Hybridization and Ring Optimization for Larger Sets of Embeddable Biomarkers

Sheridan Houghten, Member IEEE
joint work with Daniel Ashlock

1. Biomarkers or DNA error correcting codes.
2. Conway's algorithm.
3. Ring optimization.
4. Hybridization.
5. Results!

Error correcting codes in one slide.

- An error correcting code is a subset of a metric space. It has members called *codewords*. A **good** error correcting code has a large minimum distance between codewords - this is how you correct errors.
- Only codewords can be used to represent information. When errors occur due to noise, the received word is checked against all codewords, and is decoded to the closest codeword.
- Larger minimum distances permit the correction of more errors.
- There is a tension between number of codewords and minimum distance. Finding codes is often modeled as *sphere packing* in the relevant metric space. The exclusive "sphere" around a codeword w is the set of possible messages that decode to w.
- As long as we are working in the Hamming space of bit strings this is all elegant and beautiful.

What about DNA?

- DNA sequencers make errors. If we embed DNA labels in a genetic construct then we can think of it as "transmitting" that label.
- Here is where the excrement contacts the rotary air impeller: DNA sequencers don't just change bases, they can skip them or insert extra ones.

Definition: The **Levenshtein** or **edit** metric on the space of strings gives the distance between two strings as the *minimum* number of single character insertions, deletions, or substitutions that transform one string into the other.

A Killer Example: what is the distance between these strings?

CGATCGATCGATCGATCGAT
TCGATCGATCGATCGATCGA

What about DNA?

- DNA sequencers make errors. If we embed DNA labels in a genetic construct then we can think of it as "transmitting" that label.
- Here is where the excrement contacts the rotary air impeller: DNA sequencers don't just change bases, they can skip them or insert extra ones.

Definition: The **Levenshtein** or **edit** metric on the space of strings gives the distance between two strings as the *minimum* number of single character insertions, deletions, or substitutions that transform one string into the other.

A Killer Example: what is the distance between these strings?

CGATCGATCGATCGATCGAT
TCGATCGATCGATCGATCGA

Hamming distance=20, edit distance=2. Ouch!

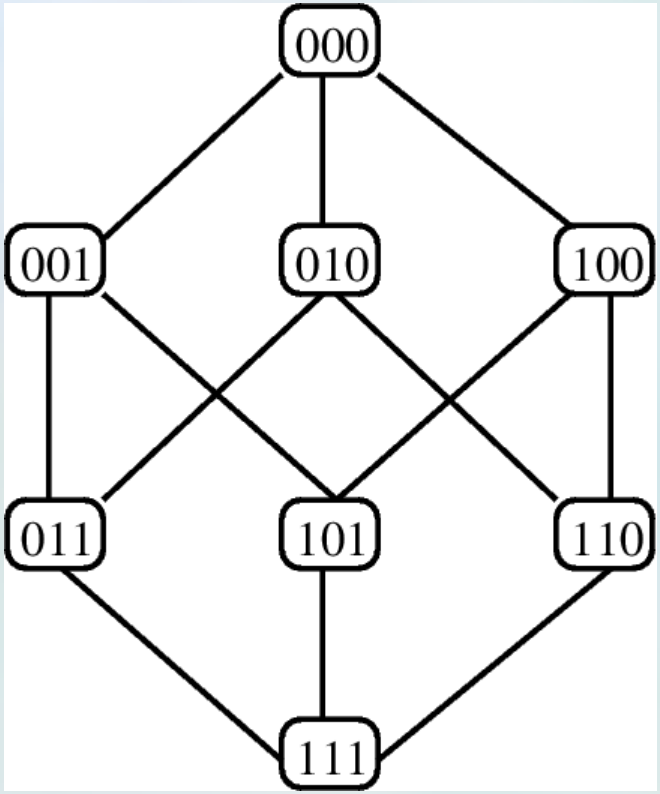
Edit metric error correcting codes

- To correct sequencing errors we need codes that measure minimum distances using edit distance.
- The number of words at Hamming distance $\leq d$ from a codeword w does not depend upon w; for the edit distance it does.
- The beautiful sphere packing theory of binary Hamming codes vanishes when we change to the edit metric.
- There are $2^n \cdot n!$ symmetries of the binary Hamming space of length n. The corresponding edit space has 4 symmetries for all n: do nothing, flip the bits, flip the words end-for-end, or do both flips.

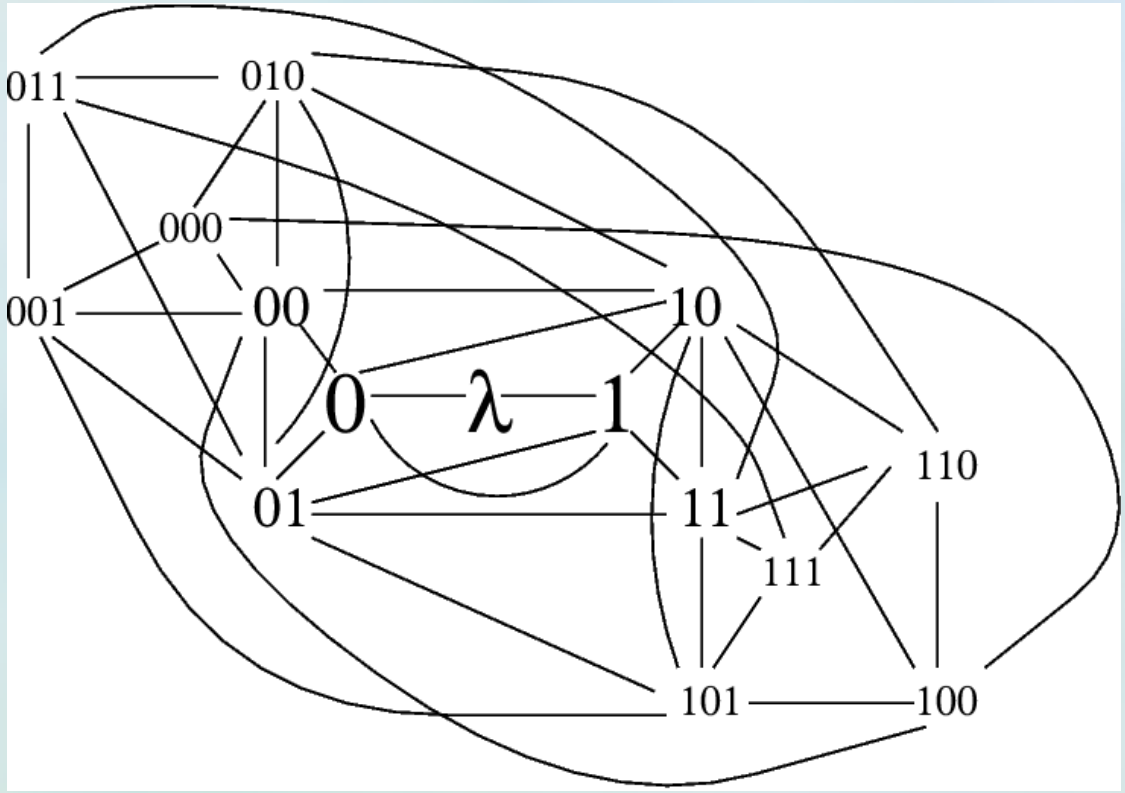
Reference

J. Campbell, **Enumeration and Symmetry of Edit Metric Spaces**, Ph.D. Thesis, Iowa State University, 2005.

Compare the 3-bits spaces



Hamming 3-Cube



Edit 3-Cube

Lines denote pairs that are distance one from one another in the Hamming and Edit spaces respectively. The edit space is far less symmetric and far more connected. The empty string is denoted by λ.

Conway's Lexicode Algorithm: a key tool.

The Lexicode Algorithm

Input: An alphabet $\mathcal{A}$, a minimum distance d and an ordered subset of $S \subset \mathcal{A}^n$.

Output: $CLA(S)$, a subset of S that has pairwise minimum distance d.

Details:

Initialize an empty set R of words.
Traverse the members $s \in S$ in order
    If s is at least distance d from every member of R,
        add s to R
Return R as $CLA(S)$.

Notice that this algorithm works for *any* metric space and returns a code with a constructive minimum distance.

Conway Crossover - a code building variation operator.

An $(n, d) - code$ is a code of length n strings with minimum distance d.

The m-ary Conway variation operator

Input: One or more (n, d)-codes $C_1, \ldots C_k$, a random material rate $R \geq 0$ (an integer), and a minimum distance d.

Output: An $(n, d) - code$.

Details:
Let Q be the union of $C_1, \ldots, C_k$.
Generate R random words and add them to Q.
Shuffle the set Q into a random order.
Apply Conway's lexicode algorithm to Q, returning $CLA(Q)$.

This is not even a vaguely normal variation operator. It operates on any number of parents and incorporates the job of mutation with the random material being added. It is a general purpose tool for finding well-spaced out sets of objects.

Two new strategies for applying the Conway operator.

Ring Optimization is an evolutionary optimizer that places the evolving population on a ring-shaped geography. On other optimization problems, ring optimization has been found to manage its own exploration/exploitation trade-off in a beneficially self-organizing manner. **The usefulness of ring optimization has proven to vary substantially from problem to problem**. It excels at locating virtual robots and finite state DNA classifiers.

Hybridization functions by seeding the initial population of an evolutionary algorithm with high fitness individuals. In this work these individuals are codes located by other evolutionary algorithms. Because of the way the Conway operator re-mixes codes it was conjectured that **the Conway operator would work well with hybridization.**

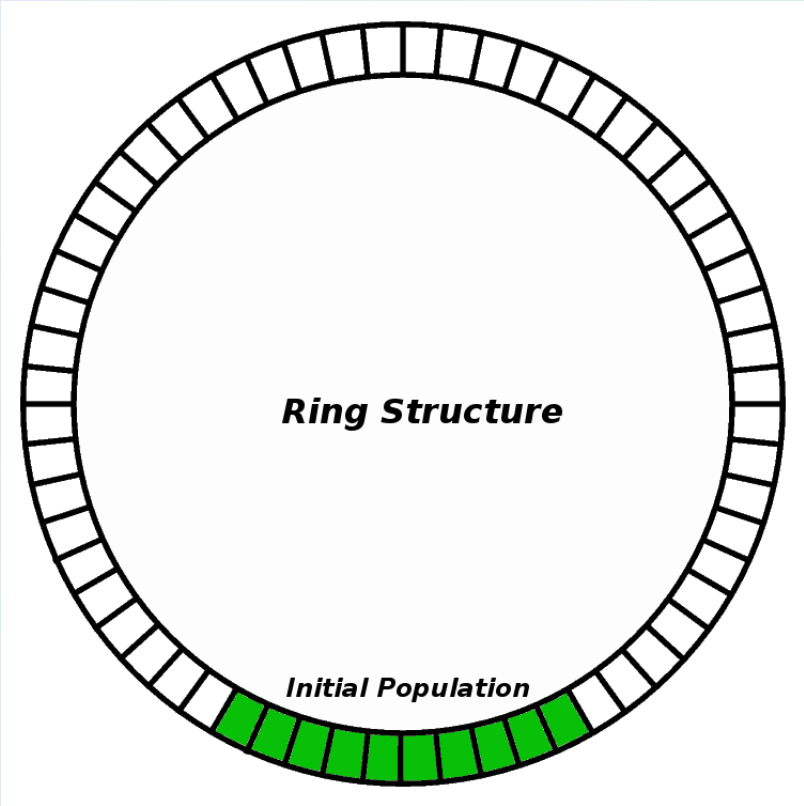
The Ring Optimizer

Algorithm

1. Populate a small segment of a ring data structure with solutions.
2. Repeat
3. Choose a parent solution at random.
4. Choose a co-parent within a mating radius M.
5. Generate a single child from the parents.
6. Choose a location within a migration radius m.
7. If empty, place child there, otherwise place the child there if it is not worse than the current occupant.
8. Until Done

A ring optimizer requires four parameters: ring size, mating radius, migration radius, and size of the initial population. Typical values are 2000-10000 positions in the ring, the radii 10-40, initial population 1-20.

Why do Ring Optimizers Work Well?



Consider how exploration and exploitation play out in ring optimization. At the beginning, almost all children survive because they are placed in empty slots. As the ring fills in, children must match or exceed the quality of an improving population. **A ring optimizer transitions smoothly from exploration to exploitation, region by region.**

The amount of exploration and the rate of transition to exploitation are controlled primarily by the ring's size and secondarily by the migration radius.

Hybridization.

Hybridization consists of **including good individuals**, in this study from previous runs of an evolutionary algorithm, **in the initial population**. In an earlier study, the practice of hybridizing controllers for a type of virtual robot was used and a substantial improvement in performance was observed.

Hybridization *can* have beneficial or detrimental effects. While the inserted genes are always high quality, they may represent the peak of a local optimum that is difficult to escape or may act, via crossover, to spark additional discovery. An important point is that **the difficulty of escaping from local optima depends on the variation operators used**. Again: we conjecture Conway crossover will be good at this sort of escape.

Results: bold entries show improved sizes of best known codes.

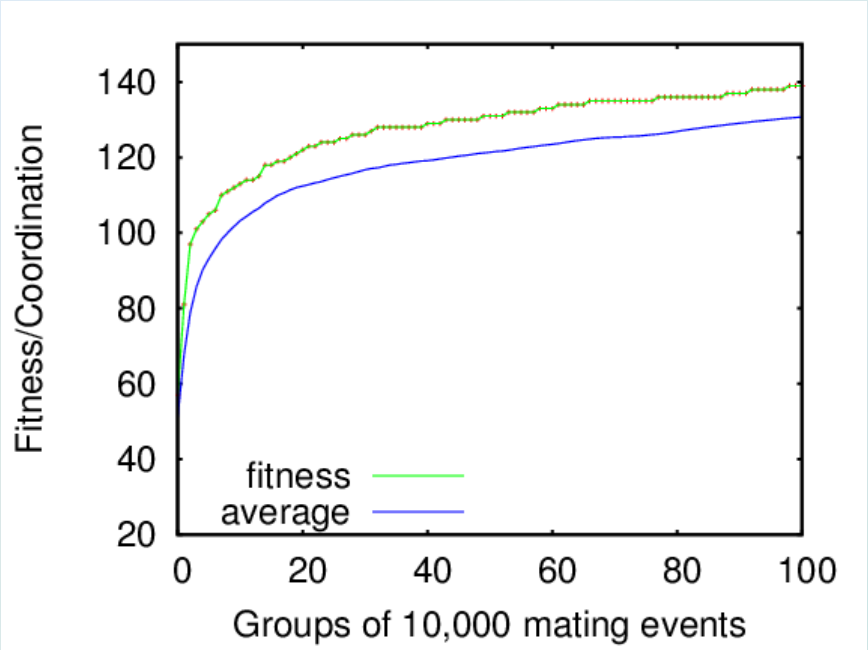
n\d	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	4	1	1	1	1	1	1	1	1
6	8	4	1	1	1	1	1	1	1
				• • •					
14	5517	879	188	54	27	13	7	5	4
15	6419	880	196	61	45	19	10	6	4
16	9220	1784	333	**196**[3]	82	32	**16**[2]	8	**6**[2]
17	9220	1784	333	**322**[1]	**143**[1]	**57**[2]	**24**[2]	**13**[2]	**8**[2]

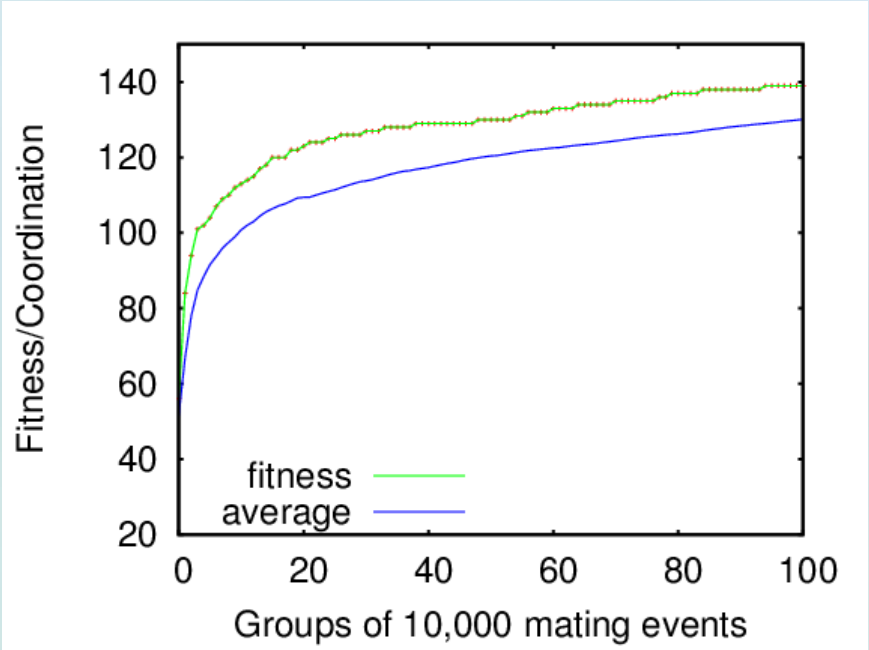
1 - results from ring optimization

2 - results from hybridization

3 - results from original CVO algorithm

Probably not done for the $(17, 9)$-codes yet?





The plots give fitness as a function of time for two runs of the ring optimizer operating on $(17, 9)$-codes. Notice that both runs are still trending upward. These were simply run for 1,000,000 fitness evaluations; the plots suggest more may be profitable.

Comments on Results

The **ring optimizer** improved the (17,8)-code, from **102 to 322**, a huge improvement, and the (17,9)-code, from **82 to 143**, also a substantial jump.

The improvements creditable to the **hybridizer** are smaller but more numerous:

- The best known size of the (16,11)-code improved from 15 to 16.
- The (16,13) code was found to have a best known size of 5, beating a theoretical lower bound of 4.
- The (17,10)-code improved from 54 to 57, the (17,11)-code from 23 to 24, and the (17,12)-code from 12 to 13.
- The (17,13)-code was found to have a best known size of 8, beating a theoretical lower bound of 4.

All of the hybridizer results of length 17 from minimum distance 10 to minimum distance 13 were built on, and improved on, ring optimization results.

Conclusions and Next Steps

- The hybridizer requires another algorithm's output to work – but given that it racked up the most improvements.
- The hybridizer **is in fact CVO-friendly**.
- When the final code is large, the ring optimizer, or the original CVO-based evolutionary algorithm, is a much better choice as the hybridizer is too slow.
- All the CVO-based algorithms are heuristics with no certificate of optimality, but for practical problems of locating codes bigger than a minimum size dictated by a biological experiment they are entirely suitable.
- The fact that the CVO permits us to avoid examining the entire space means the algorithms scale tolerably – and far better than the original algorithms for DNA edit-code location.

Many Thanks

Thanks to the *Natural Sciences and Engineering Research Council of Canada*, *Brock University* and the *University of Guelph* for support of this work. Both authors are indebted to Ling Guo, Joseph Brown, and John Orth for their help and inspiration.