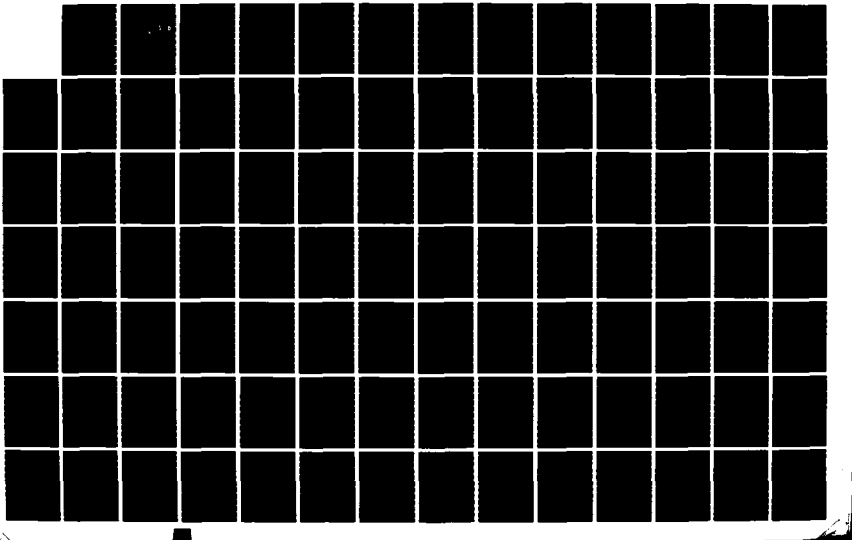
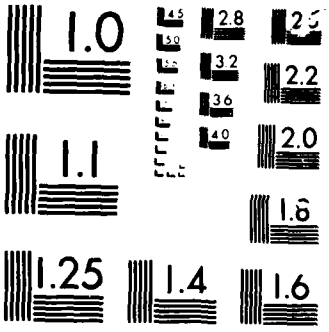
AD-A168 474 COVERING THE DE BRUIJN GRAPH(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R D BRYANT MAR 86 1/2

UNCLASSIFIED F/G 12/1 NL





MICROCOPY CHART

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

COVERING THE de BRUIJN GRAPH

BY

Roy Dale Bryant

March 1986

Thesis Advisor: Harold Fredricksen

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 53	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
COVERING THE de BRUIJN GRAPH

12 PERSONAL AUTHOR(S)
Bryant, Roy Dale

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 March	103

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Cover, de Bruijn Cycle, de Bruijn Graph, Feedback Function, Matching, Shift Register

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Random-like sequences of 0's and 1's are generated efficiently by binary shift registers. The output of n-stage shift registers viewed as a sequence of binary n-tuples also give rise to a special graph called the de Bruijn graph B_n. The de Bruijn graph is a directed graph with 2^n nodes. Each node has 2 arcs entering it and 2 arcs going out of it. Thus, there are a total of 2^{n+1} arcs in B_n.

In this thesis, we define a cover of the de Bruijn graph, different from the usual graph theoretic cover. A cover S of the de Bruijn graph is defined as an independent subset of the nodes of B_n that satisfy the following property. For each node x in B_n - S, there exists a node y in S such that either the arc $\langle x,y \rangle$ or the arc $\langle y,x \rangle$ is in B_n.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Harold Fredricksen	22b TELEPHONE (Include Area Code) (408) 646-2235	22c OFFICE SYMBOL 53Fs

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Combinatorially, we are able to place both upper and lower bounds on the cardinality of S. We find examples of covers that approach these bounds in cardinality. Several algorithms are presented that produce either a maximal or a minimal cover. Among them are Frugal, Sequential Fill, Double and Redouble, Greedy and Quartering.

Approved for public release; distribution is unlimited.

Covering the de Bruijn Graph

by

Roy D. Bryant
Captain, United States Marine Corps
B.S., Kansas State College of Pittsburg, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
MARCH 1986

Author: Roy D Bryant
Roy D. Bryant

Approved by: Harold I. Fredricksen
Harold M. Fredricksen, Thesis Advisor

Carroll O. Wilde
Carroll O. Wilde, Second Reader

Gordon E. Latta
Gordon E. Latta, Chairman, Department of Mathematics

Kneale T. Marshall
Kneale T. Marshall, Dean of
Information and Policy Sciences

ABSTRACT

Random-like sequences of Ø's and 1's are generated efficiently by binary shift registers. The output of n-stage shift registers viewed as a sequence of binary n-tuples also give rise to a special graph called the de Bruijn graph B_n. The de Bruijn graph is a directed graph with 2^n nodes. Each node has 2 arcs entering it and 2 arcs going out of it. Thus, there are a total of 2^{n+1} arcs in B_n.

In this thesis, we define a cover of the de Bruijn graph, different from the usual graph theoretic cover. A cover S of the de Bruijn graph is defined as an independent subset of the nodes of B_n that satisfy the following property. For each node x in B_n - S, there exists a node y in S such that either the arc $\langle x,y \rangle$ or the arc $\langle y,x \rangle$ is in B_n.

Combinatorially, we are able to place both upper and lower bounds on the cardinality of S. We find examples of covers that approach these bounds in cardinality. Several algorithms are presented that produce either a maximal or a minimal cover. Among them are Frugal, Sequential Fill, Double and Redouble, Greedy and Quartering.

TABLE OF CONTENTS

I. PRELIMINARIES ---------- 9

A. INTRODUCTION ---------- 9

B. BINARY SHIFT REGISTERS ---------- 10

C. de BRUIJN GRAPH PROPERTIES ---------- 13

II. COVERINGS ---------- 18

A. DECOMPOSITION OF THE de BRUIJN GRAPH -------- 18

B. REMAINING PATHS ---------- 19

C. COVERING SETS ---------- 19

D. LOWER BOUND ---------- 21

E. UPPER BOUND ---------- 22

F. EXHAUSTION ---------- 23

III. MAXIMAL COVERINGS ---------- 29

A. INTRODUCTION ---------- 29

B. FRUGAL ---------- 29

C. SEQUENTIAL FILL ---------- 32

1. n Even ---------- 32

2. n Odd ---------- 36

D. 3-COLORING ---------- 46

E. DOUBLING ---------- 50

1. Successors in B_{n+1} ---------- 51

2. Companion Pairs ---------- 56

F. DOUBLING SCHEMES ---------- 61

1. Double and Redouble ---------- 62

a. n Even ---------- 62
b. n Odd ---------- 68
2. Double and Cover ---------- 72
a. n Odd ---------- 72
b. n Even ---------- 78
G. BUILD-UP ---------- 80
H. CONCLUSIONS ON MAXIMAL COVERINGS ---------- 87
IV. MINIMAL COVERINGS ---------- 91
A. INTRODUCTION ---------- 91
B. GREEDY ---------- 91
C. QUARTERING ---------- 93
D. CONCLUSIONS ON MINIMAL COVERINGS ---------- 97
V. FURTHER STUDIES ---------- 98
LIST OF REFERENCES ---------- 101
INITIAL DISTRIBUTION LIST ---------- 102

LIST OF TABLES

II.1 EXHAUSTIVE SEARCH ---------- 28

III.1 FRUGAL ALGORITHM ---------- 31

III.2 BUILD-UP APPLIED TO FRUGAL ALGORITHM ---------- 85

III.3 BUILD-UP APPLIED TO SEQUENTIAL FILL ALGORITHM ---- 85

III.4 BUILD-UP APPLIED TO 3-COLOR ALGORITHM ---------- 86

III.5 BUILD-UP APPLIED TO DOUBLE & REDOUBLE ALGORITHM -- 86

III.6 BUILD-UP APPLIED TO DOUBLE & COVER ALGORITHM ----- 87

III.7 RESULTS OF BASIC MAXIMAL ALGORITHMS ---------- 88

III.8 BUILD-UP ALGORITHM APPLIED TO BASIC MAXIMAL ALGORITHMS ---------- 89

III.9 COST COMPARISON OF MAXIMAL ALGORITHMS ---------- 90

IV.1 GREEDY ALGORITHM ---------- 93

IV.2 RESULTS OF MINIMAL ALGORITHMS ---------- 97

LIST OF FIGURES

I.1 The n-Stage Shift Register with Feedback Function $f(x_1, x_2, \ldots, x_n)$ ---------- 11

I.2 Adjacency Quadruple ---------- 12

I.3 de Bruijn Graph for n = 3,4 ---------- 12

I.4 Pure Cycles ---------- 15

I.5 The de Bruijn Cycle (00010111) ---------- 16

I.6 de Bruijn Graph for n = 3,4 (Decimal Representation) ---------- 17

II.1 Maximal Independent Sets (a) {1,6} and (b) {2,3} in B_3 ---------- 20

I. PRELIMINARIES

A. INTRODUCTION

In todays world of rapidily advancing technology, we rely more and more on high speed computers, electronic communication equipment, etc.. Data streams are handled in these devices as a series of electronic pulses and non-pulses. Numerically, we view such data as a sequence of 1's and 0's, associating a 1 with each pulse and a 0 with each non-pulse.

With this technology, there arises a need to generate random-like sequences of 1's and 0's. Such sequences are useful in the areas of coding and communication as well as in mathematical modeling. One of the simplest and most efficient ways of generating such a sequence is with a binary shift register.

The output of these shift registers give rise to a special graph called a de Bruijn graph. Knowledge about the structure of the de Bruijn graph has proven useful in generating and analyzing the corresponding sequences of 1's and 0's.

Extending the existing theory of de Bruijn graphs, we define a cover of the de Bruijn graph and examine properties of such covers. Our main emphasis is on the cardinality of these covers. Combinatorially, we are able to place upper

and lower bounds on the cardinality of these covers. We then search for methods to form covers that approach these bounds.

B. BINARY SHIFT REGISTERS

A binary shift register of span n is a collection of n storage devices $x_1, x_2, \ldots, x_n$ each capable of holding either a Ø or a 1 and a function $f(x_1, x_2, \ldots, x_n)$, taking on values of Ø or 1, computed from the contents of the storage devices. At the pulse of an external clock, the contents of register x_{i+1} is shifted to the register x_i for i = 1, 2,..., n-1. At the same time, the value of $f(x_1, x_2, \ldots, x_n)$ is fed into the register x_n. The output of the shift register can be taken to be the sequence of bits that appears in any of the n stages of the register or it can be taken to be the whole contents of the n stages. Displayed in Fig. I.1 is a general n-stage shift register. The dependence on the clock has been suppressed.

The contents of the shift register (regarded as either an integer or a binary n-tuple) is called its state. Since each stage can be either a Ø or a 1, there are 2^n possible states for a register of n stages. Define V_n as the space of these binary n-tuples. With every pulse of the clock, a transition is made from one state to the next. Thus, the feedback function induces a mapping F from V_n to V_n.

In this mapping, the n-tuple $(x_1,x_2,\ldots,x_n)$ is mapped to $(x_2,x_3,\ldots,x_n,f(x_1,x_2,\ldots,x_n))$. Since the value of the function f is either a 1 or a 0, the state $(x_1,x_2,\ \ldots\ ,x_n)$

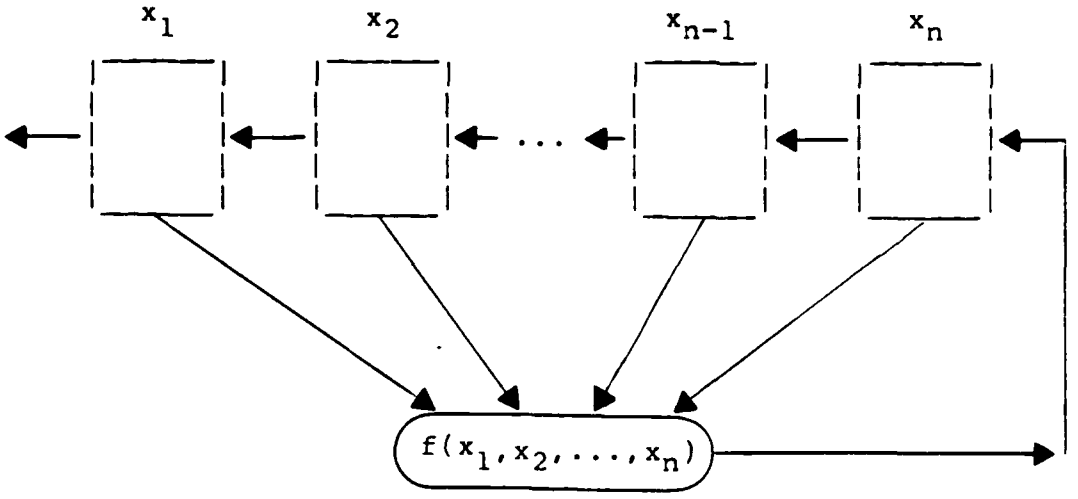


Fig. I.1 The n-stage Shift Register with Feedback Function, $f(x_1,x_2,\ldots,x_n)$

has two possible successor states, $(x_2,\ldots,x_n,0)$ and $(x_2,\ldots,x_n,1)$. We call these states conjugate states. Note that conjugate states agree in all but the last bit. In the same manner, the state $(x_1,x_2,\ldots,x_n)$ has two possible predecessor states, $(0,x_1,\ldots,x_{n-1})$ and $(1,x_1,\ldots,x_{n-1})$. We call these states companion states. Companion states agree in all bits except the first. Thus, each (n-1)-tuple forms an adjacency quadruple as seen in Fig. I.2, where a pair of companion states maps to a pair of conjugate states. The superposition of all 2^{n-1} adjacency quadruples determines a directed graph. This graph is called the de Bruijn graph B_n, named for the Dutch mathematician N.G. de Bruijn who studied

these graphs in reference 1. In Fig. I.3, we give the de Bruijn graph corresponding to n = 3 and 4.

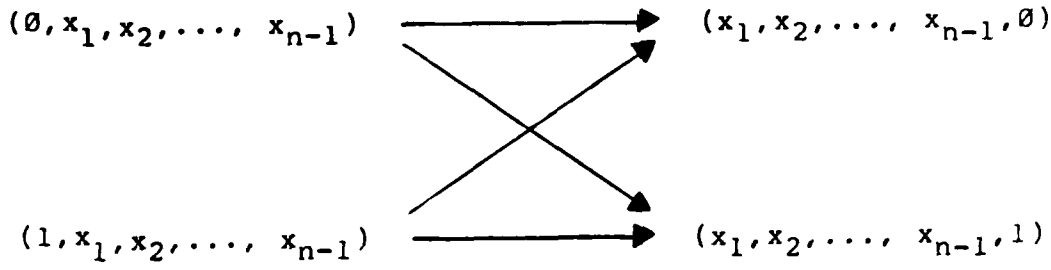


Fig. I.2 Adjacency Quadruple

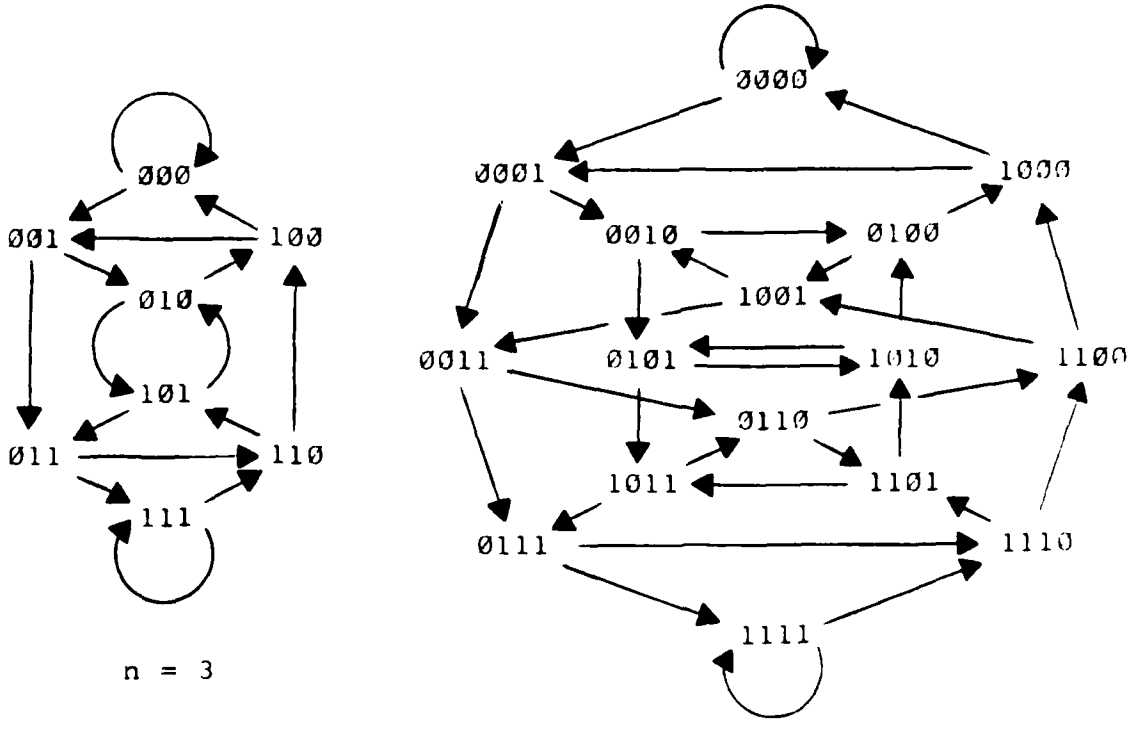


Fig. I.3 de Bruijn Graph for n = 3,4.

C. de BRUIJN GRAPH PROPERTIES

In the de Bruijn graph, each possible state (viewed as n-tuples) of the shift register is represented by a node. Thus, there are 2^n nodes in the de Bruijn graph of order n. Each possible transition from one state to another is represented by an arc connecting those two nodes. Since each state can map to one of two successor states, each node has two arcs emanating from it. These two arcs lead to conjugate nodes. Thus, there are 2^{n+1} arcs in the de Bruijn graph of order n. Each node also has two arcs coming into it. These arcs come from companion nodes. Viewing these n-tuples as n bit binary numbers, we see that the two nodes that are successors of a node x, where $x = \sum_{i=1}^{n} x_i 2^{n-i}$, are the nodes $2x \pmod{2^n}$ and $2x + 1 \pmod{2^n}$. Whether the node x is mapped to the node $2x \pmod{2^n}$ or the node $2x + 1 \pmod{2^n}$ is determined by the feedback function. The two predecessors of the node x are the nodes $\lfloor x/2 \rfloor$ and $\lfloor x/2 \rfloor + 2^{n-1}$, where $\lfloor y \rfloor$ is the greatest integer less than or equal to y. Likewise, the node that feeds into node x is determined by the feedback function.

A <u>path of length k</u> in B_n is a collection of nodes $n_1, n_2, \ldots, n_{k+1}$ such that an arc connecting n_i to n_{i+1} for $i = 1,2,\ldots,k$ exists in B_n. The path is called <u>simple</u> if no node appears on it more than once. A <u>cycle of length k</u> is a path of length k in which $n_1 = n_{k+1}$. A cycle is <u>simple</u> if no

node appears on it more than once before the start node is repeated.

Since a cycle of length k will repeat itself every k nodes, the output of the shift register will repeat itself every k bits. Thus, a cycle of length k can be described by the first k bits (or any k bits) of the shift register. If $k < n$, we repeat the k bits enough times to fill the n-long register.

The 3-cycle 00 --> 01 --> 10 --> 00 in B_2 can be represented by (001). The 3-cycle 01 --> 11 --> 10 --> 01 in B_2 can be represented by (011). These cycles, (001) and (011), appear in the graph B_n for all $n \geq 2$. However, their integer decimal representations change according as n, the length of the register, changes. Since the cyclic shifts of 001 and 011 represent all of the possible ways to write a binary 3-tuple, (001) and (011) are the only 3-cycles in B_n.

A set of disjoint cycles that include every node in B_n is called a <u>factor</u> of B_n. In a factor, no nodes or arcs are repeated. One such factor is defined by the mapping $F:(x_1,x_2,\ldots,x_n) \dashrightarrow (x_2,\ldots,x_n,x_1)$. This mapping is called the pure cycle mapping and the associated shift register is called the Pure Cycle Register (PCR). Golomb has shown that the pure cycle mapping decomposes the graph into Z(n) disjoint cycles where

$$Z(n) = 1/n \sum \emptyset(d)2^{n/d}$$

[Ref. 2:pp. 118-121]. The summation is over all divisors d of n and Ø(d) is Euler's phi-function.

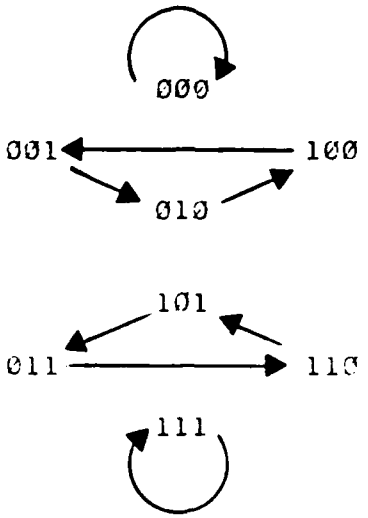


Fig. I.4 Pure Cycles (n = 3).

In a general graph, a path which goes through every node once and only once is known as a Hamiltonian path. We can always extend a Hamiltonian path in a de Bruijn graph so that the last node in the path connects with the first node, creating a cycle. This cycle is known as a de Bruijn cycle. Since a de Bruijn cycle visits all the nodes in B_n only once, it is a factor consisting of only a single cycle. It can be shown that there are 2^x such cycles in B_n, where $x = 2^{(n-1)} - n$, [Ref. 2]. One such cycle in B_3 is (00010111). The cycle can also be written as the cycle 000 --> 001 --> 010--> 101 --> 011 --> 111 --> 110 --> 100 --> 000, which can be seen in Fig. I.5.

An Eulerian path in a graph is a path which traverses each arc exactly once. If the arc connecting the nodes

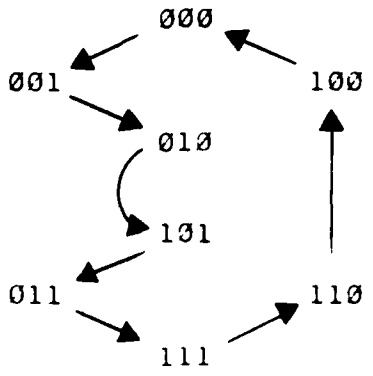


Fig. I.5 The de Bruijn Cycle (00010111).

$(x_1,x_2,\ldots,x_n)$ and $(x_2,x_3,\ldots,x_{n+1})$ in B_n is labeled with the (n+1)-tuple $(x_1,x_2,\ldots,x_n,x_{n+1})$, then the arcs in B_n correspond to the nodes in B_{n+1}. Then two nodes in B_{n+1} are connected by an arc if their corresponding arcs in B_n are such that the first arc enters a node in B_n while the second arc exits that same node. Thus, an Eulerian path in B_n defines a Hamiltonian path in B_{n+1}. As the de Bruijn graph is connected and has 2 arcs in and out of each node, there is always an Eulerian path in B_n [Ref. 3], hence a Hamiltonian path in B_{n+1}.

Two isomorphisms exist in the de Bruijn graph in which the structure of the graph is preserved. They are the binary reverse and the binary complement mappings. Under the reverse mapping, each node $(x_1,x_2,\ldots,x_n)$ is mapped to the

node $(x_n,\ldots,x_2,x_1)$. As an example, the node 001 in B_3 is mapped to the node 100. In the complementary mapping, all 0's are changed to 1's and 1's to 0's. So, the node 001 is mapped to the node 110.

For convenience, we will typically view the nodes in B_n in their decimal representation. Thus, the nodes will be numbered 0, 1, 2,..., $2^n - 1$. When more useful, we will recall the binary structure that fostered this graph.

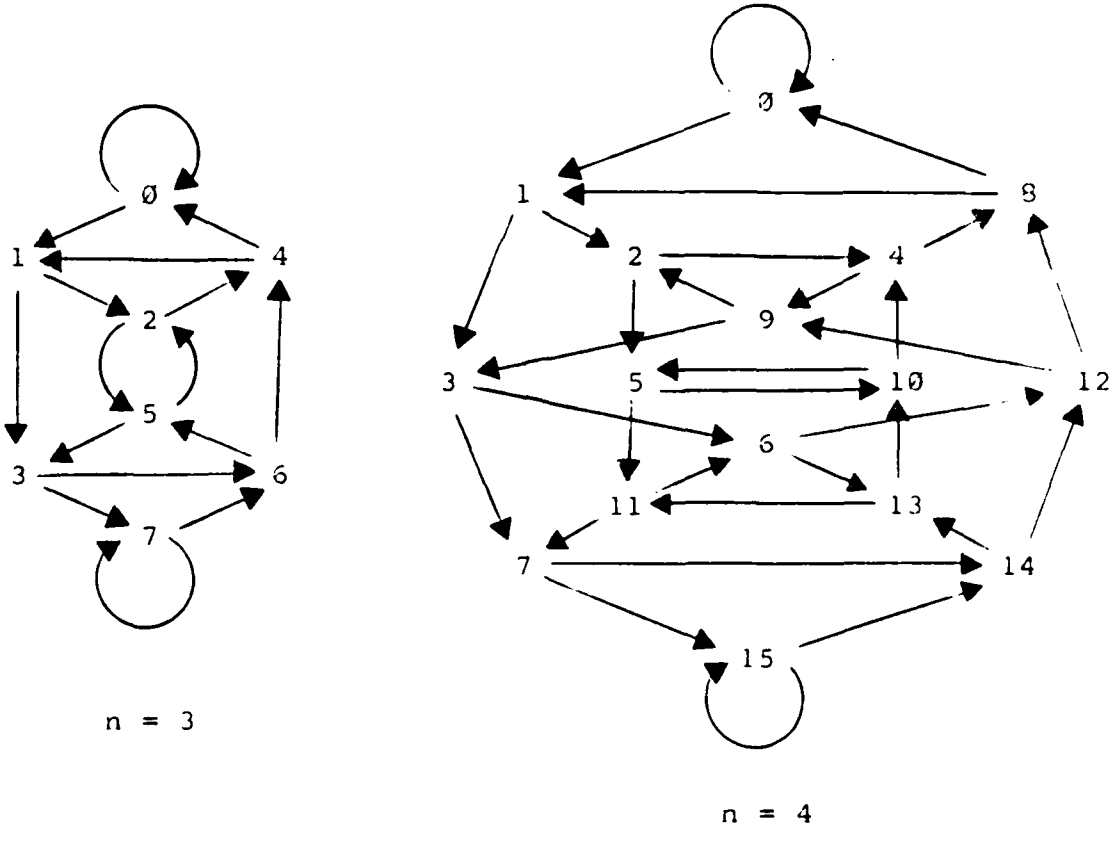


Fig. I.6 de Bruijn Graph for n = 3,4
(Decimal Representation).

II. COVERINGS

A. DECOMPOSITION OF THE de BRUIJN GRAPH

A problem that has been considered for the de Bruijn graph B_n is the maximum number of cycles into which the graph can be decomposed. Clearly, this number must be at least Z(n). In section I.C we saw that the pure cycle mapping formed a factor in the de Bruijn graph consisting of Z(n) cycles. A long standing conjecture by Golomb was that the maximum number of cycles into which B_n can be decomposed is Z(n) [Ref. 2:p. 174].

In trying to prove Golomb's conjecture, Lempel made a conjecture of his own which implied Golomb's [Ref. 4]. Lempel's conjecture was that the minimum number of nodes, which if removed from B_n that will result in an acyclic graph, is Z(n).

It is clear that at least Z(n) nodes would be required to solve Lempels conjecture. There would have to be one node lying on each of the Z(n) cycles from the pure cycle mapping. The remaining problem was to find a set of Z(n) nodes for each n that, when removed from B_n, would leave the graph acyclic. Mykkeltveit gave a constructive proof of Lempel's conjecture by providing an algorithm to find those nodes [Ref. 5]. This in turn proved Golomb's conjecture.

B. REMAINING PATHS

Once the de Bruijn graph has been decomposed in the above manner, there is some interest in the structure of the remaining tree. In particular, knowledge about the lengths of the remaining paths is desirable. Long paths in the decomposed de Bruijn graph have cryptographic application. Because of their acyclic nature, these paths can provide a cryptographic key used for encoding.

However, there are other applications of a hash coding flavor in which long paths in the decomposed graph are not desirable. For these applications we could even remove a few more nodes from the de Bruijn graph so that the remaining tree contains no unsuitably long paths. This process of removing more nodes can be continued, resulting in a forest whose maximum height tree is as short as desired.

C. COVERING SETS

Taking this deletion process to its ultimate extreme, we pose the question: "How many nodes must be deleted from B_n so that there are only paths of length 0 remaining?". In other words, we want to remove enough nodes so that no two adjacent nodes remain in the decomposed graph. Futher, the set of nodes remaining should be big enough so that no node can be added without creating a path.

By deleting only those nodes necessary so that no path exists, the remaining nodes form a maximal independent set.

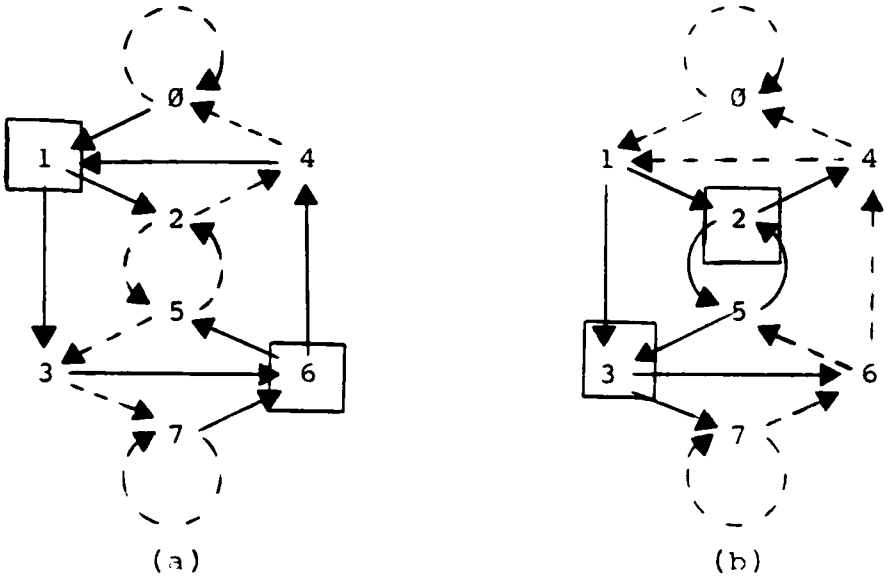


Fig. II.1 Maximal Independent Sets
(a) {1,6} and (b) {2,3} in B_3.

The set is <u>independent</u> in that if a node x is an element of the set, then neither the predecessors nor the successors of x are in the set. The set is <u>maximal</u> in the sense that no other node can be added to the set without creating a path.

In Fig. II.1a, we see that if all nodes except nodes 1 and 6 are removed from B_3, we are left with a maximal independent set. Any other node in B_3 is either a predecesor or a successor of either 1 or 6 and is therefore excluded from the set.

In Fig. II.1b, the nodes 2 and 3 form a maximal independent set in B_3. Any other node except Ø is adjacent to either 2 or 3 and cannot be in the set. The node Ø cannot be added to the set either because, if it is left in the graph, a path to itself will be created. The node 2^n-1,

which in B_3 is the node labeled 7, will also allow a path to itself.

A subset S of the nodes of B_n is defined to be a cover of B_n if S is a maximal independent set and for every node x in B_n - S, there exists a node y in S such that either an arc $\langle x,y \rangle$ or an arc $\langle y,x \rangle$ is in B_n.

As mentioned before, special consideration needs to be given to the nodes Ø and 2^n-1. Because a path exists from Ø to Ø and from 2^n-1 to 2^n-1, neither of these nodes can be an element in a cover. Yet we need to ensure that these nodes are themselves covered. For the node Ø to be covered, either 1 or 2^{n-1} must be in the cover. Likewise, for the node 2^n-1 to be covered either $2^{n-1}-1$ or 2^n-2 must be in the cover.

The set {1,6} forms a cover of B_3, while the set {2,3} does not as it fails to cover the node Ø.

It should be noted that this definition of a node cover is not the normal graph theoretic one in which nodes cover their incident arcs.

D. LOWER BOUND

When considering covering sets, the question of cardinality arises. If the nodes in the cover are chosen carefully, how large a covering set can be obtained and, similarly, how small a set can be obtained which still satisfies the requirements of being a cover?

If a node x is in a covering set S, then the four nodes adjacent to x are considered to be covered by x. If we can

form a cover for B_n in which each node in B_n - S, the relative complement of S in B_n, is covered by only a single node in S, then we will have formed a covering set S of minimum cardinality. In this case, each node in S covers a total of 5 nodes - itself and the 4 nodes adjacent to it. Thus a lower bound for the number of nodes in a cover is given by $\lceil 2^n/5 \rceil$.

E. UPPER BOUND

As noted in section I.C, a de Bruijn cycle is a Hamiltonian path through the nodes of B_n. In choosing nodes for a cover for B_n, we see that no two adjacent nodes in a de Bruijn cycle can be in the cover. Thus the nodes in a cover can be no closer on a de Bruijn cycle than every other node. This leads us to an upper bound of $(2^n)/2$ for the maximum cardinality of a cover S.

However, this upper bound on S cannot be achieved. When the de Bruijn cycle passes through the node Ø it must do so in the order ..., $2^{(n-1)}$, Ø, 1,... . The nodes $2^{(n-1)}$ and 1 are separated by a node in the de Bruijn cycle yet only one of them can be in a cover because they are also adjacent in the graph. A similar situation exists with the nodes $2^{(n-1)}-1$, 2^n-1, and 2^n-2. Thus, the upper bound can be reduced by a small amount to something less than half of the nodes of the graph.

F. EXHAUSTION

If we consider nodes in the order of a de Bruijn cycle and form a set of nodes by choosing those nodes from the ordered set that do not create a path with any other selected node, the resulting set is a maximal independent set. Each node in sequence will either be covered or can be added to the set of previously chosen nodes. If the nodes 0 and 2^n-1 are also covered, the set is a cover.

One way to reduce the work of checking that the set is a cover is to always start the de Bruijn cycle at the node 1. This will force the node 1 to be in the potential cover, ensuring that 0 is covered. No loss of generality results in choosing 1 to cover 0 because, as noted in section II.C, either the node 1 or 2^{n-1} must be in the cover. For any cover containing the node 2^{n-1}, the set consisting of the binary reverse of those nodes also forms a cover containing the node 1.

For n = 4, and the de Bruijn cycle 1, 3, 6, 13, 10, 4, 9, 2, 5, 11, 7, 15, 14, 12, 8, 0, if we follow the procedure described above, we obtain the maximal independent set {1, 6, 10, 9, 7}. Since 7 is in the set, the node 15 is covered and the set is therefore a cover.

However, if we had not chosen 6 but chosen 13 instead, we would have obtained the cover {1, 13, 4, 5, 7, 12}, of greater cardinality than that of the previous cover.

To examine characteristics of the different covers of the de Bruijn graph, the following backtracking scheme can be used to find all covering sets of the graph for a given value of n.

Algorithm: Exhaustion on Covers

Step 1: Starting with node 1 on a de Bruijn cycle, form a maximal independent set by selecting the first node in cyclic order that does not create a path with any other selected nodes. When the process is repeated, if the node $2^n - 1$ is also covered, this set is a cover.

Step 2: If the cardinality of the set is greater than one, remove the last node brought into the set. If not, stop.

Step 3: Consider the nodes remaining in the de Bruijn cycle, beyond the node just removed from the independent set. Add nodes to the set, in order, that do not form a path to previously selected nodes. If all nodes in the graph are covered, this set is a cover.

Step 4: Go to step 2.

We consider again the example n = 4, starting with the de Bruijn cycle 1, 3, 6, 13, 10, 4, 9, 2, 5, 11, 7, 15, 14, 12, 8, 0. In step 1, we would obtain the set {1, 6, 10, 9, 7}, which is a cover. Remove node 7 from the set as in

step 2. Then in step 3, we would consider the nodes 15, 14, 12, 8 and Ø to be added to the set {1, 6, 1Ø, 9}. The node 14 can be added to form the set {1, 6, 1Ø, 9, 14}. This set is a cover.

Going back to step 2, the node 14 is removed and the nodes 12, 8, and Ø are considered. None of these nodes can be added to the set without creating a path, so we are left with the independent set {1, 6, 1Ø, 9}. This set is not a cover.

The node 9 is then removed and the nodes 2, 5, 11, 7, 15, 14, 12, 8 and Ø are considered to be added to the set {1, 6, 1Ø}. This process continues until only the node 1 remains in the set. With the knowledge that the lower bound says that the cover must contain at least 4 elements for n = 4, the process could have been stopped sooner. However, the extra checking required to see if the stopping criteria had been met is more expensive than allowing the program to run to completion.

In determining the cost of the algorithm, it is noted that each node in the graph, except the four nodes adjacent to node 1, will each be chosen at some time to be the second node in the set. When the jth node in the de Bruijn cycle is chosen to be the second node in the set, the cost to fill in the remainder of the independent set is only slightly more than the cost of performing the algorithm on $V - j + 1$ nodes, where $V = 2^n$, the total number of nodes in the graph.

Since each node, starting with the third node in the de Bruijn cycle, is ultimately chosen to be the second node in the set, we can approximate the cost of exhaustively searching a graph containing V nodes by the recursion

$$f(V) = f(V-2) + f(V-3) + \ldots + f(2) + f(1).$$

Here f(x) is the cost of performing the algorithm on x nodes. Note that the term f(V-1) is not present in the expression. This is because once the first node in the de Bruijn cycle has been placed in the set, the second node in the cycle is eliminated from consideration as it is adjacent to the first node in the cycle. The cost function applies to every term in the sequence so, we could also write the expression

$$f(V-1) = f(V-3) + f(V-4) + \ldots + f(2) + f(1)$$

etc.. We introduce the constants $a_0 = 0$ and $a_1 = 1$ and add the trivial term $a_0 f(V-1)$ to f(V). This gives us

$$f(V) = a_0 f(v-1) + a_1 f(V-2) + a_1 f(V-3) + \ldots + a_1 f(2) + a_1 f(1).$$

If we substitute the equation for f(V-1) into the expression for f(V), we obtain

$$f(V) = a_1 f(V-2) + (a_0 + a_1) f(V-3) + (a_0 + a_1) f(V-4) + \ldots + (a_0 + a_1) f(2) + (a_0 + a_1) f(1).$$

By letting $a_2 = a_0 + a_1$, we can write

$$f(V) = a_1 f(V-2) + a_2 f(V-3) + a_2 f(V-4) + \ldots + a_2 f(2) + a_2 f(1).$$

A similar substitution of f(V-2) into f(V) yields

$$f(V) = a_2 f(V-3) + (a_1 + a_2) f(V-4) + (a_1 + a_2) f(V-5) + \ldots + (a_1 + a_2) f(2) + (a_1 + a_2) f(1).$$

Letting $a_3 = a_1 + a_2$, we can then write

$$f(V) = a_2 f(V-3) + a_3 f(V-4) + a_3 f(V-5) + \ldots + a_3 f(2) + a_3 f(1).$$

Continuing to make similar substitutions and defining additional constants, we obtain the equation

$$f(V) = a_{k-1} f(V-k) + a_k f(V-k-1) + a_k f(V-k-2) + \ldots + a_k f(2) + a_k f(1).$$

after substituting for $f(V-k+1)$. Here a_k is defined to be $a_{k-2} + a_{k-1}$. Continuing with similar substitutions, ultimately yields the equation

$$f(V) = a_{V-3} f(2) + a_{V-2} f(1).$$

In examining the sequence of a_k's, we see that the recursive definition and initial conditions are exactly those of the Fibonacci sequence. Thus, a closed form expression for a_k is given by

$$a_k = [((1 + \sqrt{5})/2)^k - ((1 - \sqrt{5})/2)^k]/\sqrt{5}.$$

Therefore, we see that the cost of the exhaustive search algorithm is at least $O(((1 + \sqrt{5})/2)^V)$ or $O(1.618^V)$ in complexity.

An exhaustive search was completed for $n \leq 5$. The results, seen in Table II.1, show that for $n = 5$, the covering set of maximum cardinality is only 75% of the upper bound. Also for $n = 5$, the cover of minimum cardinality

found is larger than the lower bound. Thus, neither the upper nor the lower bounds are met for the graph B_5.

An exhaustive search of the graph for a given n will

TABLE II.1 EXHAUSTIVE SEARCH

n	2^n	Lower Bound	Minimum Found	Maximum Found	Upper Bound
1	2	1	Ø	0	1
2	4	1	1	1	2
3	8	2	2	2	4
4	16	4	4	6	8
5	32	7	8	12	16

result in all possible covers of that graph. Because of computational constraints, a search was not done for the graphs with n > 5. If a faster computer had been used, graphs of larger order could have been searched. However, due to the exponential growth of the number of nodes in the graph, using a faster computer would have allowed the exhaustive search of graphs only a few orders larger. In the sequel we develop algorithms cheaper than exhaustive search to give approximate solutions to both the largest and smallest cardinality covers of the graph B_n.

III. MAXIMAL COVERINGS

A. INTRODUCTION

In this chapter, techniques are explored to find covering sets that are of maximum or nearly maximum cardinality. Methods to place nodes in the cover in a way that pack those nodes closely together yet does not violate the definition of a cover are examined. Also considered are ways to form a cover for B_n based on independent sets from lower order graphs.

B. FRUGAL

In choosing nodes to form a covering set, it seems reasonable to consider how many previously uncovered nodes will be covered when a new node is placed in the set. If the goal is to obtain a cover of maximum cardinality, a logical criterion for adding nodes to the set would be to choose the node adjacent to the fewest, as of yet, uncovered nodes. This idea is implemented in the algorithm Frugal as follows.

Algorithm: Frugal

Step 1. Select node 1 and either node $2^n - 2$ or node $2^{n-1} - 1$ to be in the set. (This is done to ensure that the resulting set will be a cover.)

Step 2. From the remaining uncovered nodes, add the node with the fewest uncovered neighbors. In case of a tie, add the smallest such node.

Step 3. If all nodes are covered the set is a cover, stop. If not, go to step 2.

In step 1, the number of combinations to ensure that the nodes Ø and $2^n - 1$ are covered has been reduced from 4 ways to 2 by forcing the node 1 to be in the cover. As noted in section II-F, this causes no loss in generality.

Once node 1 has been selected to be in the cover, there is no choice whether to include $2^n - 2$ or $2^{(n-1)} - 1$ in the cover for small values of n. For n = 1, no cover exists. For n = 2, $1 = 2^{(2-1)} - 1$ and node 1 forms a cover by itself. For n = 3, $3 = 2^{(3-1)} - 1$. However, node 3 is a descendent of node 1 and cannot be included, so the node $6 = 2^3 - 2$ must be added to the cover instead of node 3.

For n = 4, the cardinality of the cover is one greater if the node $2^{(4-1)} - 1 = 7$ is chosen instead of node 14. For n = 5, the cardinality of the cover is one greater if $2^5 - 2 = 30$ is chosen rather than node 15. For $6 \leq n \leq 11$, either choice of a node to cover $2^n - 1$ can be made with no difference in the cardinality of the resulting covers.

Most of the time spent running the Frugal algorithm is spent in step 2, choosing the next node to be added to the set. By using an array to keep track of the number of uncovered adjacent nodes of each node, this choice can be

made in O(V) time. Step 2 will have to be performed m times, where m is the cardinality of the covering set. Since V/5 < m < V/2, the cost of the frugal algorithm is $O(V^2)$.

This considerable gain in cost over the exhaustive search algorithm makes it posible to obtain a covering set for much larger values of n. The results of the Frugal algorithm for n < 12 can be seen in Table III.1. The upper bound has been adjusted to reflect the results of the exhaustive search done for n < 6. Note that the cover produced by the Frugal algorithm meets the adjusted upper bound for n < 6. Also, Frugal seems to perform better for odd values of n than for even n. In section III.H we further compare algorithm performance, considering the cardinality of the set produced by the algorithm and the cost of running the algorithm.

TABLE III.1

FRUGAL ALGORITHM

n	Upper Bound	Frugal	% Upper Bound
1	0	0	100.00
2	1	1	100.00
3	2	2	100.00
4	6	6	100.00
5	12	12	100.00
6	32	25	78.13
7	64	53	82.81
8	128	88	68.75
9	256	217	84.77
10	512	379	74.02
11	1024	917	89.55

C. SEQUENTIAL FILL

In section II-F, an initial independent set was found by choosing nodes along the order of a de Bruijn cycle. Nodes were placed in an independent set if they created no path with any other node previously placed in the set. The fact that a de Bruijn cycle is a Hamiltonian path in B_n allowed an upper bound to be placed on the cardinality of a cover. However, the algorithm made no special use of the properties of the de Bruijn cycle except that two consecutive nodes on the de Bruijn cycle cannot both be chosen for the independent set. This cuts down the time required for the search. In fact, the sequence of nodes could be considered in any order as long as each of the nodes from 1 to $2^n - 2$ is considered at some point for placement into the independent set.

Because of the ease of coding, the sequence of nodes that starts with 1 and proceeds sequentially to $2^n - 2$ was examined with some interesting results. A significant difference in the percentage of nodes that forms an independent set occurs depending on whether n is even or odd. For this reason the two cases are considered separately.

1. n Even

We illustrate the general method by considering the case for n = 4. An independent set of nodes in the graph for n = 4 is formed by sequentially including nodes in the set

when no conflict exists. By selecting node 1 for the set, nodes 2 and 3 are covered because they have node 1 as a predecessor. Nodes Ø and 8 are also covered because they have node 1 as a successor.

Proceeding sequentially to the next available node, 4 is chosen. Node 4 covers its successors 8 and 9 and its predecessors 2 and 1Ø. Likewise nodes 5, 6 and 7 are included in the set, covering their successor nodes 1Ø through 15. Their predecessors fall in the blocks 2 through 3 and 1Ø through 11, causing no conflict. As no other nodes can be added, sequentially considering the nodes yields the set {1,4,5,6,7}, which is a cover.

For n = 6, the nodes 1, 4, 5, 6 and 7 again cover the nodes Ø through 15 in essentially the same way that they did for n = 4. Nodes 16 through 31 can then be included in the set to cover nodes 32 through 63. The predecessors of nodes 16 through 31 lie in the block from 8 to 15 and in the block from 4Ø to 47. Therefore, including nodes 16 through 31 creates no path with nodes already in the set. All the nodes are covered and node 63 is covered by 31, so the set is a cover.

Proceeding sequentially, the nodes chosen for a cover for n = k can also be chosen to cover the first 2^k nodes in the graph for n = k + 2. Nodes 2^k through $2^{k+1} - 1$ can be placed in the set to cover their successors, nodes 2^{k+1} through $2^{k+2} - 1$. In the graph for n = k + 2, the

predecessors of the block $\{2^k, \ldots, 2^{(k+1)} - 1\}$, just added to the set, lie in the blocks from 2^{k-1} to $2^k - 1$ and from $2^{k-1} + 2^{k+1}$ to $2^k - 1 + 2^{k+1}$, causing no conflicts. The nodes 1 and $(2^{n-1} - 1)$ are in the independent set verifying that it is a cover.

The cardinality of the cover formed in this way for the graph $n = k + 2$ is 2^k greater than that for the cover of $n = k$. Thus, we have the recursion

$$f(n+2) = f(n) + 2^n$$

for n even, where f(n) is the cardinality of the cover for B_n. By repeated substitutions, this recurrence relation, along with the initial condition $f(2) = 1$, yields the closed form solution

$$f(n) = (2^n - 1)/3 \ .$$

Thus, for n even, we have a cover with cardinality of approximately one third the number of nodes in the graph. This is far from the upper bound of half of the nodes in the graph.

The cover for $n = k + 2$ can also be produced from the cover for $n = k$ in a recursive manner not requiring a sequential search.

For $n = 4$, we found the cover $S = \{1, 4, 5, 6, 7\}$. We define the second generation descendents of node 1, in B_6, to be the nodes 4, 5, 6 and 7. The second generation descendents of node 4 in B_6 are the nodes 16, 17, 18 and 19. Continuing in this way, we see that the second generation

descendents of all the nodes in the cover formed by the sequential fill algorithm for n = 4, form the set {4 - 7, 16 - 31} in B_6. Adding node 1 to this set yields the covering set formed by the sequential fill algorithm for the case n = 6 previously.

In general, the covering nodes in the sequential fill procedure for n = k + 2 are the second generation descendents, in B_{n+2}, of the covering nodes in the sequential fill procedure for n=k, together with the node 1.

This leads to the recursion

$$f(n) = 4f(n - 2) + 1 .$$

Solving this recursion, with the initial condition f(2) = 1, again yields the closed form solution

$$f(n) = (2^n - 1)/3 .$$

After seeing the pattern that developed in forming these covers, a very inexpensive algorithm can be developed to generate them with only a small amount of storage.

Algorithm: Sequential Fill; n even

Step 1. Place the node 1 in a queue.

Step 2. Remove the first node from the front of the queue, call it x, and add it to the cover.

Step 3. If x is equal to $2^{(n-2)}$, remove the remaining nodes from the queue, add them to the cover and stop. If not, go on to step 4.

Step 4. Find the second generation descendents of x in B_n, i.e. 4x, 4x + 1, 4x + 2 and 4x + 3. Add these to the back of the queue in ascending order.

Step 5. Go to step 2.

The number of multiplications done in step 4 is approximately one fourth the number of nodes in the cover, or one twelfth the number of nodes in the graph. Thus, we have a very fast, O(V) algorithm. Unfortunately, as noted before, the cardinality of this cover is not very close to the upper bound or the solutions found by previous algorithms.

2. n Odd

For odd values of n, the concept of forming an independent set by sequentially considering nodes of the de Bruijn graph works in the same way as it does for n even. Starting with node 1, a node is included in the set if it does not create a path with another node in the set.

For n = 3, node 1 is placed in the set, covering its successors 2 and 3 and its predecessors 0 and 4. Proceeding sequentially, 5 is the next free node. Node 5 covers its successors 2 and 3 and its predecessors 2 and 6. The next uncovered node is 7 but it cannot be included in the set, for it allows a path to itself. Therefore, the set S = {1, 5} is a maximal independent set but not a cover. However, if node 5 is deleted from S and node 6 replaces 5,

the set S = {1, 6} would still be an independent set and all nodes in the graph would be covered. No nodes become uncovered as a result of exchanging 6 for 5 for the following reason. With the exception of node 6, all of the nodes adjacent to 5 are smaller than 5. Thus, these nodes were considered for inclusion in the independent set prior to node 5, but were rejected. Therefore, they must be covered by nodes in the set other than 5.

In B_5, we begin by placing node 1 in the set. Nodes 2 and 3 are covered by node 1, so we skip them and go to node 4. Nodes 4, 5, 6 and 7 are included in the set, covering their successors, nodes 8 through 15. Node 16 is covered by node 1, so we consider node 17. Node 17's successors are nodes 2 and 3 and its predecessors are nodes 8 and 24. None of these is in the set, so 17 is added. Nodes 18 and 19 are predecessors of nodes 4 - 7, so they are excluded from the set. The successors of nodes 20 - 23 are the nodes 8 - 15. Their predecessors are the nodes 10 and 11 and the nodes 26 and 27. None of these nodes is in the set, so the nodes 20 - 23 are added to the set. Node 24 is a predecessor of 17, so it is not included. Node 25's successors are 18 and 19 and its predecessors are 12 and 28. Since these nodes are not in the set, node 25 is added. Nodes 26 and 27 are excluded because they have successors 20 - 23. Node 28 is excluded because it is a predecessor of 25. Node 29 can be added because neither its successors, 26

and 27, nor its predecessors, 14 and 30, are in the set. Finally, node 30 is excluded because it has just been covered by node 29 and node 31 is excluded because it makes a path with itself. This yields the maximal independent set S = {1, 4-7, 17, 20-23, 25, 29}. Neither node 15 nor node 30 is in the set so S is not a cover. However, again no conflict arises if node 29 is replaced by node 30. This resulting set is still independent and covers B_5 for reasons similar to that discussed concerning the cover for B_2.

0, 1, 2, 3, 4, 5, 6, 7, 8,9,10,11,12,13,14,15,
16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 26, 27,
28, 29,
30,
31

Figure III.1
Nodes in B_5 with Sequential Fill Set Underlined

In Figure III.1, the nodes in B_5 are written in order from 0 to 31. The nodes in the independent set formed by the sequential fill algorithm have been underlined. Starting with node 0, we observe that one node, 0, was excluded from the set, then one node, 1, was included. The next two nodes, 2 and 3, were excluded, then four nodes,

4, 5, 6 and 7, were included and the next eight nodes, 8 - 15, were excluded. Now the pattern begins anew with one node, 16, excluded, one node, 17, included, two nodes, 18 and 19, excluded, then four nodes, 20 - 23, included. Starting over again, we exclude one node, 24, include one node, 25, then exclude two nodes, 26 and 27. This pattern of partioning the nodes continues, each time stopping when the size of the last segment is half the size of the previous last segment. For n = 5, the partition sizes are 1, 1, 2, 4, 8; 1, 1, 2, 4; 1, 1, 2; 1, 1; 1; 1.

We see that the first sixteen nodes in B_5 are included in the first pass. The next eight nodes are included in the second pass. Four more nodes are partitioned on the third pass. Two nodes are partitioned on the fourth pass. One node is partitioned on the fifth pass and one more on the sixth pass.

In general, the pattern observed in B_5 occurs in B_n for each odd value of n. The nodes 0 through $2^{n-1} - 1$ are partitioned in segments of size 2^0, 2^0, 2^1, 2^2, ... 2^{n-2}. The first node in the pattern, 0, and all nodes in segments that are an odd power of 2 in size are excluded from the independent set. The nodes in segments that are an even power of 2 in size are included in the independent set. This partitions the first half of the nodes. Then the nodes 2^{n-1} through $2^{n-1} + 2^{n-2} - 1$ are partitioned in segments of size 2^0, 2^0, 2^1, 2^2, ... 2^{n-3}. The node 2^{n-1} and all nodes in

segments that are an odd power of 2 in size are again excluded from the independent set. The nodes in segments that are an even power of 2 in size are again included in the independent set. This partitions the next fourth of the nodes. This process is repeated with one fewer segment in each application until all of the nodes in the graph have been partitioned. Proceeding in this way, one half of the nodes are partitioned on the first pass, one fourth on the second pass, one eighth one the third pass, etc..

For the nodes $\emptyset$ through $2^{n-1} - 1$, each segment of k nodes that is added to the independent set has a 2k long segment of successor nodes subsequent to it which is excluded. Thus, each segment is twice as long as the previous one and only every other segment is allowed in the set. This pattern is broken by the node 2^{n-1}, which is a predecessor of node one, but the process repeats.

Each segment of $k > 2$ nodes in the first half of the graph also has a corresponding k/2 long segment of predecessor nodes in the next fourth of the graph. These nodes, from 2^{n-1} to $2^{n-1} + 2^{n-2} - 1$, are either excluded from the set or included in the set depending on whether their successor segments in the first half of the graph are included or excluded respectively. The nodes that are added to the independent set between 2^{n-1} and $2^{n-1} + 2^{n-2} - 1$ have predecessors in the first half of the graph and in the next eighth of the graph. The predecessors in the first half all

fall in segments that have been excluded from the set. Thus, the pattern of every other segment being added to the set is repeated with each segment being twice the size of the previous one. The pattern is broken by the node $2^{n-1} + 2^{n-2}$, which is a predecessor of node $2^{n-1} + 1$.

In a similar manner, each segment of $k/2 > 2$ nodes from 2^{n-1} to $2^{n-1} + 2^{n-2} - 1$ has a corresponding $k/4$ long segment of predecessor nodes in the next eighth of the graph. The inclusion of these segments is also determined by the status of their adjacent segments in the previous fourth. The nodes that are added to the independent set between $2^{n-1} + 2^{n-2}$ and $2^{n-1} + 2^{n-2} + 2^{n-3} - 1$ have predecessors in the first half of the graph and in the next sixteenth of the graph. The predecessors in the first half also fall in segments that have been excluded from the set.

We continue in this manner until all of the nodes in the graph are covered except $2^n - 1$. However, the node $2^n - 3$ has been included in the set and can be replaced by the node $2^n - 2$. This causes no conflicts and results in all the nodes in B_n being covered.

To find the cardinality of the resulting cover we simply add up the number of nodes in the segments that were included in the independent set. For $n = 5$, we have

$$|S| = 1 + 4 + 1 + 4 + 1 + 1 = 12 .$$

The first pass through the partitioning pattern places $1 + 4 = 5$ nodes in the independent set. The second

pass also places 1 + 4 = 5 nodes in the independent set even though there was one less segment cosidered than in the first pass. Looking again to figure III.1, we see that the last segment in the first pass was not included in the independent set. So, even though the second pass partitioned only half as many nodes as the first pass, the number of nodes added to the independent set was the same. It will be the case that every even numbered pass will place as many nodes in the set as the previous odd numbered pass.

In general, for any odd value of n, the cardinality of sequential fill set is given by

$$\begin{aligned}|S| = {} & (2^0 + 2^2 + \ldots + 2^{n-3}) + (2^0 + 2^2 + \ldots + 2^{n-3}) \\ & + (2^0 + 2^2 + \ldots + 2^{n-5}) + (2^0 + 2^2 + \ldots + 2^{n-5}) \\ & + (2^0 + 2^2 + \ldots + 2^{n-7}) + (2^0 + 2^2 + \ldots + 2^{n-7}) \\ & + \ldots \\ & + (2^0 + 2^2) \qquad\qquad + (2^0 + 2^2) \\ & + 2^0 \qquad\qquad\qquad + 2^0 .\end{aligned}$$

Letting k = (n - 1)/2 to facilitate indexing, we have

$$|S| = 2 \sum_{j=0}^{k-1} 2^{2j} + 2 \sum_{j=0}^{k-2} 2^{2j} + \ldots + 2 \sum_{j=0}^{0} 2^{2j}$$

or

$$|S| = 2 \sum_{i=1}^{k} \sum_{j=0}^{k-i} 2^{2j} .$$

Since $\sum_{j=0}^{k-i} 2^{2j} = (2^{2(k-i+1)} - 1)/3$, the cardinality of S is given by

$$|S| = 2\sum_{i=1}^{k} [2^{2(k-i+1)} - 1]/3 .$$

This summation yields

$$|S| = [8(2^{2k} - 1) - 6k]/9 .$$

Finally, substituting k = (n - 1)/2, we obtain

$$|S| = [2^{n+2} - 3n - 5]/9 .$$

As n grows, the fraction of the number of nodes in the independent set goes to 4/9. This is quite close to the upper bound of 1/2.

For the case of even values of n, it was observed that the second generation descendents of the nodes in the independent set for n = k appeared in the set for n = k + 2. This allowed us to determine the cardinality of the sequential fill sets recursively. The same methods can be applied to odd values of n to yield the sequential cover without going through the sequential search of the nodes of B_n.

For n = 5, the nodes 4, 5, 6, 7, 20, 21, 22, and 23 are the second generation descendents of nodes 1 and 5 (the independent set for n = 3 formed by the sequential fill algorithm). These nodes, together with the nodes 1, 17, 25 and 29 form the sequential fill set for n = 5. These additional nodes are the nodes of the form 1, $(1 + 2^{n-1})$, $(1 + 2^{n-1} + 2^{n-2})$, ... , $(1 + \sum_{j=2}^{n-1} 2^j)$ for odd values of n greater than 1. For each odd value of n, we add a total of

n - 1 nodes to the set of second generation descendents of the sequential fill cover for the previous odd n. Letting f(n) be the cardinality of the set formed by the sequential fill algorithm for B_n, n an odd integer, we have the recursion

$$f(n) = 4f(n-2) + n - 1$$

with the initial condition f(3) = 2. Solving this system by substitution, we get the same closed form expression obtained earlier

$$f(n) = (2^{n+2} - 3n - 5)/9 .$$

The following algorithm takes advantage of the structure of the sequential fill sets for odd values of n by extending the independent set for B_{n-2} to form the set for B_n. In doing so, we must start by forming the set for n = 3, S = {1, 5}. From that set, we form the set for n = 5, then for n = 7, and so on till we reach the desired value of n. When the last node added to the independent set is incremented by one, the set will be a cover. The set formed in this manner is the same set that is obtained by sequentially considering nodes in the graph and including them in an independent set if no path is created. However, checking each node to see if one of its four neighboring nodes is in the set is much more time consuming.

Algorithm: Sequential Fill; n odd

Step 1. Initialize k to be 3.

Step 2. Place node 1 in the set.

Step 3. Complete the set by adding the nodes

$$(1 + 2^{k-1}),\ (1 + 2^{k-1} + 2^{k-2}),\ \ldots\ ,\ (1 + \sum_{j=2}^{k-1} 2^j\ ).$$

Step 4. If k = n, increment the last node placed in the set by one and stop. If not, go on to step 5.

Step 5. Increment k by 2.

Step 6. For each node x in the independent set, replace x by the nodes 4x, 4x + 1, 4x + 2 and 4x + 3.

Step 7. Go to step 2.

The multiplication done in step 6 is the dominant factor in the cost analysis of this algorithm. Since one multiplication is required for each node in the previous set, this requires O(V) time each pass through the loop. To obtain the cover for B_n, approximately n/2 passes must be made. Therefore, the algorithm is O(nV) or O(VlogV) in cost. This gives us a fairly fast method of obtaining a covering set that is close to the upper bound in cardinality, for odd values of n.

Since multiplying a number by 4 is just a left shift of its binary representation by 2 places, if the programming language used permits the manipulation of the actual storage bits this operation is rather trivial and the algorithm becomes even faster.

D. 3-COLORING

The chromatic number of a graph is the smallest number of colors that are required to color the nodes in the graph so that no two adjacent nodes are of the same color. The de Bruijn graph B_n, for n greater than 1, contains two cycles of length 3. Therefore, the chromatic number of B_n is at least 3.

Theorem III.1

The chromatic number of B_n is 3.

Proof:

Due to the existance of 3-cycles, we see that the chromatic number of B_n is at least 3. Consider the nodes of the graph labeled with their binary representation. We can color the nodes with three colors by coloring the nodes whose binary representation ends with an odd number of ones with the color A, coloring the nodes ending with an odd number of zeros with the color B and coloring all other nodes with the color C.

The nodes colored A have a successor colored B and a successor colored C. Nodes colored B have a successor colored A and a successor colored C. Nodes colored C have a successor colored A and a successor colored B. (The nodes 0 and $2^n - 1$ are adjacent to themselves but this causes no problem).

Q.E.D.

Let us examine the set of nodes colored A. The nodes that end with a single 1 have a Ø in the next to the last bit. Since each of the remaining n - 2 bits can be either a Ø or a 1, there are 2^{n-2} nodes in B_n whose binary representations end with a single 1. Likewise, there are 2^{n-4} nodes ending with exactly three 1's. In general, there are 2^{n-2k} nodes whose binary representation ends with exactly (2k - 1) 1's. To find the cardinality of the set of nodes colored A, we simply add up the number of nodes ending with an odd number of 1's.

For even values of n, summing 2^{n-2k} for k = 1,..., n/2 yields

$$|A| = \sum_{k=1}^{n/2} 2^{n-2k} .$$

$$= (2^n - 1)/(4 - 1)$$

$$= (2^n - 1)/3.$$

For odd values of n, the summation ends at k = (n-1)/2. This accounts for all the A colored nodes ending with (n - 2) 1's or less. However, since n is odd, the all 1's n-tuple is also colored A. Thus,

$$|A| = 1 + \sum_{k=1}^{(n-1)/2} 2^{n-2k}$$

$$= 1 + (2^n - 2)/(4 - 1)$$

$$= (2^n + 1)/3.$$

The nodes that are colored B are the binary complements of the nodes colored A. Thus, $|B| = |A|$. We can find the cardinality of the nodes colored C by subtracting twice the cardinality of the nodes colored A from 2^n. For even values of n, we have $|A| = |B| = (2^n - 1)/3$ and $|C| = (2^n + 2)/3$. For odd values of n, we have $|A| = |B| = (2^n + 1)/3$ and $|C| = (2^n - 2)/3$. Thus, this coloring partitions the nodes of B_n into 3 sets, each having approximately a third of the nodes of the graph.

Since each node is adjacent only to nodes of the other 2 colors, any set of like colored nodes is adjacent to all the nodes in the graph. If this set is independent, it will be a cover. A given colored set will fail to be independent if it contains either the node Ø or the node $2^n - 1$. For even ordered graphs, the set colored A forms a cover as does the set colored B. For odd ordered graphs, the set colored C forms a cover. However, the cardinalities of these covers are not very close to the upper bound, nor are they as large as the solutions found by previous methods.

We count the number of nodes of a given color by a recursive process.

The binary representation of each non-zero node in B_n ends in either k Ø's, where $k < n$, or k 1's, where $k \leq n$. Each node is colored according to this ending. When these nodes in B_n are viewed as nodes in B_{n+1}, the ending of their binary representation is unchanged. Thus, the nodes are

colored the same color as they were in B_n. For instance, the binary representation of node 7 in B_3 is 111. Because it ends in an odd number of 1's, it is colored A. The binary representation of node 7 in B_4 is Ø111. It still ends in an odd number of 1's, so node 7 is colored A in B_4. Thus, once a non-zero node is colored, it will remain that color in all higher order graphs. (The node Ø will alternate between the color B and the color C, depending on whether n is odd or even). If we can determine the number of nodes that are added to a colored set as result of increasing n, the cardinality of that set can be found recursively.

We consider the recurrence relationship only for the cardinality of the set A. As was seen earlier, once this has been determined, the cardinalities of the sets B and C can be obtained easily. We again consider the cases for n even and n odd separately.

First, we consider the case where n is even. Recall that the nodes colored A are those whose binary representation ends with an odd number of 1's. Because n is even, we know that each node in B_n colored A has a Ø somewhere in its binary representation. One of these Ø's ends the final string of 1's. Then, for each of these A colored nodes in B_n to be viewed as a node in B_{n+2}, we are free to place either a Ø or a 1 in each of the two additional bits in the binary representation in B_{n+2}. Thus, for each A colored node in B_n, there are four A colored nodes in B_{n+2} with the same last n

bits. In addition to these, the node that has a Ø in the highest order bit, followed by (n+1) 1's, will also be colored A. If we let f(n) be the number of nodes colored A in B_n, we have the recurrence relation

$$f(n+2) = 4f(n) + 1$$

for n even, with f(2) = 1. This same recursion was solved previously to yield

$$f(n) = (2^n - 1)/3.$$

For odd values of n, the situation is similar to that of even n in that for each A colored node in B_{n+2} there are 4 nodes in B_n with the same last n bits also colored A. However, we need to pay special attention to the all 1 node in B_n. For this node, we are not free to choose any combination of 0's and 1's in the two additional bits in B_{n+2}. Since n is odd, the node that starts with a 01 and then a string of n 1's is a color C node. So, we must subtract it from our count. This gives us the recursion

$$f(n+2) = 4f(n) - 1$$

for n odd, with the initial condition f(3) = 3. The solution to this system is

$$f(n) = (2^n + 1)/3.$$

E. DOUBLING

In section III.C, we observed that an independent set in B_{n+2} could be formed by mapping each node in the sequential fill covering set of B_n to its four, second generation

descendents in B_{n+2}. This independent set can be made into a cover by adding selected nodes to it. Thus, the cardinality of the resultant cover in B_{n+2} is more than four times the cardinality of the initial cover in B_n. This result leads us to search for similar relationships between covers in B_n and in B_{n+1}.

Two methods are discussed in this section which map a cover in B_n into an independent set in B_{n+1}. Conditions under which this set can then be made into a cover are presented. For each method, the cardinality of the cover in B_{n+1} will be at least twice the cardinality of the cover in B_n.

1. Successors in B_{n+1}

The first method is very similar to the pattern observed in the sequential fill method of forming a cover. Given a cover S in B_n, an independent set S' can be formed in B_{n+1} by mapping the nodes in S to their successor nodes in B_{n+1}. That is, each node x, in S, will be mapped to the nodes 2x and 2x + 1 in B_{n+1}. Note that there is no modulus reduction necessary as the n-tuples become (n+1)-tuples by this doubling. Thus, the cardinality of S' is twice that of S. If the nodes 0 and $2^{n+1} - 1$ are covered by nodes in S', then by adding any uncovered nodes, S' can be made into a cover for B_{n+1}.

For example, the nodes of the covering set S = {3, 4} in B_3, are mapped to their successors in B_4 forming

$S' = \{6, 7, 8, 9\}$. The set S' is an independent set in B_4. By adding either node 5 or 10 to S', it is made into a cover.

The following theorem proves that the set in B_{n+1}, formed by the successors of an independent set in B_n, is always an independent set.

Theorem III.2

If the set $S = \{x_1, x_2, \ldots, x_k\}$ is an independent set in B_n, then the set $S' = \{2x_1, 2x_1 + 1, 2x_2, 2x_2 + 1, \ldots, 2x_k, 2x_k + 1\}$ is an independent set in B_{n+1}.

Proof:

Assume not. If S' is not an independent set in B_{n+1}, then there exist nodes x_i and x_j in S such that one of the following six cases is true.

Case 1. The node $2x_i$ is connected to itself in B_{n+1}.

For this to be true, the node $2x_i$ must be a descendent of itself. Thus one of the following subcases must hold.

a) $2x_i = 4x_i \pmod{2^{n+1}}$

This is true if, and only if, $x_i = 0$. However, since S is assumed to be independent, $x_i = 0$ cannot be in S.

b) $2x_i = 4x_i + 1 \pmod{2^{n+1}}$

This subcase is eliminated because $2x_i \pmod{2^{n+1}}$ is even, while $4x_i + 1 \pmod{2^{n+1}}$ is odd. Thus, the two cannot be equal.

Therefore, Case 1 cannot hold.

Case 2. The node $2x_i + 1$ is connected to itself in B_{n+1}.

By an argument similar to that of Case 1, this implies that $x_i = 2^n - 1$, which cannot appear in an independent set in B_n.

Case 3. $2x_i$ is connected to $2x_i + 1$.

This implies either that $x_i = 2^n - 1$ or that $x_i = \emptyset$. Neither condition can hold if S is an independent set in B_n.

Case 4. $2x_i$ is connected to $2x_j$ in B_{n+1}.

By reducing Case 4 to its subcases, we see that x_i must be connected to x_j in B_n for this to hold. However, this cannot occur in an independent set.

Case 5. $2x_i$ is connected to $2x_j + 1$.

Again, this implies that a path exists in B_n between the nodes x_i and x_j.

Case 6. $2x_i + 1$ is connected to $2x_j + 1$.

As before, a path must exist in B_n for this to be true.

Q.E.D.

Since, by definition, a cover is an independent set, Theorem III.2 holds for all covers. By mapping the nodes in a cover S in B_n to their successors in B_{n+1}, we form an independent set S' whose cardinality is twice that of the cardinality of S. If the end nodes, Ø and $2^{n+1} - 1$, are covered, S' can be made into a covering set by adding any uncovered nodes of B_{n+1} to it. As discussed in section II.C, the nodes Ø and $2^{n+1} - 1$ cannot be included in the cover in B_{n+1} because of the path they make with themselves. Therefore, they must be covered by neighboring nodes that are in the cover. Thus, whether an independent set S' can be made into a cover or not depends on whether the nodes Ø and $2^{n+1} - 1$ each have a neighboring node that can be included in S'. This is not always the case.

Starting with the cover S = {1,6}, in B_3, we obtain the independent set S' = {2, 3, 12, 13}, in B_4. Node 1 cannot be added to S' because nodes 2 and 3 are present. Node 8 cannot be added because it creates a path with node 12. This leaves us with no way to cover node Ø. Thus, this set cannot be made into a covering set.

As a second example, we begin with the sequential fill cover for B_4, S = {1, 4, 5, 6, 7}. By taking the successors of S in B_5, we form the independent set S' = {2, 3, 8, 9, 10, 11, 12, 13, 14, 15}. Once again, this set does not allow the node Ø to be covered. Node 1 cannot be

added to S' because it is adjacent to nodes 2 and 3. Node 16 cannot be added because it is adjacent to node 8. Therefore, S' cannot be made into a cover.

The following theorem gives certain sufficient conditions for which the end nodes are guaranteed to be covered by S'. With the nodes Ø and $2^{n+1} - 1$ covered, S' can be made into a covering set by adding any uncovered nodes to it.

Theorem III.3

If the nodes 2^{n-1} and $2^{n-1} - 1$ are in a cover S in B_n, then an independent set S' formed in B_{n+1} from the successors of the nodes in S, will cover the nodes Ø and $2^{n+1} - 1$.

Proof:

By Theorem III.2, S' is an independent set. Since the nodes 2^{n-1} and $2^{n-1} - 1$ are in S, their respective successors, nodes 2^n and $2^n - 1$, will be in S'. Thus, node Ø will be covered by 2^n and node $2^{n+1} - 1$ will be covered by $2^n - 1$.

Q.E.D.

Another sufficient condition is given by Theorem III.4.

Theorem III.4

If the nodes 1 and $2^n - 2$ are in a cover S in B_n, then an independent set S' in B_{n+1} can be formed

such that the nodes $\emptyset$ and $2^{n+1} - 1$ will be covered.

Proof:

If the nodes 1 and $2^n - 2$ are in S, S' can be formed in the following way. Map the nodes in S to their binary reverses in B_n and call that set R. R will also be a cover in B_n. The mapping will take node 1 to node 2^{n-1} and node $2^n - 2$ to node $2^{n-1} - 1$. Then, by Theorem III.3, we see that in taking the successors of R in B_{n+1}, we form an independent set in which the nodes $\emptyset$ and $2^{n+1} - 1$ are covered.

Q.E.D.

Thus, by Theorems III.2, III.3 and III.4, if the opposite corners of the de Bruijn graph (nodes 1 and $2^n - 2$ or nodes 2^{n-1} and $2^{n-1} - 1$) are in a cover for B_n, a cover for B_{n+1} can be formed that is at least twice as large in cardinality as the cover for B_n.

2. Companion Pairs

Consider an independent set S in B_n. By the following lemma, S is also an independent set in B_{n+1}.

Lemma III.5

If S is an independent set in B_n, then S is an independent set in B_{n+1}.

Proof:

Assume not. If S is not independent in B_{n+1}, then there exist nodes x and y in S such that x is a

successor of y in B_{n+1}. This can be true in one of two cases.

Case 1. $x = 2y \pmod{2^{n+1}}$

An equivalent statement is $x = 2y + k2^{n+1}$ or that $x = 2y + (2k)2^n$, where k is an integer. However, this implies that x is a successor of y in B_n, which contradicts the premise that S is independent in B_n.

Case 2. $x = 2y + 1 \pmod{2^{n+1}}$

By a similar argument, we see that for this case to be true, we must have $x = 2y + 1 \pmod{2^n}$, which again is a contradiction.

Q.E.D.

In section I.B, the companion of a node x was defined as the node whose binary representation agreed with that of x in all but the highest order bit. Since the nodes in S are in the graph B_n, they can be repressented by binary n-tuples. When viewed as nodes in B_{n+1}, their binary (n+1)-tuple representation has a 0 in the highest order bit. Thus, the companions of the nodes in S, viewed as nodes in B_{n+1}, will have a 1 in the highest order bit. In other words, the companion of a node x, in S, is the node $x + 2^n$.

Given an independent set S, we form the set S' from the nodes in S and their companion nodes in B_{n+1}. By Theorem III.6, we see that S' is an independent set in B_{n+1}.

Given an independent set S, we form the set S' from the nodes in S and their companion nodes in B_{n+1}. By Theorem III.6, we see that S' is an independent set in B_{n+1}.

Theorem III.6

If $S = \{x_1, x_2, \ldots x_k\}$ is an independent set in B_n, then the set $S' = \{x_1, \ldots, x_k, y_1, \ldots, y_k\}$, where $y_i = x_i + 2^n$, is an independent set in B_{n+1}.

Proof:

The proof is essentially equivalent to the proof of Theorem III.2.

As an example, consider the cover S = {1, 6}, in B_3. S = {1, 6} is also an independent set in B_4. We add the nodes 9 and 14 (the companions in B_4 of 1 and 6 respectively) to S to form the independent set S' = {1, 6, 9, 14}. If we add either node 5 or node 10 to S', it becomes a cover.

Since Theorem III.6 holds for any independent set, we have a second method of forming an independent set S', in B_{n+1}, from an cover S, in B_n. Again the cardinality of S' is twice the cardinality of S. If the nodes 0 and $2^{n+1} - 1$ are covered, then S' can be made into a covering set by adding any uncovered nodes. Again, conditions exist for which the end nodes cannot be covered.

The set S = {1, 4, 5, 7, 12, 13}, is a cover in B_4. By adding nodes 17, 20, 21, 23, 28 and 29 (the companion

nodes in B_5 of the nodes in S), we form the independent set S' = {1, 4, 5, 7, 12, 13, 17, 20, 21, 23, 28, 29}, in B_5. However, node 31 cannot be covered by adding nodes to S'. Node 15 conflicts with nodes 7 and 23 while node 30 conflicts with nodes 28 and 29. Thus, the set S' cannot be made into a cover in B_5 by simply adding nodes.

Theorem III.7 gives a sufficient condition so that the nodes 0 and $2^{n+1} - 1$ are covered by nodes in S'. In that case, S' can be made into a cover in B_{n+1} by adding any uncovered nodes to the set.

Theorem III.7

If the nodes 1 and $2^n - 2$ are in a cover S, in B_n, then the set S', formed from the nodes in S along with their companion nodes in B_{n+1}, will be an independent set in B_{n+1} and the nodes 0 and $2^{n+1} - 1$ will be covered.

Proof:

By Theorem III.6, S' is an independent set in B_{n+1}. Since node 1 is in S, it will be in S'. Thus, the node 0 will be covered by node 1. Since the node $2^n - 2$ is in S, $2^n - 2 + 2^n$, its companion node in B_{n+1}, will be in S'. But, $2^n - 2 + 2^n = 2^{n+1} - 2$, which covers the node $2^{n+1} - 1$ in B_{n+1}.

Q.E.D.

Theorem III.8 provides another sufficient condition that nodes in S' will cover the end nodes of B_{n+1}.

Theorem III.8

If the nodes 2^{n-1} and $2^{n-1} - 1$ are in a cover S, in B_n, then an independent set S', in B_{n+1}, can be formed from companion pairs such that the nodes 0 and $2^{n+1} - 1$ will be covered.

Proof:

If the nodes 2^{n-1} and $2^{n-1} - 1$ are in S, map the nodes in S to their binary reverses in B_n and call that set R. R will also be a cover in B_n. The mapping takes node 2^{n-1} to node 1 and node $2^{n-1} - 1$ to node $2^n - 2$. Then, by Theorem III.7, we see that by adding to the set R its companions in B_{n+1}, we form an independent set in which the nodes 0 and $2^{n+1} - 1$ are covered.

Q.E.D.

Continuing our previous example, the set S' = {1, 4, 5, 7, 12, 13, 17, 20, 21, 23, 28, 29} is not a cover in B_5 because it fails to cover node 31. However, if we replace nodes 28 and 29 with node 30, S' would satisfy all the requirements of a cover. No more than 2 nodes can conflict with a corner node that is needed to cover an end node. Thus, in exchanging nodes in S' with a node in $B_n - S'$ (such as was done in the example), the cardinality of S' will decrease no more than 1.

Thus, given a cover S, in B_n, we form an independent set S', in B_{n+1}, such that the cardinality of S' is twice that of S. This can be done either by mapping the nodes in S to their successors in B_{n+1} or by adding to the nodes in S their companion nodes in B_{n+1}. We can always choose a doubling scheme so that at least one of the two end nodes is covered. If the other end node cannot be covered without creating a conflict, the appropriate corner node can be brought into S' to cover that end node and the conflicting node(s) deleted from S'. In this way, S' can be made into a cover in B_{n+1} with cardinality no less than twice the cardinality of S minus 1.

For odd values of n, covering sets of approximately 4/9 of the nodes in the graph are obtained by the sequential fill method. By the doubling theorems, covering sets can be found in even order graphs that also have cardinality of approximately 4/9 the number of nodes in the graph. As noted before, this is quite close to the upper bound of 1/2 the nodes in the graph.

F. DOUBLING SCHEMES

In section V.E, two general methods of doubling were discussed. In each method, an independent set S' was formed in B_n from an independent set S in B_{n-1}. S' was then made into a cover of B_n by adding selected uncovered nodes.

to a cover of near maximal cardinality when applied to one starter set might not produce as large a cover when applied to another set. Likewise, a starter set might produce favorable results when doubled by one method but might produce poor results when doubled by the other method.

In this section, two schemes for doubling are presented to form a cover in B_n. For each scheme, we discuss the starter set S, the basic doubling method used and the process of covering the remaining nodes. The schemes vary for even and odd values of n. These schemes both yield covers that are close to the upper bound, yet are inexpensive to perform.

1. Double and Redouble

a. n even

For even values of n, consider the set of nodes colored A by the 3-coloring in B_{n-1}. This set is not independent in B_{n-1}, for it contains the node $2^{n-1} - 1$. Thus, we delete that node, making the set independent. This is the starter set S.

We double S to form S' by forming the successors of the nodes of S. That is, map each node x in S to the nodes 2x and 2x + 1 in B_n. Note that there is no modulus reduction necessary as the (n-1)-tuples become n-tuples by this doubling. By the doubling theorems in section V.E, we know that S' is an independent set in B_n.

reduction necessary as the (n-1)-tuples become n-tuples by this doubling. By the doubling theorems in section V.E, we know that S' is an independent set in B_n.

We take a close look at the nodes in S'. Since the nodes in S are colored A in B_{n-1}, their binary representations all end with an odd number of 1's. When these nodes are multiplied by 2, we obtain B colored nodes in B_n that end with an odd number of 1's and one Ø. Multiplying the nodes in S by 2 and adding 1 yields the C colored nodes that end with an even number of 1's. The only nodes of these types that are not in S' are the descendents of the node $2^{n-1} - 1$, which was deleted from S. These are the nodes $2^n - 2$ and $2^n - 1$.

It is noteworthy at this point to mention that neither the node Ø nor the node $2^n - 1$ is covered by the nodes in S'. Node Ø can only be covered by node 1 or node 2^{n-1}. However, node 1 is colored A and is not in S'. Node 2^{n-1} is colored B but it ends in (n-1) Ø's, so it is not in S' either. Likewise, node $2^n - 1$ can only be covered by node $2^{n-1} - 1$ or node $2^n - 2$. But, node $2^{n-1} - 1$ is colored A and is not in S' and we have already seen that node $2^n - 2$ is not in S'.

We cover $2^n - 1$ by adding $2^n - 2$ to S'. We have just shown that its predecessors, $2^{n-1} - 1$ and $2^n - 1$, are not in S'. We now need to ensure that the successors of $2^n - 2$ are not in S' for S' to remain independent.

The node $2^n - 2$ is colored B. Recall that companion nodes have the same binary representation except in the highest order bit. Thus, unless a node consists of all Ø's or all 1's, it will be the same color as its companion. So, the companion of $2^n - 2$ is also colored B. Thus,the successors of $2^n - 2$ descend from nodes that are both colored B. However, the nodes in S' descend from nodes colored A. Therefore, the successors of $2^n - 2$ are not in S' so it is not covered and can be added to S'.

The set S' now contains

1.1 All B colored nodes ending in an odd number of 1's followed by one Ø.

1.2 All C colored nodes ending in an even number of 1's except $2^n - 1$.

We consider how well S' covers B_n. Certainly, the A colored nodes in B_n that are less than 2^{n-1} are covered by the nodes in S' because they are predecessors of the nodes in S'. The companion nodes of the A colored nodes less than 2^{n-1} are the A colored nodes greater than 2^{n-1}. These nodes are also predecessors of the nodes in S'. Thus, all the A colored nodes in B_n are covered.

The nodes in 1.1 are predecessors of the C colored nodes that end with an odd number of 1's and two Ø's. The nodes in 1.2 are predecessors of the B colored nodes ending with an even number of 1's and one Ø. The nodes in S' also have successors colored A. But, since these nodes

have already been covered and no conflict exists, the details concerning them will be eliminated.

We see that the nodes that are yet to be covered are

i. All B colored nodes ending in three or more Ø's.

ii. All C colored nodes ending in four or more Ø's.

iii. The C colored nodes ending in an even number of 1's followed by two Ø's, except for the node $(2^n - 4)$. This node is covered by $(2^n - 2)$, which was included in 1.1.

Consider now the set of nodes colored A in B_{n-3} not including the node $2^{n-3} - 1$. This set is doubled by taking the successors in B_{n-2} and adding the node $2^{n-2} - 2$. We then multiply each node in the set by 4. This multiplication will shift the binary representation of these nodes 2 places to the left and fill Ø's in the vacated positions. This yields the following nodes in B_n.

2.1 The B colored nodes ending in an odd number of 1's followed by three Ø's.

2.2 The C colored nodes ending in an even number of 1's followed by two Ø's, except for the node $(2^n - 4)$.

We add these nodes to S'. There is no conflict with the nodes already in S' because the nodes just added are a subset of the nodes that were shown to be uncovered.

The nodes in 2.1 are predecessors of the nodes colored C that end with an odd number of 1's followed by four Ø's. The nodes in 2.2 are predecessors of the nodes colored B that end with an even number of 1's and three Ø's. Thus, we have the following nodes that are still uncovered.

i. All B colored nodes ending in five or more Ø's.

ii. All C colored nodes ending in six or more Ø's.

iii. The C colored nodes ending in an even number of 1's and four Ø's, except the node $(2^n - 2^4)$. The node $(2^n - 2^4)$ is covered by $(2^n - 2^3)$, which was added in 2.1 above.

So, we redouble.

On the kth doubling, we take the set colored A in B_{n-2k+1}, delete the node $2^{n-2k+1} - 1$, double by taking the successors of these nodes in B_{n-2k+2} and add the node $2^{n-2k+2} - 2$ to the set. Then, these nodes are multiplied by 2^{2k-2}, shifting their binary representation 2k-2 places to the left. The resulting set of nodes is added to S'. This process is done for $k = 1, 2, \ldots, n/2$.

If n is greater than 2k + 2, we have the following nodes that are still uncovered.

i. All B colored nodes ending in 2k + 1 or more Ø's.

ii. All C colored nodes ending in 2k + 2 or more Ø's.

iii. The C colored nodes ending in an even number of 1's and 2k Ø's, except the node $(2^n - 2^{2k})$, which is covered by the node $(2^n - 2^{2k-1})$.

On the (n/2)th doubling, we take the set colored A in B_1, {1}, and delete the node $2^1 - 1 = 1$, leaving us with the empty set. Doubling the empty set still leaves us with the empty set. We then add the node $2^2 - 2 = 2$, giving us the set {2}. The single node in this set in multiplied by 2^{n-2} to shift it n-2 places to the left. This gives us the node 2^{n-1} to add to S', which covers the node Ø. At this point, S' forms a cover of B_n.

The cardinality of the nodes colored A in B_n, for odd values of n, was given previously as $(2^n + 1)/3$. So, on the kth doubling, the cardinality of the starter set is $(2^{n-2k+1} + 1)/3$. After deleting one node, we have $(2^{n-2k+1} - 2)/3$ nodes. Doubling this set yields $(2^{n-2k+2} - 4)/3$ nodes. Finally, by adding one node, $(2^{n-2k+2} - 1)/3$ nodes are added to S'. To find the cardinality of S', we sum $(2^{n-2k+2} - 1)/3$ for values of k from 1 to n/2, yielding

$$|S'| = \sum_{k=1}^{n/2} (2^{n-2k+2} - 1)/3$$

$$= (\sum_{k=1}^{n/2} 2^{n-2k+2} - n/2)/3$$

$$= \{(2^{n+2} - 4)/3 - n/2\}/3$$

$$= (2^{n+3} - 8 - 3n)/18.$$

Asymptotically, the ratio of the number of nodes in this cover to the number of nodes in the graph is 4/9. This is fairly close to the upper bound of 1/2.

Now that the array of A colored nodes for B_{n-1} is known, the Double and Redouble algorithm can be performed in O(V) time, where V is the number of nodes in the graph. Given the array of A colored nodes, it costs O(1) time to determine any one of the nodes that will ultimately be in the cover S' as only a shift is required. Since there are O(V) nodes in S', the run time of the algorithm is O(V).

Since the nodes colored B are the binary complements of the nodes colored A, a similar method can be derived to form a cover in B_n from the B colored nodes in B_{n-1}. The method would be basically the same, with only a few modifications. Instead of deleting the node $2^{n-2k+1} - 1$ on the kth doubling and later adding the node $2^{n-2k+2} - 2$, we would delete the node Ø and later add the node 1. Also, when we shift the binary representations of the nodes in redoubling, we would fill the vacated bits with 1's instead of Ø's. So, instead of multiplying the nodes by 2^{2k-2}, we would multiply by 2^{2k-2} and add $2^{2k-2} - 1$.

b. n odd

For odd values of n, the Double and Redouble algorithm is nearly the same as for even values of n. We start with the set colored A in B_{n-1}. This set is independent, so we do not have to delete any nodes prior to

doubling to ensure an independent set S' in B_n. There is also no need to add a node after doubling. This simplifies the process so that we do not have to consider the exceptions that were present for n even.

The doubling method used is, again, the method of finding the successors in B_n of the nodes in S. This gives the following nodes in S'.

1.1 All B colored nodes ending in an odd number of 1's followed by a single Ø.

1.2 All C colored nodes ending in an even number of 1's.

Here we see that the node ending with (n-1) 1's, $(2^{n-1} - 1)$, is in S'. Thus, the node $2^n - 1$ is covered.

The following nodes are still uncovered.

i. All B colored nodes that end with three or more Ø's.

ii. All C colored nodes that end with four or more Ø's.

iii. The nodes colored C that end with an even number of 1's followed by two Ø's.

The A colored nodes in B_3 are doubled by taking their successors in B_{n-2}. These nodes are then multiplied by 4 to shift their binary representations 2 places to the left. This set is added to S'.

As in the case for even values of n, this doubling process is repeated. On the kth doubling, the set colored A in B_{n-2k+1} is doubled by taking the successors of these nodes in B_{n-2k+2}. Then, these nodes are multiplied by

2^{2k-2}, shifting their binary representation 2k-2 places to the left. The resulting set of nodes is added to S'. This process is done for k = 1, 2, ... , (n-1)/2.

If n is greater than 2k + 2, the following nodes are still uncovered.

i. All B colored nodes ending in 2k + 1 or more Ø's.

ii. All C colored nodes ending in 2k + 2 or more Ø's.

iii. The C colored nodes ending in an even number of 1's and 2k Ø's.

For k = (n-1)/2, the set colored A in B_2, {1}, is doubled. This yields the set {2, 3}. These nodes are multiplied by $2^{2(n-1)/2 - 2} = 2^{n-3}$ to shift their binary representation n-3 places to the left. Thus, the nodes 2^{n-2} and $3*2^{n-3}$ are added to S'. However, the node that ends with 2(n-1)/2 + 1 (= n) Ø's (node Ø) is still uncovered.

We cannot add node 1 to the cover because nodes 2 and 3 are present. Node 2^{n-1} cannot be added to cover node Ø because it is adjacent to the node 2^{n-2}, which has just been added. So, we delete the node 2^{n-2} from the set S'. It was added on the last doubling to cover itself and its even successor 2^{n-1}. We replace 2^{n-2} with 2^{n-1}. Thus, the node Ø will be covered and no nodes became uncovered. Now S' is a cover.

The cardinality of the nodes colored A when n is even is $(2^n - 1)/3$. So, on the kth doubling, our starter set has $(2^{n-2k+1} - 1)/3$ nodes. Doubling this yields

$(2^{n-2k+2} - 2)/3$ nodes that are added to S'. Therefore, to find the cardinality of the cover, S', we sum $(2^{n-2k+2} - 2)/3$ for values of k from 1 to (n-1)/2, yielding

$$|S'| = \sum_{k=1}^{(n-1)/2} (2^{n-2k+2} - 2)/3$$

$$= \{\sum_{k=1}^{(n-1)/2} 2^{n-2k+2} - (n-1)\}/3$$

$$= \{(2^{n+2} - 8)/3 - n + 1\}/3$$

$$= (2^{n+2} - 3n - 5)/9$$

The cardinality of the cover obtained by performing Double and Redouble for odd values of n is the same as the cardinality obtained by the Sequential Fill algorithm for n odd. These covers contain approximately 4/9 of the nodes in the graph.

A similar algorithm can be developed to form a cover for n odd that uses the nodes colored B in B_{n-1} as a starter set. This method would differ from the odd n using A colored nodes in the following ways. When shifting the binary representation of the nodes in redoubling, the vacated bits are filled with 1's instead of 0's. Thus, instead of multiplying by 2^{2k-2} we would multiply by 2^{2k-2} and add $2^{2k-2} - 1$. Also, instead of replacing node 2^{n-2} by nodes 2^{n-1} on the last doubling, we would replace node $5*2^{n-3} + 2^{n-3} - 1 = 2^{n-1} + 2^{n-2} - 1$ with node $2^{n-1} - 1$.

When the A colored nodes in B_{n-1} were used as the starter set, the following was observed. Through n = 11, after the initial doubling, if we were to sequentially fill from the remaining available nodes, the same cover that was formed by Double and Redouble would result. This holds true for both even and odd values of n.

When the B colored nodes were used as the starter set, for even values of n, we could produce the same cover formed by Double and Redouble by doubling the starter set and sequentially filling from the remaining available nodes. To produce the cover formed by Double and Redouble for odd values of n, double the starter set and fill from the remaining available nodes in reverse sequential order!

2. Double and Cover

This method of obtaining a cover in B_n is similar to Double and Redouble in that the starter set used is formed by 3-coloring B_{n-1}. We also double that starter set by taking the successors of those nodes in B_n. However, Double and Cover differs in the way that the remaining nodes in the graph are covered.

a. n odd

For odd values of n, we choose the set of nodes colored C in B_{n-1} as the starter set S. However, to make S an independent set, we must delete the nodes 0 and $2^{n-1} - 1$

from it. We then double the set S by taking the successors of these nodes in B_n to form the independent set S'.

We are now able to add the nodes 1 and $2^n - 2$ to S' to cover the end nodes, without creating a path with any of the existing nodes in S'. We do so and justify this action presently.

Since the nodes in S were colored C in B_{n-1}, the nodes in S' are

1.1 All B colored nodes ending in an odd number of Ø's greater than 1, except the node Ø.

1.2 The B colored nodes ending in an even number of 1's and one Ø.

1.3 All A colored nodes ending in an even number of 1's greater than 1, except the node $2^n - 1$.

1.4 The A colored nodes ending in an even number of Ø's and one 1.

Let us now justify the addition of the nodes 1 and $2^n - 2$ to S' to cover the end nodes.

The nodes that create a path with node 1 are the nodes 2, 3, Ø and 2^{n-1}. The last three bits of node 2 are Ø1Ø. Thus, it is colored B, but it is in neither 1.1 nor 1.2. The node 3 ends in Ø11, so it is colored C and is thus not in S'. The node Ø is in S' only if it is also in S. However, we specifically deleted node Ø from S. And, the node 2^{n-1} is in S', only if the node 2^{n-2} is in S. But, for

odd values of n, 2^{n-2} is colored B, excluding it from S. Therefore, none of the nodes adjacent to node 1 are in S'.

The nodes that are adjacent to the node $2^n - 2$ are the nodes $2^n - 4$, $2^n - 3$, $2^{n-1} - 1$ and $2^n - 1$. The node $2^n - 4$ ends in two Ø's. Thus it is colored C and is not in S'. The node $2^n - 3$ ends in 1Ø1, so it is colored A, but it is in neither 1.3 nor 1.4. Node $2^{n-1} - 1$ ends in (n - 1) 1's. Since n is odd, $2^{n-1} - 1$ is colored C and is not in S'. Finally, the node $2^n - 1$ is in S' only if the node $2^{n-1} - 1$ is in S. But, $2^{n-1} - 1$ was specifically deleted from S. Therefore, none of the nodes adjacent to the node $2^n - 2$ are in S'.

Consider now the nodes in B_n that are covered by the set S'. The nodes colored C are predecessors of the nodes in S', so they are covered. The nodes in 1.1 above cover all A colored nodes ending in an odd number of 0's greater than 1, followed by one 1. The nodes in 1.2 cover all the A colored nodes ending in an even number of 1's followed by Ø1. With these sets covered by nodes in S' and with the nodes in 1.3 and 1.4 included in S', we see that the only A colored nodes that are not covered are those ending in an odd number of 1's followed by 01. We refer to this set of uncovered A nodes as UA.

Similarly, the nodes in 1.3 cover all B colored nodes ending in an odd number of 1's greater than 1, followed by one Ø. The nodes in 1.4 cover all B colored

nodes ending in an even number of Ø's followed by 1Ø. With these sets covered by nodes in S' and with the nodes in 1.1 and 1.2 included in S', we see that the only B colored nodes that are not covered are those ending in an odd number of Ø's followed by 1Ø. We call this set of uncovered B nodes UB.

Since adjacent nodes in the graph cannot be the same color, we know that UA and UB are both independent sets. We add one of these sets, say UA, to S'.

Consider now the nodes of the form xx...xØ1Ø1, where each x can be a Ø or a 1. This is a subset of UA. The (n-4) bits represented by x's can be any combination of Ø's and 1's. To find one of the predecessors of the node xx...xØ1Ø1, we shift the bits one place to the right and fill the vacated bit with a Ø (divide by 2). This gives us Øxx...xØ1Ø. To find the other predecessor, we shift the bits one place to the right and fill the vacated bit with a 1 (divide by 2 and add 2^{n-1}). This yields 1xx...xØ1Ø, which is the companion of Øxx...xØ1Ø.. Thus, among the predecessors of the nodes in UA are the nodes ending in an odd number of Ø's followed by 1Ø. This is precisely the set UB. So, by adding the set UA to S', we cover the set UB. Similarly, adding the set UB to S', would cover the set UA. Thus, by adding either the set UA or the set UB to S', the cover is completed.

There are two exceptions that should be mentioned. If we add the set UA to S' to complete the cover, we cannot add the node ending in (n - 2) 1's followed by Ø1 (node 2^n - 3). This would form a path with the node 2^n - 2. 2^n - 2 was added to S' after doubling to cover 2^n - 1. Likewise, if we add the set UB to S', we must exclude the node ending in (n - 2) Ø's followed by 1Ø (node 2). Node 2 forms a path with node 1, which was added after doubling to cover the node Ø. However, these exceptions leave no node uncovered and allow for the end nodes to be covered.

Because of the structure of nodes in UA and UB, we can easily determine the cardinality of the cover just formed. The set UA is made up of the nodes ending in (2k - 1) 1's followed by Ø1, where k = 1,2,...,(n-3)/2. k = (n-1)/2 is disallowed because of the conflict with the node 2^n - 2. Since the bit prior to the (2k - 1) long string of 1's must be a Ø, there are (n - 2k - 2) bits that are free to be either a Ø or a 1. Thus, there are 2^{n-2k-2} nodes ending in (2k - 1) 1's followed by Ø1. To find the cardinality of UA, we sum 2^{n-2k-2} for k = 1,2,...,(n-3)/2.

$$|UA| = \sum_{k=1}^{(n-3)/2} 2^{n-2k-2}$$

$$= (2^{n-2} - 2)/(4 - 1)$$

$$|UA| = (2^{n-2} - 2)/3.$$

Since the nodes in UB are the binary complements of the nodes in UA, $|UB| = |UA|$.

The cardinality of the set of C colored nodes in B_{n-1} is $(2^{n-1} + 2)/3$ for odd values of n. Deleting nodes 0 and 2^{n-1} leaves $(2^{n-1} - 4)/3$ nodes in the starter set S. Doubling S and adding the nodes 1 and $2^n - 2$ results in $(2^n - 2)/3$ nodes in S'. Completing the cover by adding the set UA to S' yields

$$|S'| = (2^n - 2)/3 + (2^{n-2} - 2)/3$$

$$= (5*2^{n-2} - 4)/3.$$

The ratio of the number of nodes in the cover to the number of nodes in the graph is approximately 5/12. While this is fairly close to the upper bound of 1/2, it is not quite as close as 4/9, obtained by the Sequential Fill algorithm for odd values of n and the Double and Redouble algorithm for odd values of n.

The Double and Cover algorithm requires O(V) time, where V is the number of nodes in the graph, since the set of C colored nodes is known. The doubling process requires 1 multiplication and 1 addition for every node in the starter set S. While the filling process requires 1 addition for every node added to S'.

b. n even

For even n, the Double and Cover method works very much the same as it does for odd n. The starter set S is the set of C colored nodes in B_{n-1}. This set is independent, so there is no need to delete any nodes prior to doubling. We again double the set S by taking the successors of those nodes in B_n to form the independent set S'. This gives the following nodes in S'.

1.1 All B colored nodes ending in an odd number of 1's greater than 1.

1.2 The B colored nodes ending in an even number of 1's and one Ø.

1.3 All A colored nodes ending in an odd number of 1's greater than 1.

1.4 The A colored nodes ending in an even number of Ø's and one 1.

Since n is even, the node 2^{n-1} ends in an odd number of Ø's and is included in 1.1. Thus, the node 0 is covered. Likewise, the node $2^{n-1} - 1$ ends in an odd number of 1's and is included in 1.3, covering the node $2^n - 1$.

As was the case for odd n, we can partition the remaining uncovered nodes in B_n into two independent set that we will again call UA and UB. In the set UA, are the A colored nodes that end in an odd number of 1's followed by Ø1. The set UB consists of the B colored nodes that end in an odd number of Ø's followed by 1Ø. Because we did not add

any nodes to S' to cover the end nodes after doubling, there are no exceptions as there were for odd values of n. Also, the set UA is covered by the set UB and vice versa. Thus, we can complete the cover by adding either set to S'.

Again, there are 2^{n-2k-2} nodes that end with (2k-1) 1's followed by Ø1. For even values of n, k ranges from 1 to (n-2)/2. Thus, the cardinality of UA is given by:

$$|UA| = \sum_{k=1}^{(n-2)/2} 2^{n-2k-2}$$

$$= (2^{n-2} - 1)/(4 - 1)$$

$$= (2^{n-2} - 1)/3$$

The nodes in the set UB are the binary complements of the nodes in the set UA, so |UA| = |UB|.

For even values of n, the cardinality of the set of C colored nodes in B_{n-1} is $(2^{n-1} - 2)/3$. Doubling this to form S', we have $|S'| = (2^n - 4)/3$. Adding either the set UA or the set UB to S' to complete the cover, we obtain the following expression for the cardinality of S'.

$$|S'| = (2^n - 4)/3 + (2^{n-2} - 1)/3$$

$$= (5*2^{n-2} - 5)/3$$

Thus, for even values on n, the fraction of nodes in B_n that are in the cover is 5/12.

It was observed for both odd and even ordered graphs through B_{11}, that if we complete the cover by

sequentially filling from the remaining uncovered nodes rather than adding either the set UA or the set UB, we obtain a cover whose cardinality is equal to the one formed by Double and Redouble. Unfortunately, sequential filling is an expensive technique if only a portion of the cover is formed in that way. It is only when the entire cover is formed by sequential filling that the cover can be formed efficiently.

In the next section, we present a general method to increase the cardinality of a given cover under certain conditions. If the increase in cardinality is not too great, which is the case in Double and Cover, the expense of building up the cardinality is tolerable.

G. BUILD-UP

When determining the lower bound for the cardinality of a covering set S, in section II.E, each node in S was solely responsible for covering four distinct nodes in $B_n - S$. In that case, if any given node was removed from S, then the four nodes adjacent to it could be brought in, increasing the cardinality of S by three.

In most covers, a 4-for-1 exchange will not be possible. However, if a node x, in S, is solely responsible for covering k nodes in $B_n - S$, a k-for-1 exchange can be made. This increases the cardinality of S by k - 1. The new set will still be a cover. The node x is now covered by each of

the k nodes brought into S. Any nodes previously covered by x that are not brought into S are covered by other nodes in S, otherwise they would have been part of the exchange.

If no k-for-1 exchanges in which k > 1 are possible, then only 1-for-1 exchanges can be made. While making such an exchange has no immediate effect on the cardinality of the cover, it may make it possible to increase the cardinality by future exchanges.

Starting with a cover S in B_n, we repeatedly make k-for-1 exchanges where k is as large as possible. In this way, the cardinality of the covering set can be increased. We continue this process until either all of the nodes in B_n - S (except Ø and $2^n - 1$) are covered by more than one node in S or a cycle of 1-for-1 exchanges occurs.

As an example of this process, consider S = {1, 4, 11, 14}, a covering set of B_4. The node 4 is the only node covering nodes 9 and 1Ø. If 4 is deleted from S, then 9 and 1Ø can be brought into S, resulting in S' = {1,9,1Ø,11,14}.

At this point, we can do no better than the 1-for-1 exchange of removing node 1 from S' and adding node 8, leaving us with S'' = {8, 9, 1Ø, 11, 14}.

From this set, if node 9 is deleted, nodes 2 and 3 can be added, yielding S''' = {2, 3, 8, 1Ø, 11, 14}. With this cover, each node except Ø and 15 is covered by more than one node, and the algorithm stops. The cardinality of the resulting cover is 6. This is the maximum cardinality found

in the exhaustive search for n = 4, so the Build-up procedure yields a covering set of maximum cardinality for n = 4.

There are two situations which can arise that will cause the Build-up algorithm to fail to work properly. While neither is likely to occur, they are both real possibilities and must therefore be guarded against.

The first situation arises when the node to be removed from S lies on a 3-cycle, such as (4,9,18) in B_5. If one of the nodes, say node 4, is in S and none of the other nodes adjacent to nodes 9 and 18 are in S, then only node 4 covers nodes 9 and 18. The Build-up routine would attempt to increase the cardinality of S by deleting node 4 and adding nodes 9 and 18. This, however, would result in a set that is no longer independent, since a path exists from 9 to 18. Because of the relationships of the nodes adjacent to the nodes on a 3-cycle, the situation described above is impossible for n < 5 and seems unlikely to occur for larger n. Also, as there are only two 3-cycles, (ØØ1) and (Ø11), in the de Bruijn graph for each n > 1, the likelihood of this situation occurring does not increase as n gets larger.

The second difficulty to be avoided occurs as a natural process of the Build-up algorithm. As nodes are removed and subsequent nodes are added to a cover, Build-up tends to pack nodes in the cover more closely together. The process does this because then, more nodes can be added to the set.

However, in this shifting around process, there is nothing to prevent the nodes that cover the end nodes, $\emptyset$ and $2^n - 1$, from being deleted and being replaced with nodes that do not cover the ends. If this happens, the resulting set will not be a cover. One solution to this problem is to peg down the nodes that cover $\emptyset$ and $2^n - 1$ and never allow them to be removed from the set.

The process of finding the nodes in S, if any, that are solely responsible for covering nodes in $B_n - S$, is the dominant factor in the cost analysis of Build-up. For each node x, in S, the nodes adjacent to x must be checked to see if they are adjacent to any other nodes in S. Since there are O(V) nodes in $B_n - S$ to be checked against O(V) nodes in S, this process requires $O(V^2)$ time. Thus, each iteration of the procedure costs $O(V^2)$ time.

This can get quite expensive if we Build-up the cardinality of a given cover by a large amount. As an example, consider the cover obtained by 3-coloring as far as possible. It is possible to increase the cardinality of this cover from about (1/3)V to about (4/9)V. Thus, we perform the Build-Up routine at least (1/9)V times. In this case, the entire Build-up process is at least $O(V^3)$ in cost.

The Build-up routine was used in conjunction with the results from previous algorithms in an attempt to increase the cardinality of those covers. Tables III.2-6 show the results when the Build-up routine is applied to the results

of Frugal, Sequential Fill, 3-coloring, Double and Redouble, and Double and Cover respectively.

An interesting note is that neither the results from the Sequential Fill algorithm for odd values of n nor the results from the Double and Redouble algorithm could be built-up to a cover of higher cardinality. However, this does not imply that the ultimate cover obtained by these algorithms is the largest obtainable for that value of n. Covering sets found by other algorithms build-up to greater cardinalities for the same values of n. For instance, both the Sequential Fill algorithm and the Double and Redouble algorithm for n = 11 yielded a cover of cardinality 906. The Frugal algorithm, on the other hand, yielded a cover of cardinality 917 for n = 11 and was further built-up to 927.

It is also interesting to note that the Build-up routine increased the cardinality of the cover obtained by the Double and Cover method to that of the cardinality of the Double and Redouble method.

It is possible that a k-for-j exchange of nodes between S and B_n - S, where k > j > 1, could result in a cover of higher cardinality for the Sequential Fill sets for n odd or the Double and Redouble sets for any n. The same is true for the largest cover obtained by Build-up for any n. However, no program which would accomplish such an exchange has been written for this problem. We leave this for future researchers to attempt.

TABLE III.2

BUILD-UP APPLIED TO FRUGAL ALGORITHM

n	Upper Bound	Frugal	Frugal & Build-up
1	Ø	Ø	Ø
2	1	1	1
3	2	2	2
4	6	6	6
5	12	12	12
6	32	25	25
7	64	53	54
8	128	88	111
9	256	217	226
10	512	379	452
11	1024	917	927

TABLE III.3

BUILD-UP APPLIED TO SEQUENTIAL FILL ALGORITHM

n	Upper Bound	Sequential Fill	Seq. Fill & Build-up
1	Ø	Ø	Ø
2	1	1	1
3	2	2	2
4	6	5	6
5	12	12	12
6	32	21	27
7	64	54	54
8	128	85	113
9	256	224	224
10	512	341	461
11	1024	906	906

TABLE III.4

BUILD-UP APPLIED TO 3-COLOR ALGORITHM

n	Upper Bound	3-Color	3-Color & Build-up
1	Ø	Ø	Ø
2	1	1	1
3	2	2	2
4	6	5	6
5	12	1Ø	12
6	32	21	27
7	64	42	54
8	128	85	112
9	256	17Ø	224
1Ø	512	341	453
11	1Ø24	682	911

TABLE III.5

BUILD-UP APPLIED TO DOUBLE & REDOUBLE ALGORITHM

n	Upper Bound	Double & Redouble	Double & Redouble & Build-up
1	0	0	0
2	1	1	1
3	2	2	2
4	6	6	6
5	12	12	12
6	32	27	27
7	64	54	54
8	128	112	112
9	256	224	224
10	512	453	453
11	1Ø24	9Ø6	9Ø6

TABLE III.6

BUILD-UP APPLIED TO DOUBLE & COVER ALGORITHM

n	Upper Bound	Double & Cover	Double & Cover & Build-up
1	Ø	Ø	Ø
2	1	Ø	1
3	2	2	2
4	6	5	6
5	12	12	12
6	32	25	27
7	64	52	54
8	128	1Ø5	112
9	256	212	224
1Ø	512	425	453
11	1Ø24	852	9Ø6

H. CONCLUSIONS ON MAXIMAL COVERINGS

In comparing the algorithms developed to form a maximal or near-maximal cover, we look first at the cardinalities of the sets produced by the various algorithms. Table III.7 shows that for $n \leq 1Ø$, the Double and Redouble algorithm yields the largest covering set produced by any of the basic algorithms. It was shown in section III.F.1 that the ratio of the number of nodes in the cover to the number of nodes in the graph tends to 4/9 for the cover formed by Double and Redouble. This is very near the upper bound of 1/2. For odd n, the performance of Double and Redouble is equaled by Sequential Fill.

The performance of the Frugal algorithm is also noteworthy. While the Frugal algorithm performs only moderately well for even values of n, it performs quite well

TABLE III.7

RESULTS OF BASIC MAXIMAL ALGORITHMS

n	Upper Bound	Frugal	Seq. Fill	3-Color	Double & Redouble	Double & Cover
1	Ø	Ø	Ø	Ø	Ø	Ø
2	1	1	1	1	1	Ø
3	2	2	2	2	2	2
4	6	6	5	5	6	5
5	12	12	12	1Ø	12	12
6	32	25	21	21	27	25
7	64	53	54	42	54	52
8	128	88	85	85	112	1Ø5
9	256	217	224	17Ø	224	212
1Ø	512	379	341	341	453	425
11	1Ø24	917	9Ø6	682	9Ø6	852

for n odd. For n = 3, 5, 7, & 9, the cover produced by Frugal is nearly as large as the cover produced by Double and Redouble. For n = 11, the Frugal algorithm produces a cover of greater cardinality than the one produced by Double and Redouble.

By applying the Build-up routine to the covers formed by the basic algorithms, we are able to increase these covers in cardinality. The exceptions are for Double and Redouble and Sequential Fill for n odd. In Table III.8, we see the results of building-up each cover as much as the routine would take it. In each case, we form a cover that is very close in cardinality to the cover formed by Double and Redouble. For $n \geq 8$, we are able to Build-up covers that are actually larger than the ones produced by Double and Redouble. However, they are larger by only a small amount. This leads us to believe that while Double and Redouble

TABLE III.8

BUILD-UP APPLIED TO THE BASIC MAXIMAL ALGORITHMS

n	Upper Bound	Frugal	Seq. Fill	3-Color	Double & Redouble	Double & Cover
1	0	0	0	0	0	0
2	1	1	1	1	1	0
3	2	2	2	2	2	2
4	6	6	6	6	6	6
5	12	12	12	12	12	12
6	32	25	27	27	27	27
7	64	54	54	54	54	54
8	128	111	113	112	112	112
9	256	226	224	224	224	224
10	512	452	461	453	453	453
11	1024	927	906	911	906	906

does not produce a cover of maximum cardinality for all n, the upper bound on the cardinality of a cover of Bn is closer to 4/9 than it is to 1/2.

Let us now turn our attention to the cost of producing these covers. In Table III.9, we see that all of the basic algorithms, except Frugal, cost O(V) time to perform. Frugal costs $O(V^2)$ time. However, to increase the cardinality of some of these covers, the Build-up routine was applied. Each application of the Build-up routine costs $O(V^2)$ time. This drives up the cost of producing these covers to at least $O(V^2)$. The covers produced by Frugal, for n even, 3-Color and Sequential Fill, for n even can be built-up O(V) times. When this is done, the overall cost of producing the cover goes up to $O(V^3)$.

TABLE III.9

COST COMPARISON OF MAXIMAL ALGORITHMS

Algorithm	Cost	Cost to Build-Up
Frugal		
a. n Even	$O(V^2)$	$O(V^3)$
b. n Odd	$O(V^2)$	$O(V^2)$
Sequential Fill		
a. n Even	$O(V)$	$O(V^3)$
b. n Odd	$O(V)$	----
3-Color	$O(V)$	$O(V^3)$
Double & Redouble	$O(V)$	----
Double & Cover	$O(V)$	$O(V^2)$

Thus, considering both the cardinality of the cover obtained and the cost of producing that cover, the Double and Redouble algorithm is the method of choice for producing near-maximal covers. For odd n, the Sequential Fill algorithm is equally desirable.

IV. MINIMAL COVERINGS

A. INTRODUCTION

The goal of this chapter is to find a cover of B_n whose cardinality approaches the lower bound established in section II.D. We first attempt to form such covers by implementing a greedy method. We then present a method of partitioning the nodes in B_n that results in a cover whose cardinality is very near the lower bound.

B. GREEDY

The solution to a wide variety of graph theoretic problems is found by processing the graph in a greedy fashion. For our particular problem of finding a covering set of minimum cardinality in B_n, a greedy-type algorithm seems to be a very logical way to proceed. In forming the covering set S, we include nodes in S that cover the most, as of yet, uncovered nodes. The following algorithm implements this greedy method.

Algorithm: Greedy

Step 1. Select node 1 and either node $2^n - 2$ or node $2^{n-1} - 1$ to be in the set. (This is done to ensure that the resulting set will be a cover.)

Step 2. From the remaining uncovered nodes, add the node with the most uncovered neighbors. In case of a tie, add the smallest such node.

Step 3. If all nodes are covered, the set is a cover, stop. If not, go to step 2.

This algorithm is very similar to the Frugal algorithm presented in section II.B. In the Frugal algorithm, we chose the node with the fewest uncovered neighbors, while the Greedy algorithm chooses the nodes with the most uncovered neighbors.

As in Frugal, we reduce the number of combinations to cover the end nodes by forcing the node 1 to always be in the cover. Once node 1 is in the cover, we have no choice to make on how we cover the node $2^n - 1$ until $n = 4$. For $n = 1$, no cover exists. For $n = 2$, the node 1 forms a cover by itself. For $n = 3$, the node 3 is a descendent of node 1, so node 6 is included in the S to cover node 7.

For $n \geq 4$, there is a choice on how to cover the node $2^n - 1$. However, the only case for $4 \leq n \leq 11$ for which the cardinality of the covering set is affected by the choice is $n = 6$. When $2^6 - 2 = 62$ is included in S, the cardinality of S is 18. When $2^5 - 1 = 31$ is chosen to cover node 63, the cardinality of S is 16.

The cost of performing Greedy is the same as that of Frugal, i.e. $O(V^2)$. Thus, we are able to use this algorithm to find covers of the de Bruijn graph for several values of n.

The results of Greedy for $n < 12$ are seen in Table IV.1. The lower bound for $n < 6$ has been adjusted based on the

results of chapter II. Note that the cardinality of the cover formed by the Greedy algorithm is greater than the lower bound for n = 5. Since the lower bound was achieved for n = 5 by exhaustive search, Greedy is not a minimal algorithm for n = 5.

TABLE IV.1 GREEDY ALGORITHM

n	Lower Bound	Greedy
1	0	0
2	1	1
3	2	2
4	4	4
5	8	9
6	13	16
7	26	39
8	52	78
9	103	158
10	205	313
11	410	605

C. QUARTERING

We form a cover S in B_n by partitioning the nodes in B_n into 4 blocks of equal size.

Block 1. The nodes from 0 to $2^{n-2} - 1$.

Block 2. The nodes from 2^{n-2} to $2^{n-1} - 1$.

Block 3. The nodes from 2^{n-1} to $3*2^{n-2} - 1$.

Block 4. The nodes from $3*2^{n-2}$ to $2^n - 1$.

Place in the set S the nodes in block 1 whose binary representation ends in 01, the nodes in block 2 whose binary representation ends in 00, the nodes in block 3 whose binary representation ends in 11, and the nodes in block 4 whose binary representation ends in 10. Since the number of nodes

selected from each block is 2^{n-4}, there are a total of 2^{n-2} nodes in S, or 1/4 of the nodes in the graph. We will refer to the nodes in S from block i as S_i, i = 1,2,3,4.

In showing that S is an independent set, it suffices to show that the successors of the nodes in S are not also in S. Since the nodes in block 1 are between Ø and $2^{n-2} - 1$, their successors lie between Ø and $2^{n-1} - 1$, i.e. blocks 1 and 2. The successors of a node ending in Ø1 end in either 1Ø or 11. Thus, the nodes in S_1 have successors in blocks 1 and 2 that end in either 1Ø or 11. However, the nodes taken from block 1 in S end in Ø1, while the nodes from block 2 in S end in ØØ. So, the successors of the nodes in S_1 are not in S.

The successors of the nodes in block 2 lie between 2^{n-1} and $2^n - 1$, i.e. blocks 3 and 4. A node ending in ØØ has successors that end in either ØØ or Ø1. Thus, the nodes in S_2 have successors in the blocks 3 and 4 that end in either in ØØ or Ø1. But, the nodes in S_3 end in 11 and the nodes in S_4 end in 1Ø. Therefore, the successors of the nodes in S_2 are not in S.

The successors of the nodes in block 3 lie between 0 and $2^{n-1} - 1$, i.e. blocks 1 and 2. The successors of a node ending in 11 ends in either 1Ø or 11. Thus, the nodes in S_3 have successors in the blocks 1 and 2 that end in either in 1Ø or 11. However, the nodes in S_1 end in Ø1 and the nodes

in S_2 end in ØØ. So, the successors of the nodes in S_3 are not in S.

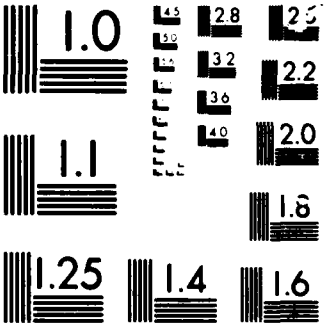
Finally, the successors of the nodes in block 4 lie between 2^{n-1} and $2^n - 1$, i.e. blocks 3 and 4. The successors of a node ending in 1Ø end in either ØØ or Ø1. Thus, the nodes in S_4 have successors in the blocks 3 and 4 that end in either in ØØ or Ø1. But, the nodes in S_3 end in 11 and the nodes in S_4 end in 1Ø. So, the successors of the nodes in S_4 are not in S. None of the successors of the nodes in S are also in S. Therefore, S is an independent set.

As stated earlier, the successors of the nodes in S_1 lie in blocks 1 and 2. Because the nodes in S_1 end in Ø1, the last 3 bits of their successor nodes are Ø1Ø for the even successors and Ø11 for the odd successors. In our partitions, a node and its companion cannot both be in the same block. Therefore, there are 2^{n-4} distinct nodes ending in Ø1Ø and 2^{n-4} distinct nodes ending in Ø11 that are successors of the nodes in S_1. Since this is the number of nodes in blocks 1 and 2 that end in this manner, all the nodes in blocks 1 and 2 ending in Ø1Ø or Ø11 are covered.

The successors of the nodes in S_3 also lie in blocks 1 and 2. They are the 2^{n-4} nodes that end in 11Ø and the 2^{n-4} nodes that end in 111. Thus, the nodes in S_1, together with the nodes in S_3, cover all the nodes in blocks 1 and 2 that end in 1Ø or 11.

AD-A168 474 COVERING THE DE BRUIJN GRAPH(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R D BRYANT MAR 86 2/2

UNCLASSIFIED F/G 12/1 NL



MICROCOPY CHART

In a similar manner, the nodes in S_2, together with the nodes in S_4, cover the nodes in blocks 3 and 4 that end in ØØ or Ø1.

We have yet to show that the following nodes are covered.

A. The nodes in block 1 that end in ØØ.

B. The nodes in block 2 that end in Ø1.

C. The nodes in block 3 that end in 1Ø.

D. The nodes in block 4 that end in 11.

The nodes in the first half of A (the nodes ending in ØØ that lie between Ø and $2^{n-3} - 1$) have successors that lie in block 1. The even successors end in ØØ and the odd successors end in Ø1. Thus, the nodes in the first half of A are covered by the nodes in S_1. The nodes in the second half of A (the nodes ending in ØØ that lie between 2^{n-3} and $2^{n-2} - 1$) have successors that lie in block 2. The even successors end in ØØ and the odd successors end in Ø1. Thus, the nodes in the second half of A are covered by the nodes in S_2. Similarly, the nodes in B are covered by their successors that are in S_3 and S_4, the nodes in C are covered by their successors that are in S_1 and S_2, and the nodes in D are covered by their successors in S_3 and S_4.

Thus, the set S, formed by this quartering scheme, covers B_n. The cardinality of S is only 2^{n-2}, or 1/4 of the nodes in B_n, which is very close to the lower bound of $\lceil 2^n/5 \rceil$. It is noteworthy that the Quartering scheme produced

a cover equal in size to the minimal covers found by the exhaustive search for $n < 6$.

In determining the nodes to be placed in S, it is only necessary that we check the contents of the last two bits in the binary expansion of the nodes in B_n. This can be done in O(V) time. Thus, we have an inexpensive method to form a cover whose cardinality is very near the lower bound.

D. CONCLUSIONS ON MINIMAL COVERINGS

Of the two methods presented to produce minimal or near-minimal covers, clearly the Quartering scheme is the method of choice. In Table IV.2, we see that for $n > 6$, the cardinality of the cover produced by Quartering is considerably less than the cardinality of the cover produced by Greedy. And the cost to perform Quartering is only $O(V)$, while the cost to perform Greedy is $O(V^2)$. While the Greedy method seemed to be a very natural way to proceed, Quartering yields much better results.

TABLE IV.2 RESULTS OF MINIMAL ALGORITHMS

n	Lower Bound	Greedy	Quartering
1	0	0	0
2	1	1	1
3	2	2	2
4	4	4	4
5	8	9	8
6	13	16	16
7	26	39	32
8	52	78	64
9	103	158	128
10	205	313	256
11	410	605	512

V. FURTHER STUDIES

In exploring the bounds on the cardinality of covering sets of the de Bruijn graph, many questions remain unanswered.

In investigating maximal covers, the parity of n played a much greater role than expected. The effect that the parity of n has on Sequential Fill, Double and Redouble, and Double and Cover is understood to a degree. However, the difference in the performance of the Frugal algorithm between even and odd n is, as of yet, unexplained.

Frugal finds the node in B_n that is adjacent to the fewest, as of yet, uncovered nodes and adds this node to the covering set. In the event of a tie, the smallest such node is taken. Perhaps, other tie-breaking schemes might improve the performance of Frugal, especially for n even. A Monte Carlo scheme that randomly chooses from among the tieing nodes might be implemented.

In the Greedy algorithm, the node in B_n that is adjacent to the most, as of yet, uncovered nodes is added to the cover. Ties are also decided in Greedy by choosing the smallest node involved in the tie. The performance of Greedy might also be improved by employing different tie-breaking schemes.

Another area for further research is the Build-up routine. Presently, the Build-up routine makes a k-for-1 exchange between the nodes in $B_n - S$ and S in an attempt to increase the cardinality of the cover S. An algorithm that makes a k-for-j exchange, where $k \geq j \geq 1$, might be able to further increase the cardinality of the cover to a level higher than the present Build-up routine does.

A process that might yield insight to both the upper and lower bounds, but has not been studied as of yet, is Random Fill. In this process, nodes are chosen at random and are placed in a cover if no conflict exists with any previously chosen nodes. Perhaps, by performing Random Fill many times for a given value of n, more can be learned about the extremes of the range of values that the cardinality of a cover can take on. We can also apply the Build-up routine to the results of Random Fill. Since the Build-up routine produces slightly different results for the different covers that were used as input, Random Fill might produce a cover that will build to a higher cardinality than previously obtained for that n.

In our research, the lower bound was not explored quite as fully as the upper bound. This is due, in part, to the fact that a Build-down routine was never developed. Such a routine would take a covering set S and from it form a cover S' of lesser cardinality. With this tool, we could further probe the lower bound.

However, it is felt that the lower bound on the cardinality of a cover of B_n should be 2^{n-2} nodes, or 1/4 of the nodes in the graph.

As noted in section I.C, the nodes in B_n have a 1-to-1 correspondence to the arcs in B_{n-1}. Consider the conditions that are imposed on a set of arcs in B_{n-1} that correspond to a covering set nodes in B_n. In these conditions, there seems to be a relationship between a minimal covering set of nodes in B_n and the arcs in B_{n-1} that deal with a graph theoretic concept known as maximal matching. It is not difficult to prove that nodes in B_n that form the Quartering cover correspond to arcs in B_{n-1} that form a maximal matching. However, as of yet, a proof has not been made which shows that the number of nodes in a cover in B_n can be no less than the number of arcs that make a maximal matching in B_{n-1}.

Certainly, a suitable topic for further research is to prove or disprove that a cover in B_n cannot be formed with fewer than 2^{n-2} nodes.

LIST OF REFERENCES

1. de Bruijn, N.G., "A Combinatorial Problem," Koninklijke Nederlands Akademie van Wetenschappen, Proceedings, v. 49 (part 2), pp. 758-764, 1946.

2. Golomb, S.W., Shift Register Sequences, Holden-Day, 1967.

3. Evan, S., Graph Algorithms, p.6, Computer Science Press, Inc., 1979.

4. Lempel, A., "On Extreme Factors of the de Bruijn Graph," Journal of Combinatorial Theory (B), v. 11, pp. 17-27, 1971.

5. Mykkeltveit, J., "A Proof of Golomb's Conjecture for the de Bruijn Graph," Journal of Combinatorial Theory (B), v.13, pp. 40-45, 1972.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 53Lz Department of Mathematics Naval Postgraduate School Monterey, California 93943-5000	1
4.	Prof. H.M. Fredricksen, Code 53Fs Department of Mathematics Naval Postgraduate School Monterey, California 93943-5000	12
5.	Prof. C.O. Wilde, Code 53Wm Department of Mathematics Naval Postgraduate School Monterey, California 93943-5000	1
6.	Captain R.D. Bryant USMC Marine Aviation Detachment Pacific Missle Test Center Point Mugu, California 93042-5010	4

END

DTIC

7-86