Pattern-Based Data Compression

Ángel Kuri[1] and José Galaviz[2]

[1] Department of Computing, ITAM.
akuri@itam.mx
[2] Department of Mathematics, Facultad de Ciencias, UNAM.
jgc@fciencias.unam.mx

Abstract. Most modern lossless data compression techniques used today, are based in dictionaries. If some string of data being compressed matches a portion previously seen, then such string is included in the dictionary and its reference is included every time it appears. A possible generalization of this scheme is to consider not only strings made of consecutive symbols, but more general patterns with gaps between its symbols. The main problems with this approach are the complexity of pattern discovery algorithms and the complexity for the selection of a good subset of patterns. In this paper we address the last of these problems. We demonstrate that such problem is NP-complete and we provide some preliminary results about heuristics that points to its solution.

Categories and Subject Descriptors: E.4 [**Coding and Information Theory**]–*data compaction and compression*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Problems; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search–*heuristic methods*.

General Terms: Algorithms, Theory

Additional Keywords and Phrases: Genetic algorithms, optimization, NP-hardness

1 Introduction

Since the introduction of information theory in Shannon's seminal paper [7] the efficient representation of data is one of its fundamental subjects. Most of the successful modern lossless methods used today are dictionary-based such as the Lempel-Ziv family [10]. These dictionary-based methods only consider strings of consecutive symbols. In this paper we will introduce a generalization by considering "strings" with gaps, whose symbols are no necessarily consecutive. This generalization will be called *pattern* in the rest of paper. A similar concept is used in [1] for approximate string matching.

In this context a pattern is a finite and ordered sequence of symbols together with an specification of the position of each symbol. For example a pattern contained in the string `It is better late than never` could be:

$$p_1 = \texttt{I\_\_\_\_\_be\_\_\_\_\_\_\_t\_\_\_\_a}.$$

R. Monroy et al. (Eds.): MICAI 2004, LNAI 2972, pp. 1–10, 2004.

We consider two possible representations of patterns:

- An ordered pair (S, O) where S is the ordered sequence of symbols in the pattern and O is the ordered sequence of absolute positions (offsets) of symbols.
- A 3-tuple (S, D, b) where S is the ordered sequence of symbols in the pattern, b is the absolute position of the first symbol in S and D is the ordered sequence of distances between symbols (positions relative to the previous symbol).

Therefore, the pattern used as an example above could be represented as: $S = \{I, b, e, t, a\}$, $O = \{0, 6, 7, 15, 20\}$ or using: $p = 0$, $D = \{6, 0, 8, 5\}$. In general the second representation is more efficient than the first one, since absolute positions are potentially larger than the relative positions.

By identifying frequent patterns we can proceed to include such patterns in a dictionary achieving data compression.

2 Pattern-Based Data Compression

In order to apply the procedure aforementioned we need to accomplish several goals:

1. Given a sample of data S, identify frequent patterns.
2. Given the set of patterns obtained in the previous step, determine the subset of such patterns that maximizes the compression ratio.
3. Determine the best way for pattern representation and encoding, and the best way for reference encoding. The compressed sample will include the, perhaps compressed, dictionary and the compressed representation of original sample.

This paper will be focused in the second step, but we will do some annotations regarding the first.

In the first step it is needed to find patterns in the sample. Such patterns must be used frequently, that is, they must appear several times in ths sample. Evidently the most frequent patterns will be individual symbols. Therefore, we need to establish another requisite: the patterns must be as large as possible. But there is a potential conflict between the pattern size (number of symbols that contains) and its frequency. Larger patterns are rare, short patterns are common. In order to avoid this conflict, we will consider the total number of symbols that are contained in all the appearances of a given pattern p. This number will be called the *coverage* of such pattern and can be calculated by the product of the pattern frequency and the pattern size. In notation:

$$\mathrm{Cov}(p) = f(p)\, t(p) \tag{1}$$

where $f(p)$ denotes the frequency of pattern p and $t(p)$ the number of symbols in p. In our example of previous section the pattern size is $t(p_1) = 5$.

It is convenient to distinguish between the nominal coverage of some pattern: the product mentioned before, and the effective coverage of some pattern in a set of patterns: the number of symbols contained in all the appearances of the pattern and *not* contained in any previously appeared pattern.

The task for this first step must be solved by algorithms of *pattern discovery* algorithms, similar to those used in the analysis of DNA sequences and biomolecular data in general. Unfortunately the reported algorithms for the discovery of patterns of the kind we are interested in, have exponential time complexity [9].

3 Selecting Good Subset of Patterns

The second step of the aforementioned process is the selection of a good subset of the whole set of frequent patterns found in the sample. The "goodness" of a subset of patterns is given by the compression ratio obtained if such subset of patterns is the dictionary.

Obviously the best subset also must cover a considerably large amount of symbols contained in the sample, since every pattern in the subset must have a good coverage. Also it must have a low amount of data symbols multiply covered. That is, the best subset must have efficient patterns: a large amount of covered symbols but a small amount of symbols covered by another patterns or by different appearances of pattern itself.

Let S be a sample of data. We will denote by $|S|$ the number of symbols in such sample (its original size). We will denote with $T(Q)$ the size of the compressed sample using the subset of patterns Q. With P we will denote the whole set of patterns found in S, therefore $Q \subseteq P$. Using this notation we will define the compression ratio.

Definition 1. The *compression ratio* obtained by using the subset of patterns Q is:

$$G(Q) = 1 - \frac{T(Q)}{|S|} \tag{2}$$

$T(Q)$ has two components: the dictionary size $D(Q)$ and the representation of the sample itself $E(Q)$. These are given by:

$$D(Q) = z \left[\sum_{p_i \in Q} t(p_i) + r(Q) \right] \tag{3}$$

$$E(Q) = 1 + \sum_{p_i \in Q} f(p_i) \tag{4}$$

Hence:

$$T(Q) = D(Q) + E(Q) \tag{5}$$

In the expression for D, $z > 1$: since for every pattern included in the dictionary, every symbol in the pattern must also appear. The distances between them

must be included and the offset of first symbol in the pattern must be included as well. We will assume that every distance or offset uses z times the space (bits for example) used for each symbol. In what follows z will be roughly estimated as $z = 2$, assuming that, for every symbol in a given pattern we need to store the symbol itself and its distance to the previous symbol in the same pattern, and both requires the same amount of data.

In expression 3, $r(Q)$ is the number of symbols not covered by patterns in Q. Such symbols must appear as a pattern in the dictionary. Once a pattern subset Q is chosen, $r(Q)$ is determined. Hence, we can equivalently think that r is included in the sum.

In the expression for E the 1 that is added to the sum at the right represents the inclusion of the "pattern" of those symbols in S not contained in any pattern of Q in the dictionary. This is the pattern with the $r(Q)$ symbols just mentioned. $E(Q)$ is the number of pattern identifiers used to represent the sample in terms of references. For every appearance of a pattern in Q, a reference must be included in the compressed sample expression. As stated a reference to the "pattern" of symbols not covered by Q must also be included in such expression.

Denoting by p_i the i-th pattern contained in a set of patterns, the coverage of a subset of patterns Q is:

$$\mathrm{Cov}(Q) = \sum_{p_i \in Q} f(p_i)\, t(p_i) \tag{6}$$

where $f(p_i)$ and $t(p_i)$ are the frequency and size of pattern p_i respectively.

We can now state our problem as follows:

Definition 2. OptimalPatternSubsetProblem.
Given:

- A data sample S with $|S|$ symbols.
- A set $P = \{p_1, \ldots, p_n\}$ of frequent patterns of S with frequencies $f(p_i) = f_i$ and sizes $t(p_i) = t_i$

Find a subset $Q \subset P$ that maximizes $G(Q)$ subject to the restriction:

$$\mathrm{Cov}(Q) \leq |S| \tag{7}$$

Hence we must find a subset of P. But there are $2^{|P|}$ of such subsets. This is a huge search space even for small values of $|P|$. In fact this is a NP-complete problem as we now prove. Similar problems have been proved NP-complete in [8,4]. However in [8] the dictionary size is not considered, and the patterns are strings of consecutive symbols. In [4] also the dictionary size is ignored and coverage of patterns is used as the only criteria to determine the best subset.

Theorem 1. OptimalPatternSubsetProblem *is NP-complete.*

Proof. First we must prove that OPSP is in NP, which means that this is verifiable in polynomial time. Given some subset $Q \subseteq P$ and the maximum compression

ratio g, then we can calculate $T(Q)$ in linear time on the size of Q, therefore we can calculate $G(Q)$ also in $O(|Q|)$. That is polynomial time verifiable.

Next, in order to do the reduction we chose the (0,1) Knapsack Problem. We must prove that any given instance of such problem can be mapped, by a polynomial time algorithm, in an instance of OPSP. The solution to knapsack instance is therefore mapped in polynomial time to the solution of some OPSP instance.

In an instance of (0,1) knapsack problem there are given:

- A set of objects $\{o_1, o_2, \ldots, o_m\}$.
- A function v that assigns to every object o_i its value: $v(o_i) > 0$.
- A function w that assigns to every object o_i its weight: $w(o_i) > 0$.
- A positive integer $C > 0$ called the *capacity*.

The problem consists in finding some subset $B \subseteq O$ such that it maximizes:

$$\sum_{o_i \in B} v(o_i)$$

with the restriction:

$$\sum_{o_i \in B} w(o_i) \leq C$$

Let $B \subseteq O$. The algorithm proceeds as follows.

For every object $o_i \in B$ compute:

$$w'(o_i) = \begin{cases} \frac{(v(o_i)-C)^2}{8\,w(o_i)}, \text{ if } w(o_i) \geq \frac{(v(o_i)-C)^2}{8} \\ w(o_i), \qquad \text{otherwise} \end{cases}$$

This means that:

$$8\,w'(o_i) < (v(o_i) - C)^2 \tag{8}$$

Now we establish the values of for sample size, the frequencies, and the sizes on the corresponding instance of OPSP:

$$C_B = \frac{C}{|B|} \tag{9}$$

$$f(o_i) = \frac{(C_B - v(o_i)) + \sqrt{(v(o_i) - C_B)^2 - 8\,w'(o_i)}}{2} \tag{10}$$

$$t(o_i) = \frac{w'(o_i)}{f(o_i)} \tag{11}$$

where (8) guarantees that the square root of (10) is real solution of:

$$f^2(o_i) + f(o_i)\,(v(o_i) - C_B) + 2\,w'(o_i) = 0$$

From this we obtain:

$$v(o_i) = \frac{C_B\,f(o_i) - 2\,w(o_i) - f^2(o_i)}{f(o_i)}$$

Finally, using (9) and (11):

$$v(o_i) = \frac{C}{|B|} - (2\,t(o_i) + f(o_i)) \tag{12}$$

Also from (11) we have:

$$w'(o_i) = f(o_i)\,t(o_i) \tag{13}$$

Therefore, in the knapsack we want to maximize:

$$\sum_{o_i \in B} v(o_i) = \sum_{o_i \in B} \left[\frac{C}{|B|} - (2t(o_i) + f(o_i)) \right] \tag{14}$$

$$= C - \sum_{o_i \in B} (2t(o_i) + f(o_i)) \tag{15}$$

If we establish $|S| = C$ for OPSP, then maximizing the last expression also maximizes:

$$\frac{\sum_{o_i \in B} v(o_i)}{|S|} = 1 - \frac{\sum_{o_i \in B} (2t(o_i) + f(o_i))}{|S|}$$

which is (2) considering (3) and (4) (excluding the terms related with the "pattern" of symbols not included in any other pattern).

The restriction is transformed as follows:

$$\sum_{o_i \in B} w'(o_i) = \sum_{o_i \in B} f(o_i)\,t(o_i) \leq \sum_{o_i \in B} w(o_i) \leq C = |S|$$

In terms of OPSP the restriction means that the joint coverage of patterns in B cannot be greater than the total size of the original sample.

Therefore the solution to knapsack, using the transformations above, can be mapped into the solution of OPSP in polynomial time.

□

Since OPSP is NP-complete we need heuristics in order to obtain an approximate solution for large samples. In what follows we will address such heuristics.

4 Heuristics for Subset Selection

4.1 Using a Genetic Algorithm

Our first heuristic approach was the use of a genetic algorithm (GA) for the selection of a good pattern subset. The use of GA to solve NP-complete problems has been used in the past [2,3].

The first step is to define a useful domain representation. Since we need to find a subset of a set of m different patterns. We can encode every subset using a binary string of length m. The i-th pattern is included in a subset if the i-th bit of its encoding string is "1", otherwise the pattern is excluded. The number

of possible binary strings of length m is 2^m, the cardinality of our search space: the power set.

The fitness function is given by (2), and the selection will be performed using a deterministic scheme called *Vasconcelos Selection* [5][1]. Such selection scheme has improved the performance of GA when combined with 2-point crossover, as is statistically proved in [6].

4.2 Using a Coverage-Based Heuristic

We propose an alternative method for the search of a good subset of patterns. The method consists in the selection of patterns using the number of symbols in the sample that are covered by them. A pattern is better the greater the number of symbols in the sample that are in the instances of such pattern. This is our concept of *coverage*. It may occur that some symbol is in the appearances of several different patterns. That is, the symbol is "overcovered". Hence we need to measure the coverage more accurately than with the product of frequency and size. Therefore, during the selection process, if some pattern covers symbols already covered by a pattern previously selected, the coverage of the last pattern should be modified to represent its *effective coverage*: the number of symbols that are covered by the pattern and not already covered by some other pattern.

Given a set of patterns P we will obtain a subset $B \subseteq P$ of selected patterns. The heuristic algorithm for the selection of subset is:

1. Set $B = \emptyset$
2. Set the current coverage $Cv = \emptyset$.
3. Select the pattern $p \in P$ with highest effective coverage (covered symbols not already in Cv).
4. Add p to B.
5. Remove p from P
6. Use the coverage of p to update the current coverage Cv.
7. Return to 3 until $|Cv|$ equals the sample size or $P = \emptyset$.

With the strategy described we will obtain the subset of patterns B with highest coverage. But good coverage does not guarantee best compression ratio, it may occur that patterns with large size and poor frequency or conversely are included in B. Since we want that the inclusion of some pattern in the dictionary will be well amortized by its use in the sample, this is not desirable.

4.3 Hillclimbing

For the improvement of the heuristic described above, we will use hillclimbing. Two different hillclimbers will be defined:

[1] The best individual (I_0) is mixed with the worst one (I_{N-1}), the second best is crossed with the second worst (I_1 and I_{N-2}, respectively), and so on.

MSHC *Minimum Step HillClimber*. Searching the better binary string in the neighborhood of radius 1 in Hamming distance. Given a binary string that encodes a subset, we perform a search for the best string between those that differ from the given one in only one bit.

MRHC *Minimum Replacement HillClimber*. Searching the better binary string in the neighborhood of radius 2 in Hamming distance. Given a binary string, we perform a search for the best string between those that exchanges the position of every bit with value 1 with the position with value 0.

To find a better subset than the one given by the coverage-based heuristic we run both hillclimbers: the string obtained by the heuristic is passed to MSHC, and the output of this climber is thereafter passed to MRHC. Every hillclimber is executed interatively until no further improvement is obtained.

5 Test Cases

In order to test the effectiveness of the heuristic methods above we define three simple test cases. These are only for testing, but a more robust statistical demonstration of effectiveness will be developed in the future.

The test cases are samples built with patterns defined *a priori*. Hence the expected resulting subset is the set of building patterns. The algorithm used for pattern discovery yields the whole set of patterns which is large in comparison with the subset we are looking for. The output of such algorithm is the input for the subset selection algorithm.

The characteristics of each sample are shown in table 1. The column labeled **Patterns** show the number of building patterns used for the sample. The column labeled **ComRa** contains the compression ratio obtained if uilding patterns are the whole dictionary. The column **Patt. found** contains the number of patterns found by the discovery algorithm, therefore the search space depends exponentially on the contents of this column.

For the genetic algorithm we use the parameter values shown in table 2. The GA was ran for 100 generations. A repair algorithm was used in order to restrict the maximum number of ones allowed in the chromosome (number of patterns in the proposed subset) and at the end of the GA execution the hillclimbers where executed.

The results are summarized in table 3. In the table the column labeled **ComRa** is the compression ratio as defined by expression (2) before the hillclimbers. The column labeled **HC ComRa** is the compression ratio obtained after both hillclimbers.

It is clear from table 3 that GA outperforms the results obtained by the application of cover based heuristics. But the solution proposed by this heuristic however can be improved more efficiently than the one obtained from the GA, since after the application of hillclimbers the best subset is the one obtained by the cover-based heuristic. In all cases the solution proposed by the cover-based+hillclimbers contains the patterns used to build the sample. The patterns obtained by GA contains al least 60% of such patterns.

Table 1. Characteristics of samples used.

Sample	Size	Alphabet	Patterns	ComRa	Patt. found
1	64	8	5	-0.1406	75
2	133	13	4	0.3233	309
3	185	37	7	0.3189	25

Table 2. Parameters used for GA.

Parameter	Value
Population size	100
Selection scheme	Vasconcelos
Crossover	2-point
Crossover probability	0.9
Mutation probability	0.05

Table 3. Summary of results for test cases.

	Genetic Algorithm				C-B Heuristic		
Sample	Gen	ComRa	HC ComRa	Patt	ComRa	HC ComRa	Patt
1	75	-0.2187	-0.2187	5	-0.3280	-0.1875	5
2	100	0.0225	0.3230	5	0.3082	0.3310	5
3	31	0.2756	0.2756	7	-0.0972	0.3189	7

6 Summary and Further Work

We have defined a problem whose solution is useful for a dictionary-based data compression technique: the OptimalPatternSubsetProblem. We have proved that such problem is NP-complete. Two different heuristic techniques have been proposed for its approximate solution: a genetic algorithm and a cover-based heuristic technique.

In order to refine the solutions proposed by these heuristics two different methods of hillclimbing where introduced: MSHC and MRHC. These hillclimbers are executed iteratively over the solutions proposed by the heuristics described until no better proposal is found.

Our very preliminary results show that, the best heuristic is the cover-based one. This is to be expected since the GA is not modified with special purpose operators. However a repair algorithm was included in every generation of GA in order to reduce the search space restricting the number of bits with value 1 in the chromosomes.

The cover-based heuristic technique does not outperform the GA by itself, at least not necessarily. The efficiency in the solution proposed by this heuristic was achieved through the use of hillclimbers. Also the cover-based heuristic is several times faster than the GA, since there is not an evolutionary process.

Further experimentation using the cover-based technique is required in order to provide statistically robust performance measurements, the results shown are not conclusive. However in order to provide a robust evaluation we need to

solve the first phase og general compression procedure. Such evaluation can also compare the methods used with some other heuristics: tabu search, simulated annealing and memetic algorithms could be used.

Once the selection of best subset is done, the third phase must be performed. The goal is to found a set of patterns which conform a meta-symbol dictionary. A similar approach is shown in [4]. Therefore, well known encoding techniques can be used on this meta-alphabet, such as those based in information theory. Then we are able to imagine that the sample we have has been produced by an information source whose alphabet is the set of meta-symbols rather than the original one.

References

[1] Burkhardt, Stefan and Juha Kärkkäinen, "Better Filtering with Gapped q-Grams", *Proceedings of 12th Annual Symposium on Combinatorial Pattern Matching CPM 2001*, Amihood Amir and Gad M. Landau (editors), Lecture Notes in Computer Science, No. 2089, 2001, pp. 73-85.

[2] DeJong, K. and W.M. Spears. "Using genetic algorithms to solve NP-complete problems". *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer (editor), 1989, pp. 124-132. `http://citeseer.nj.nec.com/dejong89using.html`.

[3] Jin-Kao Hao, J., F. Lardeux and F. Saubion, "Evolutionary Computing for the Satisfiability Problem", *Applications of Evolutionary Computing*, LNCS, No. 2611, 2003, pp. 259-268. `http://citeseer.nj.nec.com/hao03evolutionary.html`.

[4] Klein, Shmuel T., "Improving Static Compression Schemes by Alphabet Extension", *Proceedings of 11th Annual Symposium Combinatorial Pattern Matching CPM 2000*, Raffaele Giancarlo and David Sankoff (editors), Lecture Notes in Computer Science, No. 1848, 2000, pp. 210-221.

[5] Kuri, A., "A universal Eclectic Genetic Algorithm for Constrained Optimization". *Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT'98*, 1998, pp. 518-522.

[6] Kuri, A., "A Methodology for the Statistical Characterization of Genetic Algorithms", *Proceedings of MICAI 2002*, Lecture Notes in Artificial Intelligence, No. 2313, 2002, pp. 79-89.

[7] Shannon, Claude E., "A Mathematical Theory of Communication", *The Bell System Technical Journal*, vol. 27, July: pp. 379-423, October: pp. 623-656, 1948.

[8] Storer, James y Thomas Szymanski, "Data Compression via Textual Substitution", JACM, Vol. 29, No. 4, october 1982, pp. 928-951.

[9] Vilo, Jaak, *Pattern Discovery from Biosequences*, PhD Thesis, Technical Report A-2002-3, Department of Computer Science, University of Helsinki, 2002.

[10] Ziv, Jacob and Abraham Lempel, "A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, Vol. 23, No. 3, 1977, pp. 337-343.