0 | 31 | 30 | 0 | 0 | 30 |
| 16 | 198 929 | 88 | 43 | 1 | | | | | |

只列出部分数据。同样计算到第 15 次时,$R_c^*$ 是空集,可以得到结论 $S(14)=F_{14}=38$。

表 4 显示了计算 $S(22)$的结果。当计算到第 40 次时,$R_c$ 是空集。因此,$S(22)=F_{22}=71$。

**表 3 计算 $S(14)$的结果**

Table 3 The results of sorting 14 elements

| $c$ | $\|R_c\|$ | $T_{R_c}-1$ | $T_{R_c}-2$ | $\|R_c^*\|$ | $T_{R_c^*}-1$ | $T_{R_c^*}-2$ |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | | | |
| 2 | 2 | 1 | 1 | | | |
| 3 | 4 | 0 | 1 | | | |
| 4 | 9 | 1 | 0 | | | |
| 5 | 21 | 0 | 1 | | | |
| 6 | 45 | 1 | 0 | | | |
| 7 | 116 | 1 | 1 | | | |
| 8 | 335 | 0 | 1 | | | |
| 9 | 996 | 1 | 0 | | | |
| 10 | 3 075 | 1 | 1 | | | |
| 11 | 9 594 | 1 | 1 | | | |
| 12 | 29 985 | 6 | 3 | | | |
| 13 | 93 611 | 27 | 12 | | | |
| 14 | 286 713 | 139 | 55 | | | |
| 15 | 834 872 | 656 | 219 | 0 | 488 | 32 |
| 16 | 2 116 456 | 2 614 | 736 | 1 | 384 | 48 |
| 17 | 5 302 847 | 8 314 | 2 011 | 5 | 1 816 | 432 |

## 4.2 计算 $S(15)$和 $S(19)$

计算 $S(15)$时,首先用串行程序计算 $R_1$~$R_{17}$,然后把 $R_{17}$ 中的偏序排序后划分成 200 块分配给多个进程并行处理。在这里,MAXD 设为 27。在计算出 $R_{39}[i]$后,分别计算 $R_{39}^*[i]$~$R_{28}^*[i]$,再合并 $R_{28}^*[i]$;然后使用并行化的 PS 算法继续计算 $R_c^*$。其中,$c$ 大于 21 时,$M$ 设为 1;$c$ 不大于 21 时,$M$ 设为 2。在计算时,程序产生的中间数据特别巨大(超过 800 G)。考虑到在消除等价偏序时,巨大的数据量会造成过重的 I/O 负载。因此并行规模并不大,实际计算的平均并行规模大约是 20,在南开之星上运行了 18 天。表 5 所示,在计算到第 15 次时,$R_c^*$ 是空集,可以得到结论 $S(15)=F_{15}=42$。后续的并行计算数据略去。

在计算 $S(19)$时,同样先使用串行程序计算 $R_1$~$R_{19}$;然后在每次计算出一个 $R_c[i]$以后,消除等价偏序,得到结果后,再估算由 $R_c$ 计算 $R_{c+1}$ 需要的时间;

**表 4 计算 $S(22)$的结果**

Table 4 The results of sorting 22 elements

| $c$ | $\|R_c\|$ | $T_{R_c}$ | $c$ | $\|R_c\|$ | $T_{R_c}$ |
|---|---|---|---|---|---|
| 9 | 21 | 0 | 25 | 91 922 | 65 903 |
| 10 | 37 | 0 | 26 | 87 343 | 84 548 |
| 11 | 64 | 1 | 27 | 72 984 | 90 452 |
| 12 | 107 | 0 | 28 | 53 978 | 74 400 |
| 13 | 166 | 0 | 29 | 35 446 | 50 723 |
| 14 | 280 | 0 | 30 | 21 029 | 29 013 |
| 15 | 546 | 1 | 31 | 11 186 | 14 297 |
| 16 | 1 198 | 3 | 32 | 5 369 | 6 170 |
| 17 | 2 471 | 12 | 33 | 2 380 | 2 333 |
| 18 | 4 766 | 42 | 34 | 858 | 764 |
| 19 | 9 005 | 144 | 35 | 313 | 209 |
| 20 | 16 561 | 547 | 36 | 111 | 50 |
| 21 | 29 002 | 2 220 | 37 | 31 | 12 |
| 22 | 46 267 | 7 131 | 38 | 7 | 2 |
| 23 | 65 981 | 17 889 | 39 | 1 | 1 |
| 24 | 83 487 | 40 357 | 40 | 0 | 0 |

**表 5 计算 $S(15)$的结果**

Table 5 The results of sorting 15 elements

| $c$ | $\|R_c\|$ | $T_{R_c}-1$ | $\|R_c^*\|$ | $T_{R_c^*}$ |
|---|---|---|---|---|
| 1 | 1 | 0 | | |
| 2 | 2 | 1 | | |
| 3 | 5 | 0 | | |
| 4 | 11 | 1 | | |
| 5 | 26 | 0 | | |
| 6 | 59 | 1 | | |
| 7 | 151 | 1 | | |
| 8 | 433 | 0 | | |
| 9 | 1 319 | 1 | | |
| 10 | 4 230 | 1 | | |
| 11 | 13 762 | 2 | | |
| 12 | 45 645 | 9 | | |
| 13 | 155 490 | 44 | | |
| 14 | 532 955 | 272 | | |
| 15 | 1 801 701 | 1 638 | 0 | 9 456 |
| 16 | 5 939 725 | 9 125 | 1 | 22 880 |
| 17 | 18 484 389 | 50 340 | 6 | 16 224 |

表 6 计算 $S(19)$的结果

Table 6 The results of sorting 19 elements

| c | $\|R_c\|$ | $T_{R_c}$ | c | $\|R_c\|$ | $T_{R_c}$ |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 28 | 119 770 015 | 5 237 668 |
| 2 | 1 | 0 | 29 | 122 029 075 | 4 667 284 |
| 3 | 2 | 0 | 30 | 112 020 347 | 3 780 324 |
| 4 | 3 | 0 | 31 | 92 928 523 | 2 610 808 |
| 5 | 5 | 1 | 32 | 70 317 565 | 1 610 952 |
| 6 | 9 | 0 | 33 | 47 688 943 | 898 876 |
| 7 | 20 | 0 | 34 | 29 765 420 | 437 404 |
| 8 | 53 | 0 | 35 | 17 297 396 | 192 724 |
| 9 | 131 | 0 | 36 | 8 870 262 | 80 140 |
| 10 | 302 | 1 | 37 | 4 417 441 | 28 056 |
| 11 | 681 | 0 | 38 | 2 078 517 | 9 640 |
| 12 | 1 618 | 0 | 39 | 970 079 | 2 672 |
| 13 | 3 969 | 2 | 40 | 432 274 | 849 |
| 14 | 9 798 | 4 | 41 | 188 516 | 268 |
| 15 | 23 927 | 19 | 42 | 80 160 | 83 |
| 16 | 58 295 | 75 | 43 | 33 321 | 26 |
| 17 | 145 464 | 336 | 44 | 13 277 | 9 |
| 18 | 370 674 | 1 586 | 45 | 4 876 | 4 |
| 19 | 941 274 | 7 219 | 46 | 1 813 | 2 |
| 20 | 2 316 300 | 28 632 | 47 | 885 | 1 |
| 21 | 5 346 821 | 106 504 | 48 | 467 | 2 |
| 22 | 11 368 843 | 331 116 | 49 | 182 | 0 |
| 23 | 22 144 758 | 808 596 | 50 | 48 | 1 |
| 24 | 38 155 903 | 1 670 388 | 51 | 12 | 1 |
| 25 | 61 749 751 | 2 759 852 | 52 | 1 | 1 |
| 26 | 88 593 306 | 4 101 568 | 53 | 0 | 0 |
| 27 | 105 605 733 | 5 039 924 | | | |

接着根据这一估计时间来确定划分数据块的大小;最后将划分的数据提交各个进程并行处理。当 $c$ 大于 39 时,再次转换为串行算法计算。计算结果见表 6。其中,$\sum T_{R_c}$=9 959 h,$\sum |R_c|$=965 747 057,平均并行规模约 32,在南开之星上运行了 13 天。如表 6 所示,在计算到第 53 次时,$R_c$ 是空集,可以得到结论 $S(19)=F_{19}=58$。

## 5 结论和今后的工作

文章针对最小比较排序问题,在 Wells 和 Peczarski 工作的基础上,给出了一个新的 $R_c$ 计算算法。利用该算法,通过并行计算实验的验证,首次证明了 $S(15)$等于 42,$S(19)$等于 58。通过实验计算,估计下一个可能取得的结果应该是 $S(18)$。因为从排序效率的角度来看,$S(18)$的计算量可能小于 $S(16)$。这也是作者今后努力的方向。

**References:**

[1] Hugo S. Mathematical snapshots[M]. New York: Oxford University Press, 1969.

[2] Knuth D E. The art of computer programming Vol.3: sorting and searching[M]. 2nd ed. Massachusetts: Addison-Wesley, 1998.

[3] Wells M. Elements of combinatorial computing[M]. New York: Pergamon Press, 1971.

[4] Peczarski M. Sorting 13 elements requires 34 comparisons[C]//Mohring, Raman. LNCS 2461: Proc of 10th Annual European Symposium on Algorithm. [S.l.]: Springer Press, 2002:785-794.

[5] Peczarski M. New results in minimum-comparison sorting[J]. Algorithmica, 2004,40:133-145.

[6] Brightwell G, Winkler P. Counting linear extensions[J]. Order, 1991,8:225-242.

[7] Kahn J, Saks M. Balancing poset extensions[J]. Order, 1984,1:113-126.

成维一(1972–),男,甘肃兰州人,硕士生,主要研究领域为理论计算机科学。

CHENG Weiyi was born in 1972. He is a Master student in Nankai University. His current research interests include theoretical computer science.

刘晓光(1974–),男,河北安国人,博士,副教授,分别于 1996 年、1999 年和 2002 年于南开大学计算机系获学士、硕士和博士学位,目前是南开大学计算机系副教授,主要研究领域为并行计算与分布式系统。

LIU Xiaoguang was born in 1974. He received his BS degree, MS degree and PhD degree from Nankai University in 1996, 1999 and 2002 respectively. He is an associate professor of Nankai University. His current research interests include parallel computing and distributed system.

王刚(1974–),男,北京人,博士,副教授,分别于 1996 年、1999 年和 2002 年在南开大学计算机系获学士、硕士和博士学位,目前是南开大学计算机系副教授,主要研究领域为并行计算与分布式系统。

WANG Gang was born in 1974. He received his BS degree, MS degree and PhD degree from Nankai University in 1996, 1999 and 2002 respectively. He is an associate professor of Nankai University. His current research interests include parallel computing and distributed system.

刘璟(1942–),男,北京人,教授,博士生导师,1982 年于南开大学数学系获得硕士学位,目前是南开大学计算机系教授,博士生导师,主要研究领域包括算法设计和并行计算。

LIU Jing was born in 1942. He received his MS degree from Nankai University in 1982. He is a professor and PhD supervisor of Nankai University. His current research interests include algorithm design and parallel computing.