Pattern Matching and Consensus Problems on Weighted Sequences and Profiles*

Tomasz Kociumaka[1], Solon P. Pissis[2], and Jakub Radoszewski[†3]

1 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
kociumaka@mimuw.edu.pl
2 **Department of Informatics, King's College London, London, UK**
solon.pissis@kcl.ac.uk
3 **Institute of Informatics, University of Warsaw, Warsaw, Poland; and Department of Informatics, King's College London, London, UK**
jrad@mimuw.edu.pl

Abstract

We study pattern matching problems on two major representations of uncertain sequences used in molecular biology: weighted sequences (also known as position weight matrices, PWM) and profiles (i.e., scoring matrices). In the simple version, in which only the pattern or only the text is uncertain, we obtain efficient algorithms with theoretically-provable running times using a variation of the lookahead scoring technique. We also consider a general variant of the pattern matching problems in which both the pattern and the text are uncertain. Central to our solution is a special case where the sequences have equal length, called the consensus problem. We propose algorithms for the consensus problem parameterized by the number of strings that match one of the sequences. As our basic approach, a careful adaptation of the classic meet-in-the-middle algorithm for the knapsack problem is used. On the lower bound side, we prove that our dependence on the parameter is optimal up to lower-order terms conditioned on the optimality of the original algorithm for the knapsack problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases weighted sequence, position weight matrix, profile matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.46

1 Introduction

We study two well-known representations of uncertain texts: *weighted sequences* and *profiles*. A *weighted sequence* (also known as position weight matrix, PWM) for every position and every letter of the alphabet specifies the probability of occurrence of this letter at this position; see Table 1 for an example. A weighted sequence represents many different strings, each with the probability of occurrence equal to the product of probabilities of its letters at subsequent positions of the weighted sequence. Usually a threshold $\frac{1}{z}$ is specified, and one considers only strings that match the weighted sequence with probability at least $\frac{1}{z}$. A *scoring matrix* (or a profile) of length m is an $m \times \sigma$ matrix. The *score* of a string of length m is the sum of scores in the scoring matrix of the subsequent letters of the string at the respective positions. A string is said to match a scoring matrix if its matching score is above a specified threshold Z.

* Work supported by the Polish Ministry of Science and Higher Education under the 'Iuventus Plus' program in 2015–2016 grant no 0392/IP3/2015/73.
† The author is a Newton Fellow.

© Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski;
licensed under Creative Commons License CC-BY

Table 1 A weighted sequence X of length 4 over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$X[1]$	$X[2]$	$X[3]$	$X[4]$
$\pi_1^{(X)}(\mathtt{a}) = 1/2$	$\pi_2^{(X)}(\mathtt{a}) = 1$	$\pi_3^{(X)}(\mathtt{a}) = 3/4$	$\pi_4^{(X)}(\mathtt{a}) = 0$
$\pi_1^{(X)}(\mathtt{b}) = 1/2$	$\pi_2^{(X)}(\mathtt{b}) = 0$	$\pi_3^{(X)}(\mathtt{b}) = 1/4$	$\pi_4^{(X)}(\mathtt{b}) = 1$

Weighted Pattern Matching and Profile Matching. First of all, we study the standard variants of pattern matching problems on weighted sequences and profiles, in which only the pattern or the text is an uncertain sequence. In the best-known formulation of the Weighted Pattern Matching problem, we are given a weighted sequence of length n, called a text, a solid (standard) string of length m, called a pattern, both over an alphabet of size σ, and a *threshold probability* $\frac{1}{z}$. We are asked to find all positions in the text where the fragment of length m represents the pattern with probability at least $\frac{1}{z}$. Each such position is called an *occurrence* of the pattern in the text; we also say that the fragment of the text and the pattern *match*. The Weighted Pattern Matching problem can be solved in $\mathcal{O}(\sigma n \log m)$ time via the Fast Fourier Transform [5]. In a more general, indexing variant of the problem, considered in [1, 12], one can preprocess a weighted text in $\mathcal{O}(nz^2 \log z)$ time to report all *occ* occurrences of a given solid pattern of length m in $\mathcal{O}(m + occ)$ time. (A similar indexing data structure, which assumes $z = \mathcal{O}(1)$, was presented in [4].) Very recently, the index construction time was reduced to $\mathcal{O}(nz)$ for constant-sized alphabets [2].

In the classic Profile Matching problem, the pattern is an $m \times \sigma$ profile, the text is a solid string of length n, and our task is to find all positions in the text where the fragment of length m has a score which is at least Z. A naïve approach to the Profile Matching problem works in $\mathcal{O}(nm + m\sigma)$ time. A broad spectrum of heuristics improving this algorithm in practice is known; for a survey see [16]. One of the principal techniques, coming in different flavours, is *lookahead scoring* that consists in checking if a partial match could possibly be completed by the highest scoring letters in the remaining positions of the scoring matrix and, if not, pruning the naïve search. The Profile Matching problem can also be solved in $\mathcal{O}(\sigma n \log m)$ time via the Fast Fourier Transform [17].

Weighted Consensus and Profile Consensus. As our most involved contribution, we study a general variant of pattern matching on weighted sequences and the consensus problems on uncertain sequences, which are closely related to the Multichoice Knapsack problem. In the Weighted Consensus problem, given two weighted sequences of the same length, we are to check if there is a string that matches each of them with probability at least $\frac{1}{z}$. A routine to compare user-entered weighted sequences with existing weighted sequences in the database is used, e.g., in JASPAR[1], a well-known database of PWMs. In the General Weighted Pattern Matching (GWPM) problem, both the pattern and the text are weighted. In the most common definition of the problem (see [3, 12]), we are to find all fragments of the text that give a positive answer to the Weighted Consensus problem with the pattern. The authors of [3] proposed an algorithm for the GWPM problem based on the weighted prefix table that works in $\mathcal{O}(nz^2 \log z + n\sigma)$ time. Solutions to these problems can be applied in transcriptional regulation: motif and regulatory module finding; and annotation of regulatory genomic regions.

In an analogous way to the Weighted Consensus problem, we define the Profile Consensus problem. Here we are to check for the existence of a string that matches both

[1] `http://jaspar.genereg.net`

the scoring matrices above threshold Z. The Profile Consensus problem is actually a special case of the well-known (especially in practice) Multichoice Knapsack problem (also known as the Multiple Choice Knapsack problem). In this problem, we are given n classes $C_1, \dots, C_n$ of at most λ items each—N items in total—each item c characterized by a value $v(c)$ and a weight $w(c)$. The goal is to select one item from each class so that the sums of values and of weights of the items are below two specified thresholds, V and W. (In the more intuitive formulation of the problem, we require the sum of values to be *above* a specified threshold, but here we consider an equivalent variant in which both parameters are symmetric.) The Multichoice Knapsack problem is widely used in practice, but most research concerns approximation or heuristic solutions; see [14] and references therein. As far as exact solutions are concerned, the classic meet-in-the middle approach by Horowitz and Sahni [11], originally designed for the (binary) Knapsack problem, immediately generalizes to an $\mathcal{O}^*(\lambda^{\lceil \frac{n}{2} \rceil})$-time[2] solution for Multichoice Knapsack.

Several important problems can be expressed as special cases of the Multichoice Knapsack problem using folklore reductions (see [14]). This includes the Subset Sum problem, which, for a set of n integers, asks whether there is a subset summing up to a given integer Q, and the k-Sum problem which, for $k = \mathcal{O}(1)$ classes of λ integers, asks to choose one element from each class so that the selected integers sum up to zero. These reductions give immediate hardness results for the Multichoice Knapsack problem, and they can be adjusted to yield the same consequences for Profile Consensus. For the Subset Sum problem, as shown in [7, 10], the existence for every $\varepsilon > 0$ of an $\mathcal{O}^*(2^{\varepsilon n})$-time solution would violate the Exponential Time Hypothesis (ETH) [13, 15]. Moreover, the $\mathcal{O}^*(2^{n/2})$ running time, achieved in [11], has not been improved yet despite much effort. The 3-Sum conjecture [9] and the more general k-Sum conjecture state that the 3-Sum and k-Sum problems cannot be solved in $\mathcal{O}(\lambda^{2-\varepsilon})$ time and $\mathcal{O}(\lambda^{\lceil \frac{k}{2} \rceil (1-\varepsilon)})$ time, respectively, for any $\varepsilon > 0$.

Our Results. As the first result, we show how the lookahead scoring technique combined with a data structure for answering longest common prefix (LCP) queries in a string can be applied to obtain simple and efficient algorithms for the standard pattern matching problems on uncertain sequences. For a weighted sequence, by R we denote the size of its list representation, and by λ the maximal number of letters with score at least $\frac{1}{z}$ at a single position (thus $\lambda \le \min(\sigma, z)$). In the Profile Matching problem, we set M as the number of strings that match the scoring matrix with score above Z. In general $M \le \sigma^m$, however, we may assume that for practical data this number is actually much smaller. We obtain the following running times:

- $\mathcal{O}(m\sigma + n \log M)$ for Profile Matching;
- $\mathcal{O}(R \log^2 \log \lambda + n \log z)$ deterministic and $\mathcal{O}(R + n \log z)$ randomized (Las Vegas, failure with probability R^{-c} for any given constant c) for Weighted Pattern Matching.

The more complex part of our study is related to the consensus problems and to the GWPM problem. Instead of considering Profile Consensus, we study the more general Multichoice Knapsack. We introduce parameters based on the number of solutions with *feasible* weight or value: $A_V = |\{(c_1, \dots, c_n) : c_i \in C_i \text{ for all } i = 1, \dots, n, \sum_i v(c_i) \le V\}|$, that is, the number of choices of one element from each class that satisfy the value threshold; $A_W = |\{(c_1, \dots, c_n) : c_i \in C_i \text{ for all } i = 1, \dots, n, \sum_i w(c_i) \le W\}|$; $A = \max(A_V, A_W)$, and $a = \min(A_V, A_W)$. We obtain algorithms with the following complexities:

[2] The $\mathcal{O}^*$ notation suppresses factors polynomial with respect to the instance size (encoded in binary).

- $\mathcal{O}(N + \sqrt{a\lambda}\log A)$ for Multichoice Knapsack;
- $\mathcal{O}(R + \sqrt{z\lambda}(\log\log z + \log\lambda))$ for Weighted Consensus and $\mathcal{O}(n\sqrt{z\lambda}(\log\log z + \log\lambda))$ for General Weighted Pattern Matching.

Note that $a \le A \le \lambda^n$ and thus the running time of our algorithm for Multichoice Knapsack is bounded by $\mathcal{O}(N + n\lambda^{(n+1)/2}\log\lambda)$. Up to lower order terms (i.e., the factor $n\log\lambda = (\lambda^{(n+1)/2})^{o(1)}$), this matches the time complexities of the fastest known solutions for both Subset Sum (also binary Knapsack) and 3-Sum. The main novel part of our algorithm for Multichoice Knapsack is an appropriate (yet intuitive) notion of *ranks* of partial solutions. We also provide a simple reduction from Multichoice Knapsack to Weighted Consensus, which lets us transfer the negative results to the GWPM problem.

- The existence, for every $\varepsilon > 0$, of an $\mathcal{O}^*(z^\varepsilon)$-time solution for Weighted Consensus would violate the Exponential Time Hypothesis.
- For every $\varepsilon > 0$, an $\mathcal{O}^*(z^{0.5-\varepsilon})$-time solution for Weighted Consensus would imply an $\mathcal{O}^*(2^{(0.5-\varepsilon)n})$-time algorithm for Subset Sum.
- For every $\varepsilon > 0$, an $\tilde{\mathcal{O}}(R + z^{0.5}\lambda^{0.5-\varepsilon})$-time[3] solution for Weighted Consensus would imply an $\tilde{\mathcal{O}}(\lambda^{2-\varepsilon})$-time algorithm for 3-Sum.

For the higher-order terms our complexities match the conditional lower bounds; therefore, we put significant effort to keep the lower order terms of the complexities as small as possible.

Model of Computations. For problems on weighted sequences, we assume the word-RAM model with word size $w = \Omega(\log n + \log z)$ and $\sigma = n^{\mathcal{O}(1)}$. We consider the log-probability model of representations of weighted sequences, that is, we assume that probabilities in the weighted sequences and the threshold probability $\frac{1}{z}$ are all of the form $c^{\frac{p}{2^{dw}}}$, where c and d are constants and p is an integer that fits in a constant number of machine words. Additionally, the probability 0 has a special representation. The only operations on probabilities in our algorithms are multiplications and divisions, which can be performed exactly in $\mathcal{O}(1)$ time in this model. Our solutions to the Multichoice Knapsack problem only assume the word-RAM model with word size $w = \Omega(\log S + \log a)$, where S is the sum of integers in the input instance; this does not affect the $\mathcal{O}^*$ running time.

Structure of the Paper. We start with Preliminaries, where we formally introduce the problems and the main notions used throughout the paper. The following three sections describe our algorithms: in Section 3 for Profile Matching and Weighted Pattern Matching; in Section 4 for Profile Consensus; and in Section 5 for Weighted Consensus and General Weighted Pattern Matching. We conclude with a few remarks in Section 6.

2 Preliminaries

Let $\Sigma = \{s_1, s_2, \dots, s_\sigma\}$ be an alphabet of size σ. A *string* S over Σ is a finite sequence of letters from Σ. We denote the length of S by $|S|$ and, for $1 \le i \le |S|$, the i-th letter of S by $S[i]$. By $S[i..j]$ we denote the string $S[i]\dots S[j]$ called a *factor* of S (if $i > j$, then the factor is an empty string). A factor is called a *prefix* if $i = 1$ and a *suffix* if $j = |S|$. For two strings S and T, we denote their concatenation by $S \cdot T$ (ST in short).

[3] The $\tilde{\mathcal{O}}$ notation ignores factors polylogarithmic with respect to the input size.

For a string S of length n, by $lcp(i, j)$ we denote the length of the longest common prefix of factors $S[i..n]$ and $S[j..n]$. The following fact specifies a well-known efficient data structure answering such queries. It consists of the suffix array with its inverse, the LCP table and a data structure for range minimum queries on the LCP table; see [6] for details.

▶ **Fact 1.** *Let S be a string of length n over an alphabet of size $\sigma = n^{\mathcal{O}(1)}$. After $\mathcal{O}(n)$-time preprocessing, given indices i and j $(1 \le i, j \le n)$ one can compute $lcp(i, j)$ in $\mathcal{O}(1)$ time.*

The *Hamming distance* between two strings X and Y of the same length, denoted by $d_H(X, Y)$, is the number of positions where the strings have different letters.

2.1 Profiles

In the Profile Matching problem, we consider a *scoring matrix* (a profile) P of size $m \times \sigma$. For $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, \sigma\}$, we denote the integer score of the letter s_j at the position i by $P[i, s_j]$. The *matching score* of a string S of length m with the matrix P is

$$\mathrm{Score}(S, P) = \sum_{i=1}^{m} P[i, S[i]].$$

If $\mathrm{Score}(S, P) \ge Z$ for an integer *threshold* Z, then we say that the string S *matches the matrix P above threshold Z*. We denote the number of strings S that match P above threshold Z by $\mathrm{NumStrings}_Z(P)$.

For a string T and a scoring matrix P, we say that P *occurs in T at position i with threshold Z* if $T[i..i+m-1]$ matches P above threshold Z. Then $Occ_Z(P, T)$ is the set of all positions where P occurs in T. These notions let us define the Profile Matching problem:

Profile Matching Problem
Input: A string T of length n, a scoring matrix P of size $m \times \sigma$, and a threshold Z.
Output: The set $Occ_Z(P, T)$.
Parameters: $M = \mathrm{NumStrings}_Z(P)$.

2.2 Weighted Sequences

A *weighted sequence* $X = X[1] \dots X[n]$ of length $|X| = n$ over alphabet $\Sigma = \{s_1, s_2, \dots, s_\sigma\}$ is a sequence of sets of pairs of the form $X[i] = \{(s_j,\ \pi_i^{(X)}(s_j))\ :\ j \in \{1, 2, \dots, \sigma\}\}$. Here, $\pi_i^{(X)}(s_j)$ is the occurrence probability of the letter s_j at the position $i \in \{1, \dots, n\}$. These values are non-negative and sum up to 1 for a given i.

For all our algorithms, it is sufficient that the probabilities sum up to *at most* 1 for each position. Also, the algorithms sometimes produce auxiliary weighted sequences with sum of probabilities being smaller than 1 on some positions.

We denote the maximum number of letters occurring at a single position of the weighted sequence (with non-zero probability) by λ and the total size of the representation of a weighted sequence by R. The standard representation consists of n lists with up to λ elements each, so $R = \mathcal{O}(n\lambda)$. However, the lists can be shorter in general. Also, if the threshold probability $\frac{1}{z}$ is specified, at each position of a weighted sequence it suffices to store letters with probability at least $\frac{1}{z}$, and clearly there are at most z such letters for each position. This reduction can be performed in linear time, so we shall always assume that $\lambda \le z$.

The *probability of matching* of a string S with a weighted sequence X, $|S| = |X| = m$, is

$$\mathcal{P}(S, X) = \prod_{i=1}^{m} \pi_i^{(X)}(S[i]).$$

We say that a string S *matches a weighted sequence* X *with probability at least* $\frac{1}{z}$, denoted by $S \approx_{\frac{1}{z}} X$, if $\mathcal{P}(S,X) \ge \frac{1}{z}$. Given a weighted sequence T, by $T[i..j]$ we denote weighted sequence, called a *factor* of T, equal to $T[i]\dots T[j]$ (if $i > j$, then the factor is empty). We say that a string P *occurs* in T at position i if P matches the factor $T[i..i+m-1]$. The set of positions where P occurs in T is denoted by $Occ_{\frac{1}{z}}(P,T)$.

WEIGHTED PATTERN MATCHING PROBLEM
Input: A string P of length m and a weighted sequence T of length n with at most λ letters at each position and R in total, and a threshold probability $\frac{1}{z}$.
Output: The set $Occ_{\frac{1}{z}}(P,T)$.

3 Profile Matching and Weighted Pattern Matching

In this section we present a solution to the PROFILE MATCHING problem. Afterwards, we show that it can be applied for WEIGHTED PATTERN MATCHING as well.

For a scoring matrix P, the *heavy string* of P, denoted $\mathcal{H}(P)$, is constructed by choosing at each position the heaviest letter, that is, the letter with the maximum score (breaking ties arbitrarily). Intuitively, $\mathcal{H}(P)$ is a string that matches P with the maximum score.

▶ **Observation 2.** *If we have* $\text{Score}(S,P) \ge Z$ *for a string* S *of length* m *and an* $m \times \sigma$ *scoring matrix* P, *then* $d_H(\mathcal{H}(P),S) \le \lfloor \log M \rfloor$ *where* $M = \text{NumStrings}_Z(P)$.

Proof. Let $d = d_H(\mathcal{H}(P),S)$. We can construct 2^d strings of length $|S|$ that match P with a score above Z by taking either of the letters $S[j]$ or $\mathcal{H}(P)[j]$ at each position j such that $S[j] \ne \mathcal{H}(P)[j]$. Hence, $2^d \le M$, which concludes the proof. ◀

Our solution for the PROFILE MATCHING problem works as follows. We first construct $P' = \mathcal{H}(P)$ and the data structure for finding lcp values between suffixes of P' and T. Let the variable s store the matching score of P'. In the p-th step, we calculate the matching score of $T[p..p+m-1]$ by iterating through subsequent mismatches between P' and $T[p..p+m-1]$ and making adequate updates in the matching score s', which starts at $s' = s$. The mismatches are found using lcp-queries. This process terminates when the score s' drops below Z or when all the mismatches have been found. In the end, we include p in $Occ_Z(P,T)$ if $s' \ge Z$. This gives the following result.

▶ **Theorem 3.** PROFILE MATCHING *problem can be solved in* $\mathcal{O}(m\sigma + n \log M)$ *time.*

Proof. Let us bound the time complexity of the presented algorithm. The heavy string P' can be computed in $\mathcal{O}(m\sigma)$ time. The data structure for *lcp*-queries in $P'T$ can be constructed in $\mathcal{O}(n+m)$ time by Fact 1. Each query for $lcp(P'[i..m], T[j..n])$ can then be answered in constant time by a corresponding *lcp*-query in $P'T$, potentially truncated to the end of P'. Finally, for each position p in the text T we will consider at most $\lfloor \log M \rfloor + 1$ mismatches between P' and T, as afterwards the score s' drops below Z due to Observation 2. ◀

Basically the same approach can be used for WEIGHTED PATTERN MATCHING. In a natural way, we extend the notion of a heavy string to weighted sequences. Now we can restate Observation 2 in the language of probabilities instead of scores:

▶ **Observation 4.** *If a string* P *matches a weighted sequence* X *of the same length with probability at least* $\frac{1}{z}$, *then* $d_H(\mathcal{H}(X),P) \le \lfloor \log z \rfloor$.

Comparing to the solution to PROFILE MATCHING, we compute the heavy string of the text instead of the pattern. An auxiliary variable α stores the matching probability between a factor of $\mathcal{H}(T)$ and the corresponding factor of T; it is updated when we move to the next position of the text. The rest of the algorithm is basically the same as previously. In the implementation, we perform the following operations on a weighted sequence:

- computing the probability of a given letter at a given position,
- finding the letter with the maximum probability at a given position.

In the standard list representation, the latter can be performed on a single weighted sequence in $\mathcal{O}(1)$ time after $\mathcal{O}(R)$-time preprocessing. We can perform the former in constant time if, in addition to the list representation, we store the letter probabilities in a dictionary implemented using perfect hashing [8] (we build a single hash table for all positions). This way, we can implement the algorithm in $\mathcal{O}(n \log z + R)$ time w.h.p. Alternatively, deterministic dictionaries [18, Theorem 3] (one for each position) can be used to obtain a deterministic solution in $\mathcal{O}(R \log^2 \log \lambda + n \log z)$ time. We arrive at the following result.

▶ **Theorem 5.** WEIGHTED PATTERN MATCHING *can be solved in* $\mathcal{O}(R + n \log z)$ *time with high probability by a Las-Vegas algorithm or in* $\mathcal{O}(R \log^2 \log \lambda + n \log z)$ *time deterministically.*

▶ Remark. In the same complexity one can solve the GWPM problem with a solid text.

4 Profile Consensus as Multichoice Knapsack

Let us start with a precise statement of the MULTICHOICE KNAPSACK problem.

MULTICHOICE KNAPSACK PROBLEM
Input: A set $\mathcal{C}$ of N items partitioned into n disjoint classes C_i, each of size at most λ, two integers $v(c)$ and $w(c)$ for each item $c \in \mathcal{C}$, and two thresholds V and W.
Question: Does there exist a *choice* S (a set $S \subseteq \mathcal{C}$ such that $|S \cap C_i| = 1$ for each i) satisfying both $\sum_{c\in S} v(c) \le V$ and $\sum_{c \in S} w(c) \le W$?
Parameters: A_V and A_W: the number of choices S satisfying $\sum_{c\in S} v(c) \le V$ and $\sum_{c\in S} w(c) \le W$, respectively; as well as $A = \max(A_V, A_W)$ and $a = \min(A_V, A_W)$.

Indeed, we see that the PROFILE CONSENSUS problem reduces to the MULTICHOICE KNAPSACK problem. For two $m \times \sigma$ scoring matrices, we construct $n = m$ classes of $\lambda = \sigma$ items each, with values equal to the negated scores of the letters in the first matrix and weights equal to the negated scores in the second matrix; both thresholds V and W are equal to $-Z$.

For a fixed instance of MULTICHOICE KNAPSACK, we say that S is a *partial choice* if $|S \cap C_i| \le 1$ for each class. The set $D = \{i : |S \cap C_i| = 1\}$ is called its *domain*. For a partial choice S, we define $v(S) = \sum_{c\in S} v(c)$ and $w(S) = \sum_{c\in S} w(c)$.

The classic $\mathcal{O}(2^{n/2})$-time solution to the KNAPSACK problem [11] partitions $D = \{1, \dots, n\}$ into two domains D_1, D_2 of size roughly $n/2$, and for each D_i it generates all partial choices S ordered by $v(S)$. Hence, it reduces the problem to an instance of MULTICHOICE KNAPSACK with two classes. It is solved using the following folklore lemma.

▶ **Lemma 6.** *The* MULTICHOICE KNAPSACK *problem can be solved in* $\mathcal{O}(N)$ *time if* $n = 2$ *and the elements* c *of* C_1 *and* C_2 *are sorted by* $v(c)$.

The same approach generalizes to MULTICHOICE KNAPSACK. The partition is chosen to balance the number of partial choices in each domain, and the worst-case time complexity is $\mathcal{O}(\sqrt{Q\lambda})$, where $Q = \prod_{i=1}^{n} |C_i|$ is the number of choices.

Our aim in this section is to replace Q with the parameter a (which never exceeds Q). The overall running time is going to be $\mathcal{O}(N + \sqrt{a\lambda}\log A)$.

Two challenges arise when adapting the meet-in-the-middle approach: how to restrict the set of partial choices to be generated so that a feasible solution is not missed, and how to define a partition $D = D_1 \cup D_2$ to balance the number of partial choices generated for D_1 and D_2. A natural idea to deal with the first issue is to consider only partial choices with small values $v(S)$ or $w(S)$. This is close to our actual solution, which is based on the notion of *ranks* of partial choices. Our approach to the second problem is to consider multiple partitions: those of the form $D = \{1,\dots,j\} \cup \{j+1,\dots,n\}$ for $1 \le j \le n$. This results in an extra $\mathcal{O}(n)$ factor in the time complexity. However, preprocessing can assure $n = \mathcal{O}(\frac{\log A}{\log \lambda})$. While dealing with these two issues, a careful implementation is required to avoid several further extra factors in the running time. In case of our algorithm, this is only $\mathcal{O}(\log \lambda)$, which stems from the fact that we need to keep partial solutions ordered by $v(S)$.

For a partial choice S, we define $\mathrm{rank}_v(S)$ as the number of partial choices S' with the same domain for which $v(S') \le v(S)$. We symmetrically define $\mathrm{rank}_w(S)$. Ranks are introduced as an analogue of match probabilities in weighted sequences. Probabilities are multiplicative, while for ranks we have submultiplicativity:

▶ **Fact 7.** *If $S = S_1 \cup S_2$ is a decomposition of a partial choice S into two disjoint subsets, then $\mathrm{rank}_v(S_1)\,\mathrm{rank}_v(S_2) \le \mathrm{rank}_v(S)$ (and same for rank_w).*

Proof. Let D_1 and D_2 be the domains of S_1 and S_2, respectively. For every partial choices S'_1 over D_1 and S'_2 over D_2 such that $v(S'_1) \le v(S_1)$ and $v(S'_2) \le v(S_2)$, we have $v(S'_1 \cup S'_2) = v(S'_1) + v(S'_2) \le v(S)$. Hence, $S'_1 \cup S'_2$ must be counted while determining $\mathrm{rank}_v(S)$. ◀

For $0 \le j \le n$, let $\mathcal{L}_j$ be the list of partial choices with domain $\{1,\dots,j\}$ ordered by value $v(S)$, and for $\ell > 0$ let $V^{(\ell)}_{\mathcal{L}_j}$ be the value $v(S)$ of ℓ-th element of $\mathcal{L}_j$ (∞ if $\ell > |\mathcal{L}_j|$). Analogously, for $1 \le j \le n+1$, we define $\mathcal{R}_j$ as the list of partial choices over $\{j,\dots,n\}$ ordered by $v(S)$, and for $r > 0$, $V^{(r)}_{\mathcal{R}_j}$ as the value of the r-th element of $\mathcal{R}_j$ (∞ if $r > |\mathcal{R}_j|$).

The following two observations yield a decomposition of each choice into a single item and two partial solutions of a small rank. In particular, we do not need to know A_V in order to check if the ranks are sufficiently large.

▶ **Lemma 8.** *Let ℓ and r be positive integers such that $V^{(\ell)}_{\mathcal{L}_j} + V^{(r)}_{\mathcal{R}_{j+1}} > V$ for each $0 \le j \le n$. For every choice S with $v(S) \le V$, there is an index $j \in \{1,\dots,n\}$ and a decomposition $S = L \cup \{c\} \cup R$ such that $v(L) < V^{(\ell)}_{\mathcal{L}_{j-1}}$, $c \in C_j$, and $v(R) < V^{(r)}_{\mathcal{R}_{j+1}}$.*

Proof. Let $S = \{c_1,\dots,c_n\}$ with $c_i \in C_i$ and, for $0 \le i \le n$, let $S_i = \{c_1,\dots,c_i\}$. If $v(S_{n-1}) < V^{(\ell)}_{\mathcal{L}_{n-1}}$, we set $L = S_{n-1}$, $c = c_n$, and $R = \emptyset$, satisfying the claimed conditions.

Otherwise, we define j as the smallest index i such that $v(S_i) \ge V^{(\ell)}_{\mathcal{L}_i}$, and we set $L = S_{j-1}$, $c = c_j$, and $R = S \setminus S_j$. The definition of j implies $v(L) < V^{(\ell)}_{\mathcal{L}_{j-1}}$ and $v(L \cup \{c\}) \ge V^{(\ell)}_{\mathcal{L}_j}$. Moreover, we have $v(L\cup\{c\}) + v(R) = v(S) \le V < V^{(\ell)}_{\mathcal{L}_j} + V^{(r)}_{\mathcal{R}_{j+1}}$, and thus $v(R) < V^{(r)}_{\mathcal{R}_{j+1}}$. ◀

▶ **Fact 9.** *Let $\ell, r > 0$. If $V^{(\ell)}_{\mathcal{L}_j} + V^{(r)}_{\mathcal{R}_{j+1}} \le V$ for some $j \in \{0,\dots,n\}$, then $\ell \cdot r \le A_V$.*

Proof. Let L and R be the ℓ-th and r-th entry in $\mathcal{L}_j$ and $\mathcal{R}_{j+1}$, respectively. Note that $v(L \cup R) \le V$ implies $\mathrm{rank}_v(L \cup R) \le A_V$ by definition of A_V. Moreover, $\mathrm{rank}_v(L) \ge \ell$ and $\mathrm{rank}_v(R) \ge r$ (the equalities may be sharp due to draws). Now, Fact 7 yields the claimed bound. ◀

Note that $\mathcal{L}_j$ can be obtained by interleaving $|C_j|$ copies of $\mathcal{L}_{j-1}$, where each copy corresponds to extending the choices from $\mathcal{L}_{j-1}$ with a different item. If we were to construct $\mathcal{L}_j$ having access to the whole $\mathcal{L}_{j-1}$, we could proceed as follows. For each $c \in C_j$, we maintain an *iterator* on $\mathcal{L}_{j-1}$ pointing to the first element S on $\mathcal{L}_{j-1}$ for which $S \cup \{c\}$ has not yet been added to $\mathcal{L}_j$. The associated *value* is $v(S \cup \{c\})$. All iterators initially point at the first element of $\mathcal{L}_{j-1}$. Then the next element to append to $\mathcal{L}_j$ is always $S \cup \{c\}$ corresponding to the iterator with minimum value. Having processed this partial choice, we advance the pointer (or remove it, once it has already scanned the whole $\mathcal{L}_{j-1}$). This process can be implemented using a binary heap H_j as a priority queue, so that initialization requires $\mathcal{O}(|C_j|)$ time and outputting a single element takes $\mathcal{O}(\log |C_j|)$ time.

For $r \geq 0$, let $\mathcal{L}_j^{(r)}$ be the prefix of $\mathcal{L}_j$ of length $\min(r, |\mathcal{L}_j|)$ and $\mathcal{R}_j^{(r)}$ be the prefix of $\mathcal{R}_j$ of length $\min(r, |\mathcal{R}_j|)$. A technical transformation of the procedure stated above leads to an online algorithm that constructs the prefixes $\mathcal{L}_j^{(r)}$ and $\mathcal{R}_j^{(r)}$ (details will be provided in the full version). Along with each reported partial choice S, the algorithm also computes $w(S)$.

▶ **Lemma 10.** *After $\mathcal{O}(N)$-time initialization, one can construct $\mathcal{L}_1^{(i)}, \ldots, \mathcal{L}_n^{(i)}$ online for $i = 0, 1, \ldots,$ spending $\mathcal{O}(n \log \lambda)$ time per each step. Symmetrically, one can construct $\mathcal{R}_1^{(i)}, \ldots, \mathcal{R}_n^{(i)}$ in the same time complexity.*

Also the following reduction can be obtained (details are left for the full version).

▶ **Lemma 11.** *Given an instance I of the* Multichoice Knapsack *problem, one can compute in $\mathcal{O}(N + \lambda \log A)$ time an equivalent instance I' with $A'_V \leq A_V$, $A'_W \leq A_W$, $\lambda' \leq \lambda$, and $n' = \mathcal{O}(\frac{\log A}{\log \lambda})$.*

Note that we may always assume that $\lambda \leq a$. Indeed, if we order the items $c \in C_i$ according to $v(c)$, then only the first A_V of them might belong to a choice S with $v(S) \leq V$.

▶ **Theorem 12.** Multichoice Knapsack *can be solved in $\mathcal{O}(N + \sqrt{a\lambda} \log A)$ time.*

Proof. Below, we give an algorithm working in $\mathcal{O}(N + \sqrt{A_V \lambda} \log A)$ time. The final solution runs it in parallel on the original instance and on the instance with v and V swapped with w and W, waiting until at least one of them terminates.

We increment an integer r starting from 1, maintaining $\ell = \lceil \frac{r}{\lambda} \rceil$ and the lists $\mathcal{L}_j^{(\ell)}$ and $\mathcal{R}_{j+1}^{(r)}$ for $0 \leq j \leq n$, as long as $V_{\mathcal{L}_j}^{(\ell)} + V_{\mathcal{R}_{j+1}}^{(r)} \leq V$ for some j (or until all the lists have been completely generated). By Fact 9, we stop at $r = \mathcal{O}(\sqrt{A_V \lambda})$. Lemma 11 lets us assume that $n = \mathcal{O}(\frac{\log A}{\log \lambda})$, so the running time of this phase is $\mathcal{O}(N + \sqrt{A_V \lambda} \log A)$ due to Lemma 10. The preprocessing time of Lemma 11 is dominated by this complexity.

Due to Lemma 8, every feasible solution S admits a decomposition $S = L \cup \{c\} \cup R$ with $L \in \mathcal{L}_{j-1}^{(\ell)}$, $c \in C_j$, and $R \in \mathcal{R}_{j+1}^{(r)}$ for some index j. We consider all possibilities for j. For each of them we will reduce searching for S to an instance of the Multichoice Knapsack problem with $N' = \mathcal{O}(\sqrt{A_V \lambda})$ and $n' = 2$. By Lemma 6, these instances can be solved in $\mathcal{O}(n\sqrt{A_V \lambda}) = \mathcal{O}(\sqrt{A_V \lambda} \frac{\log A}{\log \lambda})$ time in total.

The items of the j-th instance are going to belong to classes $\mathcal{L}_{j-1}^{(\ell)} \odot C_j$ and $\mathcal{R}_{j+1}^{(r)}$, where $\mathcal{L}_{j-1}^{(\ell)} \odot C_j = \{L \cup \{c\} : L \in \mathcal{L}_{j-1}^{(\ell)}, c \in C_j\}$. The set $\mathcal{L}_{j-1}^{(\ell)} \odot C_j$ can be constructed by merging $|C_j| \leq \lambda$ sorted lists, each of size $\ell = \mathcal{O}(\sqrt{A_V / \lambda})$, i.e., in $\mathcal{O}(\sqrt{A_V \lambda} \log \lambda)$ time. Summing up over all indices j, this gives $\mathcal{O}(\sqrt{A_V \lambda} \log \lambda \frac{\log A}{\log \lambda}) = \mathcal{O}(\sqrt{A_V \lambda} \log A)$ time.

Clearly, each feasible solution of the constructed instances represents a feasible solution of the initial instance, and by Lemma 8, every feasible solution of the initial instance has its counterpart in one of the constructed instances. ◀

5 Weighted Consensus and General Weighted Pattern Matching

The Weighted Consensus problem is formally defined as follows.

Weighted Consensus Problem
Input: Two weighted sequences X and Y of length n with at most λ letters at each position and R in total, and a threshold probability $\frac{1}{z}$.
Output: A string S such that $S \approx_{\frac{1}{z}} X$ and $S \approx_{\frac{1}{z}} Y$ or NONE if no such string exists.

If two weighted sequences satisfy the consensus, we write $X \approx_{\frac{1}{z}} Y$ and say that X *matches* Y *with probability at least* $\frac{1}{z}$. With this definition of a match, we extend the notion of an occurrence and the notation $Occ_{\frac{1}{z}}(P, T)$ to arbitrary weighted sequences.

General Weighted Pattern Matching (GWPM) Problem
Input: Two weighted sequences P and T of length m and n, respectively, with at most λ letters at each position and R in total, and a threshold probability $\frac{1}{z}$.
Output: The set $Occ_{\frac{1}{z}}(P, T)$.

In the case of the GWPM problem, it is more useful to provide an *oracle* that finds witness strings that correspond to the respective occurrences of the pattern. Such an oracle, given $i \in Occ_{\frac{1}{z}}(P, T)$, computes a string that matches both P and $T[i..i + m - 1]$.

Our algorithms rely on the following simple observation, originally due to Amir et al. [1].

▶ **Fact 13** ([1]). *A weighted sequence has at most z different matching strings.*

The Weighted Consensus problem is actually a special case of Multichoice Knapsack. Namely, given an instance of the former, we can create an instance of the latter with n classes C_i, each containing an item $c_{i,s}$ for every letter s which has non-zero probability at position i in both X and Y. We set $v(c_{i,s}) = -\log \pi_i^{(X)}(s)$ and $w(c_{i,s}) = -\log \pi_i^{(Y)}(s)$ for this item, whereas the thresholds are $V = W = \log z$. It is easy to see that this reduction indeed yields an equivalent instance and that it can be implemented in linear time. By Fact 13, we have $A \le z$ for this instance, so Theorem 12 yields the following result:

▶ **Corollary 14.** Weighted Consensus *problem can be solved in $\mathcal{O}(R + \sqrt{z\lambda}\log z)$ time.*

The GWPM problem can be clearly reduced to $n + m - 1$ instances of Weighted Consensus. This leads to a naïve $\mathcal{O}(nR + n\sqrt{z\lambda}\log z)$-time algorithm. Below, we remove the first term in this complexity. Our solution applies the approach used in Section 3 for Weighted Pattern Matching and uses an observation analogous to Observation 4.

▶ **Observation 15.** *If X and Y are weighted sequences that match with threshold $\frac{1}{z}$, then $d_H(\mathcal{H}(X), \mathcal{H}(Y)) \le 2\lfloor \log z \rfloor$. Moreover there exists a consensus string S such that $S[i] = \mathcal{H}(X)[i] = \mathcal{H}(Y)[i]$ unless $\mathcal{H}(X)[i] \ne \mathcal{H}(Y)[i]$.*

Our algorithm starts by computing $P' = \mathcal{H}(P)$ and $T' = \mathcal{H}(T)$ and the data structure for *lcp*-queries in $P'T'$. We try to match P with every factor $T[p..p + m - 1]$ of the text. Following Observation 15, we check if $d_H(T'[p..p + m - 1], P') \le 2\lfloor \log z \rfloor$. If not, then we know that no match is possible. Otherwise, let D be the set of positions of mismatches between $T'[p..p + m - 1]$ and P'. Assume that we store $\alpha = \prod_{j=1}^{m} \pi_{p+j-1}^{(T)}(T'[p + j - 1])$ and $\beta = \prod_{j=1}^{m} \pi_j^{(P)}(P'[j])$. Then, in $\mathcal{O}(|D|)$ time we can compute $\alpha' = \prod_{j \notin D} \pi_{p+j-1}^{(T)}(T'[p+j-1])$ and $\beta' = \prod_{j \notin D} \pi_j^{(P)}(P'[j])$. Now, we only need to check what happens at the positions in D.

If $D = \emptyset$, then it suffices to check if $\alpha \geq \frac{1}{z}$ and $\beta \geq \frac{1}{z}$. Otherwise, we construct two weighted sequences X and Y by selecting only the positions from D in $T[p..p+m-1]$ and in P. We multiply the probabilities of all letters at the first position in X by α' and in Y by β'. It is clear that $X \approx_{\frac{1}{z}} Y$ if and only if $T[p..p+m-1] \approx_{\frac{1}{z}} P$.

Thus, we have reduced the GWPM problem to at most $n-m+1$ instances of the Weighted Consensus problem for strings of length $\mathcal{O}(\log z)$. By Corollary 14, solving each instance takes $\mathcal{O}(\lambda \log z + \sqrt{z\lambda} \log z) = \mathcal{O}(\sqrt{z\lambda} \log z)$ time. Our reduction requires $\mathcal{O}(R \log^2 \log \lambda)$ time to preprocess the input (as in Theorem 5), but this is dominated by the $\mathcal{O}(n\sqrt{z\lambda} \log z)$ total time of solving the Weighted Consensus instances. If we memorize the solutions to all those instances together with the underlying sets of mismatches D, we can also implement the oracle for the GWPM problem with $\mathcal{O}(m)$-time queries.

A tailor-made solution can be designed (details will be provided in the full version) to replace the generic algorithm for the Multichoice Knapsack problem, which lets us improve the $\log z$ factor to $\log \log z + \log \lambda$.

▶ **Theorem 16.** *The* GWPM *problem can be solved in* $\mathcal{O}(n\sqrt{z\lambda}(\log \log z + \log \lambda))$ *time. An oracle for the* GWPM *problem using* $\mathcal{O}(n \log z)$ *space and supporting queries in* $\mathcal{O}(m)$ *time can be computed within the same time complexity.*

A reduction from Multichoice Knapsack to Weighted Consensus (proofs will be provided in the full version) immediately yields that any significant improvement in the dependence on z and λ in the running time of our algorithm would lead to breaking long-standing barriers for special cases of Multichoice Knapsack.

▶ **Theorem 17.** Weighted Consensus *problem is NP-hard and cannot be solved in:*
1. $\mathcal{O}^*(z^\varepsilon)$ *time for every* $\varepsilon > 0$*, unless the Exponential Time Hypothesis (ETH) fails;*
2. $\mathcal{O}^*(z^{0.5-\varepsilon})$ *time for some* $\varepsilon > 0$*, unless there is an* $\mathcal{O}^*(2^{(0.5-\varepsilon)n})$*-time algorithm for the* Subset Sum *problem;*
3. $\tilde{\mathcal{O}}(R + z^{0.5}\lambda^{0.5-\varepsilon})$ *time for some* $\varepsilon > 0$ *and for* $n = \mathcal{O}(1)$*, unless there is an* $\mathcal{O}(\lambda^{2(1-\varepsilon)})$*-time algorithm for 3-*Sum*.*

Nevertheless, it might still be possible to improve the dependence on n in the GWPM problem. For example, one may hope to achieve $\tilde{\mathcal{O}}(nz^{0.5-\varepsilon} + z^{0.5})$ time for $\lambda = \mathcal{O}(1)$.

6 Final Remarks

In Section 4, we gave an $\mathcal{O}(N + a^{0.5}\lambda^{0.5} \log A)$-time algorithm for the Multichoice Knapsack problem. Improvement of either exponent to $0.5 - \varepsilon$ would result in a breakthrough for the Subset Sum and 3-Sum problems, respectively. Nevertheless, this does not refute the existence of faster algorithms for some particular values (a, λ) other than those emerging from instances of Subset Sum or 3-Sum. Indeed, we can show an algorithm that is superior if $\frac{\log a}{\log \lambda}$ is a constant other than an odd integer. We can also prove it to be optimal (up to lower order terms) for every constant $\frac{\log a}{\log \lambda}$ unless the k-Sum conjecture fails. The details will be provided in the full version.

References

1 Amihood Amir, Eran Chencinski, Costas S. Iliopoulos, Tsvi Kopelowitz, and Hui Zhang. Property matching and weighted matching. *Theor. Comput. Sci.*, 395(2-3):298–310, April 2008. doi:10.1016/j.tcs.2008.01.006.

2 Carl Barton, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Efficient index for weighted sequences. In Roberto Grossi and Moshe Lewenstein, editors, *Combinatorial Pattern Matching, CPM 2016*, volume 54 of *LIPIcs*, pages 4:1–4:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.4.

3 Carl Barton and Solon P. Pissis. Linear-time computation of prefix table for weighted strings. In Florin Manea and Dirk Nowotka, editors, *Combinatorics on Words, WORDS 2015*, volume 9304 of *LNCS*, pages 73–84. Springer, 2015. doi:10.1007/978-3-319-23660-5_7.

4 Sudip Biswas, Manish Patil, Sharma V. Thankachan, and Rahul Shah. Probabilistic threshold indexing for uncertain strings. In Evaggelia Pitoura, Sofian Maabout, Georgia Koutrika, Amélie Marian, Letizia Tanca, Ioana Manolescu, and Kostas Stefanidis, editors, *19th International Conference on Extending Database Technology, EDBT 2016*, pages 401–412. OpenProceedings.org, 2016. doi:10.5441/002/edbt.2016.37.

5 Manolis Christodoulakis, Costas S. Iliopoulos, Laurent Mouchard, and Kostas Tsichlas. Pattern matching on weighted sequences. In *Algorithms and Computational Methods for Biochemical and Evolutionary Networks, CompBioNets 2004*, KCL publications, 2004.

6 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, New York, NY, USA, 2007.

7 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science, MFCS 2015, Part II*, volume 9235 of *LNCS*, pages 287–298. Springer, 2015. doi:10.1007/978-3-662-48054-0_24.

8 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

9 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

10 Eitan M. Gurari. *Introduction to the theory of computation*. Computer Science Press, 1989.

11 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

12 Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundam. Inform.*, 71(2-3):259–277, 2006. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-07.

13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

14 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

15 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/92.

16 Cinzia Pizzi and Esko Ukkonen. Fast profile matching algorithms – A survey. *Theor. Comput. Sci.*, 395(2-3):137–157, 2008. doi:10.1016/j.tcs.2008.01.015.

17 Sanguthevar Rajasekaran, X. Jin, and John L. Spouge. The efficient computation of position-specific match scores with the fast Fourier transform. *J. Comp. Biol.*, 9(1):23–33, 2002. doi:10.1089/10665270252833172.

18 Milan Ružić. Constructing efficient dictionaries in close to sorting time. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, ICALP 2008, Part I*, volume 5125 of *LNCS*, pages 84–95. Springer, 2008. doi:10.1007/978-3-540-70575-8_8.