Information Processing Letters 109 (2009) 171–174

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Optimal algorithms for the average-constrained maximum-sum segment problem

Chih-Huai Cheng [a], Hsiao-Fei Liu [a], Kun-Mao Chao [a,b,c,*]

[a] Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106
[b] Graduate Institute of Biomedical Electronics and Bioinformatics, National Taiwan University, Taipei, Taiwan 106
[c] Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan 106

ARTICLE INFO

Article history:
Received 26 March 2008
Received in revised form 26 July 2008
Available online 25 September 2008
Communicated by F.Y.L. Chin

Keywords:
Association rules
Average
Maximum-sum segment
Algorithms

ABSTRACT

Given a real number sequence $A = (a_1, a_2, \ldots, a_n)$, an average lower bound L, and an average upper bound U, the Average-Constrained Maximum-Sum Segment problem is to locate a segment $A(i, j) = (a_i, a_{i+1}, \ldots, a_j)$ that maximizes $\sum_{i \leqslant k \leqslant j} a_k$ subject to $L \leqslant (\sum_{i \leqslant k \leqslant j} a_k)/(j - i + 1) \leqslant U$. In this paper, we give an $O(n)$-time algorithm for the case where the average upper bound is ineffective, i.e., $U = \infty$. On the other hand, we prove that the time complexity of the problem with an effective average upper bound is $\Omega(n \log n)$ even if the average lower bound is ineffective, i.e., $L = -\infty$.

1. Introduction

Given a real number sequence $A = (a_1, a_2, \ldots, a_n)$, define the length, sum, and average of a segment $A(i, j) = (a_i, \ldots, a_j)$ to be $j - i + 1$, $\sum_{h=i}^{j} a_h$, and $(\sum_{h=i}^{j} a_h)/(j - i + 1)$, respectively. The Maximum-Sum Segment problem, which finds a segment maximizing the sum, is widely formulated in pattern recognition [16,22], image processing [15], biological sequence analysis [1,13,17,20,23,25], and data mining [14,15]. It was first surveyed by Bentley in his "Programming Pearls" column of CACM [6,7] and is linear-time solvable using Kadane's algorithm [6]. Since then, there have been many variants proposed. The k Maximum-Sum Segments problem [3–5,10,12,18,19] is to locate the k segments whose sums are the k largest among all possible sums, and is solvable in $O(n + k)$ time [10,21]. The Range Maximum-Sum Segment Query (RMSQ) problem is to preprocess the input sequence such that any range maximum-sum segment query can be answered quickly, where a range maximum-sum segment query specifies two intervals $[i, j]$ and $[k, l]$ and the goal is to find a segment $A(x, y)$ with maximum sum subject to $i \leqslant x \leqslant j$ and $k \leqslant y \leqslant \ell$. Chen and Chao [11] showed that the RMSQ problem can be solved in $O(n)$ preprocessing time and $O(1)$ time per query. The Length-Constrained Maximum-Sum Segment problem is to find the maximum-sum segment among all segments satisfying the given length constraints and is solvable in $O(n)$ time [9,13,20].

For certain applications in the biological sequence analysis, it has been shown that maximizing the sum of a segment sometimes yields a lengthy segment consisting of poor regions [2,24]. To guarantee the quality of the target segment, its average should be taken into account. In this paper, we consider the Average-Constrained Maximum-Sum Segment (ACMS) problem, which is to find the maximum-sum segment among all segments satisfying the given average lower bound and upper bound. Specifically, given are a sequence $A = (a_1, a_2, \ldots, a_n)$ of real numbers, an average lower bound L, and an average upper bound U. Let $A(i, j)$ denote the segment $(a_i, a_{i+1}, \ldots, a_j)$, and let $S(i, j)$ denote the sum of $A(i, j)$. The Average-Constrained Maximum-Sum Segment problem is to find a segment $A(i, j)$ maximizing $S(i, j)$ subject to $L \leqslant S(i, j)/(j - i + 1) \leqslant U$. In Section 4 we show that Bernholt et al.'s result [9] implies an $O(n \log n)$ time

* Corresponding author at: Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106.
E-mail address: kmchao@csie.ntu.edu.tw (K.-M. Chao).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2008.09.024

algorithm for this problem. We also give an $O(n)$-time algorithm for the case where only an average lower bound is imposed and prove that no algorithm can cope with an effective average upper bound in $o(n \log n)$ time even if the average lower bound is ineffective.

2. An $O(n)$-time algorithm for the ACMS problem with an ineffective average upper bound

In this section, we assume that the average upper bound U is ineffective (i.e., $U = \infty$) and give a linear time algorithm for finding the maximum-sum segment among segments with averages no less than the average lower bound L. For ease of exposition, we assume $L > 0$ in subsequent discussion. This restriction can be overcome as follows. Let $a_{i^*} = \max_{i=1}^{n} a_i$. If $a_{i^*} < L \leqslant 0$, then report that there is no feasible solution. If $L \leqslant a_{i^*} \leqslant 0$, then output $A(i^*, i^*)$. If $L \leqslant 0$ and $a_{i^*} > 0$, then it is safe to output the segment that maximizes the sum without worrying about the average lower bound.

Let P denote the prefix-sum array of A, where $P[i] = \sum_{1 \leqslant k \leqslant i} a_k$ for $i = 1, 2, \ldots, n$ and $P[0] = 0$. It is easy to see that $S(i, j) = P[j] - P[i-1]$. Let $B = \langle a_1 - L, a_2 - L, \ldots, a_n - L \rangle$, and denote its prefix-sum array by P_B. Since $S(p, q) = P_B[q] - P_B[p-1] + (q - p + 1) \times L$, $S(p, q)/(q - p + 1) \geqslant L$ if and only if $P_B[q] \geqslant P_B[p-1]$.

Definition 1. Given an index $q > 0$, an index p is said to be a "partner" of q if and only if $p < q$ and $P_B[p] \leqslant P_B[q]$.

Definition 2. Given an index $q > 0$ with at least one partner, let $s^* = \max\{S(i+1, q)\colon 0 \leqslant i < q \text{ and } P_B[i] \leqslant P_B[q]\}$. We say that p is the "best partner" of q if and only if p is the largest partner of q with $S(p+1, q) = s^*$. Let $\pi(q)$ denote the best partner of q.

Our strategy is that for every index i, we find the segment having the largest sum among all the segments which end at i and satisfy the average constraint. Obviously, the maximum-sum segment satisfying an average constraint is the largest one of them. Therefore, we have the optimal solution when the best partner of each index is computed. It is easy to see that not every index is a best partner for some other index. With careful observations, we eliminate the indices that are not the best partner by Lemma 1.

Lemma 1. *For any index q, $\pi(q) \neq v$ if $P_B[u] \leqslant P_B[v]$ and $0 \leqslant u < v < q$.*

Proof. Suppose for contradiction that $\pi(q) = v$. It follows that $A(v+1, q)$ is the maximum-sum segment among all the segments satisfying the average constraint and ending at q. It is easy to see that $S(u+1, q) = P_B[q] - P_B[u] + (q - u) \times L > P_B[q] - P_B[v] + (q - v) \times L = S(v+1, q)$. Moreover, we know $P_B[u] \leqslant P_B[v] \leqslant P_B[q]$ by definition. Therefore, u is a partner of q better than v, which leads to a contradiction. □

To compute the best partner of each index, we construct a list C to record candidates of best partners as follows. In the beginning, the list C is empty. We then scan P and P_B, two prefix-sum arrays, on the fly and verify whether i should be added to C for every index i. Let j denote the latest index added to C. By Lemma 1, we know that i may be a best partner candidate for indices greater than i only when $P_B[i] < P_B[j]$. Thus, if $P_B[i] \geqslant P_B[j]$, we can bypass i; otherwise, we distinguish between the following two cases: $P[i] > P[j]$ and $P[i] \leqslant P[j]$. If $P[i] > P[j]$, we insert i at the end of C. If $P[i] \leqslant P[j]$, we first delete all indices x in C with $P[x] \geqslant P[i]$ and then insert i at the end of C. The deleted indices are stored in list R_i. The detailed descriptions of the construction are given in Algorithm 1. For all $i = 0, 1, \ldots, n$, let C_i denote the contents of C immediately after the ith iteration of the for-loop, $N_i = |C_i|$, and $c_{i,j}$ denote the jth element in C_i. It is easy to verify the following four properties.

Algorithm 1. ComputeCandidateL(A, L)

```
 1: B ← ⟨a_1 − L, a_2 − L, ..., a_n − L⟩;
 2: P ← the prefix-sum array of A;
 3: P_B ← the prefix-sum array of B;
 4: C ← an empty list;
 5: create n empty lists R_1, R_2, ..., R_n;
 6: initialize array HaveParners[1..n] with 0's;
 7: for i ← 0 to n do
 8:    j ← the last element of C;
 9:    if j = NULL or P_B[i] < P_B[j] then
10:       while j ≠ NULL and P[i] ⩽ P[j] do
11:          delete j from C;
12:          insert j at the beginning of R_i;
13:          j ← the last element of C;
14:       end while
15:       insert i at the end of C;
16:    else
17:       HavePartner[i] ← 1;
18:    end if
19: end for
20: output C, P, P_B, HavePartner[1..n], and R_1, R_2, ..., R_n;
```

Property 1. The values of $P_B[c_{i,1}], P_B[c_{i,2}], \ldots, P_B[c_{i,N_i}]$ are in decreasing order, and the values of $P[c_{i,1}], P[c_{i,2}], \ldots, P[c_{i,N_i}]$ are in increasing order.

Property 2. If $i \in C_k$ and $i \leqslant j < k$, then C_i is a prefix of C_j and ends with i, that is, $C_i = \langle c_{j,1}, c_{j,2}, \ldots, c_{j,N_i} \rangle$ and $c_{j,N_i} = i$.

Property 3. HavePartner$[j] = 1$ if and only if $\exists i < j$ such that $P_B[i] \leqslant P_B[j]$.

Property 4. $(C_i \backslash \{i\}) \cdot R_i = C_{i-1}$, where the symbol "$\cdot$" means concatenation.

We now analyze the time complexity. The total number of operations of the algorithm is clearly bounded by $O(n)$ except for the while-loop body in lines 10–14. In each iteration of the while-loop, we remove exactly one element from the end of C. Moreover, each index is inserted into C at most once. Therefore, the while loop totally have at most $O(n)$ iterations, and we have the following theorem.

Theorem 1. ComputeCandidateL(A, L) *runs in $O(n)$ time.*

The next lemma ensures that the best partner of any index q must reside in C_{q-1}. Therefore, by Property 1, finding $\pi(q)$ is equivalent to finding the smallest index $i \in C_{q-1}$ such that $P_B[i] \leqslant P_B[q]$.

Lemma 2. *For any index q, if $\pi(q)$ exists, it must be in C_{q-1}.*

Proof. Let $p = \pi(q)$. We first prove that p will be inserted into C in the pth iteration of the for-loop, i.e., $p \in C_p$. Suppose for contradiction that $p \notin C_p$. It follows that there exists $j < p$ such that $P_B[j] \leqslant P_B[p]$. By Lemma 1, $p \neq \pi(q)$, a contradiction. We next prove that p remains in C_i for all $i \in [p+1, q-1]$. Suppose for contradiction that p is removed from C in the ith iteration for some $i \in [p+1, q-1]$. Note that deletion of p occurs only when $P[i] < P[p]$ and $P_B[i] < P_B[p]$ are both satisfied. It implies that i is a partner better than p with respect to q, a contradiction. □

We next prove that $A(\pi(p)+1, p)$ is not an optimal solution if there exists some $q > p$ such that $\pi(q) < \pi(p)$.

Lemma 3. *$S(\pi(q)+1, q) > S(\pi(p)+1, p)$ when $\pi(q) < \pi(p)$ and $p < q$.*

Proof. Consider the following two cases. Case 1: $P[p] \leqslant P[q]$. In this case, $P[\pi(p)] > P[\pi(q)]$ for otherwise $\pi(p)$ is a better partner for q than $\pi(q)$, a contradiction. Thus, we have $S(\pi(q)+1, q) = P[q] - P[\pi(q)] > P[p] - P[\pi(p)] = S(\pi(p)+1, p)$. Case 2: $P[p] > P[q]$. In this case, $P_B[p] > P_B[q] \geqslant P_B[\pi(q)]$. Therefore, $P[\pi(p)] \leqslant P[\pi(q)]$ for otherwise $\pi(q)$ is a better partner for p than $\pi(p)$, a contradiction. It follows that $P_B[q] \geqslant P_B[\pi(q)] > P_B[\pi(p)]$, so $\pi(p)$ is a partner of q. Furthermore, since $P[\pi(p)] \leqslant P[\pi(q)]$ and $\pi(p) > \pi(q)$, $\pi(p)$ is a better partner for q than $\pi(q)$, a contradiction. Thus, Case 2 is impossible to happen. □

A linear-time algorithm for finding the maximum-sum segment satisfying an average lower bound L is given in Algorithm 2. We first call procedure ComputeCandidateL to compute C, P, P_B, HavePartner[1..n], and $R_1, R_2, \ldots, R_n$. Initialize variables r and l with $n+1$ and n, respectively, so that $C = C_l$ and $l \leqslant r-1$ hold in the beginning. We then compute $\pi(r)$ for r from n to 1. Whenever the value of r is decremented by one, we will accordingly update l and C such that $l \leqslant r-1$ and $C = C_l$ still hold. When computing $\pi(r)$, if $l = r-1$, then by Lemma 2, $\pi(r) \in C_l$. Thus we can find $\pi(r)$ by scanning the list $C = C_l = C_{r-1}$ from the end until the smallest index $i \in C$ such that $P_B[i] \leqslant P_B[r]$ is located and then reset l to $\pi(r) = i$. Otherwise, we have $l = \pi(q)$ for some $q > r$. By Property 2, $C_{\pi(q)} = \langle c_{r-1,1}, c_{r-1,2}, \ldots, c_{r-1,N_{\pi(q)}} \rangle$ is a prefix of C_{r-1}, where $c_{r-1,N_{\pi(q)}} = \pi(q) = l$. By Lemma 2, $\pi(r) \in C_{r-1}$. Thus, if we cannot find $\pi(r)$ in C_l, then $\pi(r) > \pi(q) = l$. By Lemma 3, $A(\pi(r)+1, r)$ is not the optimal solution and we can bypass the computation of $\pi(r)$. In summary, we shall first scan the list $C = C_l$ from the end until finding the smallest index $i \in C$ such that $P_B[i] \leqslant P_B[r]$. If $i \neq$ NULL, then $\pi(r) = i$ and we reset l to $i = \pi(r)$; otherwise, $A(\pi(r)+1, r)$ is not the optimal solution and we bypass the computation of $\pi(r)$.

We next analyze the running time. By Lemma 1, the procedure ComputeCandidateL takes $O(n)$ time. It is clear that the total time spent on updating C is bounded by $O(n)$. When scanning the list $C = C_l$ to find the smallest candidate $i \in C$ such that $P_B[i] \leqslant P_B[r]$, if $P_B[l] > P_B[r]$, then $i =$ NULL and only one element is scanned. If $P_B[l] \leqslant P_B[r]$, then at most $k = l - i + 2$ elements are scanned and l is reset to $i = l - k + 2$. That is, the value of l will decrease by at least $k-2$ if k elements in the list $C = C_l$ are scanned. Initially $l = n$, so the total time spent on scanning C is bounded by $O(n)$. It follows that the running time of MaxSumAvgL(A, L) is $O(n)$.

Algorithm 2. MaxSumAvgL(A, L)

```
1:  C, P, P_B, HavePartner[1..n], and
        R_1, R_2, ..., R_n ← ComputeCandidate(A, L);
2:  S_max ← −∞;
3:  r ← n + 1;
4:  l ← r − 1;
5:  while r > 1 do
6:    repeat
7:      r ← r − 1;
8:      if r − 1 < l then
9:        l ← r − 1;
10:       C ← (C\{l + 1}) · R_{l+1};
          /* Update C such that C = C_l. */
11:     end if
12:   until HavePartner[r] = 1 or r = 1
13:   scan C from the end until the smallest index i ∈ C
        such that P_B[i] ⩽ P_B[r] is located;
14:   /* If i = NULL, then π(r) > l; otherwise, π(r) = i. */
15:   if i ≠ NULL then
16:     if P[r] − P[i] > S_max then
17:       S_max ← P[r] − P[i];
18:       (α*, β*) ← (i + 1, r);
19:     end if
20:     while l > i do
21:       l ← l − 1;
22:       C ← (C\{l + 1}) · R_{l+1};
          /* Update C such that C = C_l. */
23:     end while
24:   end if
25: end while
26: output A(α*, β*);
```

Theorem 2. *The* Average-Constrained Maximum-Sum Segment *problem can be solved in $O(n)$ time if the upper bound U is ineffective.*

3. An $\Omega(n \log n)$-time lower bound for the ACMS problem with an effective average upper bound

In this section, we assume that the average lower bound is ineffective and prove that the problem of finding a segment $A(i, j)$ maximizing $S(i, j)$ subject to $(\sum_{k=i}^{j} a_k)/(j-i+1) \leqslant U$ has an $\Omega(n \log n)$ lower bound.

Definition 3 *(Element uniqueness problem).* Given a number sequence $X = (x_1, x_2, \ldots, x_n)$, determine whether there exist $i \neq j$ such that $x_i = x_j$.

Lemma 4. *(See Ben-Or [8].) The* Element Uniqueness *problem has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model.*

Theorem 3. *The problem of finding the maximum-sum segment among all segments with averages no greater than U has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model.*

Proof. We reduce the Element Uniqueness problem to our problem in $O(n)$ time as follows. Given an Element Uniqueness problem instance $X = (x_1, x_2, \ldots, x_n)$, without loss of generality, assume that $x_i \neq 0$ for $i = 1, 2, \ldots, n$. Let $A = (a_1, a_2, \ldots, a_n)$ and $U = 0$, where $a_1 = x_1$ and $a_i = x_i - x_{i-1}$ for $i = 2, 3, \ldots, n$. Let $x_0 = 0$. Then for all $1 \leqslant i \leqslant j \leqslant n$, we have $A(i, j) = x_j - x_{i-1}$. It follows that

there exists $i < j$ such that $x_i = x_j$
$\Leftrightarrow$ A has a zero sum segment
$\Leftrightarrow$ the maximum-sum segment of A satisfying average $\leqslant U$ has zero sum. □

4. An $O(n \log n)$-time algorithm for the ACMS problem

In this section we demonstrate why the Average-Constrained Maximum-Sum Segment problem can be solved in $O(n \log n)$ time by using Bernholt et al.'s algorithm [9], which is the best possible time bound for the general case by Theorem 3. Given a sequence $A = (a_1, a_2, \ldots, a_n)$ of real numbers, a set of k linear constraints on the length and sum of the located segment, and a score function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, Bernholt et al. [9] showed that locating a segment $A(i, j) = (a_i, a_{i+1}, \ldots, a_j)$ maximizing $f(j - i + 1, S(i, j))$ subject to the given constraints can be done in $O(k \log k + k \cdot n \log n)$ time as long as the score function is quasiconvex. A function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is said to be *quasiconvex* if and only if for all points $u, v \in \mathbb{R} \times \mathbb{R}$ and all $\lambda \in [0, 1]$, we have $f(\lambda \cdot u + (1 - \lambda) \cdot v) \leqslant \max\{f(u), f(v)\}$. Define the score function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ by letting $f(\ell, s) = s$ for each point $(\ell, s) \in \mathbb{R} \times \mathbb{R}$. The Average-Constrained Maximum-Sum Segment problem is equivalent to finding a segment $A(i, j) = (a_i, a_{i+1}, \ldots, a_j)$ maximizing $f(j - i + 1, S(i, j))$ subject to the following two linear constraints on the sum and length of $A(i, j)$: $S(i, j) \leqslant (j - i + 1) \cdot U$ and $S(i, j) \geqslant (j - i + 1) \cdot L$. Since the function $f(\ell, s) = s$ is quasiconvex, by Bernholt et al.'s algorithm, we have the following theorem.

Theorem 4. *(See Bernholt et al. [9].) The* Average-Constrained Maximum-Sum Segment *problem can be solved in $O(n \log n)$ time for the general case.*

Acknowledgements

We thank the anonymous referees for their careful reading of the manuscript. We thank Peng-An Chen for helpful discussions. Chih-Huai Cheng, Hsiao-Fei Liu, and Kun-Mao Chao were supported in part by NSC grants 95-2221-E-002-126-MY3 and 96-2221-E-002-034 from the National Science Council, Taiwan.

References

[1] L. Allison, Longest biased interval and longest non-negative sum interval, Bioinformatics 19 (2003) 1294–1295.
[2] A. Arslan, Ö. Eğecioğlu, P. Pevzner, A new approach to sequence comparison: Normalized sequence alignment, Bioinformatics 17 (2001) 327–337.
[3] S.E. Bae, T. Takaoka, Algorithms for the problem of k maximum sums and a VLSI algorithm for the k maximum subarrays problem, in: Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004, pp. 247–253.
[4] S.E. Bae, T. Takaoka, Improved algorithms for the k-maximum subarray problem for small K, in: Proceedings of the 11th Annual International Computing and Combinatorics Conference, 2005, pp. 621–631.
[5] F. Bengtsson, J. Chen, Efficient algorithms for k maximum sums, in: Proceedings of the 15th International Symposium on Algorithms and Computation, 2004, pp. 137–148.
[6] J. Bentley, Programming pearls: Algorithm design techniques, Communications of the ACM (1984) 865–871.
[7] J. Bentley, Programming pearls: Perspective on performance, Communications of the ACM (1984) 1087–1092.
[8] M. Ben-Or, Lower bounds for algebraic computation trees, in: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 80–86.
[9] T. Bernholt, F. Eisenbrand, T. Hofmeister, A geometric framework for solving subsequence problems in computational biology efficiently, in: Proceeding of the 23rd Annual Symposium on Computational Geometry, 2007, pp. 310–318.
[10] G.S. Brodal, A.G. Jørgensen, A linear time algorithm for the k maximal sums problem, in: Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science, 2007, pp. 442–453.
[11] K.-Y. Chen, K.-M. Chao, On the range maximum-sum segment query problem, Discrete Applied Mathematics 155 (2007) 2043–2052.
[12] C.-H. Cheng, K.-Y. Chen, W.-C. Tien, K.-M. Chao, Improved algorithms for the k maximum-sums problems, Theoretical Computer Science 362 (1–3) (2006) 162–170.
[13] T.-H. Fan, S. Lee, H.-I. Lu, T.-S. Tsou, T.-C. Wang, A. Yaom An optimal algorithm for maximum-sum segment and its application in bioinformatics, in: Proceedings of the Eighth International Conference on Implementation and Application of Automata, 2003, pp. 251–257.
[14] T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Mining optimized association rules for numeric attributes, Journal of Computer and System Science 58 (1) (1999) 1–12.
[15] T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Data mining with optimized two-dimensional association rules, ACM Transactions on Database Systems 26 (2001) 179–213.
[16] U. Grenander, Pattern Analysis, Springer-Verlag, New York, 1978.
[17] X. Huang, An algorithm for identifying regions of a DNA sequence that satisfy a content requirement, Computer Applications in the Biosciences 10 (1994) 219–225.
[18] T.-C. Lin, D.T. Lee, Efficient algorithm for the sum selection problem and k maximum sums problem, in: Proceedings of the 17th Annual International Symposium on Algorithms and Computation, 2006, pp. 460–473.
[19] T.-C. Lin, D.T. Lee, Randomized algorithm for the sum selection problem, Theoretical Computer Science 377 (2007) 151–156.
[20] Y.-L. Lin, T. Jiang, K.-M. Chao, Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis, Journal of Computer and System Sciences 65 (2002) 570–586.
[21] H.-F. Liu, K.-M. Chao, Algorithms for finding the weight-constrained k longest paths in a tree and the length-constrained k maximum-sum segments of a sequence, Theoretical Computer Science, doi:10.1016/j.tcs.2008.06.052.
[22] K. Perumalla, N. Deo, Parallel algorithms for maximum subsequence and maximum subarray, Parallel Processing Letters 5 (1995) 367–373.
[23] W.L. Ruzzo, M. Tompa, A linear time algorithm for finding all maximal scoring subsequences, in: Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology, 1999, pp. 234–241.
[24] N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller, R.C. Hardison, Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions, Nucleic Acids Research 19 (1999) 3899–3910.
[25] L. Wang, Y. Xu, SEGID: Identifying interesting segments in (multiple) sequence alignments, Bioinformatics 19 (2003) 297–298.