Gene finding with complex external information[⋆]

Marcel Kucharík, Jakub Kováč, and Broňa Brejová

Department of Computer Science, Faculty of Mathematics, Physics, and Informatics
Comenius University, Mlynská Dolina, 842 48 Bratislava, Slovakia

Abstract. *The goal of gene finding is to locate genes, which are important segments of DNA encoding proteins. Programs solving this task are based on hidden Markov models (HMMs) capturing statistical features extracted from known genes, but often also incorporate hints about the correct gene structure extracted from experimental data. Existing gene finding programs can use such external information only in a limited way. Typically, they can process only simple hints describing a single part of the gene structure, because these are relatively easy to incorporate to standard HMM algorithms, but cannot cope with complex hints spanning multiple parts. We have developed an efficient algorithm able to process such complex hints. Our experiments show that this approach slightly increases the accuracy of gene prediction. We also prove that a more general class of hints leads to an NP-hard problem.*

1 Introduction

In this paper we study a combinatorial optimization problem arising in computational biology. Although we were originally motivated by a very specific application, we formulate the problem quite generally and explore its variants that admit efficient polynomial-time algorithms as well as those that are NP hard.

Our motivation comes from the problem of gene finding. Here the goal is to locate segments of a DNA sequence that encode proteins produced by the organism. On a more abstract level, we are given a string $X = x_1 \dots x_n$ and the goal is to produce a string $A = a_1 \dots a_n$ over some output alphabet Σ of possible labels, such that a_i is a label of x_i, corresponding to its functional role. In gene finding, X is the input DNA sequence (over alphabet $\{A, C, G, T\}$) and the output alphabet could be $\Sigma = \{0, 1\}$ where 1 stands for genes and 0 for non-genic regions. We will call A a *labeling* or *annotation* of the input string X.

The desired mapping from X to A is in gene finding often characterized by a probabilistic model defining probability distribution $P(A|X)$. We then output the labeling with the highest probability. Frequently used models are hidden Markov models (HMMs) or their variants such as hidden semi-Markov models and conditional random fields [3, 5]. These models take into account statistical properties of genic and non-genic regions of DNA, such as frequency of k-tuples, length distribution of genes, and characteristic sequence motifs at gene boundaries. Genes found by HMMs are often imprecise, and to increase the accuracy, statistical models of DNA are often combined with additional external information provided by biological experiments. External information is often in the form of individual hints, each hint giving a probable location of one gene or a part of a gene.

To combine an HMM with hints, we can manipulate probabilities of individual labelings A so that the labeling gets a bonus for each hint agreeing with it. Different gene finders define the set of possible hints and their influence on the probability space differently, but the possibilities are restricted by the need for efficient algorithm finding the annotation with highest probability in the modified space. The simplest form of a hint is a *point-wise hint* which increases or decreases probabilities of all annotations that have a certain label at a certain position in the sequence [2, 16]. Such hints can be easily incorporated to the standard Viterbi algorithm for finding the most probable annotations in HMMs.

However, a single piece of evidence often suggests labels for a longer interval of the sequence. Splitting such information into multiple point-wise hints leads to information loss, because some labelings can get the bonus even if they do not agree with the evidence over the whole interval. Therefore, some gene finders [14, 5] use hints in the form of intervals such that the probability of a labeling is increased only if it has the label prescribed by the hint throughout the whole extent of the interval. We call such hints *interval hints.* The main focus of our paper is to study further generalizations of interval hints.

For most of the time we will abstract away both the HMM and the input string X and consider only a set of hints. Indeed, probabilities from the HMM can be expressed as special hints of length 2, as we explain in Section 2. Therefore in the most general setting, we can formulate our problem as follows:

Definition 1 (Optimal labeling with hints problem). *Given is positive integer n, finite alphabet Σ, and a set of hints. Each hint is a pair (γ, b) where $\gamma \subseteq \Sigma^n$ is a set of labelings and $b \in R$ is a bonus. We*

⋆ This research is funded by European Community FP7 grant IRG-231025, VEGA grant 1/0210/10, and APVV grant SK-CN-0007-09.

say that a labeling $A \in \Sigma^n$ agrees with hint (γ, b) if $A \in \gamma$. The score of a labeling $A \in \Sigma^n$ is the sum of bonuses of all hints that agree with A. The goal is to find a labeling with the maximum score.

In our work the hints will assume a special form allowing compact representation of a potentially large set γ. In particular, a *complex hint* is a four-tuple (s, e, Y, b) such that $1 \le s \le e \le n$ and $Y \in \Sigma^{e-s+1}$ is the labeling suggested for $x_s \dots x_e$ (see Figure 1). Set γ for this hint is $\gamma = \Sigma^{s-1} Y \Sigma^{n-e}$. Interval hints are a special case of complex hints with $Y = a^k$ for some $a \in \Sigma$ and point-wise hints are a special case of interval hints with $s = e$.

$h_1 = (1, 4, 0001, 2)$	`0001...`
$h_2 = (2, 3, 00, 1)$	`.00....`
$h_3 = (2, 6, 00100, 1)$	`.00100.`
$h_4 = (3, 6, 0001, 1)$	`..0001.`
$h_5 = (6, 7, 02, 1)$	`.....02`
A	`0001002`

Fig. 1. A set of complex hints for $n = 7$ and $\Sigma = \{0, 1, 2\}$ and an optimal labeling with score 5, agreeing with hints h_1, h_2, h_3 and h_5. Note that h_2 is an interval hint.

In this work we describe an efficient algorithm for finding the optimal labeling for a set of complex hints, which is a non-trivial extension of algorithms for interval hints (Section 2). We also show that additional natural generalization of the problem where some positions between s and e are allowed to vary, leads to an NP-hard problem (Section 3). Finally in Section 4, we describe the use of our algorithm in the context of gene finding and show experimental results on real and simulated data.

2 An algorithm for complex hints

In this section we give a polynomial-time algorithm which finds the optimal labeling for a set of complex hints. We will first discuss several simpler cases, later generalizing the algorithm to the full problem.

Interval hints. Gene finders employing interval hints typically proceed by dynamic programming. For each prefix $x_1 \dots x_i$ of the input string X, they consider all possibilities for the last region of this prefix labeled by one label (that is, all possible $j \le i$ and $a \in \Sigma$ such that $a_j = a_{j+1} = \dots = a_i = a$ and $a_{j-1} \ne a$). For each such interval (j, i), it is easy to compute the sum of bonuses of all interval hints agreeing with label a.

This type of algorithm is dictated by the fact that gene finders combine hints with hidden semi-Markov models or their variants [14, 5] and the Viterbi algorithm for inference in these models already has this form with running time $O(n^2 |\Sigma|^2)$. With appropriate data structures, it is possible to implement the algorithm so that processing of hints adds only an additive term $O(m)$ where m is the number of hints. However, a more efficient algorithm would be desirable if we wanted to use hints with regular HMMs, where the Viterbi algorithm is linear in n. Running time of our algorithm for complex hints with positive bonuses will not depend on the sequence length, only on the number and total length of hints.

Complex hints with positive bonuses. In order to introduce our algorithm for a set of complex hints with positive bonuses, we start with several necessary definitions. Let $h = (s, e, y_1 \dots y_{s-e+1}, b)$ be a complex hint. Then for $i \in [s, e]$ expression $\ell(h, i)$ denotes label suggested by h for x_i, that is, $\ell(h, i) = y_{i-s+1}$. Now let us consider two hints $h_1 = (s_1, e_1, Y_1, b_1)$ and $h_2 = (s_2, e_2, Y_2, b_2)$. We say that h_1 and h_2 are *compatible* if they agree in the intersection of their intervals, that is, if for every $i \in [s_1, e_1] \cap [s_2, e_2]$ we have $\ell(h_1, i) = \ell(h_2, i)$ (non-intersecting hints are always compatible). Hint h_1 is a *subset* of hint h_2 if they are compatible and $[s_1, e_1] \subseteq [s_2, e_2]$. Hint h_2 is an *extension* of h_1 if they are compatible and $e_2 > e_1$ and $s_2 > s_1$. Note that if $s_1 = s_2$, $e_1 = e_2$, and $Y_1 = Y_2$, these two hints can be replaced by a single hint with bonus $b_1 + b_2$. We will assume that such a transformation is done on all applicable pairs.

In our algorithm, we transform the input to create a weighted directed acyclic graph (DAG) such that the optimal labeling corresponds to the path with maximum weight between given two vertices s and t. Such longest paths can be found in DAGs efficiently, in $O(|V| + |E|)$ time [4]. Our algorithm could be also expressed directly as dynamic programming, but the graph representation is more convenient for proving correctness and considering variants of the algorithm.

Our graph has one vertex for each hint and two special vertices s and t. There is an edge from h_1 to h_2 if and only if h_2 is an extension of h_1. The weight of this edge is the sum of the bonus of h_2 and bonuses of all hints that are subset of h_2 but are not a subset of h_1 (see Figure 2). Vertices s and t are connected with other vertices as if they corresponded to sentinel intervals beyond the end of the sequence, namely $(0, 0, a, 0)$ and $(n + 1, n + 1, a, 0)$. Clearly the graph created in this way is acyclic, because vertices on each path have increasing coordinates of their endpoints. Correctness of the algorithm is given by the following theorem.

Theorem 1. *The weight W^* of the maximum-weight path from s to t in the graph described above is equal to the score S^* of an optimal labeling for the input set of hints.*

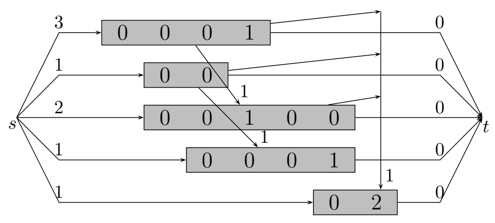

Fig. 2. Directed acyclic graph created by our algorithm for hint set from Figure 3. Vertices for hints are shown as grey boxes.

Proof. First, we will prove that for every labeling A with score S there is a path with weight S and therefore $W^* \geq S^*$. Let $\mathcal{H}$ be the set of hints that agree with A. Let $\mathcal{H}_S$ be the set of all hints from $\mathcal{H}$ that are a subset of another hint from $\mathcal{H}$, and let $\mathcal{H}_R = \mathcal{H} \backslash \mathcal{H}_S$. Let h_1 and h_2 be two different hints from $\mathcal{H}_R$ such that $s_1 < s_2$. Then also $e_1 < e_2$ because no hint in $\mathcal{H}_R$ is a subset of another, and these two hints are compatible because they both agree with A. Therefore, h_2 is an extension of h_1.

Our path will start in s, pass through all vertices in set $\mathcal{H}_R$ ordered from left to right by their left endpoints and end in t. As we have shown, all necessary edges on the path exist. Bonus of hint $h \in \mathcal{H}_R$ is included in the weight of the edge entering h and bonus of hint $h \in \mathcal{H}_S$ is included in the weight of the edge entering the leftmost hint $h' \in \mathcal{H}_R$ such that h is a subset of h'. Bonuses of no other hints are included in the weight of the path, and therefore its weight is exactly S.

Now we will prove that for any path π from s to t with weight W, there is a labeling with score at least W, implying that $W^* \leq S^*$. Let $\mathcal{H}'$ be the set of all hints corresponding to vertices of π except s and t. We will construct a labeling $A = a_1 \dots a_n$ as follows. If a position i is not covered by any hint in $\mathcal{H}'$, its label a_i can be chosen arbitrarily. If i is covered by some hints $h_1, \dots, h_k$ from $\mathcal{H}'$, they suggest the same label for position i and this label is chosen as a_i.

Now let $\mathcal{H}''$ be the set containing all hints from $\mathcal{H}'$ and all hints that are subsets of hints from $\mathcal{H}'$. The score S of path π is the sum of bonuses of hints from $\mathcal{H}''$. Also, each hint from this set agrees with A. The score of A is the sum of all bonuses that agree with it, therefore $S \leq W$. □

We have proved that the weight of the optimal path is equal to the score of the optimal labeling and the proof also implies an algorithm to convert the optimal path to a labeling with the same score. Note however that weights of some suboptimal paths do not correspond to the score of any labeling, as a labeling that agrees with all vertices on a path may also agree with other hints that the path avoids.

Our graph has $O(m)$ vertices and $O(m^2)$ edges where m is the number of hints. As we will show below, it can be constructed in $O(m^2 + \ell)$ time, where ℓ is the sum of lengths of all hints. The running time of the whole algorithm is therefore $O(m^2 + \ell + n)$ where the dependence on n comes only from the necessity to produce a labeling of length n (using arbitrary labels for parts of the sequence not covered by any hint).

Running time improvement for practical instances. In practical gene finding instances, we expect hints to be much shorter than n and to be spread along the length of the sequence. We can modify the graph so that it has fewer edges if every position is covered by at most d hints for some $d < m$. The construction shown above will create edges even between hints that are far apart, and those can be eliminated. The weight of an edge from hint h' to hint h depends on mutual position of these hints, because we include only those subset hints of h that end after the end of h'. However, the weight will be the same for all hints h' that do not intersect h. Our goal is to remove such edges from the graph and replace them with a smaller number of edges linear in m.

We will say that a hint h_2 is a *strict extension* of a hint h_1 if h_1 is an extension of h_2 and the two hints overlap. Our new graph will have a vertex for each hint, two special vertices s and t for sentinel hints and a *skipping vertex* v_i for every position i which is a right endpoint of at least one hint (including position 0 where the sentinel hint for s ends). Skipping vertices are connected to a chain going from left to right with edges of weight 0. Vertex for a hint h is connected to all hints that are its strict extensions, with the same edge weights as before. It is also connected to the skipping vertex for its right endpoint with an edge of weight 0. Finally, each skipping vertex v_i is connected to all hints that start in the interval $(i, j]$ where v_j is its neighbor in the chain of skipping vertices. The edge from v_i to h will contain the bonus of h and all its subset bonuses. Paths in this new graph have 1-1 correspondence with the paths in the original graph, where an edge between two non-overlapping hints is now replaced by a path through skipping vertices. The number of vertices is $O(m)$ and the number of edges $O(md)$, since from a vertex for hint h there are at most d outgoing edges: one to a skipping vertex and at most $d-1$ to other hints, because all these hints cover the right endpoint of h. Similarly the number of incoming edges is at most d. Finally, the number of edges between skipping vertices is $O(m)$. The overall time of the algorithm with this graph will be $O(md + \ell + n)$; we still need to demonstrate that the graph can be constructed within this running time.

The critical ingredient is the ability to check in constant time whether two hints are compatible. This can be achieved by building a suffix tree of the strings associated with individual hints and preprocessing the suffix tree for longest extension queries. These queries allow us to take two suffixes of any of the strings and determine the length of their common prefix. In our scenario we take two hints (s_1, e_1, Y_1, b_1), (s_2, e_2, Y_2, b_2) and if they overlap by some length $d > 0$, we determine the positions of the start of the overlap in strings Y_1 and Y_2. If the suffixes starting at these two positions have a common prefix of length at least d, the two hints are compatible. They are also compatible if $d = 0$. The preprocessing of necessary structures can be done in $O(\ell)$ time, and compatibility of each pair of hints can be then assessed in $O(1)$ time [8].

To find the edges between hint vertices, we first sort all hints by left endpoints. This can be done in $O(n+m)$ time by counting sort. Next, we traverse the sorted list and maintain a set of *active hints.* A hint is active if it overlaps the left endpoint of the current hint in the list. In every step, the list contains at most d hints, and therefore we can in each iteration traverse the list and remove hints that are no longer active. The current hint h is a strict extension of any active hint h' such that h' and h are compatible and h is not a subset of h'. This can be checked in $O(1)$ time per pair of hints. If the current hint h is a subset of some active hint h', we will append h to a list of subsets of h'.

At the end of traversal, we have for each hint h the list of its subset hints and the list of incoming edges from other hint vertices. We sort both these lists by right endpoints by counting sort in time proportional to the length of h (the right endpoints of all these hints are within h). Now we use a merge-like algorithm with two pointers to the two lists to find for every incoming edge from hint h' the bonus sum of subset hints that end after the end of h'. Edges incident to skipping vertices can be easily constructed within the desired running time. The overall running time of the graph construction is thus $O(md + \ell + n)$ where the n term comes from sorting and can be replaced by $O(m \log m)$.

Complex hints with arbitrary bonuses. The algorithm described above is efficient, but can only handle hints with positive bonus values. If a hint h has a negative bonus value, the path could skip its vertex in the graph, even if other visited vertices imply that the labeling agrees with h and its bonus should be counted. The graph may thus contain paths with weight higher that the score of the optimal labeling.

As we will show, we can transform a set of hints with arbitrary bonuses to an equivalent set of hints with positive bonuses. Each hint (s, e, Y, b) with a negative bonus b is transformed to a set of hints of the form $(s, e', Y', |b|)$ such that $s \le e' \le e$ and Y' is the same as Y on the first $e' - s$ positions and differs from Y on its last position. We create all $(|\Sigma| - 1)$ $(e - s + 1)$ combinations of e' and Y' of this form. Clearly, these hints are mutually incompatible, and therefore at most one of them will agree with any annotation. An annotation A agrees with one of these hints if and only if it does not agree with the original hint h. This transformation increases the score of every annotation by $|b|$: the annotations agreeing with h will increase by $|b|$ because hint h is omitted from the set and all other annotations agree with one of the new hints, thus getting an additional bonus of $|b|$. When we transform all hints with negative values in this way, we increase the total score of every labeling by the sum of absolute values of all negative bonuses. This is a constant and therefore the optimal labeling will remain the same and can be found by the algorithm described above. Unfortunately, the number of hints m as well as their total length ℓ increase by a factor of $L|\Sigma|$ where L is the length of the longest negative hint, but the running time is still polynomial.

Combination of complex hints with an HMM. In gene finding, we typically (although not exclusively [1]) combine the score from hints with a probability value from some model capturing typical sequence features of genes and their constituents. Probability of a particular labeling A in hidden Markov models or conditional random fields can be written as

$$P(A) = \prod_{i=1}^{n} p(i, a_i, a_{i+1}, X),$$

where factors $p(i, a_i, a_{i+1}, X)$ depend only on two adjacent labels a_i and a_{i+1} (and some portion of the input string X). They are computed from the emission and transition probabilities of the model. We will consider the case when the total score of A has the form

$$\mathcal{B} + \log P(A) = \mathcal{B} + \sum_{i=1}^{n} \log p(i, a_i, a_{i+1}, X),$$

where $\mathcal{B}$ is the sum of bonuses of all agreeing hints. The log values in the sum can be written as hints of length 2 of the form $(i, i+1, a_i a_{i+1}, \log p(i, a_i, a_{i+1}, X))$. Overall, we will need $n|\Sigma|^2$ such hints and they typically all have negative score. After applying the transformation described in the previous paragraphs and combining together hints with the same values of s, e and Y, we get $O(n|\Sigma|^2)$ additional hints of length 1 and 2 with positive bonuses.

This generic procedure works only if the states of the HMM correspond exactly to the symbols of the output alphabet Σ. Gene finding HMMs typically have a much larger state space, with several states corresponding to the same output label. This situation could be expressed using more general subset hints described in Section 3. Although subset hints lead in general to NP-hard problems, special cases necessary for handling large state spaces can be solved efficiently, as discussed at the end of the next section. Moreover, it is also possible to create specialized algorithms for combining complex hints with a hidden Markov model [12]. By taking the special structure of the problem into account, we may obtain more efficient solutions than by converting the HMM to a collection of hints.

3 NP-hardness for subset hints

A subset hint is a four-tuple $h = (s, e, Y, b)$ such that numbers s, e, and b are defined as for a complex hint and Y is a string of length $e - s + 1$ over an extended alphabet Σ_+ containing all non-empty subsets of Σ. Expression $\ell(h, i)$ now denotes the set of labels suggested by h for x_i and a labeling A agrees with h if for every position $i \in [s, e]$ the label a_i is in the set $\ell(h, i)$. Unfortunately, our algorithm for complex hints cannot be straightforwardly extended to this scenario, because the relation of extension is not transitive, which is crucial for the existence of a labeling satisfying all hints on a path in our graph.

As we will prove, it is unlikely that the problem can be solved efficiently in the full generality by other methods, since it is NP-hard.

Theorem 2. *The problem of testing whether there is a labeling A with score at least t for a set of subset hints is NP-complete, even for a binary alphabet Σ.*

Proof. Clearly the problem is in NP (if the size of input is much smaller than n, we can specify only portions of A covered by hints and this is sufficient for efficient computation of the score).

To prove NP-hardness, we will use a reduction from the 3-SAT problem. Consider a 3-SAT instance with N variables $y_1, \ldots, y_N$ and M clauses, each clause consisting of three literals. We will encode it as a set of hints over the alphabet $\Sigma = \{0, 1\}$ such that the formula has a satisfying assignment if and only if there is a labeling with score at least $M + MK$ where $K = 3M + 1$.

The length of the annotated sequence will be $n = N + 3M$. The first N labels correspond to a truth assignment to all variables, that is, $a_i = 1$ corresponds to y_i being true. Each of the remaining labels corresponds to one literal from the formula, namely $a_{p_{i,j}}$ corresponds to jth literal from ith clause, where $p_{i,j} = N + 3(i-1) + j$. For each such literal we will add an *assignment hint* of the form $(k, p_{i,j}, z\{0,1\}^{p_{i,j}-k-1}1, 1)$ where k is the index of the variable forming this literal and $z = 1$ if the literal is y_k and $z = 0$ if the literal is $\neg y_k$. Notation $\{0, 1\}^c$ represents c copies of the set $\{0, 1\} \in \Sigma_+$. In other words, the hint connects the literal with its satisfying variable assignment, and if the literal is *selected* by $a_{p_{i,j}} = 1$ and the variable y_k has the satisfying assignment, the labeling gets a bonus 1. We will also add three *selector hints* for every clause, that enforce that exactly one literal from the clause is selected by $a_{p_{i,j}} = 1$. These hints have the form $(p_{i,1}, p_{i,3}, Y, K)$ where Y is a binary string of length three containing exactly two zeroes. Note that for each clause at most one of the selector hints can agree with a given labeling.

In order to achieve total score $M + MK$, the annotation has to agree with one selector hint for every clause. The loss of even one such selector could not be compensated because the sum of bonuses of all assignment hints is only $3M$, which is less than K. Selector bonuses that contribute bonus MK enforce that exactly one literal in each clause is selected. The selected literal has a potential to contribute bonus 1 if it is indeed satisfied in the assignment specified by the labeling of the first N symbols. If the formula has score at least $M + MK$, every clause has exactly one literal which is simultaneously selected and satisfied. Note that multiple literals per clause may be satisfied, but if they are not selected, they will not influence the score. Annotation with score $M + MK$ thus corresponds to a satisfying assignment of the formula and vice versa. □

This proof is a slight modification of our earlier proof for the use of RT-PCR queries in gene finding [10]. In this earlier work we have considered hints of a special form that suggest a specific label at both ends and allow arbitrary labels in the middle. However the output labeling was constrained to obey an additional DAG of possible gene structures. Here we have adapted the proof for the simpler case where the annotation can be arbitrary.

Partition subset hints. Subset hints often occur in practice because our methods for obtaining additional information about the labeling cannot distinguish between some labels from Σ. In some situations we can represent subset hints as complex hints over an alphabet $\Sigma' \subseteq \Sigma_+$ that forms a partition of Σ into equivalence classes (in other words, every two elements in Σ' are disjoint subsets of Σ). We will now consider a situation where every hint contains only characters from Σ or only characters from Σ' (but cannot use characters from both). We will show that optimal labeling (over Σ) for such hint sets can be found in polynomial time.

Briefly, we first create a directed acyclic graph G for hints over Σ similarly as in Section 2. We add new hints of the form $(i, i, a, 0)$ for all positions i and labels $a \in \Sigma$. Both the original hints and these new hints will be vertices. We will have an edge from hint h_1 to hint h_2 if h_2 is an extension of h_1 and they either overlap or h_2 starts immediately after h_1 ends. In this graph, every path consists of hints that completely cover the sequence. In the same way we also build a graph G' for hints over Σ'. Finally, we create graph G'' in which each vertex is a pair (h, h') such that h is a vertex in G, h' is a vertex in G', hints h and h' overlap and are compatible. If hint h' ends before h, this vertex will be connected to vertices of the form (h, h'') where (h', h'') is an edge in G', the weight of the edge is also copied from G'. Similarly, if h ends before h', (h, h') will be connected to vertices of the form (h'', h') where (h, h'') is an edge in G. Finally, if the two hints end together, they will be connected to vertices of the form (h'', h''') where (h, h'') is an edge in G and (h', h''') is an edge in G'. The weight of the new edge will be the sum of the two source edges. The best path in G'' then corresponds to the optimal annotation (proof omitted due to space).

The algorithm can be extended to several groups of hints, each over a different partition of Σ, but the running time would increase exponentially in the number k of such groups, because the vertices in the final graph will be k-tuples of vertices of graphs for individual groups of hints.

4 Use of complex hints in gene finding

We have implemented a variant of the algorithm from Section 2 and applied it to the problem of gene finding. Recall that the goal of gene finding is to find genes, regions encoding proteins in DNA sequences. A gene in eukaryotic organisms can be divided into several segments called *exons* and *introns*, and only exons encode the protein (see Figure 3). Thus the goal of gene finding is to find the position of each gene and its exact partitioning to exons and introns. We use output alphabet $\Sigma = \{C, I, X\}$, where C stands for coding exons, I for introns, and X for intergenic regions.

The algorithm was implemented in C++ as a standalone console application called GRAPHHMM. In order to process long DNA sequences, we implemented several practical improvements. For example, the graph is generated on the fly and unnecessary information is immediately deleted to free up memory.

We have tested the program on genomic data from *Drosophila melanogaster* (fruit fly). This species is an important model organism in genetics and its genome is relatively well sequenced and annotated. Genomic sequences and reference annotations were downloaded from the UCSC genome browser [7]. We have used 44 MB of DNA sequence containing 5 164 genes. They were split to 2 MB parts and divided to training (≈ 28 MB, 3368 genes) and testing (≈ 16 MB, 1796 genes) data sets.

The HMM has been taken from the work of [15] and retrained on our training data set by standard procedures [6]. It has 265 states, but it is quite simple compared to state-of-the-art gene finders. For example, it does not allow arbitrary length distributions of exons, introns and intergenic regions and uses simpler models of sequence motifs at exon boundaries. In our experiments we compare the accuracy achieved on the testing set by the HMM alone and with different sets of hints.

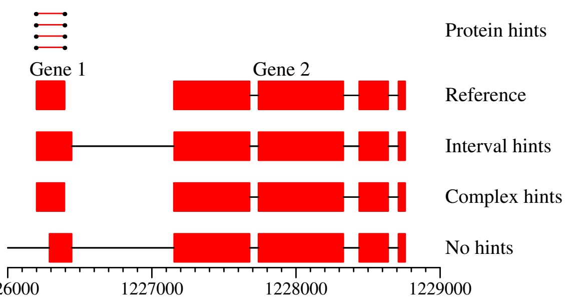

Fig. 3. Example of a prediction obtained with different protein hint sets in a region containing two genes. Lines depict introns, rectangles exons (gene 1 has one exon, gene 2 has four exons). Four identical complex hints span the whole length of gene 1 and suggest short intergenic region beyond its boundary. Prediction with complex hints is correct. Prediction with interval hints does not agree with the intergenic hint at the right end of the gene. Prediction without hints predicts a long gene extending beyond the displayed region.

Experiment with protein hints. In the first experiment, we have used a real set of hints originating from a database of known proteins. We have downloaded all known proteins from several insect species (*D. melanogaster, D. simulans, D. pseudoobscura, Anopheles gambiae*, and *Bombyx mori*) from the NCBI RefSeq database [13] (67, 893 protein sequences in total). We have used the BLAT program [9] and scripts from ExonHunter distribution [2, 11] to find regions in the testing set that could encode identical or similar proteins to those in the database. Weak matches were were filtered out to increase the specificity. As a result of this strict filtering, only 11% of the sequence is covered by hints, whereas genes cover almost 45% of the sequence. Hints contain exon and intron labels (exons are regions where the protein seems to be encoded and introns are separating parts encoding the same protein). If the whole source protein could be aligned to the genome, a single intergenic position is added to the beginning and to the end of the complete gene hint.

Our goal was to compare prediction accuracy obtained using complex hints with accuracy obtained using simpler interval and point-wise hints. Therefore we have converted our hint set to these simpler forms by splitting each complex hints to several shorter hints as necessary. Overall we have thus used three different hint sets:

– *complex hint set* (15,057 training hints, 5,545 testing hints)
– *interval hint set* (39,900 training hints, 14,358 testing hints)
– *point-wise hint set* (8,651,653 training hints, 4,042,685 testing hints)

Bonuses of hints have great influence on the gene prediction accuracy: if they are too low, the hints will be ignored; if they are too high, wrong hints will decrease the gene prediction accuracy. For simplicity, all hints in one hint set have the same bonus b, and we choose optimal value of b to maximize the accuracy on the training set for each set of hints separately. The exception are complex hints, where the bonus of a hint is bk where k is the number of contiguous regions in the hint labeled by one label (or in other words the number of interval hints comprising this complex hint). Again the value of b was optimized using the training set.

Evaluation of these three hints sets on the test sequences is shown in Table 1. Complex hints slightly increase the number of completely correctly predicted genes compared to the interval hints, but the difference is less than 1%. On the other hand, difference in accuracy between prediction without hints and with one of the hint sets is quite significant.

Figure 3 shows an example of a gene where complex hints lead to a better prediction than interval hints. The algorithm with interval hints chooses an annotation agreeing with only two out of three parts comprising a single complex hint. With complex hints only annotations agreeing with the whole hint get the bonus, and one such annotation is indeed the optimum.

Experiment with simulated hints. To better control various parameters of the hint set, we have also generated artificial hints. All hints in this set have the length 1000, and the number of hints was chosen so that the sum of their lengths is approximately $n/2$. Half of the hints were created so that they start at a random place within some exon and agree with the reference annotation along their whole length. The second half of hints was generated so that they start at a random place in the sequence and suggest label for intergenic regions along their whole length. Some of these hints are correct, since by chance they are contained in real intergenic regions, whereas others overlap real genes. Note that we have chosen intergenic hints, because it is non-trivial to randomly generate other reasonable wrong hints that have a non-zero probability in the HMM. Overall, 74% of hints in our set were correct. In this experiment, we have compared complex and interval hints, again training their bonus on the training set and testing their performance on the testing set. Results shown in Table 2 are analogous to the protein hint experiment – slight improvement in the prediction accuracy at the gene level for complex hints compared with interval hints.

Hint set	No hints	Point-wise	Interval	Complex
Bonus	–	8	8	6
Gene sn.	48.83%	61.08%	61.30%	61.86%
Gene sp.	40.40%	46.00%	46.42%	46.61%
Exon sn.	71.18%	75.89%	76.20%	76.14%
Exon sp.	69.01%	71.73%	72.31%	72.29%
Base sn.	92.81%	94.35%	94.15%	94.14%
Base sp.	91.86%	92.00%	92.10%	92.08%

Tab. 1. Comparison of hint sets created from protein sequences. For each set, we show the trained value of the bonus parameter as well as several standard measures of gene finding accuracy. Gene sensitivity (sn.) is a fraction of real genes that were predicted completely correctly. Gene specificity (sp.) is the fraction of predicted genes that are completely correct. Similarly, we measure sensitivity and specificity at the exon and base level, where we count completely correctly predicted exons and individual symbols in coding exons.

Hint set	Interval	Complex
Bonus	6	12
Gene sensitivity	60.75%	61.53%
Gene specificity	49.43%	50.41%
Exon sensitivity	77.31%	77.92%
Exon specificity	76.51%	76.92%
Base sensitivity	94.98%	94.69%
Base specificity	93.52%	93.39%

Tab. 2. Comparison of gene prediction accuracy with artificial hints sets, using the same accuracy measures as in Table 1.

5 Conclusion and open problems

In this paper, we have explored several variants of the problem of finding optimal sequence annotation that agrees with hints with the highest total score. If the hints completely specify annotation within some interval and all hints have positive scores, the problem can be solved quite efficiently, in time $O(md + \ell + n)$ where n is the length of the annotated sequence, ℓ is the total length of all hints, m is the number of hints and d is the maximum number of hints overlapping a single position. If some hints have negative scores, the problem can still be solved in polynomial time. Finally, we can also solve the case where hints are over two different alphabets, one being a partition of the other. These algorithms can be also extended to combine hints with hidden Markov models or conditional random fields. We have also shown that if we allow wildcards in hints, the problem becomes NP-hard even for binary output alphabet.

Our results might be applicable in various fields where HMMs and their variants need to be combined with external information. However, our original motivation stems from gene finding, where complex and subset hints are the most natural form of expressing hints from various sources. Our experiments with a simple gene finder show that compared to simpler interval hints used before, complex hints may lead to slight increases in prediction accuracy. More experiments with different information sources or in different species may lead to more significant improvements.

Several open problems remain in this area. Our algorithms are in the worst case quadratic in the number of hints. The question is whether more efficient algorithms exist at least for the case of interval hints. It might be also useful to consider special classes of subset hints that can be processed in polynomial time. For example in our earlier work the RT-PCR hints were NP-hard in general but solvable efficiently if their relative position was constrained [10]. Finally, one could also generalize complex hints in other ways, for example, allowing some uncertainty in the exact place where labeling changes from one label to another.

References

1. J.E. Allen, and S.L. Salzberg: . *JIGSAW: integration of multiple sources of evidence for gene prediction.* Bioinformatics, 21(18), 2005, :3596–3603.
2. B. Brejová, D.G. Brown, M. Li, and T. Vinař: *ExonHunter: a comprehensive approach to gene finding.* Bioinformatics, 21 Suppl 1:, 2005, 57–65.
3. C. Burge, and S. Karlin: *Prediction of complete gene structures in human genomic DNA.* Journal of Molecular Biology, 268(1), 1997, 78–94.
4. T.H. Cormen, C.E. Leiserson, and R.L. Rivest: *Introduction to algorithms, second edition.* The MIT Press and McGraw-Hill Book Company, 1989.
5. D. DeCaprio, J.P. Vinson, M.D. Pearson, P. Montgomery, M. Doherty, and J.E. Galagan: *Conrad: gene prediction using conditional random fields.* Genome Research, 17(9), 2007, 1389–1398.
6. R. Durbin, R. Eddy, A. Krogh, and G. Mitchison: *Biological sequence analysis.* Cambridge University Press, 1998.
7. P.A. Fujita, B. Rhead, A.S. Zweig, et al. (2011). *The UCSC Genome browser database: update 2011.* Nucleic Acids Research, 39(Database issue), 2011, D876–882.
8. D. Gusfield: *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge University Press, 1997.
9. W.J. Kent: *BLAT* – the BLAST-like alignment tool. Genome Research, 12(4), 2002, 656–664.
10. J. Kováč, T. Vinař, and B. Brejová: *Predicting gene structures from multiple RT-PCR tests.* In Algorithms in Bioinformatics, 9th International Workshop, WABI 2009, volume 5724 of LNCS, pp. 181–193. Springer, 2009.
11. P. Kováč: *Implementácia externých zdrojov dát v hl'adaní génov.* Bachelor Thesis, Department of Computer Science, Comenius University in Bratislava, 2010.
12. M. Kucharík: *A new algorithm for using external information in gene finding.* Master's Thesis, Department of Computer Science, Comenius University in Bratislava. Submitted 2011.
13. K.D. Pruitt, T. Tatusova, and D.R. Maglott: *NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins.* Nucleic Acids Research, 35(Database issue), 2007, D61–65.
14. M. Stanke, M. Diekhans, R. Baertsch, and D. Haussler: *Using native and syntenically mapped cDNA alignments to improve de novo gene finding.* Bioinformatics, 24(5), 2008, 637–644.
15. R. Šrámek, B. Brejová, and T. Vinař: *On-line Viterbi algorithm for analysis of long biological sequences.* In Algorithms in Bioinformatics: 7th International Workshop (WABI), volume 4645 of Lecture Notes in Computer Science, pp. 240–251. Springer.
16. R.F. Yeh, L.P. Lim, and C.B. Burge: *Computational inference of homologous gene structures in the human genome.* Genome Research, 11(5), 2001, 803–806.