Fast Algorithms for Finding Disjoint Subsequences with Extremal Densities*

Anders Bergkvist

Department of Molecular Biology, Göteborg University, P.O. Box 462,

40530 Göteborg, Sweden,

abk@molbio.gu.se

Peter Damaschke†

School of Computer Science and Engineering, Chalmers University,

41296 Göteborg, Sweden,

ptr@cs.chalmers.se

Abstract

We derive fast algorithms for the following problem: Given a set of n points on the real line and two parameters s and p, find s disjoint intervals of maximum total length that contain at most p of the given points. Our main contribution consists of algorithms whose time bounds improve upon a straightforward dynamic programming algorithm, in the relevant case that input size n is much bigger than parameters s and p. These results are achieved by selecting a few candidate intervals that are provably sufficient for building an optimal solution via dynamic programming. As a byproduct of this idea we improve an algorithm for a similar subsequence problem of Chen, Lu and Tang (2005). The problems are motivated by the search for significant patterns in biological data. Finally we propose several heuristics that further reduce the time complexity in typical instances. One of them leads to an apparently open subsequence sum problem of independent interest.

*Part of the results have been presented in preliminary form in: Proceedings of the *16th International Symposium on Algorithms and Computation ISAAC 2005*, *Lecture Notes in Computer Science* (Springer), vol. 3827, pp. 714-723.

†Second author is the correspondence author. Contact information: Email: ptr@cs.chalmers.se. Fax: 0046 31 165655. Phone: 0046 31 772 5405.

Keywords: holes in data, range prediction, protein torsion angle, protein structure prediction, dynamic programming, selection algorithms, time complexity

Summary: We give a scheme for fast algorithms for problems where disjoint subsequences are sought that optimize the sum of certain characteristic values (lengths, densities). Such problems appear naturally in the analysis of biological sequences and other linear data sets, such as projections of slices from scatterplots to one axis. We are using them for range predictions of correlated data. (Our specific motivation is the prediction of protein backbone torsion angles from nuclear magnetic resonance chemical shifts, however the present paper is mainly devoted to the algorithmic problems.) We study a problem called DIMaxL in depth, where we seek s intervals containing at most p points from a given set of n points. Dynamic programming algorithms with a preselection phase give the currently best time bounds for this problem if s, p are small compared to n. We also propose several heuristics and give evidence that they save even more time in typical irregular instances. Some open problems (improved worst-case bounds, complexity of two-dimensional analogues, impact of our heuristics, and another computational problem resulting from one of our heuristics) are left for further theoretical and experimental research.

1 Problem Statement and Results

Finding large empty regions (big holes) in data sets is interesting in data mining and statistical inference. The problem in its most general form is described by a space X, a family $\mathcal{F}$ of subsets of X, an additive size function that assigns a positive real number to every nonempty set in $\mathcal{F}$, and a finite set D of n data points. The goal is to find the largest set in $\mathcal{F}$ disjoint to D. In a commonly considered case, X is a finite-dimensional real vector space, $\mathcal{F}$ is the family of axis-parallel boxes, and the size of a box is its volume.

It is natural to generalize the problem in two directions: We are interested in several disjoint big holes in X rather than only one largest hole, and we tolerate a limited number of data points in these holes. That is, we introduce two parameters s and p, and look for s disjoint sets with maximum total size which may contain up to p data points.

Already in two dimensions, i.e., for axis-parallel rectangles in the plane, we do not know of complexity results for this parameterized problem. It is even unclear whether the problem is polynomial. However, in the present paper we study the one-dimensional case where $\mathcal{F}$ is the set of intervals contained in a fixed finite-length interval X, and where the size measure is the length of intervals. This one-dimensional case is already interesting for some applications, as we outline in Section 2.

Basically, the one-dimensional problem is easy to solve by dynamic programming, however the question of algorithms being as fast as possible for given n, s, p turns out to be nontrivial. This is the main subject of our paper.

Note that an optimal solution always consists of s open intervals with data points at their ends. (The term "open" means that the ends are excluded, i.e., not considered as points of the interval. We also remark that the two outermost intervals of a solution may reach the ends of X.) Therefore we get the following, more convenient formal description. In a sequence of positive real numbers $x_0, x_1, x_2, \ldots, x_n$ called *items*, we call any set of consecutive items $x_i, \ldots, x_j$ $(i \leq j)$ an *interval of length* $x_i + \ldots + x_j$. Sometimes we say "item x_i" for "item i with length x_i" if there is no confusion. Obviously, x_i (for $0 < i < n$) means the distance of the ith and $(i+1)$st data point in their natural ordering in X, while x_0 and x_n is the distance of the first and last data point to the left and right end, respectively, of X. Now our problem can be stated as follows:

Disjoint Intervals of Maximum Length (DIMaxL)

Given a sequence $x_0, x_1, x_2, \ldots, x_n$ and integers $s \geq 1$ and $p \geq 0$, find s pairwise disjoint intervals with a total of $s+p$ items and maximum total length.

A batch of remarks is in order here. If $x_i \geq 0$ for all i, an optimal solution will always use exactly $s+p$ items, otherwise we could add another item to some interval. Hence, the problem would not change if we wrote "at most $s+p$ items" in the specification. In our formulation with exactly $s+p$ items, the x_i can be arbitary real numbers, and an optimal solution may have to take some negative items, in order to connect large positive items to intervals. Also, a problem instance does not change if any constant is added to all x_i. On the other hand, an optimal solution may have chains of, say, t incident intervals. In such cases, a chain can be split arbitrarily in t other intervals without changing s, p and the total length. For certain instances, a trivial solution exists and can be verified in $O(n)$ time: If the $s+p$ largest items form (at most) s intervals, this is obviously an optimal solution. In particular, DIMaxL with $p = 0$ is a trivial problem. However, in general the $s+p$ largest items can be scattered in the sequence, and then the optimum is no longer obvious. Parts of an optimal solution may consist of medium-lenght items that are clustered, and then we have to find the best among many candidates.

We also mention the relation to independent sets in interval graphs. (We assume some background in graph algorithms. However the reader may skip this paragraph. The statements will not be used further, we just had to make sure that DIMaxL has not already been considered in that context, under a different name.) Given a sequence of numbers, consider the interval graph $G(p)$ with intervals of at most p items as vertices, where two vertices are adjacent if the intervals intersect. We sloppily identify intervals and vertices in $G(p)$. DIMaxL can be rephrased as a special case of finding a maximum weight independent set of s vertices in an interval graph which is given as an interval model. Although Independent Set and related problems in interval graphs and other intersection graphs are well studied (we refer to [2] for some special results and more pointers to the vast literature), we cannot directly make use of earlier results, since we have to apply the special features of our instances to make our algorithms as fast as possible. In our particular case, every vertex of an interval graph $G(p)$ belongs to at most p consecutive cliques, and the vertex weights are the interval lengths in the interval model. To our best knowledge there is no previous work on Weighted Independent Set in interval graphs with these special restrictions.

Organisation of the paper and results: In Section 2 we motivate DIMaxL by statistical inference tasks. Our particular bioinformatics motivation is discussed in more detail, but there may be more, given that our approach is rather general and conceptually simple. Section 3 is the core of the paper: We develop our basic algorithms for DIMaxL. A naive dynamic programming algorithm solves DIMaxL in $O(spn)$ time. By some preprocessing that selects possible intervals for an optimal solution, without missing any candidate, we can achieve a time bound $O(pn + s^2p^3)$. Since we have $s, p \ll n$ in statistically relevant cases, this alternative algorithm is then an improvement. As a byproduct we also improve an algorithm for a similar numerical subsequence problem: Define the density of an interval $x_i, \ldots, x_j$ to be the ratio $(x_i + \ldots + x_j)/(j - i + 1)$. Let us call the following problem that has been introduced and attacked in [7]

Disjoint Intervals of Maximum Density (DIMaxD)

Given a sequence $x_1, x_2, \ldots, x_n$ and integers $k \geq 1$ and $l \geq 1$, find k disjoint intervals, each with *at least* l items, so as to maximize the sum of their densities.

(Loosely speaking, our problem DIMaxL is looking for low-density regions in point sets, opposite to DIMaxD.) Chen, Lu and Tang [7] gave an $O(kln)$ time algorithm, as well as algorithms running in time $O(n)$ and $O(n + l^2)$ in the special cases $k = 2$ and $k = 3$, respectively. They explicitly asked if there is an $o(kln)$ algorithm for general k. We give an affirmative answer. We can solve DIMaxD in $O(ln + k^2l^2)$ time, using the same idea of candidate selection followed by dynamic programming on the small candidate set. Actually, this scheme would be applicable to any similar problems where one has to find a set of disjoint subsequences that optimizes some objective function. The main property we need from the problem is that the candidate intervals are of limited length.

In Section 4 we further elaborate on our approach for DIMaxL. For different ranges of parameters s, p we can further reduce the "n-free" term of our time bound. Although these extra improvements are small (third root of a parameter), we find the matter interesting from the algorithmic point of view: Better time bounds are achieved by suitable combinations of the two basic algorithms. While the analysis becomes somewhat tricky, implementation is not harder. Basically we treat short and long candidate intervals differently. We also expect that our current bounds are not yet the ultimate ones.

On the other hand, it is not clear whether the pn term can be reduced, too. In Section 5 we propose a few heuristics to avoid $p + 1$ independent candidate selection routines (one for every number of items in an interval). These heuristics do not remedy factor p in the worst-case bound, however we argue that they take effect in "usual" instances, i.e., that only malicious instances represent the worst case. One heuristic leads to an apparently open sum-selection problem that looks interesting in itself. We quest for a non-trivial subquadratic algorithm.

2 Motivations and Related Work

Estimating distributions in a space of numerical attributes from random samples is fundamental in data mining and statistical inference. Sparsely populated regions give evidence that certain ranges of attribute values or combinations thereof are unlikely, if not impossible, and statistically significant holes indicate systematic reasons which exclude such values, and can therefore be used to derive inference rules and models. Finding big holes (or "anti-clusters") in data sets is an established approach that complements the more common and more traditional clustering approaches. The algorithmic questions are well investigated both in theory and practice [1, 3, 6, 12, 18, 22]. But apparently very little has been done for the more general, two-parameter problem introduced in Section 1.

To motivate DIMaxL, suppose that we have sampled n data points (real numbers) from an unknown probability distribution on a finite-length real interval X, and we want to identify large sub-intervals of X with low probability density. Since there can be several disconnected holes, we choose some s and ask for the s disjoint holes with maximum total length. Parameter s must be chosen small compared to n: To discuss the extreme case, if $s = n + 1$ then we can obviously not conclude that the intervals between the data points (and the ends of X) all have low probability density. However we can conclude low density for the s largest intervals, if $s \ll n$ and if these intervals are significantly larger than the expected maximum length in a sample from a uniform distribution. The rationale for tolerating a certain small number $p \ll n$ of data points in holes is simply that even a small positive probability can actually yield a few data points. (More precisely, the number of data points follows a Poisson distribution.) Another reason is that data sets may contain outliers, e.g., due to measurement errors. In either case it is sensible to consider large, almost void intervals as holes, despite a few occasional data points.

Determining holes in one-dimensional data sets can constitute a useful step in the prediction of numerical attributes from others, if they are joint by complicated correlations. The general scenario is that certain objects are characterized by pairs of real values x, y (of whatever meaning), where x is easy to measure while it is hard or expensive to measure y. Furthermore, suppose we have a database with a sample of such pairs, giving data points (x, y) in the plane. When we get a new measured value x_0 belonging to a pair (x_0, y_0), we would like to predict y_0, or more modestly, to restrict the range of likely values y_0. For this prediction we consider a set of coordinates y of our sampled data points (x, y), namely those y belonging to points with coordinates x around the measured x_0. (The maximum difference $|x - x_0|$ may be related to the measurement accuracy, or be user-defined, depending on the sample size.) Under mild smoothness conditions on the underlying distribution from which the data points are sampled, we can conclude that, with high probability, y_0 is not in any of the big holes in this set. The approach can be readily extended to more dimensions. An "orthogonal" way to use holes for range predictions is to partition the data set into slices along the y-axis and to compute within each slice the big holes in the projection to the x-axis. Then we can for a measured x_0 discard such values y where x_0 is in a big hole in the slice of y. With either method we discard upon each measurement x_0 the true y_0 with some limited error probability only.

We became interested in the problem by a project aiming at the prediction of backbone torsion angles in proteins from measured nuclear magnetic resonance (NMR) chemical shifts. Determining protein structures is of fundamental importance in molecular biology, and NMR spectroscopy analysis is one of the most important experimental techniques to do this (only X-ray crystallography is of equal or greater importance). In a nutshell, the chemical shift is a measure for the resonance frequency of atoms in a strong external magnetic field. It is well-known that chemical shifts are sensitive indicators of local molecular structure since the effective magnetic field sensed by a resonating nucleus is modulated by its molecular, electromagnetic environment. Until recently, various prediction methods have been developed; refer to [5, 10, 20, 21, 27, 28, 29] for more background information. Nevertheless, the structure determination process is usually dominated by indirect techniques that require more advanced analysis than the simple assignment of NMR chemical shifts to corresponding atomic nuclei. The reason for this may be due to a low precision of structural predictions from chemical shifts. Even so, the low cost of chemical shift information compared to structural information from indirect NMR techniques, suggests an advantage of using measured chemical shifts for predicting structural parameters. Those structural parameters, such as torsion angle restraints at each residue, can then be input in existing computer programs for protein structure prediction. In addition, a more precise prediction heuristic for structural parameters from chemical shift may allow a more extensive direct use of chemical shifts for structure determination purposes.

The analytical form of dependencies between NMR chemical shifts and protein torsion angles is complicated. Several attempts to quantify these correlations include ab initio quantum mechanical calculations [29], empirically derived shielding surfaces [21], and a trained neural network [20]. Predefined classificatons of ϕ and ψ torsion angles (protein secondary structures) from chemical shifts have been predicted in [28] by a statistical method using averages of empirical distributions and sequence homology, based on a rather small sample of proteins.

The idea of our project is to predict again the torsion angles directly, now by exploiting the full information in the empirical distributions that are nowadays available from large public protein structure databases, in order to achieve the best possible prediction accuracy on a fine scale. For correct and efficient structure reconstruction of whole protein sequences it is crucial that the predicted local restraints satisfy two demands at the same time: they should exclude large ranges of torsion angle values, but also keep the unknown true torsion angles with large probability. The hole-based range prediction outlined above is a natural approach to accomplish these goals. It is conceptually simple but does not rely on simplifying assumptions about the distributions of chemical shifts versus torsion angles. Rather, we arrange the empirical distributions in scatterplots and take the patterns as they are. Although holes in scatterplots meet the eye, the hole recognition must run automatically in applications like this, since *many* instances have to be solved (see below). Thus it is desirable to have *fast* and easy-to-implement algorithms that discard unlikely torsion angle intervals. This situation stimulated the present algorithmic study.

On the application side, we have downloaded and arranged torsion angle and chemical shift data for several hundreds of proteins from databases PDB and BMRB. (More careful specifications of how the raw data have been processed shall appear in a planned separate study, as well as a systematic report of the results. This is beyond the scope of the present, purely methodological paper.) From a data source we get in principle 240 scatterplots, namely for each of 20 amino acids, six nuclei, and two torsion angles. We get a multiple of this number if different data sources are used. Given a value of some chemical shift we may consider data points in a stripe of width determined by, e.g., the measurement accuracy of a few ppm, and then take their projection to the torsion angle axis as a one-dimensional problem instance. One can see that the number of instances is really large.

In order to give at least an impression of these data we insert one representative example (Figure 1). It is obvious that intervals of unlikely angles, i.e., those containing a few sporadic points, differ a lot for various chemical shifts. Parameters s and p can be experimentally adjusted both to the typical patterns in the application and to the amount of data points. Anyway it will become clear that, since our algorithms work with dynamic programming, they also yield optimal solutions for all pairs of s, p smaller than the chosen parameters, hence the choice of (maximal) s, p is more a matter of time complexity. Since the horizontal stripes at various chemical shifts have at most 3 clearly identifiable holes (which is typical not only in Figure 1), we may fix $s = 3$. Parameter p may be chosen very small, too. However with a larger p, say some $p > 5$, the algorithms will also distinguish the sparsely populated interval between -180 and -90 degrees from the cluster at -60 degrees, if the chemical shift is around 55 ppm, without splitting that interval into several smaller artificial holes.

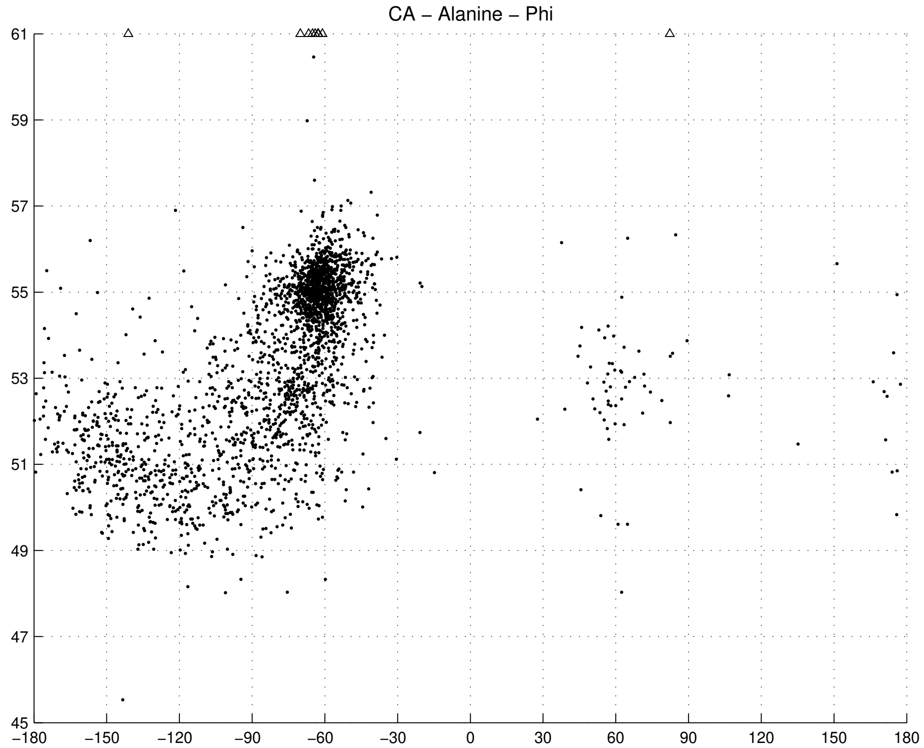

Figure 1: A scatterplot of NMR chemical shifts of the C^{α} nucleus versus the torsion angle ϕ for the alanine amino acid. The data points are chemical shift/torsion angle pair occurences from alanine amino acids in a database built from 601 proteins (proteins often have several occurences of alanine in their sequences). Triangles at the top of the plot indicate torsion angle positions of statistically insignificant rare outliers.

Of course, torsion angle prediction is only the one application we came across. Since the proposed algorithmic approach is conceptually simple, it may in general supplement other analysis tools used for value prediction of interesting attributes from observable ones. It should also be useful just for data reduction, "binning" and summarizing data.

The situation that relevant attributes depend on each other in complicated ways (that rule out simple analytic formulae) seems to be prevalent in important studies in life sciences. Among candidates for future applications we mention a few, arbitrarily picked examples: Frequencies of amino acids, as well as geometric parameters, show correlations to gene expression levels [9, 24], and these have been used to predict expression levels to a certain extent. (The method in [24] is a particular machine learning algorithm applied to structured scatterplots.)

- The functionally important curvature of DNA depends on GC-content and external conditions like temparature [16]. - A genuinely one-dimensional prediction problem is the identification of regions of mutation hotspots (or rather, the subsequences with low mutation frequencies) on nucleotide sequences [25].

The problem of [7] we called DIMaxD originates from several other pattern recognition applications in molecular biology, such as locating GC-rich regions and CpG islands in DNA, see [7] for more pointers to the literature.

Detection of special or unusual subsequences in biological sequences (DNA, proteins) has led to various problems similar to ours. One example is: Given a sequence of scores, find non-overlapping contiguous subsequences with maximum total scores [26]. However this problem is quite different from ours, in that the number of items is not limited. Therefore it requires different techniques. New algorithmic ideas for the problem of finding the k largest intervals in a sequence of real numbers have been developed in [8, 17].

A data partitioning problem that is accessible to dynamic programming in one dimension but whose complexity is open in the two-dimensional case (just as our problem) is reported in [15]. Finally we mention that the principal idea of sparse dynamic programming is not new. It has been successfully applied to string distance computations [13, 14, 19] which are, of course, also relevant in molecular biology. But apparently we apply it for the first time to density problems.

3 Selection as Preprocessing for Faster Dynamic Programming

First we give a simple dynamic programming algorithm for DIMaxL. Since it is straightforward, we omit the implementation details.

Proposition 1 *DIMaxL can be solved in $O(spn)$ time.*

Proof. The algorithm scans the sequence and maintains, for every position i and for $t = 1, \dots, s$ and $q = 1, \dots, p$, two solutions for the instance consisting of the first i items. Namely, both solutions have t intervals and $t + q$ items. One solution includes the ith item, the other solution does not, and either solution has maximum length among all solutions for the triple (t, q, i) that do include/do not include the ith item. When we step from i to $i + 1$, we can easily update these $2sp$ solutions by $O(sp)$ additions and comparisons of real numbers: Note that upon insertion of the new item we have to increase either q or t by 1, if the ith item is or is not in the previous solution. QED

Since this $O(spn)$ time algorithm builds up solutions incrementally by adding single items, we call it the *item-based algorithm* for later reference.

The item-based algorithm is not the most efficient way to solve DIMaxL if parameters s, p are small compared to n. Intuitively, we should be able to restrict the search inside dynamic programming to the largest sums of at most $p+1$ consecutive items x_i, rather than considering all possible local combinations all the time. It is not possible to limit from below the length of candidate *items* that may appear in an optimal solution, since it may be necessary to insert a very short item that links two long intervals. (Small examples are easy to construct.) But we can restrict the candidate *intervals* instead, as shown in the following lemma. We have to introduce some new notation first.

For $q = 1, \ldots, p+1$ let $I(q)$ denote the set of intervals with exactly q items. Formally we consider a solution S as the *set* of the s designated intervals appearing there. We will show that some optimal solution consists only of intervals among the $r(q)$ largest from each $I(q)$, where $r(q)$ is some bound that solely depends on q and parameters s, p but no longer on the input size n. Moreover, $r(q)$ is a moderate polynomial in s, p.

Whenever we refer to a set of intervals sorted by non-increasing lengths, ties are broken arbitrarily, that is, we fix one ordering of any intervals with equal lengths. With respect to such an ordering of $I(q)$, an interval has *rank* i if it is the ith largest interval in $I(q)$. An interval is said to be *blocked* by a set S' of intervals if it intersects some interval of S'. In particular, a member of $I(1)$ is blocked if the single item is covered by some interval of S'.

Lemma 2 *There exists an optimal solution S where all intervals from $I(q)$ in S have rank at most $r(q) = (s-2)q + p + (s+p)/q + 1$ in $I(q)$.*

Proof. Consider an optimal solution S. For any $v \in S \cap I(q)$ and $u \in I(q) \setminus S$ longer than v, interval u is blocked by $S \setminus \{v\}$, since otherwise $(S \setminus \{v\}) \cup u$ would be a better solution. For $j = 1, \ldots, p+1$ let s_j denote the number of intervals in $S \cap I(j)$. Clearly, $\sum_j s_j = s$ and $\sum_j j s_j = s + p$. Every interval in $(S \setminus \{v\}) \cap I(j)$ can block at most $q + j - 1$ intervals in $I(q)$. Hence the number of intervals blocked by $S \setminus \{v\}$ in $I(q)$ is at most $(\sum_j s_j(q + j - 1)) - (2q - 1)$, the last term is subtracted because $v \in I(q)$, but factor s_q appears in the sum. Since at most $s_q - 1$ intervals in $S \cap I(q)$ have lower ranks than v, the rank of

v in $I(q)$ is at most

$$s_q - 2q + 1 + \sum_j s_j(q+j-1) = s_q - 2q + 1 + (q-1)s + (s+p) = (s-2)q + p + s_q + 1.$$

Finally, observation $s_q \leq (s+p)/q$ gives the result. QED

Due to Lemma 2, only the $r(q)$ top intervals in each $I(q)$ need to be considered as candidates for an optimal solution. Bounding $\sum_q r(q)$ yields (H_k is the kth Harmonic number):

Corollary 3 *The number of candidate intervals is at most*

$$(s-2)(p+1)(p+2)/2 + (p+1)^2 + (s+p)H_{p+1}$$

$$= s(p^2 + 3p + 2)/2 + (s+p)H_{p+1} - p + 1 = (1/2 + o(1))sp^2.$$

Factor 1/2 will not be important for the time bound in O-notation, but constant factors can be practically relevant when huge data sets are processed. Now we use these findings to develop a more efficient algorithm for $s, p \ll n$. One idea would be to run the item-based algorithm on the set of items occuring in the candidate intervals. However, the following dynamic programming scheme is faster. We refer to it as the *candidate-based algorithm*, since the objects it works with are whole candidate intervals.

As a prerequisite, it is well known that the r largest elements in an (unsorted!) set of n numbers can be extracted via a Selection algorithm in $O(n)$ time, independent of r. (For an overview of results on Selection cf. [11, 23].) Now we give a high-level description of the candidate-based algorithm.

Algorithm and analysis:

(1) Extract the $r(q)$ candidate intervals from every set $I(q)$. In more detail: Compute all sums of q consecutive items in an obvious way in $O(n)$ time, and then find the $r(q)$ largest sums. Altogether, this phase needs $O(pn)$ time.

(2) Sort all $O(sp^2)$ candidate intervals by their rightmost items. This costs $O(sp^2(\log s + \log p))$ time, or alternatively $O(n + sp^2)$ time if bucketsort is applied.

(3) Run dynamic programming on the sorted list of candidate intervals as follows. For increasing i being the right end of some candidate interval, and for each $t = 1, \ldots, s$, $q = 1, \ldots, p$, compute the optimal solution with at most

t intervals, $t+q$ items, and rightmost item before or at i. Note that, for any two solutions S and S' with the same t,q but with different i and i' ($i < i'$), solution S is worse than S', or we can discard the latter one. In other words, it suffices to maintain a set of solutions in which, for any fixed t,q, the lengths are monotone increasing in i.

In every step of dynamic programming, take the next candidate interval $next$ from the list and append it to solutions whose rightmost item is to the left of $next$. By the monotonicity property, for every t,q it suffices to append $next$ to only one solution: that with i immediately preceding the left end of $next$. Then add 1 to t, add to q the number of new items minus 1, and discard new solutions that violate the monotonicity, by comparison to the solution with the previous end-item and same parameters t,q. It is straightforward to fill in the details. We spend $O(sp)$ time on every new candidate interval, hence $O(s^2p^3)$ time is needed for all of them.

Summing up the time bounds we obtain the main result of this section:

Theorem 4 *DIMaxL can be solved in $O(pn + s^2p^3)$ time.*

This is an improvement upon the item-based algorithm provided that $sp^2 = o(n)$. In Section 4 we will further reduce the polynomial term in s,p for certain ranges of these parameters. However the most noticeable fact is that we have saved a factor s in the $O(n)$ term. The next obvious question is whether we can get rid of factor p, too. Phase (1) of our algorithm processes all $I(q)$, $q = 1,\ldots,p{+}1$ in isolation, although it always works on the same input sequence. This situation suggests to look somehow at all q simultaneously. We come back to the question in Section 5.

Some data structure issues:

Theorem 4 holds for any $O(n)$ time Seletion algorithm used inside. In practice, a Selection algorithm with large hidden constant factor or with complicated internal administration would destroy the effect of having saved factor s, or make things even worse. However we can keep the auxiliary data structures simple and the constant factor in the Selection procedure small. (This statement is somewhat informal, as we do not specify the constants, but we argue that the structures proposed below are well known to be very efficient.)

Recall that, for each q, our $r(q) \approx sp^2/2$ is assumed to be small compared to n. To select the $r(q)$ largest elements in $I(q)$, we may construct a heap (see any textbook on data structures) with the largest sums on top. This is easily done in $O(n)$ time. Then, we first remove the root, which splits the heap in two disjoint heaps. After $j-1$ steps we have j heaps, and the jth largest sum is the maximum at their j roots. Using another small heap for the sums at the current roots, which supports insert and delete-max operations, we manage Selection in $O(n + r(q) \log r(q))$ time, with a small hidden constant factor.

Another idea is to sample k random elements and to determine a pivot element with a rank slightly above $r(q)k/n$ in the sample. Its expected rank in $I(q)$ is then slightly above $r(q)$. Now, simply compare each member of $I(q)$ to the pivot. In the event that the selected set is smaller than $r(q)$, repeat the process similarly on the remaining set. But with high probability it will be larger than $r(q)$, if a suitable margin has been added to the expected rank. Sample size k must give a good compromise between the time for sampling/pivot selection and the deviation of the pivot rank from $r(q)$. For this purpose, $k \approx \sqrt{n}$ seems to be a reasonable choice.

Solving DIMaxD faster:

Finally we apply the same principal idea to DIMaxD and improve the earlier $O(kln)$ time bound of [7] for arbitrary parameters. A simple lemma from [7] states:

Lemma 5 *There exists an optimal solution where each interval has less than $2l$ items.*

The following result beats the $O(kln)$ bound. Note that $kl < n$, since otherwise the problem would not make any sense.

Theorem 6 *DIMaxD can be solved in $O(ln + k^2l^2)$ time.*

Proof. By Lemma 5 it is enough to consider candidate segments of length less than $2l$, these are $O(ln)$ segments. Due to the bounded length $2l$, each candidate segment intersects only $O(l)$ other candidate segments. We say that the candidate segment with rth largest density has rank r, ties are broken arbitrarily. Now assume that we have an optimal solution U containing a segment s of rank larger than $O(kl)$ (with an appropriate constant factor). The other $k-1$ segments in U intersect only $O(kl)$ other candidate segments. Hence there exists a segment v' which is no worse than v and disjoint to the $k-1$ other segments in U. Substitute v with v' in U. Iterating this argument, it follows the existence of an optimal solution where all segments have a rank $O(kl)$.

Select the $O(kl)$ candidate segments in $O(ln)$ time. The candidate segments have only $O(kl)$ different endpoints, rather than n. Hence, a dynamic programming algorithm as in [7] but applied to this restricted set costs only $O(k^2l^2)$ time. QED

4 Complexity of DIMaxL Further Improved

In Section 3 we presented two different algorithms for DIMaxL in a "pure" form: the item-based and candidate-based algorithm with time bound $O(spn)$ and $O(pn + s^2p^3)$, respectively. Next we develop certain combinations of both, and obtain better complexity results if p is not too large compared to s, which will be the normal case according to the motivations.

First, remember that DIMaxL is trivial for $p = 0$, since we would simply take the s largest items. In the following we generalize this observation to any $p < s$ (and the resulting improvement will vanish as p approaches s). Recall also the notions of rank and blocking.

Lemma 7 *Any instance of DIMaxL with $p < s$ has an optimal solution that blocks all items (i.e., members of $I(1)$) with ranks $1, \ldots, s - p$.*

Proof. Among s intervals with $s + p$ items, at least $s - p$ intervals consist of single items. In particular, an optimal solution S has at least $s - p$ intervals consisting of single items. Suppose that any of the $s - p$ largest items, say x_i, is not blocked by S. Then there must be an item x_j with larger rank than x_i, that forms an interval in S. Replacing x_j with x_i gives obviously a solution no worse than S. QED

We refer to the $s - p$ items addressed in Lemma 7 as *top items*. For the subsequent argument we distinguish the following types of intervals: A *bound interval* contains one or more top items. A *halfbound interval* does not contain a top item, but is at most p items away from some top item x (that is, at most p items are between x and the end of the halfbound interval directed towards x). An *unbound interval* is more than p items away from every top item. Since unbound and halfbound intervals have together at most $2p$ items, at most $2p$ such intervals can exist in any solution. Moroever, since an interval in a solution has at most $p + 1$ items, an unbound interval cannot intersect a bound interval. With these facts in mind, we adapt Lemma 2 to the case $p < s$ as follows.

Lemma 8 *Any instance of DIMaxL with $p < s$ has an optimal solution S such that all unbound intervals from $I(q)$ in S are among the $2(pq + p/q - q) + 1$ largest unbound intervals in $I(q)$. Consequently, at most $(1 + o(1))p^3$ unbound intervals need to be considered as candidates for an optimal solution.*

Proof. Consider an optimal solution S that blocks all the top items. Such S exists due to Lemma 7. Let u, v be unbound intervals with $v \in I(q) \cap S$, $u \in I(q) \setminus S$, and u longer than v. Then interval u is blocked by $S \setminus \{v\}$, since otherwise $(S \setminus \{v\}) \cup \{u\}$ would be a better solution. Since u is unbound, it can be blocked only by halfbound or other unbound intervals. For $j = 1, \ldots, p+1$ let s_j denote the number of halfbound and unbound intervals in $S \cap I(j)$. As mentioned above, we have $\sum_j s_j \leq 2p$ and even $\sum_j js_j \leq 2p$. Every halfbound or unbound interval in $(S \setminus \{v\}) \cap I(j)$ can block at most $q + j - 1$ intervals in $I(q)$. Hence the number of unbound intervals in $I(q) \setminus S$ blocked by $S \setminus \{v\}$ is at most $(\sum_j s_j(q+j-1)) - (2q-1)$, the last term is subtracted because $v \in I(q)$, but factor s_q is used in the sum. Since at most $s_q - 1$ intervals in $S \cap I(q)$ have a rank lower than v in $I(q)$, the rank of v among the unbound intervals in $I(q)$ is at most $s_q - 2q + 1 + \sum_j s_j(q+j-1) = s_q - 2q + 1 + 2p(q-1) + 2p = 2(p-1)q + 1 + s_q$. Observation $s_q \leq 2p/q$ gives the result. QED

The following proof relies on the results of Section 3, therefore the presentation focuses on what is new.

Theorem 9 *Instances of DIMaxL with $p < s$ can be solved in $O(pn + sp^3 + p^5)$ time.*

Proof. First we select the top items and the unbound candidate intervals for all q (from Lemma 8) in $O(pn)$ time. For constructing the optimal solution we scan the input sequence from left to right and maintain a best solution (i.e., one with maximum total length of selected items) for the subsequence to the left of the current position, for each combination of certain counters of objects used so far. Specifically we use the following counters:
f: number of items, without the top items.
b: number of intervals minus the number of top items.

Clearly, $0 \leq f \leq 2p$. As mentioned above, at most $2p$ intervals are halfbound or unbound, hence $b \leq 2p$ at any time. However b can become negative if many top items are present in the bound intervals. With a trick we can limit the "relevant" values of b to $-p \leq b \leq 2b$ as follows. In any final solution we will have $b = s - (s - p) = p$. During the successive selection of intervals to be added to a solution, b can grow only if a halfbound or unbound interval is selected. But as we noticed above, their total number is at most $2p$. Thus b can grow at most $2p$ times, and it suffices to consider $b \geq -p$ at any time. That means, if b falls below $-p$ in a partial solution S', we would finally get fewer than s intervals and could split some of the intervals of S' arbitrarily. Hence it is equally good to do this splitting already now, at the considered moment, which simply means to raise b immediately back to $-p$ without changing the set of selected items in S'.

We apply the item-based algorithm in subsequences of $O(p)$ items around every top item, and apply the candidate-based algorithm to the unbound candidate intervals in the subsequences in between. In contrast to Section 3 we work with counters f, b rather than p, s. In fact, we can easily glue together the two algorithms, since both work from left to right on a set of partial solutions organized in the same way, only their update methods are different. Clearly, the subsequences where we run the item-based algorithm have total length $O(sp)$. Since we are using two counters, each of range $O(p)$, these phases need in total $O(sp^3)$ time. Similarly, we spend $O(p^2)$ time for insertion of each of the $O(p^3)$ unbound candidate intervals (Lemma 8), hence these phases need in total $O(p^5)$ time. QED

By combining the two basic algorithms in a more sophisticated way we can improve the s^2p^3 part of Theorem 4 even for $p < s^2$. For this new algorithm we define several parameters whose values will be specified later:
t: some length threshold. Intervals with at most (with more than) t items are called *short (long)* intervals.
u: the maximum number of long intervals that any fixed solution may contain.
r: the maximum number of short intervals *with equal number of items* that may be blocked by a solution.

Lemma 10 *The parameters satisfy $r \leq st + p$ and $u \leq (s + p)/t$.*

Proof. Observe that the first items of the s intervals in any solution block at most st different intervals of a fixed length up to t, and the p remaining items block at most another p such intervals, hence $r \leq st + p$. The second inequality is obvious from the number of items, the minimum length, and disjointness of intervals in any solution. QED

We will derive a time bound in terms of parameters t, u, r and finally compute, for the given s and p, valid values of t, u, r that minimize the time bound.

Until then, we assume that the parameter values are already known and can be used by the algorithm.

Lemma 11 *Any instance of DIMaxL has an optimal solution S such that every interval in $S \cap I(q)$, $q \leq t$ (that is, every short interval), has rank at most r in $I(q)$.*

Proof. By iterated replacements we can rule out all $x \in S \cap I(q)$ with rank larger than r as follows. By definition of r, at most r members of $I(q)$ are blocked by S. Hence some $y \in I(q)$ with rank at most r is not blocked by S, and $S \setminus \{x\} \cup \{y\}$ is a feasible solution no worse than S. QED

According to Lemma 11 we refer to the intervals with ranks $1, \ldots, r$ in each $I(q)$, $q \leq t$, as the *short candidate intervals.* Similarly as above, we distinguish the following types of intervals with at most $p+1$ items: A *bound interval* intersects one or more short candidate intervals. A *halfbound interval* does not intersect any short candidate interval, but it is at most p items away from some. An *unbound interval* is more than p items away from any short candidate interval. Note that an unbound interval cannot intersect a bound interval. Furthermore these definitions yield immediately that, among the halfbound and unbound intervals, only long intervals need to be considered as candidate intervals. By definition of u, at most u of these long intervals are in any fixed solution. Since at most $u+p$ items can belong to any u intervals in a solution, the halfbound and unbound long intervals in a solution have together at most $u+p$ items. After these preparations we are now able to limit also the rank of the unbound long candidate intervals, by a variation of Lemma 8.

Lemma 12 *There exists an optimal solution S such that all unbound long intervals from $I(q)$ in S have rank at most $(u-2)q+p+(u+p)/q+1$ in the set of unbound intervals in $I(q)$.*
Consequently, at most $(1/2+o(1))(u(p^2-t^2)+(u+p)\ln(p/t))$ unbound long intervals need to be considered as candidate intervals for an optimal solution.

Proof. Let S be an optimal solution, and x, y be unbound long intervals such that $x \in I(q) \cap S$, $y \in I(q) \setminus S$, and y has a smaller rank than x. Then y is blocked by $S \setminus \{x\}$. Since y is unbound, it can be blocked only by halfbound or unbound long intervals. For $j = t+1, \ldots, p+1$ let s_j denote the number of halfbound and unbound intervals in $S \cap I(j)$. As mentioned above, we have $\sum_j s_j \leq u$ and $\sum_j js_j \leq u+p$. Every halfbound or unbound interval in $(S \setminus \{x\}) \cap I(j)$

can block at most $q+j-1$ intervals in $I(q)$. Hence the number of unbound intervals in $I(q)$ blocked by $S \setminus \{x\}$ is at most $\sum_j s_j(q+j-1)) - (2q-1)$, the last term is subtracted because $x \in I(q)$, but factor s_q is used in the sum. At most $s_q - 1$ unbound intervals in $S \cap I(q)$ can have smaller ranks than x. In addition, the above mentioned number of unbound intervals in $I(q)$ blocked by $S \setminus \{x\}$ may have smaller ranks than x. Together, the rank of x is limited by $s_q - 2q + 1 + \sum_j s_j(q+j-1) = s_q - 2q + 1 + (q-1)u + (u+p) = s_q + p + 1 + (u-2)q$. Observation $s_q \leq (u+p)/q$ gives the result. QED

Once more, only the main term $O(up^2)$ will be relevant for the time bound, but constant $1/2$ can make a difference in practice. Anyway, we shall prove the following:

Theorem 13 *DIMaxL can be solved in $O(pn + s^2p^{8/3})$ time if $s^{6/7} \leq p \leq s$, and in $O(pn + s^{4/3}p^{10/3})$ time if $s \leq p \leq s^2$.*

Proof. We proceed similarly as in Theorem 9 and apply the item-based algorithm in subsequences of $O(p)$ items around every short candidate interval, and the candidate-based algorithm to the unbound long candidate intervals in the subsequences in between. Since we have, by Lemma 11, only $O(tr)$ short candidate intervals, the number of items on which the item-based algorithm runs is $O(trp)$. By Lemma 12, the number of unbound long candidate intervals is $O(up^2)$. Unlike the method in Theorem 9 we use the original counters, one for the number of intervals and one for the number of extra items (i.e., items minus intervals). Since they range from 0 to s and p, respectively, the time for all dynamic programming phases is $O(trsp^2 + usp^3)$. Using Lemma 11 we obtain $O(t^2s^2p^2 + tsp^3 + s^2p^3/t + sp^4/t)$.

For any given s and p, we define constant a and variable x $(0 \leq x \leq a)$ by $p = s^a$ and $t = s^x$, and write the four terms as powers of s. Their exponents are $e_1 = 2+2a+2x$, $e_2 = 1+3a+x$, $e_3 = 2+3a-x$, $e_4 = 1+4a-x$. Since the e_i are monotone in x, their maximum can be minimized only if some of $x = 0$, $x = a$, $e_1 = e_3$, $e_1 = e_4$, $e_2 = e_3$, $e_2 = e_4$ holds. Hence the only relevent values of x are $0, a, a/3, -(1/3) + (2/3)a, 1/2, a/2$. In these six cases, the e_i have the following different values (written in one line for each case, where obviously non-maximal values are omitted):

$2+3a$, $1+4a$.
$2+4a$.
$2+(8/3)a$, $1+(11/3)a$.
$(7/3)+(7/3)a$, $(4/3)+(10/3)a$, $(2/3)+(11/3)a$.
$3+2a$, $(3/2)+3a$, $(1/2)+4a$.
$2+3a$, $1+(7/2)a$.

For any fixed a we determine the maximum in each case.

$2+3a$ if $a \leq 1$, $1+4a$ if $1 \leq a$.
$2+4a$.
$2+(8/3)a$ if $a \leq 1$, $1+(11/3)a$ if $1 \leq a$.
$(7/3)+(7/3)a$ if $a \leq 1$, $(4/3)+(10/3)a$ if $1 \leq a \leq 2$, $(2/3)+(11/3)a$ if $2 \leq a$.
$3+2a$ if $a \leq 5/4$, $(1/2)+4a$ if $5/4 \leq a$.
$2+3a$ if $a \leq 2$, $1+(7/2)a$ if $2 \leq a$.

Given a, we choose x where this term is minimized. We find that the minimum is $2+(8/3)a$ if $a \leq 1$, $4/3+(10/3)a$ if $1 \leq a \leq 2$, and $1+(7/2)a$ if $2 \leq a$. Finally we compare the new bounds to those in Theorem 4 and 9, which have exponents $1+3a$ if $a \leq 1/2$, $5a$ if $1/2 \leq a \leq 1$, and $2+3a$ if $1 \leq a$. Taking the minimum yields the improvements $2+(8/3)a$ if $6/7 \leq a \leq 1$, and $4/3+(10/3)a$ if $1 \leq a \leq 2$. QED

The new twechnique does not help if $p \geq s^2$. It remains open whether the s^2p^3 term can be reduced also in that case. We finally mention a simple result for $s=2$. (Case $s=1$ is trivially solvable in $O(n)$ time.) For $s=2$, dynamic programming and even candidate selection is useless.

Proposition 14 *DIMaxL can be solved in $O(pn)$ time if $s=2$.*

Proof. For every $q \leq p+1$ and each of n positions in the given sequence it suffices to determine the largest interval to the left and to the right, with q items and $p+2-q$ items, respectively. All prefix and suffix maxima for each q can be precomputed in $O(n)$ time. Finally, the best couple is chosen. QED

5 Acceleration Heuristics for DIMaxL

5.1 Joint Selection and an Open Problem

We come back to the question of possible improvements of the $O(pn)$ term in Theorem 4. Note that Corollary 3 bounds only the *number* of candidate intervals, but still we have all freedom regarding the way to compute this set. Instead of doing Selection on all q separately, we may also exploit locality in the problem's structure and determine the longest intervals in subsequences of $O(p)$ items, for all q simultaneously, and then pick the best from all these subproblems.

More precisely, we cover the input sequence by overlapping intervals which we call *windows*. The ith window is $x_{(i-1)(p+1)+1}, \ldots, x_{(i+1)(p+1)}$ (except the last window which may have up to $3p$ items, depending on n). Obviously, every interval of $q \leq p+1$ items is entirely in some of these windows.

Proposition 15 *Assume that we can compute the longest interval from $I(q)$, for all $q = 1, \ldots, p+1$, within each window in $T(p)$ time. Then we could also solve DIMaxL in $O(nT(p)/p + s^2p^3)$ time.*

Proof. Recall that Phase (1) of our candidate-based algorithm has to find the $r(q)$ longest intervals in each $I(q)$. Lemma 2 gives $r(q) = O(sp)$. Once we have the longest intervals in all $O(n/p)$ windows, we can select, for each q, the $r(q)$ longest of them in $O(n/p)$ time, thus we need $O(n)$ time for all q. At this stage we know, for each q, that all candidates are in the $O(sp)$ windows that contain these $r(q)$ top intervals. All other windows can be disregarded since already the maximum from $I(q)$ therein is too bad. Thus we collect the $O(sp^2)$ intervals of $I(q)$ from the candidate windows and do the final selection in $O(sp^2)$ time. This takes $O(sp^3)$ time for all q. The rest works as in Theorem 4. QED

The idea may also be combined with the more sophisticated dynamic programming schemes in Section 4. Anyway, the question is what is $T(p)$. Trivially we have $p \leq T(p) = O(p^2)$. Note that any bound $T(p) = o(p^2)$ would speed up the candidate selection. We cannot deliver such a subquadratic worst-case bound nor prove an $\Omega(p^2)$ lower bound. However we give a heuristic that should be faster in almost all practical cases, as it needs $O(p^2)$ time only if the input exhibits some periodicities that are unlikely in real data.

First we give a nicer formulation of the essence of the problem. In the following, m is the window size.

MAXIMUM CONSECUTIVE SUMS (MaxCS)

Given a sequence $x_1, \ldots, x_m$, find for each $q = 1, \ldots, m$ the longest interval with q items.

The question is now whether there exists an $o(m^2)$ algorithm for MaxCS. Another equivalent formulation uses the prefix sums $\sum_{i=1}^{j} x_i$ and suffix sums

$\sum_{i=m-k+1}^{m} x_i$ which are computable in $O(m)$ time. Instead of any interval of q items, consider the prefix to the left and the suffix to the right of that interval, say with $j \geq 0$ and $k \geq 0$ items, respectively. Note that $j + k = m - q$. Thus, substituting $m - q$ with q gives following problem with the same complexity:

Lowest Midpoints (LMP)

Given two sequences $u_1, \ldots, u_m$ and $v_1, \ldots, v_m$, find for each $q = 1, \ldots, m$ the minimum of $u_j + v_k$ with $j + k = q$.

Our naming LMP comes from a geometric interpretation: Given red and blue points in the plane with coordinates (j, u_j) and (k, v_k) respectively, find for each q the lowest midpoint, with abscissa $q/2$, of a red and a blue point.

This geometric view suggests a simple heuristic: Consider some straight line, in the following called a *separator*. Compute the set M of all midpoints of all red-blue pairs of points below the separator. For every abscissa $q/2$ appearing in M we only check the red-blue pairs where at least one partner is below the separator. Note that the lowest midpoint at $q/2$ cannot come from two points above the separator, since there is already a better pair below it. For every $q/2$ not appearing in M we finally compute the lowest midpoint naively, by checking all $O(m)$ red-blue pairs. This easily gives:

Proposition 16 *If, for some line, we have b points below the line, and their red-blue pairs have midpoints with a different abscissa values, then the given instance of LMP is solvable in $O(ab + (m - a)m)$ time.*

It is computationally simple to try several separators with fixed slope, since this requires only one linear transformation of coordinates. It is advisable to cut off $b = \Theta(\sqrt{m})$ points in a first attempt. If we are most lucky, the $b = \Theta(\sqrt{m})$ points below the separator generate nearly all abscissa values. In this case we need only $O(m^{3/2})$ computations. Of course, the set of abscissa values of points below the separator may have periodicities, so that many pairwise averages are equal. The worst case is that they form arithmetic sequences. Then we are stuck with $a = O(b)$ and have to try more separators upwards. In the worst case the time is still $O(n^2)$. However we argue that such bad cases should be rare in irregular real data. Moreover, the slope can be arbitrarily changed, too. This will most probably break existing regularities, since the order in which points are encountered is changed. We may also vertically shift all red or blue points by the same amount. This does not affect the result but it can mix the two colors better, so that we get many red-blue pairs below the separators.

Example: A toy example may illustrate how a fraction of the ca. $m^2/2$ comparisons between interval lengths are saved, at cost of $O(m)$ auxiliary computations. We have chosen an example with $m = 8$ for quick explanation, the effect is more significant for larger m. Consider the sequence of items $(3, 5, 1, 6, 3, 1, 6, 4)$ in a window of size $m = 8$. The prefix and suffix sums are
$(u_0, \ldots, u_7) = (0, 3, 8, 9, 15, 18, 19, 25)$ and
$(v_0, \ldots, v_7) = (0, 4, 10, 11, 14, 20, 21, 26)$,
respectively. Of course these sequences are always monotone, so that the smallest u_j and v_k generate only a few distinct index sums $j+k$. Therefore we change the slope and set, for instance, $u_j := u_j - 3j$ and $v_k := v_k - 3k$. For fixed $j + k$ this subtracts the same term $3(j+k)$ from all u_j+v_k. In general, a slope close to the avarage of items is obviously a good choice. We get the linearly transformed prefix and suffix sums
$(0, 0, 2, 0, 3, 3, 1, 4)$ and
$(0, 1, 4, 2, 2, 5, 3, 5)$,
respectively, which are already well mixed. (Otherwise we could have added a nonzero constant to all v_k.) Now let us use, for instance, a line separating the $u_j, v_k < 1$ from the $u_j, v_k > 1$. Positions $j = 0, 1, 3, 6$ and $k = 0, 1$ of the $u_j, v_k < 1$ are not equidistant, reflecting irregularities in the input sequence. This is also "sufficiently irregular" for our purpose: Among the 8 sums $j + k$ of these positions, 7 are distinct, and the corresponding sums $u_j + v_k$ for $j + k = 0, 1, \ldots, 7$ are $(0, 0, 1, 0, 1, ?, 1, 2)$. The question mark appears since we have no u_j, v_k with $j + k = 5$ below this separator. Given the height of our separator, we see that the six sums $u_j + v_k \leq 1$ are already optimal. Only for $j + k = 5$ and 7 it remains to determine the smallest $u_j + v_k$ by more comparisons. We finally get $(0, 0, 1, 0, 1, 2, 1, 2)$. These optimal sums came from the following pairs $j + k$:
$0 + 0, 1 + 0, 1 + 1, 3 + 0, 3 + 1, 1 + 4, 6 + 0, 3 + 4$.
It means that the longest subsequence of $7, 6, 5, 4, 3, 2, 1$ items starts after position $0, 1, 1, 3, 3, 1, 6, 3$, respectively, which is confirmed by the given sequence.

Still, it would be very nice to develop an algorithm with worst case time $o(m^2)$. We hardly expect that MaxCS and LMP belong to the class of 3SUM-complete problems where the trivial $O(m^2)$ bound is hard to beat. (See [4] for recent progress in 3SUM-completeness.) Recent techniques for the related maximum-sums problem [8, 17] give some hope.

5.2 Taking Advantage of Structural Restrictions

Optimal solutions S have several structural properties that may prove useful also algorithmically. In the following, an *end item* is the first or last item of an interval in S, and a *side item* is incident to an end item, but does not occur in S. (We remark that a chain of incident intervals in S can always be considered as one interval, because merging does not affect $s+p$ but only decreases s.)

Observe that if an end item is shorter than the side item at the opposite end of the same interval, we could exchange the former by the latter item and get a better solution. Hence such intervals can be removed from the candidate set that we previously selected based on total lengths. If several candidates are affected, the dynamic programming phase becomes faster. Back to our illustrating example, if $(3,5,1,6,3,1,6,4)$ is in the input sequence then, e.g., intervals $(1,6,3)$ and $(3,1,6)$ are ruled out.

Similarly, presence of an interval v in S excludes all later candidates with side items larger than some end item of v. If we keep track of the shortest end item used so far, we need not consider such candidates anymore. However this also inserts a new parameter, thus it remains open whether these pairwise exclusion relations are helpful algorithmically.

5.3 Taking Advantage of Very Different Lengths

All previous techniques were based on intersections of intervals and on length comparisons. But we may also use the numerical values of lengths more directly.

Intuitively, the largest items are likely to be in an optimal solution. On the other hand, points in real data often form clusters, giving long consecutive subsequences of small x_i. Assume there is an optimal solution S which does *not* contain *all* q largest items, for some fixed $q \leq p$. Then, all intervals in S must be longer than the qth largest x_k, or an obvious replacement improves S. Consequently, if the interval of $2p+1$ items centered at x_i is shorter than x_k, it follows that item x_i cannot appear at all in S. Thus we might be able to remove many short items in $O(n)$ time and drastically reduce our instance. In order to find S, we would then apply one of the previous algorithms to the remaining instance with smaller n.

However, due to the assumption on q, we have to consider a second instance where S has to include *all* q largest items. This immediately reduces p (not s!) to $p-q$, which is nice because p appears quadratically in the number of candidate intervals (see Corollary 3, Lemma 12). The dynamic programming procedure must be slightly adjusted so as to choose the q largest items in each case. Altogether, the sum of times spent on both instances can be smaller than in the previous algorithms. We may choose our threshold q depending on the distribution of lengths x_i.

6 Conclusions and Further Research

We devised sparse dynamic programming algorithms for finding, in a one-dimensional data set, disjoint intervals with maximum total length that contain a limited number of data points. The algorithms for some parameter constellations are already quite sophisticated, but further improved time bounds seem to be possible.

In addition we proposed several useful heuristics which do not help the current worst-case time bounds but use typical features of real data to run faster. Complexities of algorithms are studied in the usual framework of asymptotic worst-case analysis. It remains to pursue a comprehensive experimental study to compare the running times of different implementations. Since the theoretical bounds differ in relatively small factors, subtle implementation details can be decisive.

An intriguing open theoretical problem is a subquadratic algorithm, deterministic or randomized, that finds for all q the largest sums of q consecutive items in a sequence. It seems that only certain regularities in the sequence make the problem difficult. An evaluation of our current separator line heuristic in the recently introduced framework of smoothed analysis would therefore be interesting, too. A more probability-theoretic question is which problem instances are typically produced by random sampling from certain simple distributions, e.g., such with piecewise constant densities.

Last but not least, the algorithms shall be applied to prediction tasks with huge data sets in bioinformatics, in our concrete case to the protein torsion angle prediction from NMR chemical shifts, and compared to the performance of other statistical approaches.

The complexity of multi-dimensional analogues of our main problem is open, e.g.: How difficult is it to compute the s disjoint axis-parallel rectangles with maximum total area that contain at most p given points within a finite rectangle?

Acknowledgments

The first author has been supported by the Knut and Alice Wallenberg Foundation, Carl Tryggers Foundation, and Assar Gabrielsson Foundation. The second author has been partially supported by the Swedish Research Council (Vetenskapsrådet), project "Algorithms for searching and inference in genetics", grant no. 621-2002-4574. He would also like to thank Ferdinando Cicalese for constructive discussions during a stay with his group "Combinatorial Search Algorithms in Bioinformatics" at the University of Bielefeld. We are indebted to the students and project assistants Can Gencer, Rina Sripomo Kocadag and Christin Christin for their work in processing the protein structure data.

References

[1] A. Aggarwal, S. Suri: Fast algorithms for computing the largest empty rectangle, *Symp. on Comput. Geometry 1987*, 278-290

[2] G. Agnarsson, P. Damaschke, M.M. Halldórsson: Powers of geometric intersection graphs and dispersion algorithms, *Discrete Applied Mathematics* 132 (2003), Special Issue *Stability in Graphs and Related Topics* (eds. V. Lozin, D. de Werra), 3-16. Preliminary version in: Proc. of SWAT 2002, *LNCS* 2368, 140-149

[3] M.J. Atallah, G.N. Frederickson: A note on finding a maximum empty rectangle, *Discrete Applied Math.* 13 (1986), 87-91

[4] I. Baran, E. Demaine, M. Patrascu: Subquadratic algorithms for 3SUM, *9th WADS 2005*, *LNCS* 3608, 409-421

[5] R.D. Beger, P.H. Bolton: Protein ϕ and ψ dihedral restraints determined from multidimensional hypersurface correlations of backbone chemical shifts and their use in the determination of protein tertiary structures, *J. of Biomolecular NMR* 10 (1997), 129-142

[6] B. Chazelle, L.R.S. Drysdale, D.T. Lee: Computing the largest empty rectangle, *SIAM J. Comp.* 15 (1986), 550-555

[7] Y.H. Chen, H.I. Lu, C.Y. Tang: Disjoint segments with maximum density, *International Workshop on Bioinformatics Research and Applications IWBRA 2005* (within *ICCS 2005*), *LNCS* 3515, 845-850

[8] C.H. Cheng, K.Y. Chen, W.C. Tien, K.M. Chao: Improved algorithms for the k maximum-sums problem, *16th ISAAC 2005*, *LNCS* 3827, 799-808

[9] F. Chiaromonte, W. Miller, E.E. Bouhassira: Gene length and proximity to neighbors affect genome-wide expression levels, *Genome Research* 13 (2003), 2602-2608

[10] G. Cornilescu, F. Delaglio, A. Bax: Protein backbone angle restraints from searching a database for chemical shift and sequence homology, *J. Biomolecular NMR* 13 (1999) 289-302

[11] D. Dor: Selection algorithms, PhD thesis, Tel-Aviv Univ. 1995

[12] J. Edmonds, J. Gryz, D. Liang, R.J. Miller: Mining for empty rectangles in large data sets, *Theor. Computer Science* 296 (2003), 435-452

[13] D. Eppstein, Z. Galil. R. Giancarlo, G.F. Italiano: Sparse dynamic programming I: Linear cost functions, *J. of the ACM* 39 (1992), 519-545

[14] Z. Galil, K. Park: Dynamic programming with convexity, concavity and sparsity, *Theor. Computer Science* 92 (1992), 49-76

[15] B. Jackson et al.: An algorithm for optimal partitioning of data on an interval, preprint, `arxiv.org/abs/math/0309285` (2004)

[16] L. Kozobay-Avraham, S. Hosid, A. Bolshoy: Curvature distribution in prokaryotic genomes, *In Silico Biology* 4:0029 (2004)

[17] T.C. Lin, D.T. Lee: Randomized algorithm for the sum selection problem, *16th ISAAC 2005*, *LNCS* 3827, 515-523

[18] B. Liu, L.P. Ku, W. Hsu: Discovering interesting holes in data, *15th IJCAI 1997*, 930-935

[19] V. Mäkinen, G. Navarro, E. Ukkonen: Algorithms for transposition invariant string matching, *20th STACS 2003*, *LNCS* 2607, 191-202

[20] J. Meiler: PROSHIFT: protein chemical shift prediction using artificial neural networks, *J. of Biomol. NMR* 26 (2003), 25-37

[21] S. Neal, A.M. Nip, H. Zhang, D.S. Wishart: Rapid and accurate calculation of protein ^1H, ^{13}C, ^{15}N chemical shifts, *J. of Biomol. NMR* 26 (2003), 215-240

[22] M. Orlowski: A new algorithm for the largest empty rectangle problem, *Algorithmica* 5 (1990), 65-73

[23] M.S. Paterson: Progress in selection, *5th SWAT'96*, *LNCS* 1097, 368-379

[24] G.P.S. Raghava, J.H. Han: Correlation and prediction of gene expression level from amino acid and dipeptide composition of its proteins, *BMC Bioinformatics* 6:59 (2005)

[25] I.B. Rogozin, V.N. Babenko, L. Milanesi, Y.I. Pavlov: Computational analysis of mutation spectra, *Briefings in Bioinformatics* 4 (2003), 210-227

[26] W.L. Ruzzo, M. Tompa: A linear time algorithm for finding all maximal scoring subsequences, *7th Int. Conf. Intelligent Systems for Molecular Biology 1999*, AAAI, 234-241

[27] B.R. Seavey, E.A. Farr, W.M. Westler, J.L. Markley: A Relational Database for Sequence-Specific Protein NMR Data, *J. of Biomol. NMR* 1 (1991), 217-236

[28] Y. Wang, O. Jardetzky: Probability-based protein secondary structure identification using combined NMR chemical-shift data, *Protein Science* 11 (2002), 852-861

[29] X.P. Xu, D.A. Case: Probing multiple effects on ^{15}N, $^{13}C^{\alpha}$, $^{13}C^{\beta}$ and $^{13}C'$ chemical shifts in peptides using density functional theory, *Biopolymers* 65 (2002), 408-423

Correction

Proof of Theorem 6.

By Lemma 5 it is enough to consider candidate segments of length less than $2l$, these are $O(ln)$ segments. Due to the bounded length $2l$, each candidate segment intersects only $O(l^2)$ other candidate segments. We say that the candidate segment with rth largest density has rank r, ties are broken arbitrarily. Now assume that we have an optimal solution U containing a segment v of rank larger than $O(kl^2)$ (with an appropriate constant factor). The other $k-1$ segments in U intersect only $O(kl^2)$ other candidate segments. Hence there exists a segment v' which is no worse than v and disjoint to the $k-1$ other segments in U. Substitute v with v' in U. Iterating this argument, it follows the existence of an optimal solution where all segments have a rank $O(kl^2)$.

Select the $O(kl^2)$ candidate segments in $O(ln)$ time. A candidate-based dynamic programming algorithm applied to this set costs only $O(k^2l^2)$ time, since we need only one counter of range k for the number of intervals. QED