Evolutionary Construction of de Bruijn Sequences

Meltem Sönmez Turan
National Institute of Standards and Technology
Maryland
meltem.turan@nist.gov

ABSTRACT

A binary de Bruijn sequence of order n is a cyclic sequence of period 2^n, in which each n-bit pattern appears exactly once. These sequences are commonly used in applications such as stream cipher design, pseudo-random number generation, 3-D pattern recognition, network modeling, mainly due to their good statistical properties. Constructing de Bruijn sequences is of interest and well studied in the literature. In this study, we propose a new randomized construction method based on genetic algorithms. The method models de Bruijn sequences as special type of traveling salesman tours and tries to find optimal solutions to this special type of the traveling salesman problem (TSP). We present some experimental results for $n < 14$.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

Keywords

De Bruijn sequences, Genetic algorithms, Traveling salesman problem

1. INTRODUCTION

A binary de Bruijn sequence of order n is a cyclic sequence of period 2^n, in which each n-bit pattern appears exactly once. De Bruijn sequences are balanced, i.e., have same number of 1's and 0's, and have good randomness properties. These sequences are commonly used in applications such as stream cipher design, pseudo-random number generation, 3-D pattern recognition, network modeling etc.

De Bruijn sequences of order n exist for $n \geq 2$ and for a given n, the number of de Bruijn sequences is $2^{2^{n-1}-n}$ [8]. There are various methods to construct de Bruijn sequences [4, 15]. Some of these methods start with an n-bit pattern and append a new bit to the sequence based on a pre-determined criteria, whereas some methods are recursive and use lower-order de Bruijn sequences as input. These construction methods are capable of generating a subset or all of the possible sequences for a given order n, with varying time and memory complexity. It is of interest to find an efficient construction method that is not limited to a subset of all sequences.

Genetic Algorithms (GAs), developed by Holland [10], are a part of evolutionary computation which is a subfield of artificial intelligence. These heuristic search algorithms are based on the survival of the fittest and natural selection concept of natural genetics. They simulate the natural evolution with the goal of finding the best solution to problems having a large and non-linear search space.

In this study, we propose a new method to construct de Bruijn sequences using genetic algorithms. First, we define a special type of the traveling salesman problem (TSP), denoted as TSP_n^* with 2^n nodes and a predefined distance matrix. Then, we use our genetic algorithm to find an optimal tour for TSP_n^* and convert the output tour to a de Bruijn sequence.

The outline of the paper is as follows. In Sect. 2, literature surveys on de Bruijn sequence construction methods and genetic algorithms are provided. In Sect. 3, the definition and some basic properties of TSP_n^* are given. In Sect. 4, the details of the genetic algorithm are explained. The experimental results are summarized in Sect. 5. Finally, the results are discussed in the final section.

2. PRELIMINARIES

The preliminaries part of this study consists of two main parts; constructions of de Bruijn sequences, and genetic algorithms especially focusing on their application to TSP.

2.1 Constructions of de Bruijn Sequences

Simplest construction method is the Prefer-one method which starts with n zeros and adds the bit 1 to the sequence whenever possible. For $n = 4$, the prefer-one method generates the following sequence;

$$0000111101100101.$$

Prefer-same [4] and prefer-opposite [1] methods are similar to prefer-one method using different bit insertion criteria. Given the same initial state, these methods generate only one de Bruijn sequence.

De Bruijn sequences can also be generated using feedback shift registers (FSRs). A *FSR* is a device that shifts its contents into adjacent positions within the register and fills the position on the other end with a new value generated by the *feedback function.* A FSR is uniquely determined by its length n and feedback function f. The output sequence $\mathbf{S}= \{s_0, s_1, s_2, \ldots\}$ of a FSR satisfy the following recursion

$$s_{n+i} = f(s_i, \ldots, s_{n-1+i}),\ i \geq 0 \qquad (1)$$

given the initial state $(s_0, s_1, \ldots, s_{n-1})$. To guarantee that every state has a unique predecessor and successor, f should be written in the form $f(x_1, \ldots, x_n) = x_1 + g(x_2, \ldots, x_n)$ [8]. Some necessary conditions on f and g to generate a de Bruijn sequence are given as follows;

1. To avoid all zero cycle, $f(0, \ldots, 0) = 1$.
2. To avoid all one cycle, $f(1, \ldots, 1) = 0$.
3. To avoid the cycle $(00\ldots01)$, not all of the linear terms exists in f [9].
4. The parity of the truth table of g is 1 [8].
5. The function g is non-symmetric [2], i.e.

$$g(x_2, \ldots, x_n) \neq g(x_n, \ldots, x_2). \qquad (2)$$

As n gets larger, finding an FSR with maximum length becomes very inefficient. One way to construct de Bruijn sequences using FSRs is to use linear feedback shift registers (LFSRs) with period $2^n - 1$. De Bruijn sequences are constructed by simply appending 0 to the $(n-1)$-bit $(00\ldots0)$ pattern in LFSR output. The number of distinct de Bruijn sequences generated by this method is bounded by the number of degree n primitive polynomials over $GF(2)$ which is equal to $\phi(2^n - 1)/n$, where ϕ is the Euler-phi function.

Fredricksen and Maiorana [5] proposed an efficient method to generated the lexicographically minimal de Bruijn sequence. Etzion and Lempel [3] provided construction methods that can generate de Bruijn sequences with minimal linear complexity. Games [6] provided a recursive construction that inputs two de Bruijn sequence of order n to produce a de Bruijn sequence of order $n + 1$. Fredricksen [4] and Ralston[15] give a survey of these construction methods.

2.2 Genetic Algorithms

GAs operate on a population of chromosomes by an iterative procedure. A chromosome represents a candidate solution for the problem of interest, its genes corresponding to solution elements. As given in Figure 1, a GA starts by generating an initial population. A fitness function is used to evaluate the ability of each chromosome to survive over the generations. In each generation, population members undergo selection, crossover, and mutation to generate new offspring, according to predetermined crossover and mutation probabilities. Some of the chromosomes are removed from the population in order to reduce the population size back to its initial size. This is repeated until given stopping conditions are satisfied. When GA stops, the best solution found so far is given as the output.

```
Generate initial population;
Until stopping condition is satisfied
  Select parents from the population;
  Apply crossover operator to produce children;
  Apply mutation operator to the children;
  Extend the population by adding the children to it;
  Reduce the extended population back to its original size;
Output the best chromosome;
```

Figure 1: The pseudocode of a generic GA

2.2.1 Genetic Algorithms for TSP

GAs can be used to solve the TSP which is an NP-hard sequence-based combinatorial optimization problem. Given n nodes and their pairwise distances, TSP aims to find a shortest closed tour visiting each node exactly once. Although the exact methods (such as evaluating all possible tours) fall short when the problem size gets large ($n > 30$), heuristic methods obtain near-optimal solutions for real-world applications in a reasonable computation time.

There are various successful implementations of GAs to TSP and their performance highly depends on the selection of the crossover operator. Some crossovers examples can be given as; PMX [7], CX [14], EAX [13], NX [11], NEX [12], NNX [16]. It seems that the crossover operators that make use of problem specific information perform better.

3. A SPECIAL TYPE OF TSP: TSP$^*_N$

Let $(a_1, \ldots, a_n)$ be the binary representation of integer A $(< 2^n)$. We define $msb_s(A)$ and $lsb_s(A)$ to be the first and last s $(s \leq n)$ bits of $(a_1, \ldots, a_n)$, respectively , i.e.,

$$msb_s(A) = (a_1, a_2, \ldots, a_s),$$

$$lsb_s(A) = (a_{n-s+1}, a_{n-s+2}, \ldots, a_n).$$

Definition 1. *TSP^*_n is a special type of TSP with 2^n nodes where the distance between two distinct nodes $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ is defined to be*

$$d_{A,B} = n - max_s\{s | msb_s(A) = lsb_s(B)\}, \quad s \geq 0.$$

The distance between A and B corresponds to the number of edges in the shortest path from A to B in the Good's diagram and can take values between 1 and n. Since the diagram is directed, the distances are asymmetric. As an example, the distance between node 2 with binary representation (0,1,0) and node 7 with binary representation (1,1,1) for TSP$^*_3$ is 3, as illustrated in Figure 2.

The tour lengths for TSP$^*_n$ vary between 2^n and $n2^n$. Each cyclically-distinct optimal solution (i.e., the tours with length 2^n) corresponds to a de Bruijn sequence. Given an optimal tour $(A^{(1)}, A^{(2)}, \ldots, A^{(2^n)})$, de Bruijn sequence is generated as

$$(a_1^{(1)}, a_1^{(2)}, \ldots, a_1^{(2^n)})$$

where $a_1^{(i)}$ corresponds to the first bit in the binary representation of the node $A^{(i)}$. Selecting different bit positions only results in the rotation of the same de Bruijn sequence.

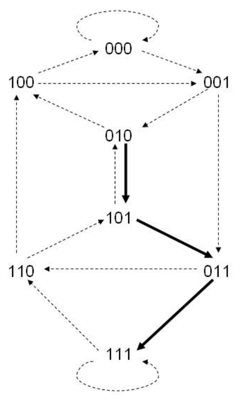

Figure 2: Good's graph for n=3, showing the shortest path from (0,1,0) to (1,1,1)

Example 1. *TSP_3^* has 8 nodes $\{0, 1, \ldots, 7\}$, with the distance matrix given in Figure 3. The length of the tour (0 1 2 5 3 7 6 4) = (000, 001, 010, 101, 011, 111, 110, 100) is 8 and corresponds to the following de Bruijn sequence (00010111).*

From \ To	0	1	2	3	4	5	6	7
0	-	1	2	2	3	3	3	3
1	3	-	1	1	2	2	2	2
2	2	2	-	2	1	1	3	3
3	3	3	3	-	2	2	1	1
4	1	1	2	2	-	3	3	3
5	3	3	1	1	2	-	2	2
6	2	2	2	2	1	1	-	3
7	3	3	3	3	2	2	1	-

Figure 3: Distance matrix for the TSP_3^* problem.

4. PROPOSED GENETIC ALGORITHM

In this section, we describe the main components of the proposed GA.

Representation: The population members can be represented using either binary or path representation. The binary representation is more natural to generate de Bruijn sequences and requires less memory. However, since our underlying problem is the TSP, we used the path representation which is more suitable to solve TSPs.

Initial Population: The GA starts by generating the initial population that contains N_p members. To generate the initial population, we used two different methods;

- *Method I* simply generates random tours, therefore the quality of the initial population is low in terms of the tour lengths.
- *Method II* uses an heuristic approach which results in higher quality members. This method first starts with a random node and selects the next node randomly from the nodes whose distance is 1 to the current node, if possible, otherwise, an unused node is selected randomly.

Crossover: We use the nearest neighbor crossover (NNX), defined in [16], as our crossover operator. NNX is originally proposed for symmetric TSP, in which the distances are undirected, however with slight modification, it can also be used for asymmetric TSPs. NNX randomly selects the starting node and finds the successors of this node in each parent. The closest unused successor is used as the next node, if possible. If both successors are used before, NNX randomly selects an unused node.

Example 2. *Let Parent I = (1 2 4 7 6 3 0 5) and Parent II = (5 2 6 7 0 3 4 1) with tour lengths 17 and 19, respectively. The initial node is randomly chosen to be 3,*

$$(3 * * * * * * *).$$

The successors of 3 are 0 and 4 from Parent I and Parent II, respectively. Since $d_{3,4} < d_{3,0}$, the next node is selected as 4,

$$(3\ 4 * * * * * *).$$

Continuing this way, the offspring is obtained as

$$(3\ 4\ 1\ 2\ 6\ 7\ 0\ 5)$$

with tour length 16, with better fitness value compared to its parents.

Mutation: Two different mutation operators are used.

- *Mutation I* simply randomly selects two nodes and swaps their positions.
- *Mutation II* is an improvement of *Mutation I*, in which one of the nodes to be swapped (let's say $A^{(i)}$), is selected when $d_{A^{(i-1)},A^{(i)}} + d_{A^{(i)},A^{(i+1)}} > 2$, which means that

  $$(\ \ldots\ A^{(i-1)}\ A^{(i)}\ A^{(i+1)} \ldots\ )$$

  part of the member should be updated in order the member to be optimal. Second node to be swapped is selected randomly. The swap operation is applied to the member, if it results in an improvement int the fitness value.

Selection and Replacement: At each iteration, N_p couples are selected randomly, and from each couple, one offspring is produced. The new offsprings are added to the population and the population size doubles. Then, population members are sorted based on the fitness values. The best N_p members are moved to the next generation.

Fitness function and Stopping Condition: The fitness value for the problem is the tour length, for which the optimal value is 2^n. Since the optimal solution for TSP_n^* is known unlike other random TSP instances, the GA stops whenever the optimal solution is found. Limiting the iteration number to 500 is used as an alternative stopping condition, to stop the GA whenever the population converges to a non-optimal solution.

5. EXPERIMENTAL RESULTS

In this section, we provide the details of our experimental results. For $n = 3, \dots, 10$, we repeat our experiments 100 times for each choice of population size, initial population type and mutation operator. The parameters of our experiments are summarized in Table 1.

Population size, N_p	2^5
	2^8
Initial Population	Method I - Random
	Method II - Heuristic
Crossover	NNX
Mutation	No Mutation
	Mutation I
	Mutation II
Mutation rate	Fixed to 0.05

Table 1: Parameter selection of the experiments for $n = 3, \dots, 10$

Table 2 and 3 summarizes the results of our experiments for $n = 3, \dots, 10$ using the success rate (out of 100 trials), average number of iterations for successful trials and the number of distinct de Bruijn sequences in all trials. As shown in Table 2, whenever the initial population is generated randomly, the algorithm can be considered to be successful for $n \leq 5$ and $n \leq 8$ for population size 2^5 and 2^8, respectively. Using the second method, the success rate of the genetic algorithm is greater than 90 percent for all $n \leq 10$. For small n values, even the initial population generation method managed to generate optimal solution, however as n gets larger, the heuristic method seems to fall short. The GA manages to generate optimal solutions after small number of iterations, usually less than 10 iterations on the average.

For larger n values, we repeat our experiments with $N_p = 2^5$ and 10 trials. Since the mutation operators do not have a significant effect on the results, mutation operator is not used for larger problems. The results are summarized in Table 4.

6. DISCUSSION AND CONCLUSION

In this study, we propose a new evolutionary construction method for de Bruijn sequences that is not limited to a subset of sequences. The method models the de Bruijn sequences as traveling salesman tours, which corresponds to Hamiltonian tours in Good's graph.

After some experiments, the algorithm managed to generate many different sequences for small values of n. For $n > 14$, the algorithm becomes inefficient mainly due to the memory requirement. The proposed GA requires n, $n2^n$ and $n2^nN_p$ bits to represent each node, member and the population, respectively. More efficient representations will be studied as future work.

7. ACKNOWLEDGMENTS

The author would like to thank Çağdaş Çalik for his help while implementing the code.

8. REFERENCES

[1] A. Alhakim. A simple combinatorial algorithm for de bruijn sequences. Retrieved April 30, 2009 from http://people.clarkson.edu/ aalhakim/Mypapers/deBruijn.pdf.

[2] Çağdaş Çalik, M. Sönmez-Turan, and F. Özbudak. On feedback functions of maximum length nonlinear feedback shift registers. *IEICE Transactions*, 93-A(6):1226–1231, 2010.

[3] T. Etzion and A. Lempel. Construction of de bruijn sequences of minimal complexity. *IEEE Transactions on Information Theory*, 30(5):705–708, 1984.

[4] H. Fredricksen. A survey of full length nonlinear shift register cycle algorithms. 24(2):195–221, 1982.

[5] H. Fredricksen and J. Maiorana. Necklaces of beads in k colors and k-ary de bruijn sequences. *Discrete Mathematics*, 23:207–210, 1978.

[6] R. A. Games. A generalized recursive construction for de bruijn sequences. *IEEE Transactions on Information Theory*, 29(6):843–849, 1983.

[7] D. E. Goldberg and R. Lingle. Alleles, Loci and the TSP. In *In Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159. (Edited by Grefenstette, J.J.), Lawrence Erlbaum, Hillsdale, NJ, 1985.

[8] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981.

[9] R. Gonzalo, D. Ferrero, and M. Soriano. Some properties of non linear feedback shift registers with maximum period. *Proc. Sixth Int. Conf. Telecommunications Systems*, 1998.

[10] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[11] S. Jung and B. R. Moon. Toward Minimal Restriction of Genetic Encoding and Crossovers for the Two-Dimensional Euclidean TSP. *IEEE Transactions on Evolutionary Computation*, 6(6):557–565, 2002.

[12] P. Merz. A Comparison of Memetic Recombination Operators for the Traveling Salesman Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 472–479. Morgan Kaufmann, San Francisco, 2002.

[13] Y. Nagata and S. Kobayashi. Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 450–457. Morgan Kaufmann, San Mateo, 1997.

[14] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 224–230, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[15] A. Ralston. De bruijn sequences-a model example of the interaction of discrete mathematics and computer science. *Mathematics Magazine*, 55(3):pp. 131–143,

		No Mutation			Mutation I			Mutation II		
n	N_p	Success Rate	Avr. Iter.	Distinct	Success Rate	Avr. Iter.	Distinct	Success	Avr. Iter.	Distinct
3	2^5	91	2.16	2	100	3.28	2	100	1.17	2
4	2^5	77	16.35	16	88	22.46	16	98	22.52	16
5	2^5	51	36.64	51	58	59.08	59	64	43.98	68
6	2^5	0	-	0	0	-	0	0	-	0
3	2^8	100	0.02	2	100	0.47	2	100	0	2
4	2^8	97	6.46	16	97	3.23	16	99	5.16	16
5	2^8	96	8.56	143	99	9.64	132	97	11.07	137
6	2^8	88	22.92	119	88	22.82	111	89	27.46	130
7	2^8	79	39.82	92	73	41.15	87	77	51.23	91
8	2^8	45	116	49	31	95.46	35	33	99.63	36
9	2^8	0	-	0	0	-	0	0	-	0

Table 2: The summary of results obtained using a random initial population

		No Mutation			Mutation I			Mutation II		
n	N_p	Success Rate	Avr. Iter.	Distinct	Success Rate	Avr. Iter.	Distinct	Success	Avr. Iter.	Distinct
3	2^5	100	0	2	100	0	2	100	0	2
4	2^5	100	0	16	100	0.06	16	100	0.02	16
5	2^5	100	0.54	199	100	0.61	194	100	0.42	219
6	2^5	100	2.81	155	100	2.69	128	100	2.9	131
7	2^5	100	6.8	113	100	7.74	117	100	6.96	111
8	2^5	100	14.39	103	100	16.83	107	99	16.67	105
9	2^5	100	32.4	100	96	34.19	98	99	27.32	99
10	2^5	97	59.27	97	95	63.84	95	90	60.26	90
3	2^8	100	0	2	100	0	2	100	0	2
4	2^8	100	0	16	100	0	16	100	0	16
5	2^8	100	0	1190	100	0	1191	100	0	1163
6	2^8	100	0	881	100	0	867	100	0	901
7	2^8	100	0.07	208	100	0.04	428	100	0.05	420
8	2^8	100	0.85	205	100	1.19	222	100	1.06	203
9	2^8	100	4.32	129	100	4.21	144	100	3.37	142
10	2^8	100	7.98	110	100	8.49	117	100	9.63	110

Table 3: The summary of results obtained using a non-random initial population

n	Success rate	Avr. Iter.	Distinct
11	8 /10	136.00	8
12	5 /10	118.40	5
13	2 /10	152.00	2
14	3 /10	262.33	3

Table 4: The summary of results obtained for larger n values

1982.

[16] H. Süral, N. E. Özdemirel, I. Önder, and M. S. Turan. An evolutionary approach for the tsp and the tsp with backhauls. In L. M. Hiot, Y. S. Ong, Y. Tenne, and C.-K. Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, volume 2 of *Adaptation, Learning, and Optimization*, pages 371–396. Springer Berlin Heidelberg, 2010.