# A fast algorithm to generate Beckett-Gray codes
### Extended Abstract

## Joe Sawada [1,2]

*Computing and Information Science*
*University of Guelph*
*Guelph, Canada*

## Dennis Chi-Him Wong [3]

*Computing and Information Science*
*University of Guelph*
*Guelph, Canada*

## Abstract

We provide several heuristics to improve the efficiency of exhaustively generating Gray codes on length $n$ binary strings. We then apply the heuristics to a restricted class of Gray codes on binary strings known as Beckett-Gray codes. The resulting algoirthm has a speed-up by a factor of 60 over the fastest previously known algorithm when $n = 6$. It was also used to discover over 9500 new Beckett-Gray codes for $n = 7$ in the equivalent of three months of computation time.

*Keywords:* Beckett-Gray codes, Code generation, Enumeration, Gray codes

# 1    Introduction

A *Gray code* is an ordering of combinatorial objects such that any two successive objects differ by some pre-specified constant amount. The amount of change between elements is called the *Hamming distance*. The following is an example of a Gray code on length 3 binary strings with Hamming distance 1, where the bit that differs with the previous element is underlined:

$$[000, 00\underline{1}, 0\underline{1}1, \underline{1}11, 10\underline{1}, 10\underline{0}, 1\underline{1}0, \underline{0}10]$$

A Gray code is said to be *cyclic* if the last element and the first element in the sequence also differ by some pre-specified constant amount. The Gray code on length 3 binary strings above demonstrates the cyclic property. It is well known that a cyclic Gray code on length $n$ binary string corresponds to a Hamilton cycle in the $n$-dimensional hypercube. Figure 1 shows the 3-dimensional hypercube $Q_3$ with the Hamilton cycle corresponding to the example Gray code on length 3 binary strings above.
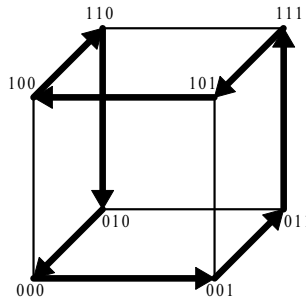


Fig. 1. A Hamilton cycle in $Q_3$ corresponds to a cyclic Gray code on length 3 binary strings.

In this paper, we are interested in finding a special type of Gray code on binary strings named after Irish playright Samuel Beckett. One of his plays, "Quad" [Bec84], had 4 actors and was divided into a sequence of time periods. At the end of each time period, Beckett wished to have one of the four actors either entering or exiting the stage; he wished the play to begin and end with an empty stage and he wished each subset of actors to appear on stage exactly once. Observe that this problem is equivalent to finding a cyclic Gray code on length 4 binary strings. However, Beckett wanted an additional restriction on the scripting: the actor that leaves the stage must be the one who has currently been on stage for the longest time.

If we apply this final restriction to cyclic Gray codes on binary strings we obtain what is known as a *Beckett-Gray code*. Unfortunately, Beckett was

unable to find a 4-bit Beckett-Gray code (and indeed none exist), so instead he repeated some subsets of actors and used 24 time periods. The following is an example of a 5-bit Beckett-Gray code where the bit that has been a 1 for the longest time is underlined:

$$[00000, 0000\underline{1}, 000\underline{1}1, 000\underline{1}0, 00\underline{1}10, 00\underline{1}11, 00\underline{1}01, 01\underline{1}01,$$
$$0100\underline{1}, 0\underline{1}000, 0\underline{1}010, 0\underline{1}011, 1\underline{1}011, 100\underline{1}1, 101\underline{1}1, 1010\underline{1},$$
$$\underline{1}0100, 00\underline{1}00, 01\underline{1}00, 11\underline{1}00, 1\underline{1}000, 1\underline{1}010, \underline{1}0010, \underline{1}0110,$$
$$1\underline{1}110, 011\underline{1}0, 011\underline{1}1, 111\underline{1}1, 11\underline{1}01, 1\underline{1}001, 1000\underline{1}, \underline{1}0000]$$

There is no known constructive way for building Beckett-Gray codes and finding Beckett-Gray codes is listed as a hard problem in [Knu06]. Beckett-Gray codes are known to exist for $n = 2, 5, 6, 7, 8$ and they do not exist for $n = 3, 4$. No Beckett-Gray code is known for any $n \geq 9$.

A recursive algorithm for generating Beckett-Gray codes has been developed by Cooke et al. [CNS07]. However, for $n = 6$ their algorithm currently requires more than a month of computation time to produce an exhaustive list, and for $n = 7$ a single Beckett-Gray code was found only after several months of computing time. They also found the first Beckett-Gray code for $n = 8$ by using simulated annealing techniques. In this paper, we describe some heuristics that can improve the runtime for the exhaustive generation of all cyclic Gray codes on length $n$ binary strings and apply the results to Beckett-Gray codes. Our first two heuristics consider the Hamiltonian property of cyclic Gray codes on binary strings and our third heuristic considers equivalence under reversal. There is an improvement by a factor of over 60 on the runtime for an exhaustive search for $n = 6$ when compared to [CNS07], and over 9500 new Beckett-Gray codes for $n = 7$ were also discovered by our algorithm in about 3 months of computing time.

## 2   Generating Gray codes on binary strings

We begin this section by describing a simple recursive algorithm to find all Gray codes on length $n$ binary strings. Then we will outline some heuristics to improve the running time. The basic idea is to extend a partial Gray code listing on length $n$ binary strings by considering all strings that differ by a single bit from the last string in the partial listing. For each such string, the listing is extended and a recursive call is made. Since equivalence classes of size $n!$ are formed by permuting the bit positions, we only generate the lexicographically smallest Gray code on length $n$ binary strings in each equivalence class [Wel61]. This can be done by starting with the string corresponding to 0 and maintaining the largest bit position *maxpos* that has at some point been

**procedure** GC($s, x, maxpos$ :integer)
**local** $i$

1: **if** $s \geq 2^n$ **then**
2:     Print()
3: **else**
4:     **for** $i = 0$ **to** Min($n-1, maxpos$) **do**
5:         $x = $ Flip($x, i$)
6:         **if** $avail[x]$ **then**
7:             $avail[x] = $ **false**
8:             $bgc[s] = x$
9:             GC($s + 1, x, $ Max($maxpos, i$))
10:            $avail[x] = $ **true**
11:        $x = $ Flip($x, i$)

Fig. 2. GC($s, x, maspos$).

set to 1. Pseudocode for this algorithm is shown in Figure 2. The structures used by the algorithm are as follows:

- *bgc*: a global array to store the Gray code listing on length $n$ binary strings,

- *avail*: a global boolean array to keep track of which strings are still available,

- $x$: the integer value of the string currently at the end of the listing,

- $s$: the length of the partial listing,

- *maxpos*: the largest bit position that has at some point been set to 1,

- Flip($x, i$): a function that returns the integer value obtained by flipping the $i$-th bit in the binary representation of $x$,

- Print(): a function that prints out the Gray code *bgc*.

To initialize the algorithm we set $avail[i] = $ **true** for $i = 1$ to $n-1$, set $avail[0] = $ **false**, set $bgc[0] = 0$, and then call GC($1, 0, 0$).

The algorithm will generate all Gray codes on length $n$ binary strings. To generate only the cyclic Gray codes on length $n$ binary strings, the Print function is only called if the last string in the listing is a power of 2 (it differs by one bit from 0).

## 2.1 Improvements for cyclic Gray codes on binary strings

As mentioned earlier, a cyclic Gray code on length $n$ binary strings corresponds to a Hamilton cycle in the hypercube $Q_n$. Let $P = \{v_0, v_1, v_2, \ldots, v_k\}$ be a

simple path in $Q_n$ where $k < 2^n$. Such a path may correspond to a prefix of some Gray code on length $n$ binary strings. Now consider the induced subgraph $G_P$ that is obtained from $Q_n$ by removing all the vertices of $P$ except $v_0$ and their incident edges. Since a Hamilton cycle needs to visit all vertices and return to the starting vertex, if $G_P$ is disconnected then there will exist no Hamilton cycle starting with $P$.

One way to test this connectivity is to apply a breadth-first search every time we add a vertex to the path. However, since there are $n \cdot 2^{n-1}$ edges in $Q_n$, this test would take time $O(n \cdot 2^n)$ which adds a large overhead. Instead, we apply two heuristics that test for partial connectivity. The first heuristic focuses on finding pendant vertices in $G_P$ and can be implemented in $O(n)$ time per recursive call. The second heuristic applies an Eulerian property on a related graph and can be implemented in $O(1)$ per recursive call.

### 2.1.1  Pendant vertices

A *pendant vertex* is a vertex with degree 1. Except for $v_0$, if there are any pendant vertices in $G_P$ then there will be no way to extend $P$ to a Hamilton cycle. To efficiently find pendant vertices in $G_P$, we maintain a counter for the number of available neighbors for each vertex. When a vertex $v_i$ is added to $P$, we decrement this value for each of its neighbors. If one of its neighbors $u$ gets decremented to 1, it means that $u$ must be the next vertex in the path since it requires one edge to get to the next vertex in the path. If two or more neighbors get decremented to 1, then there will be no Hamilton cycle starting with $P$. The exception to this is vertex $v_0$ which only gets added at the end. If its degree is reduced to 0, then $P$ cannot be extended to a Hamilton cycle. Implementation of these tests to the $n$-bit cyclic Gray code algorithm can be easily implemented in $O(n)$ time per recursive call.

### 2.1.2  The Eulerian property

Consider a subgraph of $Q_n$ consisting of all vertices in $Q_n$ but containing only the edges of some Hamilton cycle $Ham$. Let this graph be denoted by $G_{Ham}$. Figure 3 shows such a $G_{Ham}$ for $n = 5$ illustrating the Hamilton cycle as directed edges. Now partition the set of vertices into $n + 1$ different subsets $V_0, V_1, \ldots, V_n$ such that each subset contains the vertices which have the same number of bits set to 1. Thus, the number of vertices in each $V_i$ is $\binom{n}{i}$, where $i$ is the number of bits set to 1. Edges exist only between elements in the subsets $V_i$ and $V_j$ where $| \, i - j \, | = 1$ since the Hamming distance between successive elements in a Gray code on length $n$ binary strings (Hamilton cycle) is one. The vertical alignment of vertices in Figure 3 illustrate this partitioning.
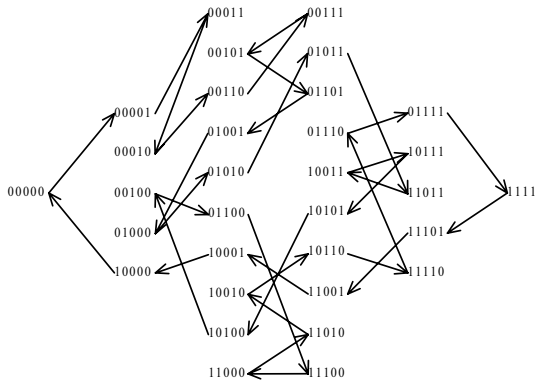
Fig. 3. A Hamilton cycle in $G_{Ham}$ corresponds to a cyclic Gray code on length 5 binary strings.

Now consider a directed multi-graph constructed by collapsing the vertices in each partition of $G_{Ham}$ and using the directed edges from the Hamilton cycle. In the resulting graph $G_{Eul}$, the vertices are $V_0, V_1, \ldots, V_n$ with a directed edge between $V_i$ and $V_j$ for every directed edge between the vertices of $V_i$ and $V_j$ in $G_{Ham}$. Observe the Hamilton cycle $G_{Ham}$ now corresponds to an Euler cycle in $G_{Eul}$.

Using the Eulerian property, we follow the approach of Fleury's algorithm [Luc91] to test for connectivity. A *bridge* of a connected graph is an edge whose removal disconnects the graph. The main idea of Fleury's algorithm is to never cross a bridge of the reduced graph unless there is no other choice. Thus, if we have a path $P$ that starts from $V_0$ and ends at $V_i$, we can detect a bridge if the number directed edges from $V_{i+1}$ to $V_i$ is one (the bridge), but the number of edges from $V_{i+1}$ to $V_{i+2}$ is greater than 0.

Using some extra data structures, this partial connectivity test of $Q_n$ can be implemented in $O(1)$ time per recursive call.

### 2.1.3   Equivalence under reversal

In addition to the symmetry with respect to the bit positions, cyclic Gray codes on length $n$ binary strings also have equivalence under reversal. Although it is less obvious, the reversal of a Beckett-Gray code is also a Beckett-Gray code. As long as $n \neq 2^j$, we can generate the non-isomorphic Gray codes on length $n$ binary strings by adding a constant amount of work per recursive call by considering the position of the string $111 \cdots 11$. Applying this symmetry, we can further reduce the time it takes to generate (non-isomorphic) $n$-bit cyclic Gray codes.

# 3   Generating Beckett-Gray codes

To generate Beckett-Gray codes we can modify the algorithm outlined in Figure 2 to apply the final Beckett restriction: if a bit is changed from a 1 to a 0 in successive strings, it must be the bit that has been a one the longest. This can be done by maintaining a first-in, first out queue of the bit positions set to 1. If this constraint is added, the resulting algorithm to generate $n$-bit Beckett-Gray codes will be very similar to the one outlined by Cooke et al. [CNS07].

However, if we apply the additional improvements for cyclic Gray codes on binary strings that were outlined in the previous section, we obtain some significant improvements. In our experiments, we evaluate the performance of the algorithm in [CNS07] against our improved algorithm that generates only non-isomorphic Beckett-Gray codes. For $n = 6$, we achieved a speedup by a factor of approximately 60, allowing us to generate all non-isomorphic Beckett-Gray codes in under 15 hours of computation. (It is interesting to note that the number of regular Gray codes on binary strings where $n = 6$ is estimated to be $7 \times 20^{22}$ [SVS83].) For $n = 7$ we distributed our program on sixteen processors using SHARCNET [4]. This allowed us to find over 9500 new Beckett-Gray codes in the equivalent of 3 months of computation time.

# References

[Bec84] S. Beckett. Collected shorter plays of Samuel Beckett. *faber and faber, London*, 1984.

[CNS07] M. Cooke, C. North, and B. Stevens. Beckett-Gray codes. *Discrete Mathematics, submitted*, 2007.

[Knu06] Donald E. Knuth. *The Art of Computer Programming*, volume 4. Addison-Wesley, 2006.

[Luc91] E. Lucas. Récréations Mathématiques. *Gauthier-Villares, Paris*, 1891.

[SVS83] J. Silverman, V.E. Vickers, and J.L. Sampson. Statistical estimates of the $n$-bit Gray codes by restricted random generation of permutations of 1 to $2^n$. *IEEE Transactions of Information Theory 29*, pages 894–901, 1983.

[Wel61] M. B. Wells. Generation of permutations by transposition. *Mathematics of Computation 15*, pages 192–195, 1961.

---

[4] SHARCNET is a consortium of colleges and universities in a "cluster of clusters" of high performance computers, linked by advanced fibre optics (`http://www.sharcnet.ca`).