BioSystems 110 (2012) 1–8

Contents lists available at SciVerse ScienceDirect

BioSystems

journal homepage: www.elsevier.com/locate/biosystems

On the synthesis of DNA error correcting codes

Daniel Ashlock[a,*,1], Sheridan K. Houghten[b], Joseph Alexander Brown[c], John Orth[d]

[a] Department of Mathematics and Statistics, University of Guelph, Guelph, Ontario, N1G 2W1, Canada
[b] Department of Computer Science, Brock University, St. Catharines, Ontario, L2S 3A1, Canada
[c] School of Computer Science, University of Guelph, Guelph, Ontario, N1G 2W1, Canada
[d] Department of Computer Science, Brock University, St. Catharines, Ontario, L2S 3A1, Canada

ARTICLE INFO

Article history:
Received 11 May 2011
Received in revised form 4 June 2012
Accepted 20 June 2012

Keywords:
Evolutionarycomputation
Bioinformatics
Representation
Error correcting code
Nucleic acids
Next-gen sequencing

ABSTRACT

DNA error correcting codes over the edit metric consist of embeddable markers for sequencing projects that are tolerant of sequencing errors. When a genetic library has multiple sources for its sequences, use of embedded markers permit tracking of sequence origin. This study compares different methods for synthesizing DNA error correcting codes. A new code-finding technique called the *salmon algorithm* is introduced and used to improve the size of best known codes in five difficult cases of the problem, including the most studied case: length six, distance three codes. An updated table of the best known code sizes with 36 improved values, resulting from three different algorithms, is presented. Mathematical background results for the problem from multiple sources are summarized. A discussion of practical details that arise in application, including biological design and decoding, is also given in this study.

1. Introduction

An *error correcting code* is a collection of strings over a given alphabet that are well separated from one another. The separation property means that small numbers of errors in transmission of a code word can be both detected (by noting that the word received is not a code word) and corrected (by assuming the code word transmitted is the one most similar to the one that was received). In this study we are creating a code where the transmission channel consists of incorporating the code word, in the form of an oligonucleotide, as a label in a genetic construct and later reading it out when the entire construct is sequenced. In this situation the *Hamming metric* Gusfield (1997), which only counts substitutions, is inappropriate because sequencers can skip a base or read one that is not there. This means that codes relative to the *edit metric* or *Levenshtein distance* Levenshtein (1966) are required for the detection and correction of sequencing errors.

The edit distance between two words is the smallest number of single character substitutions, insertions, or deletions that can transform one string into the other. The edit metric is, in fact, a metric in the mathematical sense Gusfield (1997). Edit metric codes are startlingly unlike codes over the Hamming metric Campbell (2005). Very little of the beautiful algebraic theory of Hamming metric codes applies to edit codes. While the edit metric permits code words with variable length, we will, for reasons of simplicity in both coding and biological applications, work with code words of a fixed length.

An (n, M, d)-*code* is a code with M members whose words are of length n with pairwise minimum distance between code words of d. In fact, in general, only a very small number of additional code words may be obtained by permitting word length to vary Baker et al. (2007). In essence, the longer words are spaced out from one another more and so shorter words obstruct a far larger share of the edit-metric space than longer words.

The first application of DNA-error correcting codes Qiu et al. (2003) used the code words as embedded tracking labels in an expressed sequence tag (EST) project in *zea mays* (corn). The biochemistry used in the preparation of the genetic constructs used in that project limited the codewords to a length of six. Since more than twenty different genetic libraries were pooled, this meant that a (6,*M*,3)-code was the logical choice, balancing number of code words available with maximum possible error correction within the resource constraint. Such a code corrects $(3-1)/2 = 1$ error, a number that was sufficient to recover 50% of the data in which sequencing errors had made the source of the sequenced EST otherwise unreadable. The advent of next-gen sequencing, with runs that obtain millions of sequence reads, forms a new application domain for application of error-correcting embedded sequence tags. This new application domain motivates this manuscript which pulls

* Corresponding author. Tel.: +1 519 824 4120x53453; fax: +1 519 837 0221.
E-mail addresses: dashlock@uoguelph.ca (D. Ashlock), houghten@brocku.ca (S.K. Houghten), jb03hf@gmail.com (J.A. Brown), jo05bi@brocku.ca (J. Orth).
[1] The authors thank the National Science and Engineering Research Council of Canada (NSERC) for supporting this work.

0303-2647/$ – see front matter © 2012 Elsevier Ireland Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.biosystems.2012.06.005

together diverse research on the creation of error correcting tags for genetic constructs. The practicality of including tags depends on the size of the tag while its utility depends on the number of errors the code can correct. The maximum size of a code with a specified length and error correction capacity is a critical planning variable if such tags are to be incorporated in a sequencing project. As before, error correcting tags will enhance the recovery of information about the source of given sequences.

The first studies on edit metric error correcting codes used a fitness function with exponential time complexity Ashlock et al. (2002). This algorithm solved an immediate biological problem but is computationally unsatisfying. This high time complexity was acceptable only because a relatively short code (of length 6 and able to correct a single error) was required. The algorithm used to find these codes operates by evolving small collections of code words called *code seeds* that are completed to a full code with a greedy algorithm. If we view the greedy algorithm as a type of optimization, then this algorithm is a Baldwinian one. The technical details of this initial algorithm for finding edit metric codes appears in Ashlock et al. (2002). The fitness of a code seed is the size of the code located by the greedy algorithm starting with the seed. This definition of fitness, resulting code size, is retained in the current study.

In general, when we fix the length of the code words and the number of errors that must be corrected, codes are better if they are bigger (contain more code words). To understand why bigger codes are better, consider the application of tagging biological constructs. A larger code provides more available labels at a given level of error correction and (word length based) difficulty of synthesis.

In Ashlock and Houghten (2009) the authors established that the crossover operator in the original (code seed representation) evolutionary algorithm was actually counterproductive and showed enhanced performance with a crossover-free evolution strategy. The lack of scalability of the code-seed representation led to the development of a novel binary variation operator, one that is not really a crossover operator. This new operator incorporates aspects of both crossover and mutation and exploits the fact that the order in which words are fed into the *lexicode algorithm* (Algorithm 1) has a substantial impact on the size of the resulting code. In a preliminary study Ashlock and Houghten (2005) a number of forms of this variation operator were explored. The one exhibiting best performance, a modified evolution strategy (ES), is used in this study. It applies the new variation operator to code lengths it had not been tested on previously. In Ashlock and Houghten (2010) the new variation operator was tested in a spatially structured structured evolutionary algorithm called a *ring optimizer* Ashlock and VonKonigslow (2008), Ashlock et al. (2008). The standard algorithm exhibited superior performance. In this study, the ES is run for longer times on a broad variety of code parameters and improves the known bound for thirteen different sets of code parameters.

The most recently developed algorithm for searching for DNA error correcting codes is the *Salmon* algorithm Orth and Houghten (2011), inspired by the spawning behavior of salmon. This algorithm is similar to ant colony optimization Dorigo and Stützle (2004), except that the algorithmic analog of pheromones does not evaporate. The algorithm is configured to search for cliques in a graph with vertices that are strings in the DNA alphabet and edges where pairs of words meet the minimum distance restriction for a code. In Orth and Houghten (2011) the algorithm's performance was optimized on a single set of code parameters. In this study we run the algorithm on several sets of code parameters. The algorithm is memory intensive and so most useful to perform a thorough search for codes with relatively short word length. Since these words are synthesized into biological constructs Qiu et al. (2003), codes with short word length are the most useful in applications. The algorithm improves the best known code size for six different code parameters.

The remainder of this study is structured as follows. Section 2 gives mathematical background that permits us to leverage the experimental information used to build the table of best code sizes. Section 3 surveys the techniques used to locate large codes over the DNA alphabet. Section 4 presents new best code sizes and a table of best-known code sizes. Section 5 gives application notes and discusses the results and future directions.

2. Mathematical background

Readers only interested in the application of DNA error correcting codes may safely skip this section. It is included because it is needed to justify many of the entries in Table 3 which gives the best known sizes of codes for various lengths and minimum distances.

2.1. Preliminaries

Recall that an $(n, M, d)_q$ edit code is a q-ary code consisting of M codewords each of which has length n, and in which all codewords are at edit distance at least d from each other. This study focuses primarily on DNA codes for which $q = 4$. Define $E_q(n, d)$ to be the maximum M for which there is an $(n, M, d)_q$ edit code. Such a code is *optimal* if $M = E_q(n, d)$.

2.2. Automorphism group and equivalence

The automorphism group of a q-ary edit code is described in Houghten et al. (2006). In that paper words are described in terms of their *block structure*. A *block* is defined as a maximal run of a single character within a word. It is shown that the automorphism group is dependent on the block structure of words, with the only allowed operations being to simultaneously permute all symbols of all words, or to simultaneously reverse all words. Two codes C_1 and C_2 are *equivalent* if one can be transformed into the other by any combination of these operations. That paper also establishes several relationships between different types of codes and between different parameter sets.

2.3. Bounds on sizes of codes

The results in this section fill in gaps in the table of best known code sizes by using knowledge of the size of a code for one parameter to estimate the size for another parameter.

2.3.1. Relationships to other types of codes

Definition 1. The insertion–deletion distance between two strings is the minimum number of insertions and deletions needed to transform one string into another. Insertion–deletion codes are codes that use the insertion–deletion distance.

Observation 1. *Since edit distance encompasses insertion–deletion distance, the number of codewords M in an* $(n, M, d)_q$ *edit code cannot exceed that in an optimal insertion–deletion correcting code of the same parameters.*

Observation 2. *Since edit distance encompasses Hamming distance, the number of codewords M in an* $(n, M, d)_q$ *edit code cannot exceed that in an optimal Hamming distance code of the same parameters.*

An implication of this observation is that one can use Hamming distance, which is faster to compute than edit distance, as a bounding function in the computation of edit distance. If two words are at Hamming distance $<d$ apart, then they are also at edit distance $<d$