SPACE–TIME TRADE–OFFS IN PRODUCING CERTAIN PARTIAL ORDERS

PETER RUŽIČKA

(Received October 24, 1979)

1. Introduction

From everyday programming experience it is well-known that one can often speed up programs at the expence of using extra storage space and alternatively that one can often reduce the storage space at the expence of increasing the computing time. Thus, it is quite natural and from the point of view of computing fairly important to investigate whether these phenomena correspond to some general computational trade-off.

In this paper we deal with problems of producing partial orders (on some set) with one particular element, the centre, around which the set minus the centre is divided into a subset of elements less than the centre and a subset of elements greater than the centre. Many of the well known selection problems can be understood as tasks of producing certain partial orders with the centre.

The classical time complexity problem was formulated as the one to find time-optimal algorithms without paying any attention to the space needed for the computation. In this paper we investigate the problem of time complexity in limited space. We start with determining the "allowable" space interval in which it is sufficient to find time-optimal algorithms. We do it by examining the minimal space required by a time-optimal algorithm and the minimal space required by any algorithm solving a problem. For certain problems the space interval is trivial, i.e. there does not exist a space-time trade-off for this problem. We say there exists a space-time trade-off for a problem if there does not exist an algorithm solving this problem simultaneously in minimal space and in minimal time. Our major goal here is to investigate a continuous space-time trade-off, i.e. we study the effect of the added space on the time complexity of a problem. Finally, we investigate the number of memory transfers done by time-optimal space -minimal algorithms producing partial orders with central elements.

2. Problem and model of computation

Consider we are given a totally ordered finite set X, the reservoir. The order is not known initially and can be determined only by performing successive binary comparisons between elements of X. In this paper we shall investigate the problem of producing the partial order S_m^k on $k+m+1$ elements, where a particular element, the centre, is smaller than k other elements and greater than the remaining elements.

Algorithm solving our problem can be represented as a finite computation tree with a finite number of memory cells (so called working space) and a one-way read-only input tape. Inputs, i.e. elements of the reservoir, are read from the input tape into a specified memory cell. Comparisons may be made between the contents of any pair of memory cells. In a computation tree each interior node is associated with an operation and the output takes place at the leaves of the tree. Comparison operation have either two branches ($\leq$, $>$ comparisons) or three branches ($<$, $=$, $>$ comparisons), the input operation has one branch. A problem is described relative to a size parameter n (which can be the number of elements to be processed). For each n there is a finite tree with a finite number of memory cells to describe the computation. The computation obviously begins at the root of the tree and proceeds from node to node until a leaf is reached. A path of greatest length (i.e. with the greatest possible number of comparisons) indicates the worst-case time complexity of an algorithm* and the number of memory cells used indicates the space complexity of an algorithm. The time (space) complexity of a problem is the minimum time (space) complexity over all algorithms solving this problem.

The notation $P^{f(n)}(n)$ is used to denote the time complexity of the problem P over the n-element set under a space restriction $f(n)$; $P_{f(n)}(n)$ denotes the space complexity of the problem P solved by doing at most $f(n)$ binary comparisons.

Furthermore, we use the following notation:

O $f(n) = O(g(n))$ iff there are constants $c>0$ and $k>0$ such that for all $n>c$ $f(n)\leq k\cdot g(n)$ holds.

Ω $f(n) = \Omega(g(n))$ iff there are constants $c>0$ and $k>0$ such that for all $n>c$ $f(n)\geq k\cdot g(n)$ holds.

Θ $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ holds.

o $f(n) = o(g(n))$ iff the limit of $f(n)/g(n)$, as n tends to infinity, is 0.

3. Minimum-space algorithms

3.1 (Partial order S_{n-1}^0) A widely used algorithm producing the partial order S_{n-1}^0 is optimal with respect to the space and time requirements simultaneously. This yields a non-existence of a space-time trade off for this

* Throughout this paper the notion time complexity of the algorithm is often interchanged synonymously with number of comparisons used by this algorithm.

problem. Furthermore, $n-1$ memory transfers are necessary for each algorithm using impli cite data structure (i.e. a data structure without explicite indices or pointers) and producing S^0_{n-1} simultaneously in minimal time and in minimal space.

3.2 (Space-optimal producing of a partial order S^1_{n-2}). It is fairly easy to find an algorithm producing a partial order S^1_{n-2} in minimal space. We present, as an example, the following algorithm which requires 3 memory cells A [1...3] and $2n-3$ binary comparisons:

```
procedure SECMINSPACE(n);
begin
  read (A[1], A[2]);
  if A[1]< A[2] then exchange (A[1], A[2]);
  M←1; m←2; t←3;
  while the input tape is not empty
  do
    begin
      read (A[t]);
      select
        A[t]> A[M]: begin
                      exchange (t, M); exchange (t, m)
                    end;
        A[t]> A[m]:   exchange (t, m)
      end
    end;
  return (A[M], A[m])
end
```

It is not hard to determine the lower bound on time required by the space optimal algorithm for this problem. Consider the directed acyclic graph (dag) whose nodes correspond to elements in memory cells and whose edges correspond to the ordering between elements in memory cells as specified by the partial order underlying the previous computation. As the computation proceeds, the dag is changing in the following way. Initially, it consists of an empty set of nodes and at each step of the computation an egde between two compared elements is added and oriented to minimize the number of paths of length greater than one. Whenever the paths of length two occur in a dag they are shorten by eliminating all terminal nodes together with edges leading to them. These nodes are then replaced by singletons from the reservoir.

If a dag with 3 nodes is considered and there are additional $n-3$ singletons in the reservoir, then 3 comparisons are necessary for the first elimination and 2 comparisons for each of the following $n-3$ eliminations are necessary, thus giving

Claim 1

$$SEC^3(n) = 2n-3.$$

Furthermore, each algorithm producing S^1_{n-2} in minimal space makes at least $\Omega(n)$ memory transfers.

3.3 (Minimum-space required by time optimal producing of S^1_{n-2}) The time-optimality of the $SEC(n)$ problem has been proved by Schreier and Kislitsyn (see in Knuth (73)). Time-optimal algorithm can be exhibited in the following manner:

procedure *SECMINTIME* (n);
begin

1. Set up a balanced binary tree for a knock-out tournament for a set X of size n;
   comment the winner of the tournament is denoted as first;
2. $sec \leftarrow MAXIMUM$ {the set of all elements compared in step 1 directly with the element *first*};
3. **return** (*first*, sec)

end

This algorithm makes $n-2+\lceil \log n \rceil$ binary comparisons and a straightforward implementation by means of tree data structure requires $0(n)$ space. We investigate the minimum space required by the time optimal algorithm for this problem. We prove that the problem of producing the partial order S^1_{n-2} can be computed simultaneously making $n-2+\lceil \log n \rceil$ binary comparisons and using $1/2 \log^2(n)+\Theta(\log n)$ memory cells. The existence of such an algorithm together with Claim 1 implies a space-time trade-off for the $SEC(n)$ problem.

We first show that for a time optimal algorithm $0(\log^2 n)$ space suffice.

Lemma 1

$$SEC_{n-2+\lceil \log n \rceil(n)}(n) \leq 1/2 \log^2 n + 0(\log n).$$

Proof

Assume that $n = 2^k$ for some $k \geq 1$, other cases follow easily. We present an algorithm to produce a partial order S^1_{n-2} which is optimal relative to the number of comparisons and which requires space for maintaining $0(\log^2 n)$ elements. This algorithm uses an implicit data structure based on the binomial trees T_k which are defined inductively as follows: a single node forms a tree T_0, and two copies of a T_k with an additional edge between two roots form a T_{k+1}. An alternative and very useful representation of a binomial tree T_{k+1} can be viewed as the root of $k+1$ sons being roots of $T_k, T_{k-1}, \ldots, T_0$, respectively. Our algorithm needs to save only two upper levels of each binomial tree. Therefore it is sufficient to consider two-level binomial trees which will have the form: $B_i = S^0_i$ for $i = 0,1,2, \ldots, \lceil \log n \rceil$. Initially this algorithm contains n elements on the input tape. At each step of the computation two elements of the input are read into the working space, and whenever possible, two copies of B_i are used to construct B_{i+1}.

```
procedure SEC1 (n);
begin
  comment the number n of elements on the input tape is even and at least
          two;
  repeat
    read  two elements from the input tape into the working space;
    i←0:
    while there are two copies of tree B_i in the working space
    do
      begin
        put together two copies of B_i to give B_{i+1};
        i←i+1
      end
  until the input tape is empty;
  return (the root of tree B_i, maximal son in tree B_i)
end
```

To bound the number M of elements used by this algorithm it may be observed that the *while* cycle is the only step of the algorithm which requires any space. At the worst case it requires (when all the trees B_i are present at the working space)

$$M(4) = 4$$

$$M(2^k) = k + M(2^{k-1}) \qquad \text{for } k \geq 3$$

and so

$$M(n) = 1/2 \log^2 n + 0(\log n).$$

To complete the proof of an $1/2 \log^2 n + 0(\log n)$ bound on the space it remains to show that maintaining a forest of two-level binomial trees implemented compactly in an array will need no additional space. Consider an implementation of the forest $\{B_{k_1}, \ldots, B_{k_s} | k_1 > \ldots > k_s \geq 1\}$ in an array A in the following way:

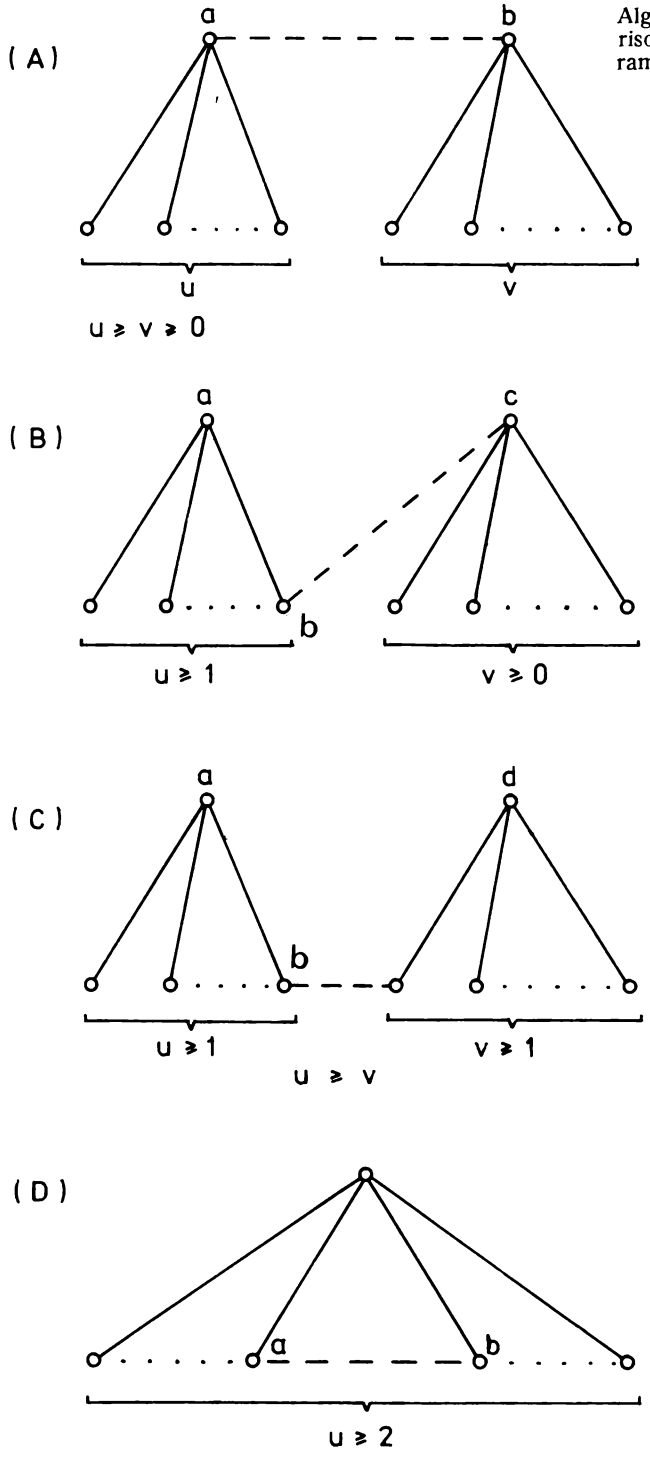
B_{k_1} is saved in $A[1..k_1+1]$ with the root in $A[1]$;

B_{k_t} is saved in $A[k_1+\ldots+k_{t-1}+t..k_1+\ldots+k_t+t]$

with the root in $A[k_1+\ldots+k_{t-1}+t]$ for $t = 2, \ldots, s$.

Given the number N of processed input elements, the position of each two-level binomial tree from the forest can be determined by computing the binary decomposition $N = \sum_{i=1}^{\log n} b_i \, 2^i$ with $b_i \in \{0,1\}$. For each $b_i = 0$ in the binary decomposition of N one B_i is present in the working space. □

Further we prove that this bound is optimal to within a lower ordered term. In order to show the asymptotical space-optimality of the previous algorithm we give a characterization lemma applicable for optimal algorithms over all time optimal algorithms (abreviated as (t, s)-optimal algorithms).



Algorithm comparison Hasse diagram

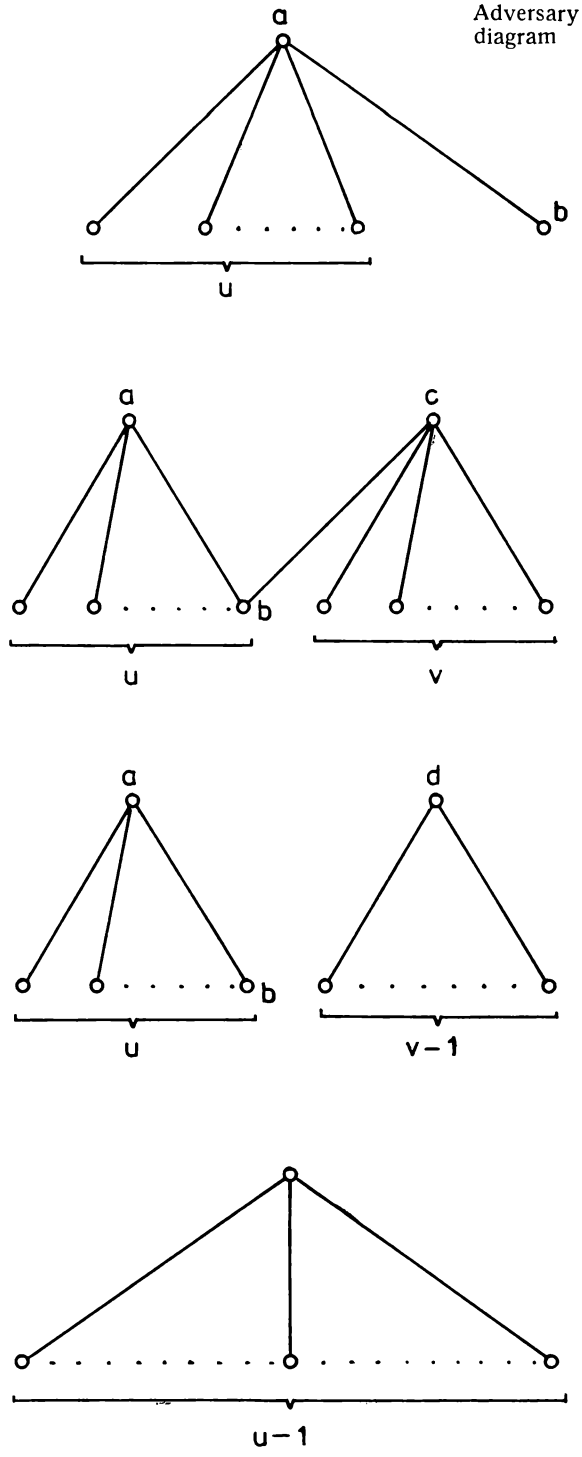


Figure 1

This characterization lemma is a key result for obtaining lower bounds on space and lower bounds on time for space-limited cases.

Lemma 2

Each time optimal algorithm solving the *SEC* (n) problem for $n = 2^s$, $s \geq 1$, performs only two kinds of comparisons:

(a) comparisons between roots of two B_k for $k = 0, \ldots, s-1$;
(b) comparisons between sons of B_s.

Proof

For each time optimal algorithm solving *SEC*(n) problem it is sufficient to maintain the forest of two level binomial trees. Consider an adversary with the following responding strategy *(Figure 1)*:

If the adversary's respond yields some non-tree structure, then it will certainly contain a subgraph of the form

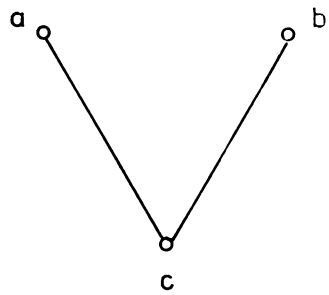


Figure 2

and after computing maximum either $a : c$ or $b : c$ will be a superfluous comparison, thus contradicting time-optimality condition. Hence, comparisons of the type (B) and (C) are illegal in time optimal computation. We see that this argument can be extended to comparisons of the type (D). A comparison of the type (A) for $u \neq v$ will also be superfluous by contradiction with time optimality as well as comparisons of the type (D) for $u \neq s$ using the assumption $n = 2^s$. Thus, we have constructed an adversary which forces each time optimal algorithm solving the *SEC* (n) problem for $n = 2^s$, $s \geq 1$, to perform either comparisons of the type (A) for $u = v$ and of type (D) for $u = s$ or yields the contradiction with time optimality condition. □

We turn to a proof of the asymptotical space-optimality of the previous algorithm.

THEOREM 1

$$SEC_{n-2+\lceil \log n \rceil}(n) = 1/2 \log^2 n + \Theta(\log n).$$

Proof

1. The case $n = 2^s$ for some $s \geq 1$. We do the proof by observing that each (t, s)-optimal algorithm for the SEC (n) problem has initially all elements situated on the input tape and using permissible comparisons between roots of two-level binomial trees B_k (by Lemma 2) a space is necessary for saving a copy of B_k (i.e. $k+1$ memory cells) and further space for generating a copy of B_k (i.e. $S(B_k)$). We obtain recursively equation

$$S(B_{k+1}) = k+1+S(B_k) \qquad \text{for } k = 2,3, \ldots, s$$
$$S(B_1) = 1$$

Therefore, each time-optimal algorithm for $SEC(n)$ problem requires $\Omega(s^2)$ space.

2. The extention of the proof to the case $n \neq 2^s$, $s \geq 1$, follows directly from the observation that

$$SEC_{2^t-2+t}(2^t) \leq SEC_{n-2+t}(n) \text{ holds for } t = \lceil \log n \rceil. \ \square$$

3.4 (Memory transfer complexity of (t, s)-optimal algorithms producing S^1_{n-2}) Consider now a (t, s)-optimal algorithm $SEC1$ for $n = 2^s$, $s \geq 1$. The number of comparisons between two level binomial trees is $n-1$. A straightforward application of this fact gives a rough estimation $0(n \cdot \log^2 n)$ for memory transfers. It can be seen that by a finer analysis of the $SEC1$ algorithm a logarithmic factor can be eliminated. We start with counting the number of memory transfers done by $SEC1$ during binomial tree comparisons. Consider a comparison between two roots of B_k to give B_{k+1} (denoted as $B_k : B_k$). This comparison needs k memory transfers in an array to compact B_{k+1}. All memory transfers made before $B_k : B_k$ can be computed by the following prescription:

> All memory transfers made for the second copy of B_k plus
> all memory transfers made for the first copy of B_k plus memory transfers made during the $B_k : B_k$.

If we denote memory transfers made for B_k $(B_k : B_k)$ by $M(B_k)$ $(M(B_k : B_k))$, then

$$M(B_{k+1}) = 2 \cdot M(B_k) + k \qquad \text{for } k \geq 2$$
$$M(B_2) = M(B_1 : B_1) = 2.$$

The solution is $M(B_{\log n}) = 0(n \cdot \log n)$.

Further we count memory transfers made by the $SEC1$ in order to compact elements in an array after each tree comparison. Again, the number of memory transfers in proportional to the number of the resulting comparison

structures and hence the entire number of memory transfers counted in this step is $0(n \cdot \log n)$. Thus, the overall number of memory transfers made by the *SEC*1 algorithm is $0(n \cdot \log n)$. This is the best we can do using implicit data structure.

We show how to do better for a special case. For the *SEC*(n) problem we give the following (c, s)-optimal algorithm which uses $0(n)$ memory transfers, but two indices for a working space. Thus, the following algorithm uses a non-implicit data structure.

```
procedure SEC2 (n);
begin
  comment the number n of the input elements is even and greater than two;
          the working space is denoted by
          A[1..(log² n)/2+3·(log n)/2];
  p←1; q←2;
  comment p(q) is a position of the old (a new) copy of B_0 in an array A;
  repeat
    read two elements from the input tape into memory cells
         A[p] and A[q];
    i←0;
    comment i is an index of the two-level binomial tree considered in the
            tree comparison;
    while there are two copies of B_i in the working space
    do
      begin
        i←i+1;
        p←position of the old copy of B_{i-1};
        compare the old copy of B_{i-1} with the root in A[p]
        with a new copy of B_{i-1} with the root in A[q] such that
          if A[p]<A[q]
          then
            if there is an old copy of B_i in the working space
            then
              begin
                exchange (A[p], A[q]);
                exchange (A[q-1], A[p]);
                q←q-1
              end
            else
              begin
                A[p+1..p+i-1]←A[q+1..q+i-1];
                exchange (A[p], A[q])
              end
      end;
      rename all new copies for the old one
    until the input tape is empty
end
```

We count the amount of memory transfers of the *SEC*2 algorithm. The **then** part in the second **if** statement contains the constant number of transfers in the memory and it is done once for each two level binomial tree comparison. Consider $n = 2^s$ for some $s \geq 1$. There are $0(n)$ binomial tree comparisons in the **while** statement and therefore $0(n)$ memory transfers are done in the **then** part of the second **if** statement in the worst case. Memory transfers in the **else** part of the second **if** statement are made once for each sequence of binomial tree comparisons and the number of transfers in this step is proportional to the number of elements in the underlying sequence which further equals the number of binomial tree comparisons made by the entire program. Hence there are $0(n)$ memory transfers made by the *SEC*2 algorithm.

THEOREM 2

The memory transfer complexity of the (t, s)-optimal $SEC(n)$ problem is $0(n)$.

We now turn our attention to the lower bound on the number of memory transfers in the case of compact implementation (i.e. when continuous memory cells are used) of data structures.

THEOREM 3

Each (t, s)-optimal algorithm for the $SEC(n)$ problem using a compact implementation of each B_i and a compact implementation of the entire working space requires $\Omega(n)$ memory transfers.

Proof

1. Let $n = 2^s$ for some $s \geq 1$. Consider two copies of two-level binomial tree B_k, placed in a working space of any (t, s)-optimal algorithm and let the first copy be placed in $A[i_1 .. i_{k+1}]$ with a root in $A[i_p]$, $1 \leq p \leq k+1$, and the second copy of B_k be placed in $A[j_1 .. j_{k+1}]$ with a root in $A[j_q]$, $1 \leq q \leq k+1$. Let the algorithm perform a comparison between these two copies. We construct an adversary which responds for this comparison according to the following scheme:

$$A[i_p] < A[j_q] \quad \text{iff} \quad i_p < j_q.$$

To estimate the number of memory transfers forced by this adversary it is sufficient to observe that for each sequence of binomial tree comparisons, number of memory transfers proportional to the length of the underlying sequence is necessary and that there are $n-1$ binomial tree comparisons for each (t, s)-optimal algorithm solving $SEC(n)$ problem.

2. Again, the case $n \neq 2^s$ can be obtained from the fact that the number of memory transfers for each (t, s)-optimal algorithm solving SEC $(2^{\log n})$ gives a lower bound on the number of memory transfers for a (t, s)-optimal algorithm for $SEC(n)$ problem. □

The following problem remains open: determine the lower bound on the number of memory transfers for the (t, s)-optimal $SEC(n)$ problem for the case in which two-level binomial trees need not to be necessarily compactly implemented and in which the forest of all two level binomial trees in the working space must be implemented compactly only if the minimum space is overloaded. Another unsolved problem is to prove whether there exists an effective implementation of comparison-optimal algorithms "in-situ" using only $o(n)$ memory transfers. Note that Theorem 2 gives an $0(n)$ solution to the "in-situ" problem and that Theorem 3 answers only a special case of the lower bound problem for (t, s)-optimal algorithms.

3.5 (Minimum space algorithms producing $S^{pn}_{(1-p)n}$ for $0<p<1/2$) Now consider a problem of producing the partial order $S^{pn}_{(1-p)n}$ for $0<p<1/2$.

It is well-known that there is an algorithm using absolute minimum space and making $0(n \cdot \log n)$ binary comparisons. This algorithm is using a heap data structure of $pn+1$ elements as a priority queue, throwing away the smallest remaining element as each new element is read. The algorithm based on this idea solves the problem in $n \cdot \log n + 0(n)$ binary comparisons and $0(n \cdot \log n)$ memory transfers.

To prove the lower bound on the number of comparisons required for determining the partial order $S^{pn}_{(1-p)n}$ for $0<p<1/2$, an adversary argument was used by Dobkin an Munro (78). They considered the data on the input tape to be arranged in such a way that the first $2pn$ input elements would be the $2pn$ smallest elements of the entire set. If the algorithm eliminates:

a) the minimum of $pn+1$ elements in the working space, then after pn eliminations, the pn smallest elements of the original set are sorted.

b) the maximum of $pn+1$ elements in the working space, then $(1-2p)n$ elements are sorted.

Thus an adversary is constructed which forces any algorithm producing the partial order $S^{pn}_{(1-p)n}$ to sort $\min(p, (1-2p))\, n$ elements. Furthermore, in this adversary at least one memory transfer is forced to be done as each element is eliminated. Hence we have

THEOREM 4

For any rational $p \in (0,1/2)$ there is an integer $c_p > 0$ such that for producing the partial order $S^{pn}_{(1-p)n}$ in minimal space, simultaneously $c_p \cdot n \cdot \log n - 0(n)$ binary comparisons and $\Omega(n)$ memory transfers are necessary.

There still remains the task to determine the exact number of memory transfers done for producing $S^{pn}_{(1-p)n}$, $p \in (0,1/2)$, in minimal space. The problem is whether the adversary argument can be augmented also for $\Omega(n \cdot \log n)$ memory transfer complexity.

3.6 (Minimum space algorithm producing $S^{n/2}_{n/2}$) A surprising result has been obtained for the case of median partial order. Dobkin and Munro (78) gave an algorithm *MEDMINSPACE* for computing median partial order in mi-

nimal space. This algorithm is asymtotically optimal with respect to time requirements and optimal with respect to space requirements.

procedure *MEDMINSPACE* (n);
begin
comment assume that $n = 2^k+1$ for $k \geq 1$;
1. read $2^{k-1}+1$ input elements into the set U;
2. for $i \leftarrow 0$ **downto** $k-3$ **do**
find the $n/2^{i+3}$ largest and smallest elements remaining in U and place them in L_{k-3-i} and S_{k-i-3} respectively;
3. for $i \leftarrow 0$ **to** $k-1$ **do**
read 2^i elements to R; then discard the largest and smallest 2^i elements of $L_i \cup S_i \cup R$ and the rest transfer into R;
4. read the rest of the input elements;
find the median partial order of the remaining elements by a standard linear time algorithm
end

To count the total number of memory transfers made by *MEDMINSPACE* algorithm, a standard selection algorithm of Paterson, Schönhage and Pippenger (76) is used which needs $3n+o(n)$ binary comparisons, $4n+o(n)$ memory transfers and n memory cells. We observe that the only steps of the algorithm which require any transfers in the memory are steps 2, 3 and 4. In step 2 each iteration makes linear number of memory transfers for the elements actually considered. Since this number is effectively halved on each iteration step, the total number of memory transfers is linear. In step 3 the number of memory transfers made in the i-th iteration step is proportional to the number of elements $(2^{i+2}-1)$ considered, thus it is proportional to $\sum_{i=0}^{\log n} M \cdot (2^{i+2}-1)$ for some constant $M>0$. Since step 4 is linear due to the result about the standard selection algorithm, the total cost is $0(n)$.

We have obtained

THEOREM 5

There is an algorithm producing the partial order $S_{n/2}^{n/2}$ simultaneously in minimal space, linear time and a linear number of memory transfers.

4. TIME-EFFICIENT ALGORITHMS IN LIMITED SPACE

In the previous section the analysis of two algorithms for the $SEC(n)$ problem has been presented. The *SECMINSPACE* algorithm required minimum (constant) space, $2n-3$ binary comparisons and $0(n)$ memery transfers. The *SEC2* algorithm required a minimum number $n+\lceil \log n \rceil - 2$ of binary comparisons, $1/2 \log^2 n + \Theta(\log n)$ space and $0(n)$ memory transfers. In addition to this, space otimal and time optimal algorithms for $SEC(n)$ problem has been analyzed.

This section is devoted to the solution of the following question: what is the time complexity of the problem under some space restriction? Such a kind of time characterization of the problem for each space constraint is called continuous space-time trade-off.

As a first result of this kind, a space-time trade off for the $SEC(n)$ problem is presented. The essence of this result is exhibited by the following algorithm in limited space. Let us have 4 memory cells $A[1..4]$ at our disposal. The computation starts by reading two elements, comparing them and locating them into $A[1..2]$ in ascending order. The computation then proceeds in $(n-2)/2$ cycle by reading and comparing another two input elements, comparing the greater input with $A[1]$ and placing the greater one into $A[1]$, then comparing the candidates for the second element and placing the greater one into $A[2]$. The number of comparisons done by this algorithm is $1+3\cdot(n-2)/2 = 3n/2-2$.

We can generalize this idea to obtain an upper bound on a continuous space-time trade-off result.

THEOREM 6

If M is an integer, $M \geq 4$, then there is an algorithm producing S^1_{n-2} using M memory cells and making

$$n\cdot(1+1/g(M))+\sqrt{2\cdot M}+0(1)$$

binary comparisons where $f(M) = \lfloor(\sqrt{8\cdot M-7}-1)/2\rfloor$ and

$$g(M) = 2^{f(M)} - \lfloor 2^{(f^2(M)+3\cdot f(M)-2\cdot M)/2}\rfloor - 1.$$

Proof

Consider the following algorithm:

procedure *SECTRADEOFF* (M, n);
begin
comment consider $n = s\cdot g(M)+2^{f(M)}$;

1. Find an integer $i\in\langle 2, \log n\rangle$ such that $i^2/2+i/2+1 \leq M < i^2/2+3\cdot i/2+2$;
2. Create a partial order of the type S^0_i from 2^i elements using memory cells $A[1], \ldots, A[M]$ in the same way as in the $SEC1$ algorithm and place it into continuous memory cells $A[1..i+1]$ with the center in $A[1]$;

repeat

i. Create a partial order of the type S^0_i from $g(M)$ elements using $A[i+2..M]$ memory cells in the same way as in $SEC1$ algorithm and place it into memory cells $A[i+2..2i+2]$ with the center in $A[i+2]$;

```
    ii. if A[1] < A[i+2] then
          for j ← 1 to i+1 do
            exchange (A[j], A[j+i+1]);
        if A[i+1] < A[i+2] then
          exchange A([i+1], A[i+2])
  until the input tape is empty;
  A[2] ← MAXIMUM {A[2], ..., A[i+1]};
  return (A[1], A[2])
end
```

The space used by this algorithm is M memory cells. To count the number of comparisons we see that step 2 needs $2^i - 1$ comparisons, step i needs $g(M) - 1$ and step ii needs 2 comparisons. Steps i and ii are repetead s times, that means $(n - 2^{f(M)})/g(M)$ times. Thus, the total number is

$$2^{f(M)} - 1 + \frac{n - 2^{f(M)}}{g(M)} \cdot (g(M) + 1) + f(M) - 1 =$$

$$= n \cdot (1 + 1/g(M)) - 2^{f(M)}/g(M) + f(M) - 2$$

and since

$$2^{f(M)}/g(M) < 2$$

we obtain the result.
The case $n \neq s \cdot g(M) + 2^{f(M)}$ for some $s \geq 1$ is obtained from enlarging the set of n elements by additional $-\infty$ elements to obtain a "new" set on which the *SECTRADEOFF* algorithm can be applied. "Superfluous" comparisons with $-\infty$ elements are not made and thus are not counted. □

We close by deriving lower bounds on the continuous space-time trade-off.

THEOREM 7

Let $M = s^2/2 + s/2 + 1$ for some $s \geq 2$ and let $n = k \cdot 2^s$ for some $k > 1$. Each algorithm producing S^1_{n-2} in space M requires at least

$$n + n/2^{s-1} + s - 0(1)$$

binary comparisons.

Proof

To prove this assertion, an adversary argument is used. We construct an adversary which can be viewed as an algorithm working on an acyclic directed graph which has been defined in the section 3.2. The lower bound can be achieved by computing the minimal number of edges entering terminal nodes of paths longer than two, over all algorithms determining S^1_{n-2}. It can be ob-

served that there are terminal nodes with either one or two entering edges. We compute

$$T = \underset{\mathcal{A}}{\text{MINIMUM}} \{p/q \mid p+q = n-2,\ q(p) \text{ is the number of terminal nodes with one (two) entering edge } (s) \text{ for an algorithm } \mathcal{A} \text{ determining } S^1_{n-2} \text{ in space } s^2/2+s/2+1,\ s \geq 2, \text{ and } n = k \cdot 2^s \text{ for } k > 1\}.$$

Following the idea of the proof of Lemma 2 we obtain

$$T \geq \frac{n/2^{s-1}+s-3}{n(1-1/2^{s-1})-s+2}$$

wnich yields the assertion of the Theorem. □

4.2 (Space-time trade-off for $S^{pn}_{(1-p)n}$, $0<p<n/2$) Consider the problem of producing the partial order $S^{pn}_{(1-p)n}$ for $0<p<1/2$. Dobkin and Munro (78) showed that only a small increase in space is necessary to make this problem linear in the number of binary comparisons, as it is shown in the following Theorem.

THEOREM 8

For each rational $r>0$ there is an algorithm producing the partial order $S^{pn}_{(1-p)n}$ for $0<p<1/2$ which requires a linear number of binary comparisons, a linear number of memory transfers and $(p+r)n$ memory cells.

Proof

The upper bound is obtained by the following algorithm. Extra space is used to eliminate r.n elements at each step of the computation using a standard linear time selection algorithm to find $S^{rn}_{(p+r)n}$. After $\frac{1}{r}$ steps the computation terminates with $c \cdot (p+r) \cdot n$ binary comparisons made in one step for some integer constant $c>0$. The linearity in the number of memory transfers is obtained by the same argument as in Theorem 6. □

Theorem 8 and Theorem 4 can be combined to obtain a continuous space-time trade-off.

THEOREM 9

Let $f(n) = o(n)$. Then for each $p: 0<p<1/2$ there is an integer $c_p>0$ such that each algorithm producing the partial order $S^{pn}_{(1-p)n}$ in space $p \cdot n + f(n)$ requires at least $c_p \cdot n \cdot \log(n/f(n)) - O(n)$ binary comparisons.

5. Conclusions

We investigated the space-time trade-off in producing partial orders with center. We discussed the efficiency of time optimal algorithms in limited space for these problems by determining their memory transfer complexity. Among other results we have shown that the number of comparisons is a suitable criterion for time complexity of producing partial orders with centre even in limited space.

Other computational models have been considered relative to producing partial orders in limited space. Multipass algorithms have been discussed by Paterson and Munro (78) and Ružička (79). Here the trade-off between the amount of internal space available and the number of passes over the input tape required is studied. On-line algorithms have been studied by Ružička (81). At each step of computation these algorithms produce partial orders with center for all processed elements.

REFERENCES

Dobkin, D. P., and *J. I. Munro* – Time and space bounds for selection problems. Lecture Notes in Computer Science 62. Springer – Verlag, 192 – 204, 1978.

Knuth D. E. – The art of computer programming. Vol. 3., Sorting and searching. Reading, Mass., Addison – Wesley, 1973

Munro, J. I., and *M. S. Paterson* – Selection and sorting with limited storage. J. of computer and system sciences *13*, 184 – 199, 1976.

Paterson, M. S. Schönhage, A., and *Pippenger, N.* – Finding the Median. 19-th Annual Symposium on Foundations of computer science, Ann Arbor, Oct. 16 – 18, 253 – 258, 1978.

Ružička, P. – Space-optimal multipass algorithms for selection. Send for publication. 1979.

Ružička, P. – Bounds for on-line selection, Kybernetika, Vol. 17,, No. 2., 147 – 157, 1981.

VVS, Dubravska 3
88531 Bratislava, Czechoslovakia