A String Pattern Regression Algorithm and Its Application to Pattern Discovery in Long Introns

Hideo Bannai[1] bannai@ims.u-tokyo.ac.jp
Shunsuke Inenaga[2] s-ine@i.kyushu-u.ac.jp
Ayumi Shinohara[2,3] ayumi@i.kyushu-u.ac.jp
Masayuki Takeda[2,3] takeda@i.kyushu-u.ac.jp
Satoru Miyano[1] miyano@ims.u-tokyo.ac.jp

[1] Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokane-dai, Minato-ku, Tokyo 108-8639, Japan
[2] Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
[3] PRESTO, Japan Science and Technology Corporation (JST)

Abstract

We present a new approach to pattern discovery called *string pattern regression*, where we are given a data set that consists of a string attribute and an objective numerical attribute. The problem is to find the best string pattern that divides the data set in such a way that the distribution of the numerical attribute values of the set for which the pattern matches the string attribute, is most distinct, with respect to some appropriate measure, from the distribution of the numerical attribute values of the set for which the pattern does not match the string attribute. By solving this problem, we are able to discover, at the same time, a subset of the data whose objective numerical attributes are significantly different from rest of the data, as well as the splitting rule in the form of a string pattern that is conserved in the subset. Although the problem can be solved in linear time for the substring pattern class, the problem is NP-hard in the general case (i.e. more complex patterns), and we present an exact but efficient branch-and-bound algorithm which is applicable to various pattern classes. We apply our algorithm to intron sequences of human, mouse, fly, and zebrafish, and show the practicality of our approach and algorithm. We also discuss possible extensions of our algorithm, as well as promising applications, such as microarray gene expression data.

Keywords: pattern discovery, string pattern matching, regression, long introns

1 Introduction

Pattern discovery from the current wealth of biological sequences is one of the most important problems in the field of bioinformatics. The aim is to find patterns which are conserved across different biological sequences, since such patterns might indicate that the sequences share the same or similar functions. Computational analyses which provide such candidates can be very helpful, since they can guide the biologists in the task of identifying and experimentally confirming the actual sequence in play.

In a broad perspective, traditional settings in pattern discovery can be classified into two cases [3]. The first is when we are given a single set of sequences, or a *positive data set* and we would like to find patterns which are conserved in the sequences, generally preferring *longer* patterns which appear in *most* of the sequences in the set. Solving such a problem ranges from a plethora of multiple alignment methods, and numerous other methods such as Gibbs sampling [10] or MEME [1]. Another situation is when we have more information at hand, and are given a second set of sequences as an explicit *negative data set* that is known not to have characteristics of (or known to have different characteristics from) the first set, and the problem is to find a pattern which match most of the sequences in one set, while not matching those of the other set [8, 13, 14].

In this paper, we consider a new situation where we are given a single set of sequences, but this time, we are presented with more information in the form of a numerical value expressing some objective measure of interest, paired with each sequence. We formalize the situation and define the *string pattern regression* problem for a general pattern class. The numerical values in our situation gives us quantitative information concerning the sequences, and our intention is to find the best pattern which is conserved in a subset of the sequences for which the distribution of numerical values of the subset is most *different* from the distribution of the numerical values of the rest. Making use of both the string attribute and numerical attribute, we are able to discover, at the same time, a subset of the data whose objective numerical attributes are significantly different from rest of the data, as well as the splitting rule in the form of a string pattern that is conserved in the subset. Although the problem can be solved in linear time for the substring pattern class, the problem is NP-hard in the general case, and we will present an efficient branch-and-bound algorithm which solves the problem exactly. We can apply our algorithm with various pattern classes, such as subsequence patterns, approximate patterns, VLDC patterns [8], and many others.

Compared with the previous situations, the string pattern regression problem can be viewed as a generalization of the situation where we have both a positive and negative sequence set, since we could set the numerical attributes to 1 for the positive set, and -1 for the negative set, which would then represent an equivalent problem. A natural advantage of our new situation is that more general values can be assigned to each sequence. Also, the data set does not need to be divided *a priori* into positive and negative data sets, for example, according to some threshold for the numerical attributes, which we would have to do in the previous situation. Another way to view the problem is in the context of regression, where it can be seen as trying to predict the numerical value from a binary value (whether a pattern matches or not), and the aim is to find the pattern which best predicts the numerical value.

In Section 2, we will define the problem of string pattern regression for a general pattern class. In Section 3, we will present the branch-and-bound algorithm which solves the problem exactly, and give examples of pattern classes to which our algorithm can be applied. To show that our methods are useful in practice, in Section 4, we describe computational experiments of applying our algorithm to the acceptor site sequences of introns, paired with the logarithm of the (whole) intron length. The results strengthen our recent observation that there is a correlation between the intron length and its base composition in some species [2] (U over C, and A over G for longer introns), and especially the importance of U's in the polypyrimidine tract near the acceptor site for longer introns. We discuss possible extensions to our algorithm, as well as promising applications of our approach, in Section 5.

2 Preliminaries

2.1 Notation

Let $\mathbf{N}$ be the set of non-negative integers, and $\mathbf{R}$ the set of real numbers. Let Σ be a finite set of characters called the *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $l(w)$. The empty string is denoted by ε, that is, $l(\varepsilon) = 0$. Strings x, y, and z are said to be a *prefix*, *substring*, and *suffix* of string $w = xyz$, respectively. For a set $T \subseteq \Sigma^*$ of strings, the number of strings in T is denoted by $|T|$ and the total length of strings in T is denoted by $l(T)$. Let 2^Σ denote the power set of Σ, which is the set of all subsets of Σ.

A *pattern class* is a pair $\mathcal{C} = (\Pi, m)$, where Π is a set called the *pattern set* and $m : \Sigma^* \times \Pi \to \{-1, 1\}$ is the *pattern matching function*. An element $p \in \Pi$ is called a *pattern*. For a pattern $p \in \Pi$ and string $t \in \Sigma^*$, we say *p of class $\mathcal{C}$ matches t* if $m(t, p) = 1$, and *p of class $\mathcal{C}$ does not match t*, otherwise.

For example, the *substring pattern class* is defined with the pattern set Σ^* and the pattern matching function *strstr* given by:

$$strstr(t, p) = \begin{cases} 1 & p \text{ is a substring of } t \\ -1 & \text{otherwise} \end{cases}$$

Other examples of pattern classes will be given in Section 3.2. For a pattern p, we denote by $L_{\mathcal{C}}(p)$ the set of strings in Σ^* which p of class $\mathcal{C}$ matches. We will write $p \preceq_{\mathcal{C}} p'$ if $L_{\mathcal{C}}(p) \supseteq L_{\mathcal{C}}(p')$.

2.2 Problem Definition

Here, we define the problem of string pattern regression. We assume a set of data having two attributes: a string attribute and an objective numerical attribute. Let $D \subset \Sigma^* \times \mathbf{R}$ denote this data set. For a pattern $p \in \Pi$, the data set D can be split into two sets: D_p, which represents the subset of D for which the pattern p matches the string attribute, and $D_{\bar{p}}$, where p does not match the string attribute. The aim of our approach is to find the pattern for which the distributions of the objective numeric attribute of D_p is "best distinguished" from that of $D_{\bar{p}}$.

There can be many measures for the *goodness* of such splits, but we will define the problem as follows:

Definition 1 (String pattern regression) Given a data set $D \subset \Sigma^* \times \mathbf{R}$ and a pattern class $\mathcal{C} = (\Pi, m)$, the *string pattern regression problem* is to find $p \in \Pi$ such that the mean squared error defined below is minimized:

$$MSE(D,p) = \frac{\sum_{(s,r)\in D_p}(\mu(D_p) - r)^2 + \sum_{(s,r)\in D_{\bar{p}}}(\mu(D_{\bar{p}}) - r)^2}{|D|},$$

where $D_p = \{(s,r) \mid (s,r) \in D \text{ and } s \in L_{\mathcal{C}}(p)\}$ is the subset of D for which p matches the string attribute, $D_{\bar{p}} = D - D_p$, and $\mu(D') = \sum_{(s,r)\in D'} r/|D'|$ represents the average of the objective numerical attribute values of the data in $D' \subseteq D$.

Minimizing the mean squared error can be achieved by maximizing the interclass variance (ICV) [12], defined by:

$$ICV(D,p) = |D_p|(\mu(D) - \mu(D_p))^2 + |D_{\bar{p}}|(\mu(D) - \mu(D_{\bar{p}}))^2 \tag{1}$$

This measure is more convenient for our calculations, and will be used for the remaining of the paper.

For the substring pattern class, the problem can be solved in linear time, in the summed length of all the string attributes, by a clever use of suffix trees for strings, extending the algorithm of [7] to incorporate the numerical attributes. However, the problem is NP-hard in general, for example, for more complex pattern classes such as subsequence patterns. This paper will focus on giving an efficient but exact branch-and-bound algorithm for solving this problem in a general case, applicable to various pattern classes, such as subsequence patterns, approximate patterns (with substitution/insertion/deletion), patterns with *don't care* characters, patterns with classes of alphabets, VLDC patterns, etc.

3 Branch-and-Bound Algorithm

In this section, we present a branch-and-bound algorithm for solving the string pattern regression problem. In principle, the algorithm proceeds by enumerating all of the patterns in Π. The idea behind our branch-and-bound heuristics is as follows: at a given point in the algorithm, when we have calculated the subset of the data for which the pattern p matches, we can calculate the upperbound of the interclass variance for any pattern p' such that $p \preceq_{\mathcal{C}} p'$. If the upper bound is less than the current maximum value, we can stop considering such patterns and therefore prune the search space. Since the overall strategy for the enumeration of patterns is essentially the same as [8, 14], we will only briefly describe this procedure. However, the calculation of the upperbound is somewhat different, which we will describe in detail.

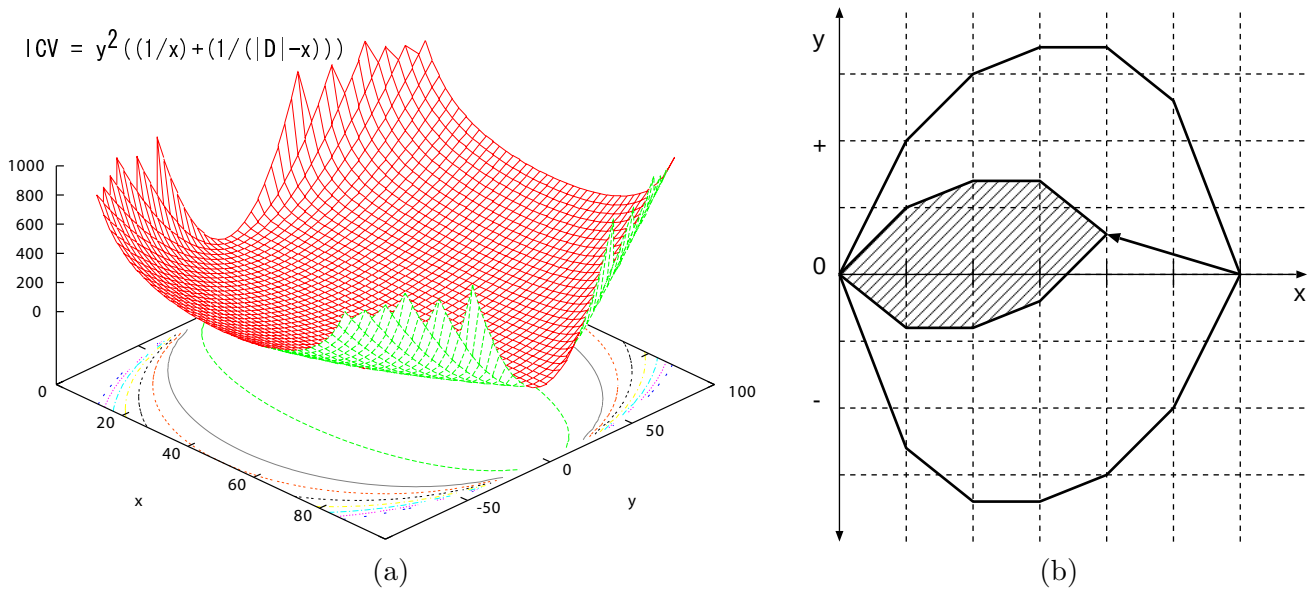

Figure 1: (a) The interclass variance ICV as a function of x and y for $|D| = 100$. (b) The region depicting the possible values of x, y for any subset of a given data set D. The shaded region depicts a smaller region, after 2 elements are removed from D. An upper bound for the ICV of any point inside of the shaded region can be obtained by calculating the maximum ICV for the vertices of the convex hull enclosing the shaded region.

3.1 Upperbound Calculation

We can normalize the numeric values so that their average will be 0, that is, $\mu(D) = 0$. Under this conversion, let $x_p = |D_p|$ and $y_p = \sum_{(s,r)\in D_p} r$ for a pattern p. Since $\mu(D_p) = \frac{y_p}{x_p}$ and $\mu(D_{\bar{p}}) = \frac{-y_p}{|D|-x_p}$, the interclass variance can be written as:

$$ICV(D,p) = f_{|D|}(x_p, y_p) = \begin{cases} 0 & \text{if } x_p = 0 \text{ or } x_p = |D| \\ y_p^2 \left(\frac{1}{x_p} + \frac{1}{|D|-x_p}\right) & \text{otherwise} \end{cases}$$

For any subset $D' \subseteq D$, we can define the pair $(x_{D'}, y_{D'}) = (|D'|, \sum_{(s,r)\in D'} r)$ and we call it the *stamp point* of D'. If $D' = D_p$ for some pattern p, we also call it the *stamp point* of pattern p.

Figure 1(a) shows the function $f_{100}(x, y)$. Figure 1(b) shows the region of possible values for (x, y), for some subset of the data. For any set D_p, let $H(D_p)$ denote the convex hull enclosing the region of the possible stamp points for all $D' \subseteq D_p$. Although the number of stamp points inside $H(D_p)$ can be exponential in $|D_p|$, $H(D_p)$ can be obtained by first sorting the data set according to the objective numerical attribute, and then summing the values in order (i.e. given $x = k$, the maximum (minimum) possible value for y is obtained by using the k largest (smallest) elements).

From the convexity of f [12], the following property holds:

Lemma 1 For any $0 \leq x \leq c$ and $y_1 \leq y \leq y_2$, we have $f_c(x, y) \leq \max(f_c(x, y_1), f_c(x, y_2))$.

This gives us the following lemma.

Lemma 2 For a given data set $D_p \subseteq D$, the stamp point (x, y) for any $D' \subseteq D_p$ lies inside of, or is a vertex of $H(D_p)$. The maximum value for $f_{|D|}(x, y)$ for points in $H(D_p)$ is given by a vertex of $H(D_p)$.

Since if $p \preceq_{\mathcal{C}} p'$, then $L_{\mathcal{C}}(p) \supseteq L_{\mathcal{C}}(p')$,

Lemma 3 Given a pattern p of class $\mathcal{C}$ and data set D, for any p' such that $p \preceq_{\mathcal{C}} p'$, the stamp point (x'_p, y'_p) of p' lies inside of, or is a vertex of $H(D_p)$.

For our problem, since we must use a string pattern to divide the data set, we do not know immediately whether there exists a pattern whose stamp point is a given pair (x, y). Nevertheless, we can use the maximum value of the ICV for the vertices in $H(D_p)$ as an upperbound for any point inside $H(D_p)$. From the above lemmas, we are able to calculate the upperbound of any p' which satisfies $p \preceq_{\mathcal{C}} p'$ by calculating the maximum value of f for the vertices of $H(D_p)$. The calculation takes $O(|D| \log |D|)$ time for the initial sorting of the data set, and $O(|D_p|)$ for each p.

3.2 Pattern Classes and Pattern Enumeration

We briefly describe several pattern classes and enumeration methods to which our branch-and-bound algorithm can be applied.

substring pattern: $\mathcal{C} = (\Sigma^*, strstr)$. Defined in Section 2.

subsequence pattern: $\mathcal{C} = (\Sigma^*, subseq)$. $subseq(t, p) = 1$ if p can be obtained by removing any number of characters from t.

pattern with don't care characters: $\mathcal{C} = (\{\Sigma \cup \{.\}\}^*, dcstrstr)$. '.' is the don't care character. $dcstrstr(t, p) = 1$ if a substring of t can be obtained by substituting each don't care character with a character in Σ.

VLDC pattern: $\mathcal{C} = (\{\Sigma \cup \{\star\}\}^*, vldc)$. $\star$ is the variable length don't care symbol, or a wild card that can represent 0 or more characters. $vldc(t, p) = 1$ if t can be obtained from p by substituting each of the $\star$'s in p with an element of Σ^*.

pattern with classes of alphabets: $\mathcal{C} = (\{2^{\Sigma}\}^*, cstrstr)$. A pattern $p = p_1 \cdots p_m$ is essentially a sequence of subsets of the alphabet Σ. $cstrstr(t, p) = 1$ if there exists a substring $t_i \ldots t_{i+m-1}$ of t such that $t_{i+k-1} \in p_k$ for $1 \leq k \leq m$.

Patterns in the above pattern classes consist only of strings of a *pattern alphabet*, and can be enumerated simply by first starting from the empty string as a *seed* pattern, and then concatenating a new character of the pattern alphabet, for all members of the alphabet. The new patterns are then considered as new seed patterns, and the process is repeated (of course we should be careful not to create useless patterns such as '$\star\star\star$').

Adding a new character c to a pattern p results in the decrease in the number of strings for which the pattern matches, and it is easy to see that $p \preceq_{\mathcal{C}} p'$ for any pattern p' which is obtained by elongating a seed pattern p.

approximate pattern: $\mathcal{C} = (\Sigma^* \times \mathbf{N}, astrstr)$. For $p = \langle s, n \rangle$, $astrstr(t, p) = 1$ if t contains a substring such that the edit distance (with insertion, deletion, substitution operations) from s is within n.

episode pattern: $\mathcal{C} = (\Sigma^* \times \mathbf{N}, esubseq)$. For $p = \langle s, n \rangle$, $esubseq(t, p) = 1$ if there exists a substring t' of t where $|t'| \leq n$, for which $subseq(t', p) = 1$. Notice that $\langle s, \infty \rangle$ is the same as the subsequence pattern s. Moreover, $\langle s, |s| \rangle$ is the same as the substring pattern.

VLDC pattern within a window: $\mathcal{C} = (\{\Sigma \cup \{\star\}\}^* \times \mathbf{N}, evldc)$. For $p = \langle s, n \rangle$, $evldc(t, p) = 1$ if there exists a substitution for the $\star$'s in s with Σ^* that yields t, and for the substitution, the length between the corresponding positions, in t, for the first and last non-star characters of s, is less than n. Notice that $\langle s, \infty \rangle$ is the same as the VLDC pattern s.

Patterns in the above pattern classes have two elements. The enumeration for the first element can be done in the same way as the patterns described previously. Given the first element s, we can calculate the best second element in the following way. For each text t_i, we are able to calculate the minimum value k_{t_i} for which the pattern $\langle s, k_{t_i} \rangle$ matches the text for a given first element of the above classes [8, 14]. If we sort the data set according to the k_{t_i}'s, we can calculate in one pass, all possible scores, just by calculating the scores for each threshold, therefore obtaining the integer value giving the best score [8, 14]. For a pattern $p = \langle s, n \rangle$ and any pattern $p' = \langle s', n' \rangle$ such that s' is a descendant of s and $n \geq n'$, we have $p \preceq_{\mathcal{C}} p'$. We can use $\langle s, n_{\max} \rangle$ to calculate the upperbound for p' where $n_{\max}$ can be ∞, or a predefined parameter to limit the space of possible patterns.

4 Computational Experiments

The algorithm was implemented in the Objective Caml language [16]. All experiments were conducted on a dual Xeon 2.2 GHz desktop computer with 1GB of main memory running Debian Linux. We have currently implemented all of the pattern classes mentioned in the previous section, except for patterns with classes of alphabets. The enumeration of patterns was done in a guided depth first search, where all characters of the alphabet are added to the seed, and the pattern which gives the best upperbound is then used as the next seed.

4.1 Patterns from Intron Sequences

Introns are non-coding regions of a eucaryotic gene which are transcribed to RNA, but are subsequently *spliced* out, to form the mature mRNA. There have been observations that a significant fraction of mutations in human genes that cause diseases affect the pre-mRNA splicing [4, 9], and elucidating the mechanism of pre-mRNA splicing is an important problem. Although the mechanism of splicing has been studied extensively, and canonical splicing signals are known (e.g.: the GU-AG rule and branch sites with an adjacent polypyrimidine tract (PPT)), computational prediction of the splice sites for a given pre-mRNA sequence is still difficult due to many *pseudo-signals* which are detected. Splice site predictions for short introns seem to be fairly accurate [11], but a deeper understanding of long introns - which can even be longer than 100,000 bases - seems to be necessary.

In an attempt to find differences between short and long introns, we have recently observed that there is a preference for U over C, and A over G in longer introns of human and mouse, especially near the acceptor site of the intron [2]. In this paper, we run our algorithm, on the data set created as described below, to see if we could find string patterns which are characteristic of long (or short) introns. Intron sequences of several species were taken from the Ensembl database [6] (human release 4.28, mouse release 4.1, fly release 4.3, and zebrafish release 4.06.). The intron sequences of genes with only one known transcript, obey the GU-AG rule, whose lengths are greater than or equal to 20, and do not contain characters other than 'A', 'C', 'G', or 'U' (e.g. 'N'), were collected. This resulted in a set of 1063, 5929, 44833, 114669 introns for zebrafish, fly, mouse, and human, respectively. For each intron, we use the pair of: 20 bases of the intron adjacent to its acceptor site (including the AG), and the logarithm of the length of the entire intron.

We use the "VLDC pattern within a window" (defined in the previous section) as the pattern class, and a guided depth first enumeration of patterns. Figure 2 shows the histograms, for human and mouse, of log intron lengths for the sets of introns which the best VLDC pattern discovered by our algorithm matches or does not match the intron acceptor site sequence. The discovered patterns $\langle$⋆U⋆U⋆U⋆U⋆U⋆U⋆A⋆, 18$\rangle$, and $\langle$⋆U⋆U⋆U⋆U⋆U⋆U⋆, 15$\rangle$ which match the longer introns, interestingly consists of many U's, consistent with our previous observation concerning base content. We can see that although the distributions of the log intron length for the matching and non-matching set is similar for shorter introns (e.g. ∼256 for human), there is a clear difference for the longer introns. The pattern discovered for zebrafish was $\langle$⋆U⋆U⋆UU⋆UUGCAG⋆,15$\rangle$. This pattern shows that U is

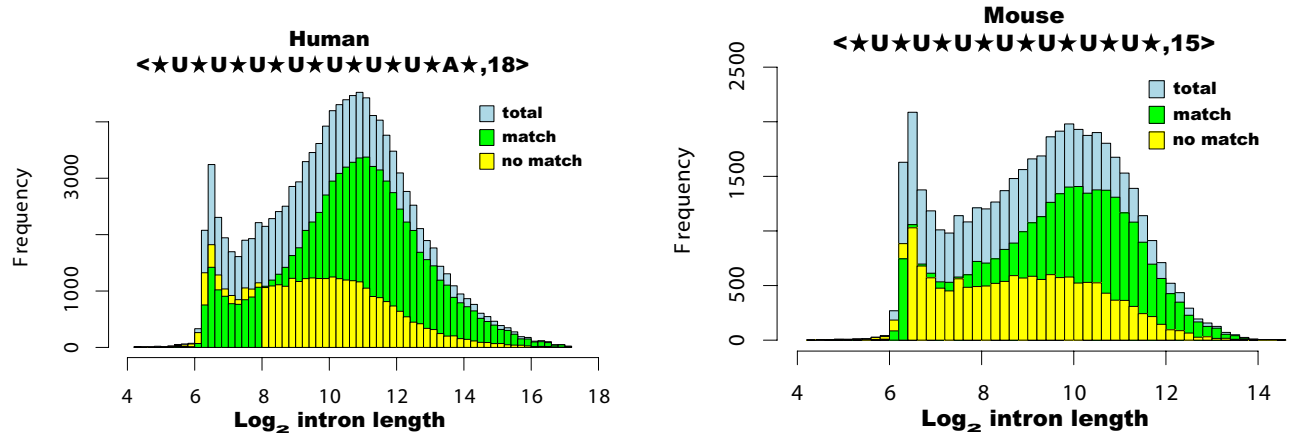

Figure 2: Splitting the log intron lengths with the best VLDC pattern within a window. The histograms of long intron lengths of 1) all introns, 2) introns which the pattern matches, and 3) introns which the pattern does not match, are overlayed. (a) The human introns are split with the pattern $\langle\star$U$\star$U$\star$U$\star$U$\star$U$\star$A$\star$, 18$\rangle$. (b) The mouse introns are split with the pattern $\langle\star$U$\star$U$\star$U$\star$U$\star$U$\star$U$\star$, 15$\rangle$.

an important element in longer introns for zebrafish as well, which was not evident when just looking at the base frequencies. The pattern discovered for fly was $\langle\star$A$\star$A$\star$A$\star$U$\star$,18$\rangle$, which is more abundant in shorter introns this time, is yet to be interpreted. When using substring patterns, the patterns discovered were UUU for human and mouse, UUGCAG for zebrafish, and AA (matching the shorter introns) for fly.

The discovered patterns are interesting since there has been a report that the strength of a PPT (a region near the acceptor site of an intron, abundant in pyrimidines (C or U)), depends greatly on the amount of U's (the more the stronger) [5]. Assuming this is true for most PPT's, this implies that acceptor sites of longer introns tend to be stronger. This may imply that, it is harder for long introns to be recognized after all, and they require a stronger acceptor site to be spliced correctly.

4.2 Performance

The execution time was about 47 hours and 42 hours respectively, using "VLDC pattern within a window" for human and mouse. This may not seem very fast, but we should consider the fact that the calculation with this pattern class would not have been feasible without the pruning of the search space.

Table 1 shows the comparison of execution times for the zebrafish introns, with and without the branch-and-bound search, while varying the maximum length of the pattern searched for (the length of a VLDC pattern is defined as the number of non-wild card characters). The run time for the exhaustive search is exponential as expected while the run time for the branch-and-bound algorithm seems to converge, even showing a slight decrease when not specifying a maximum length. This is probably because a longer but higher scoring pattern was found at an earlier stage of the search, which resulted in the pruning of a larger portion of the search space.

5 Discussion

We defined a new approach to string pattern discovery and presented an efficient branch-and-bound algorithm for solving the problem exactly. We conducted computational experiments on intron sequences, and showed the practicality of our approach and methods.

Table 1: Execution times (in seconds) for branch-and-bound and exhaustive algorithms on zebrafish introns

max len	exhaustive	b&b	best pattern	ICV
1	0.02	0.02	⟨C⋆,1⟩	8.35
2	0.12	0.13	⟨⋆C⋆G,8⟩	55.96
3	0.69	0.76	⟨⋆G⋆C⋆G,9⟩	77.80
4	3.57	3.83	⟨⋆G⋆C⋆AG⋆,9⟩	86.55
5	16.10	16.52	⟨⋆UUG⋆C⋆G,7⟩	111.77
6	64.52	61.09	⟨⋆UUG⋆C⋆G,7⟩	111.77
7	237.56	183.45	⟨⋆UUG⋆C⋆G,7⟩	111.77
8	896.47	418.11	⟨⋆UUG⋆C⋆G,7⟩	111.77
9	3136.85	761.35	⟨⋆UUG⋆C⋆G,7⟩	111.77
10	12230.58	1086.91	⟨⋆U⋆U⋆UU⋆UUGCAG⋆,15⟩	115.09
11	43925.63	1345.59	⟨⋆U⋆U⋆UU⋆UUGCAG⋆,15⟩	115.09
12	N/A	1454.61	⟨⋆U⋆U⋆UU⋆UUGCAG⋆,15⟩	115.09
13	N/A	1483.77	⟨⋆U⋆U⋆UU⋆UUGCAG⋆,15⟩	115.09
14	N/A	1617.47	⟨⋆U⋆U⋆UU⋆UUGCAG⋆,15⟩	115.09
15	N/A	1685.52	⟨⋆U⋆U⋆UU⋆UUGCAG⋆,15⟩	115.09
∞	N/A	1511.27	⟨⋆U⋆U⋆UU⋆UUGCAG⋆,15⟩	115.09

Our problem definition can be easily extended for finding patterns from strings associated with n numerical attributes. However, subtleties arise in the process of calculating the upper bound of a given pattern, where finding the maximum and minimum sizes of the sum vector (i.e. $|\vec{y}|^2$ in $|\vec{y}|^2(1/x + 1/(|D| - x)))$ does not seem to be straightforward. Still, the problem can be efficiently solved for the substring pattern class, again in linear time, since pruning of the search space is not necessary in this case. The score function used in this paper is the interclass variance (1), but any other function f which satisfies Lemma 1 could be used in place. Our algorithm can also be easily modified to find the k highest scoring patterns.

Another promising application of our approach is for microarray gene expression data, to discover putative regulatory elements contained in the upstream regions of the genes. Preliminary experiments have been conducted with the publicly available cell cycle data of [15], using the pair of upstream sequence of the gene, and the expression level for that gene, as data. Our algorithm was able to capture several known motifs, for example, the well known MCB element ACGCGT, related to cell cycle.

Acknowledgements

This research was supported in part by Grant-in-Aid for Encouragement of Young Scientists, and Grant-in-Aid for Scientific Research on Priority Areas "Genome Information Science" from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] Bailey, T. L. and Elkan, C., Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proc. Second International Conference on Intelligent Systems for Molecular Biology*, 28–36, AAAI Press, 1994.

[2] Bannai, H., Tamada, Y., Ott, S., Nakai, K., and Miyano, S., Long introns tend to have stronger splice acceptor sites. submitted

[3] Brazma, A., Jonassen, I., Eidhammer, I., and Gilbert, D., Approaches to automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):279–305, 1998.

[4] Cartegni, L., Chew, S. L., and Krainer, A. R., Listening to silence and understanding nonsense: exonic mutations that affect splicing, *Nat. Rev. Genet.*, 3(4):285–298, 2002.

[5] Coolidge, C. J., Seely, R. J., and Patton, J. G., Functional analysis of the polypyrimidine tract in pre-mRNA splicing, *Nucleic Acids Res.*, 25(4):888–896, 1997.

[6] Hubbard, T., Barker, D., Birney, E., Cameron, G., Chen, Y., Clark, L., Cox, T., Cuff, J., Curwen, V., Down, T., Durbin, R., Eyras, E., Gilbert, J., Hammond, M., Huminiecki, L., Kasprzyk, A., Lehvaslaiho, H., Lijnzaad, P., Melsopp, C., Mongin, E., Pettett, R., Pocock, M., Potter, S., Rust, A., Schmidt, E., Searle, S., Slater, G., Smith, J., Spooner, W., Stabenau, A., Stalker, J., Stupka, E., Ureta-Vidal, A., Vastrik, I., and Clamp, M., The Ensembl genome database project, *Nucleic Acids Res.*, 30(1):38–41, 2002.

[7] Hui, L., Color set size problem with applications to string matching, *Proc. Third Anual Symposium on Combinatorial Pattern Matching*, *LNCS* 644:230–243, 1992.

[8] Inenaga, S., Bannai, H., Shinohara, A., Takeda, M., and Arikawa, S., Discovering best variable-length-don't-care patterns, *Proc. 5th International Conference on Discovery Science*, 2002, in press.

[9] Krawczak, M., Reiss, J., and Cooper, D. N., The mutational spectrum of single base-pair substitutions in mRNA splice junctions of human genes: causes and consequences, *Hum. Genet.*, 90(1–2):41–54, 1992.

[10] Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., and Wootton, J. C., Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment, *Science*, 262(5131):208–214, 1993.

[11] Lim, L. P. and Burge, C. B., A computational analysis of sequence features involved in recognition of short introns, *Proc. Natl. Acad. Sci.*, 98(20):11193–11198, 2001.

[12] Morimoto, Y., Ishii, H., and Morishita, S., Efficient construction of regression trees with range and region splitting, *Machine Learning*, 45:235–259, 2001.

[13] Shimozono, S., Shinohara, A., Shinohara, T., Miyano, S., Kuhara, S., and Arikawa, S., Knowledge acquisition from amino acid sequences by machine learning system BONSAI, *Transactions of Information Processing Society of Japan*, 35(10):2009–2018, 1994.

[14] Shinohara, A., Takeda, M., Arikawa, S., Hirao, M., Hoshino, H., and Inenaga, S., Finding best patterns practically, *Progress in Discovery Science*, *LNAI* 2281:307–317, 2002.

[15] Spellman, P. T., Sherlock, G., Zhang, M. Q., Iyer, V. R., Anders, K., Eisen, M. B., Brown, P. O., Botstein, D., and Futcher, B., Comprehensive identification of cell cycle-regulated genes of the yeast *saccharomyces cerevisiae* by microarray hybridization, *Mol. Biol. Cell*, 9:3273–3297, 1998.

[16] Objective Caml - `http://caml.inria.fr/ocaml/`.