Research Interests

My research interests include discrete mathematics and theoretical computer science, and my favourite results lay at their intersection. This document provides an informal overview of my previous and future work in Gray codes, de Bruijn cycles, Combinatorial Optimization, Graph Theory, and Bioinformatics. Citations refer to my publication list, which appears at the end of this document for convenience.

Sample Result. For an illustration of one of my basic discoveries, consider the following operation for rearranging the bits in a binary string.

> Move the first bit to the right until it passes over a 01 substring or the last bit.

For example, the operation transforms the binary string $\overline{1}100\underline{01}100011$ into $1000\underline{11}\overline{1}100011$, since the first bit (overlined) is moved to the right until it passes over a 01 substring (underlined). Similarly, if the operation is repeated it will transform $\overline{1}00\underline{11}100011$ into $00\underline{11}\overline{1}1100011$. Despite its simplicity, this operation has a surprising property: It rearranges the bits in all possible ways. In other words, if you start with a binary string containing s copies of 0 and t copies of 1, then repeatedly applying the operation will produce a list of strings that includes all $\binom{s+t}{t}$ possible arrangements of these bits, and one more application of the operation will produce the initial string [5]. The next two sections explain how this "curiosity" has a number of interesting generalizations and applications.

Gray Codes

My PhD thesis focused on an aspect of discrete mathematics and computer science known as *combinatorial generation* or simply "generating all possibilities". The goal is to efficiently iterate through every instance of a combinatorial object, such as the permutations of $\{1, 2, \ldots, n\}$, or some subset of binary strings (as in the sample result). Three important concepts are listed below:

- *Gray code order.* When the instances are ordered so that there is a constant difference from one to the next.
- *CAT algorithm.* An algorithm that generates successive objects in constant-amortized time (ie amortized O(1)-time). These algorithms may or may not use a Gray code order.
- *Loopless algorithm.* An algorithm that generates successive objects in worst-case O(1)-time. These algorithms must use a Gray code order.

In my thesis I uncovered a simple and practical method for creating Gray code algorithms for a wide variety of combinatorial objects. To illustrate the binary case, consider this new definition [1]:

> A *binary bubble language* is a set of binary strings with the following property: The first 01 of any string in the set can be replaced by 10 to create another string in the set.

For example, consider the feasible solutions to a knapsack problem with capacity C and element weights $w_1 \leq w_2 \leq \cdots \leq w_n$. Since the weights are — without loss of generality — sorted in non-decreasing order, an element i in a feasible solution can be replaced by element $i-1$ and the result will still be feasible. In particular, the minimum pair of consecutive elements with $i-1$ outside the knapsack and i inside the knapsack can be interchanged in this manner. Therefore, the feasible incidence vectors form a binary bubble language. One of my main results is that any binary bubble

language can be generated by a simple operation that is similar to the one given in the sample result of this document [1]. Furthermore, this operation can be efficiently implemented to create simple CAT [6,8] and loopless algorithms [10]. The advantage of this framework is that it can be applied to other objects, as illustrated by Figure 1. There is

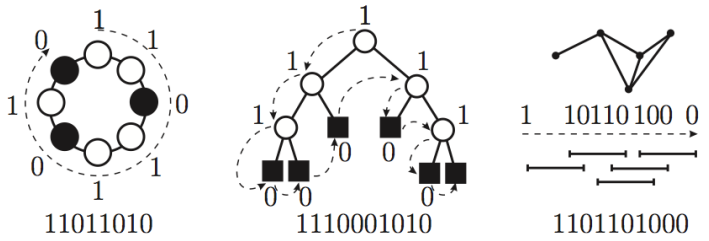


Figure 1: Many combinatorial objects can be represented by binary bubble languages including (from left-to-right) binary necklaces, binary trees, and connected unit interval graphs.

- My thesis contains a generalization of these bubble language Gray codes from binary strings to strings with any fixed-content (ie f_1 copies of 1, f_2 copies of 2, and so on). However, only the generalized sample result has been written for publication thus far [15].
- A collaboration with Brett Stevens has generalized the sample result to n-tuples, although the implications of this have not yet been explored.

de Bruijn Cycles

Binary Strings

A *de Bruijn cycle* is a circular binary string of length 2^n that contains every binary string of length n exactly once as a substring. More generally, a de Bruijn cycle with a *density upper-bound* is a circular binary string of length $\binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{d}$ that contains every binary string of length n so long as it has at most d copies of 1 (the number of ones in a binary string is often known as its *density*). For example, the following is an example with $n = 5$ and $d = 2$ (where the latter strings "wrap-around" the cycle).

$$\text{cycle: } 0000011000101001 \qquad \text{substrings: } \underbrace{00000, 00001, 00011, \ldots, 00100, 01000, 10000.}_{\text{binary strings of length } n=5 \text{ with at most } d=2 \text{ copies of } 1}$$

Standard techniques from graph theory can be used to prove that these generalized de Bruijn cycles exist for all values of n and d. However, the underlying graphs have exponential size, so they are not useful in constructing particular examples that are needed for applications. The open problem of efficiently constructing a single de Bruijn cycle with a density upper-bound was recently solved [7,11] . Surprisingly, the key ingredient of the construction is efficiently generating the special Gray code order for binary necklaces discussed in the previous section [1,8]. Currently, there is one open problem that would "complete" this line of research:

- Efficiently create de Bruijn cycles that have both a lower-bound and an upper-bound on the density of the substrings.

Permutations

When the notion of de Bruijn cycles is applied to other combinatorial objects, the resulting circular string is known as a *universal cycle*. One object that does not have a universal cycle is permutations

in one-line notation. For example, it is impossible to organize the permutations of $\{1, 2, 3\}$ — 123, 132, 213, 231, 312, 321 — around a circular string of length 6. On the other hand, each symbol in a permutation is redundant, and "shorthand" universal cycles for permutations do exist. For example, the shorthand representations of the aforementioned strings — 12, 13, 21, 23, 31, 32 — each appear once around the circular string 321312. These *shorthand universal cycles* have a number of potential applications, and Knuth asked for an efficient algorithm to create one of these cycles in an early draft of Volume 4 of *The Art of Computer Programming*. This problem was again solved by a novel yet simple construction that can be generated efficiently [4], and this solution now appears in *The Art of Computer Programming*. Further investigation has revealed an interesting connection between the shorthand universal cycles and the spanning trees of the permutohedron [2].

Combinatorial Optimization

A classic result in combinatorial optimization is the max-flow min-cut theorem:

Theorem: The maximum *st*-flow in a weighted directed graph is equal to the minimum *st*-cut.

In my Masters thesis I investigated a relation with the following differences:

- The *st*-cuts are replaced by dicuts (directed cuts). That is, we only consider cuts in which all of the arcs are directed from 'inside' the cut to 'outside' the cut (with no 'backward' arcs), and the cuts are not constrained by a fixed source node s or sink node t.
- The *st*-flows are replaced by dijoins (directed joins). That is, we only consider sets of arcs whose contraction will make the directed graph strongly-connected. That is, we only consider sets of arcs that intersect every directed cut.

The analogous relation (given below) has another fundamental distinction: Sometimes it is not true.

Conjecture: The minimum weight of a dicut equals the maximum number of disjoint dijoins.

Prior to my thesis there were three known counterexamples, as shown in Figure 2 a)-c). By using an

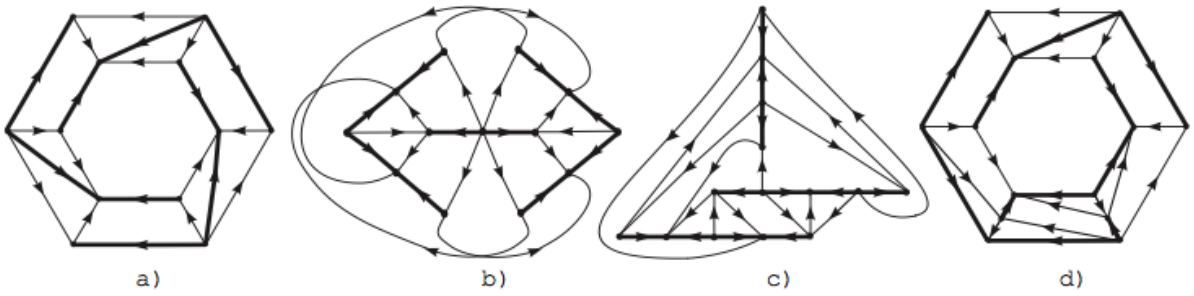


Figure 2: Weighted directed graphs in which the min-weight dicut is greater than the max number of disjoint dijoins. Thin/thick arcs represent weights of 0/1 and d) is an alternate embedding of c).

alternate embedding of the counterexample in Figure 2 c), I was able to find an additional operation that reduces the number of minimal counterexamples from two to three. (See Figure 2 d) for the alternate embedding, and observe its similarity to Figure 2 a).) Although this was an exciting development, I have not had a chance to follow up on it. In lieu of a complete characterization (which may be NP-Hard) I am interested in the following questions.

- The conjecture is true for source-sink connected graphs, and this result implies several previous results. In a *source-since connected graph*, every source is a *super source* meaning that there is a directed path from it to every sink, and every sink is a *super sink* meaning there is a directed path from every source to it. I would like to investigate the conjecture for graphs that contain one super-source and one super-sink (one of the counterexamples contains a super-sink).
- The conjecture is true for graphs that do not have a minor that is K_5 minus an edge [18]. It may be possible to extend this result with further work.
- Currently there is no known bound for the difference between the min-weight dicut and the maximum number of disjoint dijoins.

Graph Theory

Hamilton Cycles in Cayley Graphs

Shorthand universal cycles for permutations (see the de Bruijn cycle section) can also be viewed as Hamilton cycles in the Cayley graph of the symmetric group S_n generated by the operations σ_n and σ_{n-1}. (The symmetric group S_n is simply the permutations of $\{1, 2, \ldots, n\}$ in one-line notation, and the operations σ_n and σ_{n-1} move the first symbol of a permutation into the last or second-last position, respectively.) This Cayley graph is denoted by $\mathsf{Cay}(S_n, \{\sigma_{n-1}, \sigma_n\})$ and is shown in Figure 3 for $n = 4$. This connection has led to an interest in the general problem of finding Hamilton

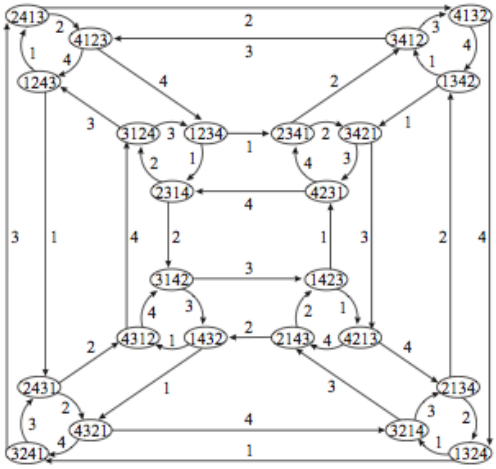


Figure 3: The Cayley graph for S_n with generators σ_n (straight arcs) and σ_{n-1} (curved arcs) for $n = 4$. By labelling the arcs with the first symbol of the node from where it originates (as shown) the Hamilton cycles are precisely the shorthand universal cycles for the permutations of $\{1, 2, 3, 4\}$.

cycles in Cayley graphs. Recently, this problem was solved in another case $\mathsf{Cay}(S_n, \{\sigma_2, \sigma_3, \sigma_n\})$ [9]. Related problems I am interested in appear below:

- Find and efficiently generate a Hamilton cycle in the Cayley graph $\mathsf{Cay}(S_n, \{\sigma_2, \sigma_n\})$. Currently there is a very difficult solution for $\mathsf{Cay}(S_n, \{\sigma_2, \sigma_n, \sigma_n^{-1}\})$ due to Corbett and Williamson.
- More generally, find and efficiently generate a Hamilton cycle for an arbitrary set of σ_i generators. (For some sets this is not possible, such as for $\{\sigma_3, \sigma_n\}$ and odd n.)

Graceful Tree Conjecture

A well-known conjecture attributed to Ringel and Kotzig is that every tree is graceful. That is, every tree has a graceful labelling. For a tree with n vertices, a *graceful labelling* is obtained by assigning each value in $\{0, 1, \ldots, n-1\}$ to a unique vertex in the tree, such that each value in $\{1, 2, \ldots, n-1\}$ is the absolute difference between the vertex labels on exactly one edge. The conjecture has attracted so much work without yielding much insight that it has been called a "disease" by Kotzig. Previous results have shown the conjecture to be true for various subclasses of trees including paths, caterpillars, lobsters, and so on. Unfortunately the techniques used for each subclass are often unrelated to each other, and thus do not give much insight into the general conjecture. Wendy Myrvold and I instead investigated the result of reducing graceful labels modulo 2. When n is even the resulting *binary graceful labelling* will have

- $\frac{n}{2}$ vertices will be labelled 0 and $\frac{n}{2}$ vertices will be labelled 1, and
- $\frac{n}{2} - 1$ edges will be labelled 0 and $\frac{n}{2}$ edges will be labelled 1

For example, Figure 4 contains an example of a tree with a graceful labelling together with the result of reducing the labels modulo 2.

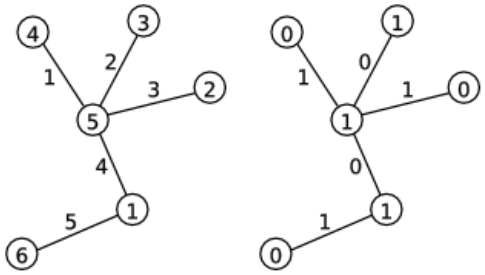


Figure 4: A tree with a graceful labelling (left) and the resulting binary graceful labelling (right) obtained by reducing the labels modulo 2.

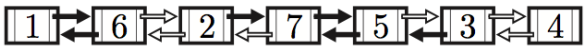
We proved that every tree has a binary graceful labelling. Moreover, we extend the definition to forests, and prove that a forest has a binary graceful labelling if and only if it is not made up of k components, where k is congruent to 2 or 3 mod 4, and each component is a tree whose vertices all have odd degree. This result gives some insight into how graceful labels for trees can and cannot be built in general trees. This work will not be submitted until the summer of 2011.

Bioinformatics

A textbook that is closely related to my previous research is *The Combinatorics of Genome Rearrangements* by Fertin, Labarre, Rusu, Tannier, and Vialette. The primary problem tackled in this book is to determine the minimum number of operations of a given type required to sort a given combinatorial object such as a permutation or signed permutation. For example, the number of prefix-reversals required to sort a permutation is widely known as the *pancake problem*. (A prefix-reversal of length 4 transforms abcdefg into dcbaefg.)

One variation of this type of problem is to actually sort the combinatorial object efficiently inside of a data structure. For example, suppose a given permutation of $\{1, 2, \ldots, n\}$ can be sorted by applying prefix-reversals of length $\ell_1, \ell_2, \ldots, \ell_k$. To actually apply these prefix-reversals could take $\Omega(nk)$-time if the permutation is stored in an array or linked list since each prefix-reversal of length ℓ will take $\Omega(\ell)$-time. A data structure that permits O(1)-time reversals of any length is

relaxation of a doubly-linked list in which the forward and backward pointers of any node can be interchanged. For example, the following data structure stores the permutation 1627534.



Notice that each node has a black/white pointer, although they may point forward/backward or backward/forward. This ambiguity makes the data structure difficult to work with, but In a paper presented at the *Fun with Algorithms* conference in 2010 I showed that it can be used in a loopless algorithm for generating permutations [13]. This data structure may also be useful in efficiently reversing signed permutations.

Another variation of the genome rearrangement problem is to efficiently create all solutions. For example, according to recent work by Anne Bergeron at UQAM, it is possible to count the number of minimum double-cut and join solutions, but there is of yet no algorithm for efficiently creating each of these possible solutions. This type of result could be useful in determining the most plausible sequence of sorting operations subject to some additional criteria.

My previous results have also been used in practical sequence alignment software. For example, my generalization of this document's sample result from binary strings to strings with arbitrary fixed-content [15] was implemented by Erik Garrison for the FreeBayes software package in Gabor Marth's lab at Boston College.

Reception

I have been fortunate to win a "best paper" award at an international conference (CATS 2008) and my thesis was nominated by the University of Victoria for the best Canadian doctoral thesis in 2009. My work has also gained positive feedback from two notable researchers:

- Ron Graham was the external examiner for my PhD thesis and commented to my supervisor Frank Ruskey that it was one of the best theses he had read in the past decade.
- Don Knuth commented by email that "I have terrific respect for your work – you're responsible for at least two of the most exciting improvements in the entire field of combinatorial generation during the past five years" and included several results in *The Art of Computer Programming*. (This was written in January 2010 with regards to some preliminary results.)

For references from those who I have worked closely with, please contact

- Frank Ruskey `ruskey@cs.uvic.ca` (PhD supervisor),
- Joe Sawada `jsawada@uoguelph.ca` (co-author),
- Brett Stevens `brett@math.carleton.ca` (postdoctoral supervisor), or
- Bertrand Guenin `bguenin@math.uwaterloo.ca` (Masters supervisor).

Publications and Submissions

Published Journal Articles

[1] F. Ruskey, J. Sawada, and A. Williams, *Binary Bubble Languages and Cool-lex Order*. Journal of Combinatorial Theory Series A [JCTA] (accepted) 2011, 13 pages.

[2] A. Holroyd, F. Ruskey, and A. Williams, *Shorthand Universal Cycles for Permutations*. Algorithmica (accepted) 2011, 30 pages.

[3] F. Ruskey, and A. Williams, *The Feline Josephus Problem.* Theory of Computing Systems (accepted) 2011, 13 pages.

[4] F. Ruskey and A. Williams, *An Explicit Universal Cycle for the* $(n-1)$*-Permutations of an* n*-Set.* ACM Transactions on Algorithms, Volume 6, Issue 3, Article 45 (2010).

[5] F. Ruskey and A. Williams, *The Coolest Way to Generate Combinations*, Discrete Mathematics, Volume 309, Issue 17 (2009), 5305-5320.

Submitted to Journal

[6] J. Sawada and A. Williams, *A Gray Code for Fixed-Density Necklaces and Lyndon Words in Constant Amortized Time.* Submitted to Theoretical Computer Science (Special Issue for GASCOM 2010) in March 2011.

[7] F. Ruskey, J. Sawada, and A. Williams, *Fixed-Density de Bruijn Sequences.* Submitted to SIAM Journal of Discrete Mathematics in September 2010.

[8] J. Sawada, and A. Williams, *Efficient Oracles for Generating Binary Bubble Languages.* Submitted to Electronic Journal of Combinatorics in September 2010.

Published Refereed Conference Proceedings

[9] B. Stevens and A. Williams, *Hamilton Cycles in Restricted Rotator Graphs.* Submitted to the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011).

[10] S. Durocher, P. C. Li, D. Mondal, and A. Williams, *Ranking and Loopless Generation of k-ary Dyck Words in Cool-lex Order.* Submitted to the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011).

[11] J. Sawada, B. Stevens, and A. Williams, *De Bruijn Sequences for Binary Strings with Maximum Density*, the 5th Workshop on Algorithms and Computation (WALCOM 2011), New Dehli, India, 6552 Lecture Notes in Computer Science (2011).

[12] A. Holroyd, F. Ruskey, and A. Williams, *Faster Generation of Shorthand Universal Cycles for Permutations*, The Sixteenth International Computing and Combinatorics Conference (COCOON 2010), Nha Trang, Vietnam, 2010, Lecture Notes in Computer Science, 6196 (2010) 298-307.

[13] A. Williams, *O(1)-Time Unsorting by Prefix-Reversals in a Boustrophedon Linked List*, Fifth International Conference on FUN with Algorithms (FUN 2010), Ischia Island, Italy, 2010, Lecture Notes in Computer Science, 6099 (2010) 368-379.

[14] F. Ruskey and A. Williams, *The Feline Josephus Problem*, Fifth International Conference on FUN with Algorithms (FUN 2010), Ischia Island, Italy, 2010, Lecture Notes in Computer Science, 6099 (2010) 343-354.

[15] A. Williams, *Loopless Generation of Multiset Permutations using a Constant Number of Variables by Prefix Shifts.* SODA 2009, ACM-SIAM Symposium on Discrete Algorithms, New York, United States (2009).

[16] F. Ruskey and A. Williams, *Generating Balanced Parentheses and Binary Trees by Prefix Shifts*, Proc. Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, Australia, 2008. Theory of Computing, 77 (2008). 107-115.

[17] G. Lee, F. Ruskey, and A. Williams, *Hamming Distance from Irreducible Polynomials over* $\mathbb{F}_2$, International Conference on Analysis of Algorithms (AofA 2007), Juan-les-pins, France, 2007. DMTCS (2007). 169-180.

[18] O. Lee and A. Williams, *Packing Dicycle Covers in Planar Graphs with no* $K_5 - e$ *Minor*, Theoretical Informatics 7th Latin American Symposium (LATIN 2006), Valdivia, Chile, 2006. Lecture Notes in Computer Science, 3887 (2006) 677-688.

[19] F. Ruskey and A. Williams, *Generating Combinations by Prefix Shifts*, The Eleventh International Computing and Combinatorics Conference (COCOON 2005), Kunming, China, 2005. Lecture Notes in Computer Science, 3595 (2005) 570-576.

[20] B. Guenin and A. Williams, *Advances in Packing Directed Joins*, GRACO 2005, 2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics, Angra dos Reis, Brazil, 2005. Electronic Notes in Discrete Mathematics, 19 (2005) 249-255.