THE SNAKE-IN-THE-BOX PROBLEM: A PRIMER

by

THOMAS E. DRAPELA

(Under the Direction of Walter D. Potter)

ABSTRACT

This thesis is a primer designed to introduce novice and expert alike to the *Snake-in-the-Box* problem (SIB). Using plain language, and including explanations of prerequisite concepts necessary for understanding SIB throughout, it introduces the essential concepts of SIB, its origin, evolution, and continued relevance, as well as methods for representing, validating, and evaluating snake and coil solutions in SIB search. Finally, it is structured to serve as a convenient reference for those exploring SIB.

INDEX WORDS: Snake-in-the-Box, Coil-in-the-Box, Hypercube, Snake, Coil, Graph Theory, Constraint Satisfaction, Canonical Ordering, Canonical Form, Equivalence Class, Disjunctive Normal Form, Conjunctive Normal Form, Heuristic Search, Fitness Function, Articulation Points

THE SNAKE-IN-THE-BOX PROBLEM: A PRIMER

by

THOMAS E. DRAPELA

B.A., George Mason University, 1991

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2015

THE SNAKE-IN-THE-BOX PROBLEM: A PRIMER

by

THOMAS E. DRAPELA

Major Professor:	Walter D. Potter
Committee:	Khaled Rasheed
	Pete Bettinger

Electronic Version Approved:

Julie Coffield
Interim Dean of the Graduate School
The University of Georgia
May 2015

DEDICATION

To my dearest Kristin: For loving me enough to give me a shove.

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to Dr. Potter for introducing me to the Snake-in-the-Box problem, for giving me the freedom to get lost in it, and finally, for helping me to find my way back. Heartfelt thanks go, too, to Dr. Michael Covington for (repeatedly) reminding me that a thesis is *just* a thesis (and not a dissertation).

I also thank the members of my committee for their valuable insights and feedback regarding my research and its presentation herein—and perhaps more importantly for still remembering me after I took so long to complete my program of study. I must also thank the Graduate School for granting me the necessary extensions of time so that I might at last graduate.

Finally, thanks go out to all my fellow "snake hunters," especially Karthik Nadig (M.S., A.I. '12), Tom Horton (Ph.D., C.S. in progress), and Ananta Palani (M.S., A.I. '10) for making our discussions in the "Snake Pit" memorable and meaningful; and to Seth Meyerson (M.S., C.S. '15) for drawing me across the finish line.

I am the better for knowing you all.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

AN INTRODUCTION TO THE SNAKE-IN-THE-BOX PROBLEM

This thesis is a primer designed to introduce novice and expert alike to the *Snake-in-the-Box* problem (SIB). As such, its objectives are manifold.

First, it presents in plain language the essential aspects of the problem, while easing the reader into the lingo of SIB. As necessary, introductions to prerequisite concepts are included. Furthermore, for ease of reference, relevant terms appear **bolded** when defined.

Second, it untangles much of the literature devoted to the problem—and reported since SIB's introduction in 1958—presenting the problem in historical and technical contexts. This includes, addressing conflicts in the SIB lexicon, organizing research into general approaches, and discussing the paired origins of the problem.

Third, it presents methods for representing both the SIB search space and potential solutions, for validating and manipulating potential solutions, exploiting symmetry to reduce search effort, as well as evaluating in-progress solutions and remaining search space for continued growth potential.

Lastly, it is written to be easy—even enjoyable—to read.

1.1 A SIMPLE ANALOGY

At its simplest, SIB is a puzzle game. The objective of this game is to find the longest possible path (the snake) which may be plotted along the edges of a hypercube graph (the box). The path follows special rules (constraints) which make the puzzle more

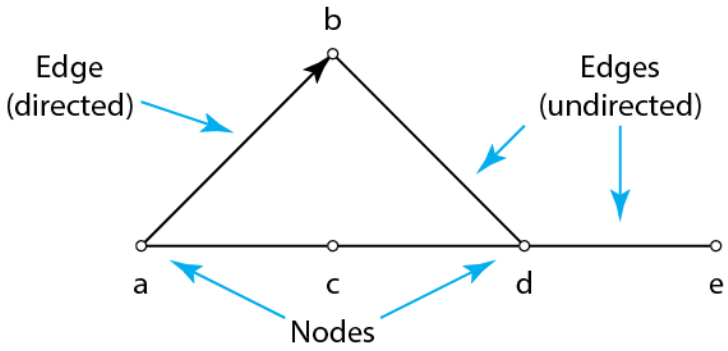

Figure 1: A sample graph with nodes and edges indicated.

challenging. This game has an unlimited number of levels, and a winner is declared for each level solved. Each new level is exponentially more difficult to solve than the previous level. In fact, years may pass between an individual or team discovering a level's longest path and the verification of the win.

Equipped with this simplest understanding of SIB, let us leave this analogy behind and delve into the problem in greater detail.

1.2 SIB

SIB is a *graph theory* problem concerned with finding the longest possible *induced path* that can be plotted along the edges of an *n*-dimensional *unit hypercube graph*. It is also a *constraint satisfaction problem*, due to the constraints placed on the path.

Graph theory is concerned with the study of mathematical graphs; structures that represent associations among a collection of interrelated objects. In a graph, each object is represented by a vertex, or *node*, and each relationship—between pairs of objects—is indicated by a line, or *edge*, connecting the objects together. Edges may be directed (one-way) or undirected. Figure 1 illustrates. In SIB, all hypercube graph edges are undirected.

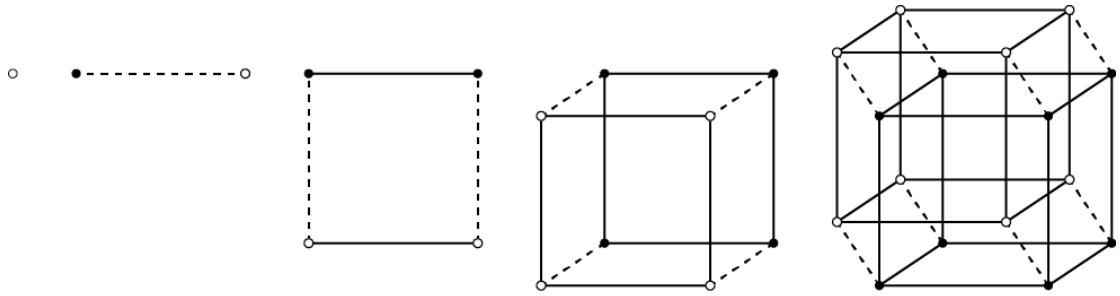

Figure 2: Hypercube graphs for $0 \leq n \leq 4$ (left to right respectively). Solid nodes joined by solid edges show previous n-cube in current n-cube. Hollow nodes joined by solid edges show duplication of previous n-cube in current n-cube. Dashed edges show new edges in the current n-cube.

1.2.1 THE HYPERCUBE

A **hypercube graph** is a graph whose nodes and edges are arranged such that they form a *geometric hypercube*. From geometry, we are all familiar with points, lines, squares, and cubes. We understand too that these objects have different degrees of dimensionality. Cubes are 3-dimensional, squares are 2-dimensional, lines are 1-dimensional, and points have 0 dimensions. Using n to represent dimension, the **geometric hypercube** (or n-cube) is the n-dimensional analog of all these objects and more. For shapes having more than 3 dimensions, names are less familiar, increasingly complicated, or even unassigned. To keep things simple, the term n-cube is used to refer to each hypercube by its dimensionality. Hence, the 0-cube describes a point, the 1-cube a line, the 2-cube a square, the 3-cube a cube, and so on. Figure 2 illustrates. As a graph, the vertices of the geometric hypercube are analogous to the nodes of the hypercube graph. Edges remain edges.

Every **hypercube** has 2^n total vertices. Each of these vertices is connected to n neighboring vertices via n edges—edges which are all orthogonal (perpendicular) to each

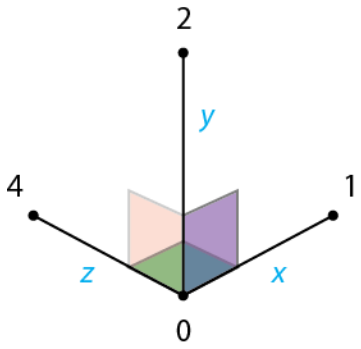

Figure 3: Example of orthogonal (mutually perpendicular) edges. $z \perp y \perp x \perp \cdots$

other. Figure 3 illustrates. This relation can be seen easily in the 2-cube and 3-cube, but is harder to conceptualize in higher dimensions.

A graph wherein every node connects to the same number of neighbors is called a **regular graph**. The hypercube graph is a regular graph. It is also a **bipartite graph**, meaning it is possible to divide its nodes into two disjoint (exclusive) sets such that every edge connects a pair of disjoint nodes. Figure 4 illustrates.

Finally, a **unit hypercube** is one whose edges are all 1 *unit* in length. A **unit** is an arbitrary expression of length that does not imply any specific unit of measure, but rather the *uniformity* of lengths; in this case, of edges.

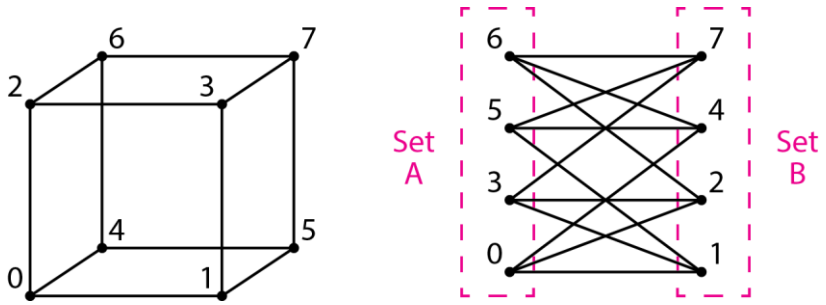

Figure 4: Hypercube graphs are regular (both), and bipartite (right).

Figure 5: Hamming distance. The Hamming distance of each comparison (left to right) is 2, 4, 3, and 1, because each pair of strings differs in that number of positions respectively.

1.2.2 The Path

Borrowing a term from Information Theory, **Hamming distance** is a count of the number of differences between two strings of data of the same length [Hamming 1950]. Figure 5 illustrates. When those strings are the binary node labels of a hypercube graph, Hamming distance indicates the distance between the nodes in the graph—by counting the number of non-matching bits between the nodes. (Binary labels are discussed in more detail in Section 1.2.3.) In the unit hypercube, the Hamming distance between any two adjacent nodes is 1, and the distance between any two non-adjacent nodes is greater than 1; or more specifically: equal to the fewest number of edges which must be traversed in

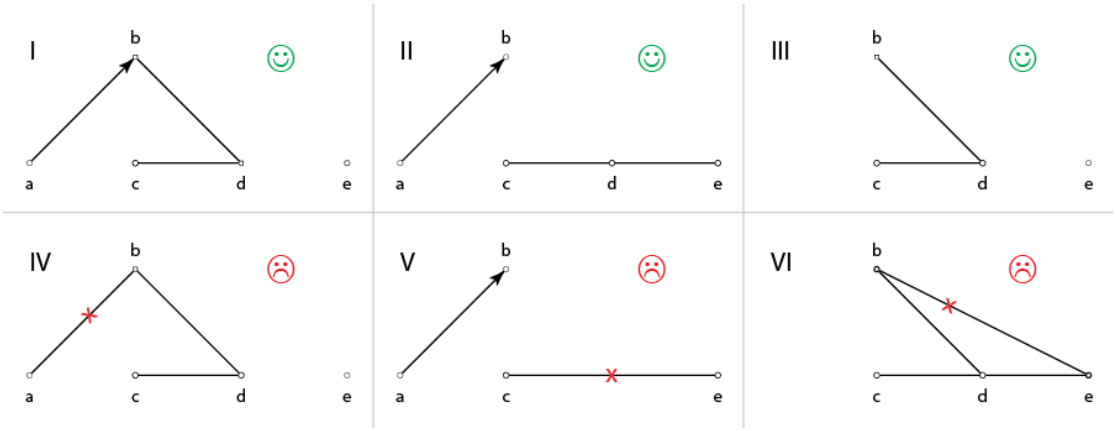

Figure 6: Sample subgraphs. I, II, and III are valid subgraphs of the sample graph in Figure 1. IV, V, and VI are invalid subgraphs of same. An ‘x’ indicates the violations in each invalid subgraph.

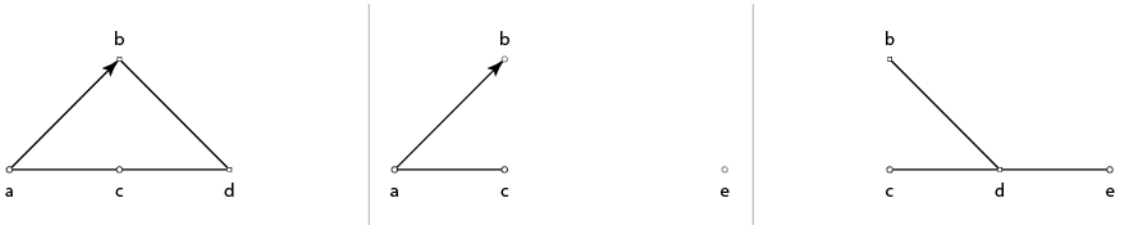

Figure 7: Sample induced subgraphs. These three subgraphs are valid induced subgraphs of the sample graph in Figure 1.

order to bridge the distance between the nodes. In the n-cube, the maximum distance between any two nodes is n.

A **subgraph** is a graph whose nodes and edges form a subset of a larger graph. That is, the entire subgraph exactly matches all or part of the larger graph. Figure 6 illustrates. An **induced subgraph** is a subgraph in which every pair of nodes in the subgraph is connected by an edge if and only if the pairs are similarly connected in the larger graph. That is, edges must exist in the subgraph wherever edges exist in the larger graph. Figure 7 illustrates.

A **path** is an ordered set of connected nodes or edges in a graph. A path is a type of subgraph. An **induced path** is a type of induced subgraph in which no two non-

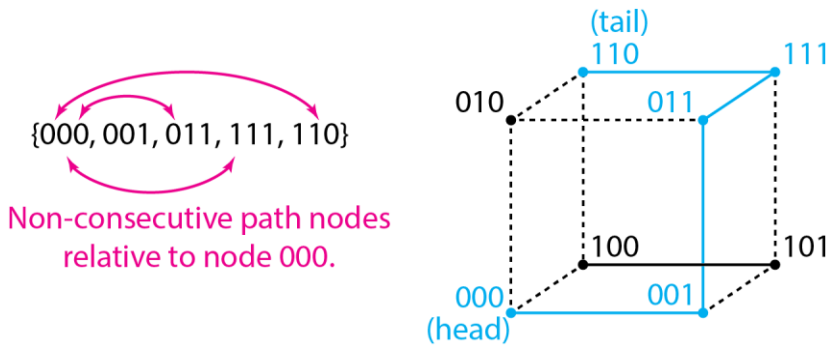

Figure 8: Sample induced path as an ordered set (left) and in the 3-cube graph (right). In the 3-cube, the dashed edges here indicate chords connecting to non-path nodes.) In an induced path all non-consecutive path nodes are ≥ 2 distant.

adjacent nodes in the path are themselves connected in the n-cube. Such additional adjacencies are called *chords*. As such, an induced path is also known as a **chordless path** or an **achordal path**. The ordered set in Figure 8 illustrates. Furthermore, as with the hypercube graph, the Hamming distance between the consecutive nodes in an induced path is always 1, while the Hamming distance between non-consecutive nodes in the path is always greater than 1. This *minimum distance* between non-consecutive path nodes is known as **spread**. An induced path is further known as a **spread 2 path**; meaning that all non-consecutive nodes in the path maintain a distance *not less* than 2. The 3-cube graph in Figure 8 illustrates. The *maximum distance* between any two nodes in n is n, hence spread n is the maximum possible spread in n. SIB is concerned with spread $k = 2$ paths; however, paths of $k > 2$ are also sought. Such searches remain within the domain of SIB, as all spread j paths, for $j > k$, are subsets of spread k paths [Singleton 1996].

The **length** of a path is equal to the number of edges it describes when plotting the path. The initial node of a path is its *head*, and the terminal node of a path is its *tail*. A path whose head and tail are adjacent is a *closed* path; otherwise, it is an *open* path.

In SIB, an open induced path is a **snake**, and a closed induced path is a **coil**. Coils which may be split down the middle into two identical snake subsequences are **symmetrical coils**. Symmetrical coils are also referred to as **double coils**. Those who exclusively search for coils in the hypercube sometimes refer to SIB as the *Coil-in-the-Box* problem [Casella & Potter 2005c]. Any snake which can be extended is a sub-snake of a longer snake or coil. A snake which can no longer be extended is a **maximal snake**. Likewise, a coil which can no longer be extended is a **maximal coil**. The longest of the maximal snakes and coils in dimension n are called the **longest maximal snake** and

Table 1: Path representations compared.

Notation Type	**Sequence notations for the same snake in the 4-cube**
Node Sequence (integer labels)	0, 1, 3, 7, 6, 14, 12, 13
Node Sequence (binary labels)	0000, 0001, 0011, 0111, 0110, 1110, 1100, 1101
Transition Sequence	0, 1, 2, 0, 3, 1, 0
Binary Sequence	1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0

longest maximal coil—or the *absolute snake bound* and *absolute coil bound* for n—respectively. Finding the absolute bounds for n is the goal of SIB.

1.2.3 Node Labeling and Path Representation

For plotting paths, it is helpful for each node of the n-cube graph to be uniquely identifiable. One common convention is to use a *Gray code* numbering scheme which assigns a unique n-bit binary label to each graph node such that each label differs from its n neighboring nodes by exactly one bit. In fact, when constrained to binary values, the Gray code numbering method by default describes a hypercube. The 3-cube in Figure 8 illustrates. For any n-cube, binary node labels will run consecutively from 0 to $(2^n - 1)$. Because of the regularity (symmetry) of the hypercube, any node may be designated as Node 0. Thus, an arbitrary node is first designated as Node 0 with all other nodes labeled relative to it.

There are three common notations for representing an induced path on a hypercube graph. Two explicitly describe a path from head to tail, while the third's description is implicit and requires additional processing to extract an explicit path. The two explicit representations are the *node sequence* and the *transition sequence*. The implicit representation is the *binary sequence*.

In **node sequence** (NS) notation, a path is described from head to tail by an ordered list of the labels of the nodes through which it passes. Table 1 illustrates. Node values range from 0 to $2^n - 1$. Path length (number of edges) in NS is always one less than the length of the node sequence.

In **transition sequence** (TS) notation, a path is described from head to tail by the position of the bit which changes between the labels (in binary) of the nodes through which its passes. Table 1 illustrates. As Gray code node labels are each n-bits long, the position of each bit in the label can be likened to a dimension of travel in the n-cube—i.e., a dimensional transition. Transition values range from 0 to $n - 1$. Path length (number of edges) in TS is equal to the length of the transition sequence itself.

Both NS and TS representations have their advantages and limitations. As *path encoding schemes*, both describe complete paths; with a NS describing a snake exactly as it appears in the hypercube, allowing any point along a path to be easily identified within the hypercube; and a TS generalizing the node references, allowing for the identification of patterns present within the relative *shape* of a path.

In **binary sequence** (BS) notation, a path is indirectly described via a bit string that indicates which nodes of the hypercube are included in the path. The BS contains 2^n bits, one for each node in the n-cube. For each node through which the path passes, its corresponding bit is set to 1. All other bits are set to 0. The order in which the path passes through the nodes is not encoded in BS—though paths generally start at Node 0—so additional processing is required to extract an explicit path—typically to either NS or TS.

Directly representing the hypercube and not a path, BS representation is a *search-space reduction* representation, effectively limiting the size of the search space (the

hypercube graph) to a subgraph of the whole. As such, the BS is not necessarily limited to describing a single path; and conversely, a path need not be described by all of the "1" bits in the BS. In cases where it is guaranteed that a BS contains the minimal number of "1" bits to describe a single path, the number of edges in the BS path is equal to one less than the *cardinality* of the binary sequence. **Cardinality** here refers to the number of "1" bits set in a binary sequence. The BS in Table 1 has a cardinality of 8.

1.3 The Challenge

At a glance, the SIB problem may appear to be a simple puzzle with a simple (enough) solution. However, this is an elegant deception, as SIB falls into a category of problems which suffer from *combinatorial explosion.*

Combinatorial explosion refers to the doubling—at least—of a measurable complexity for every occurrence of a measurable increment. For SIB this doubling occurs in its search space. As n increases linearly, the dimensional complexity of the hypercube grows exponentially. Because of this, SIB is an excellent proving ground for testing the mettle of would-be innovative heuristic search techniques. But the significance of SIB is not limited to being a challenging sandbox.

1.4 Significance

Knowing the maximum number of elements which may be packed into a code type is useful in code design [Kautz 1958]. Longer SIB codes mean greater error detection accuracy in the systems which use them [Paterson & Tuliani 1998]. This is the chief motivation behind SIB.

Today, SIB codes continue to find relevant application in many science and engineering fields. Some of these areas include: coding theory, electrical engineering,

Table 2: Current lower bounds of the Snake-in-the-Box problem ($k = 2$). Shaded cells denote proven solutions (absolute bounds) for n [Potter 2015].

Dimension (n)	**Length**		
	Snakes	**Coils**	**Symmetrical Coils**
1	1	0	0
2	2	4	4
3	4	6	6
4	7	8	8
5	13	14	14
6	26	26	26
7	50	48	46
8	98	96	94
9	190	188	186
10	370	358	362
11	707	668	662
12	1302	1276	1222
13	2520	2468	2354
14+			

analog-to-digital conversion, precision high speed rotational sensors, pattern recognition and classification [Preparata & Nievergelt 1974], disk sector encoding [Blaum & Etzion 2002], charge modulation schemes for multi-level flash memory [Yehezkeally & Schwartz 2011], and systems biology, particularly in gene regulatory networks modeling [Zinovik et al. 2008].

As a testament to its enduring challenge, since the introduction of SIB by [Kautz 1958], only dimensions $n \leq 8$ have been definitively solved; with solutions for $n = 8$ snakes [Carlson & Hougen 2010] and coils [Paterson & Tuliani 1998] being proved by [Östergård & Pettersson 2014b] and [Östergård & Pettersson 2014a] respectively. Previously, solutions for $n = 7$ snakes [Potter et al. 1994] and coils [Eastman via Even 1963] were proved by [Kochut 1996]. Table 2 lists the current snake, coil, and

symmetrical coil lower bounds for $1 \leq n \leq 13$. For lower bounds of spreads $k > 2$, [Hood et al. 2013] includes a recently updated table.

CHAPTER 2

HISTORY OF THE SNAKE-IN-THE-BOX PROBLEM

This chapter presents a brief history of SIB, what lead to its development and the continuing search to solve it for ever increasing values of n.

2.1 PRELUDE

The transmission of messages (data) over distances without the physical exchange of a tangible medium, such as a letter or photograph, is known as *telegraphy*. Data transmission via telegraphy requires messages to be encoded using methods which are both appropriate to a given telegraphic medium and known to the sender (encoder) and receiver (decoder) alike. Telegraphy—and methods of encoding messages—have existed since ancient times. From the humble beginnings of bonfire beacons and smoke signals, through interim developments like flag semaphore and the heliograph, telegraphy came of age with the introduction of electrical and wireless telegraphy in the late 19^{th} and early 20^{th} centuries. Out of these last developments, and up through the present day, the march of technology has continued to increase telegraphy's capacities to handle *more* messages of *greater* complexity at *faster* transmission speeds.

2.1.1 DIGITIZATION

To share a single wired or wireless connection, messages must be woven together for transmission. Enter *sampling*. **Sampling** is the process of digitally representing a continuous (analog) signal—or data stream—by reducing it to a discrete (digital) series of snapshots. The first data samplings were performed in order to interlace messages from

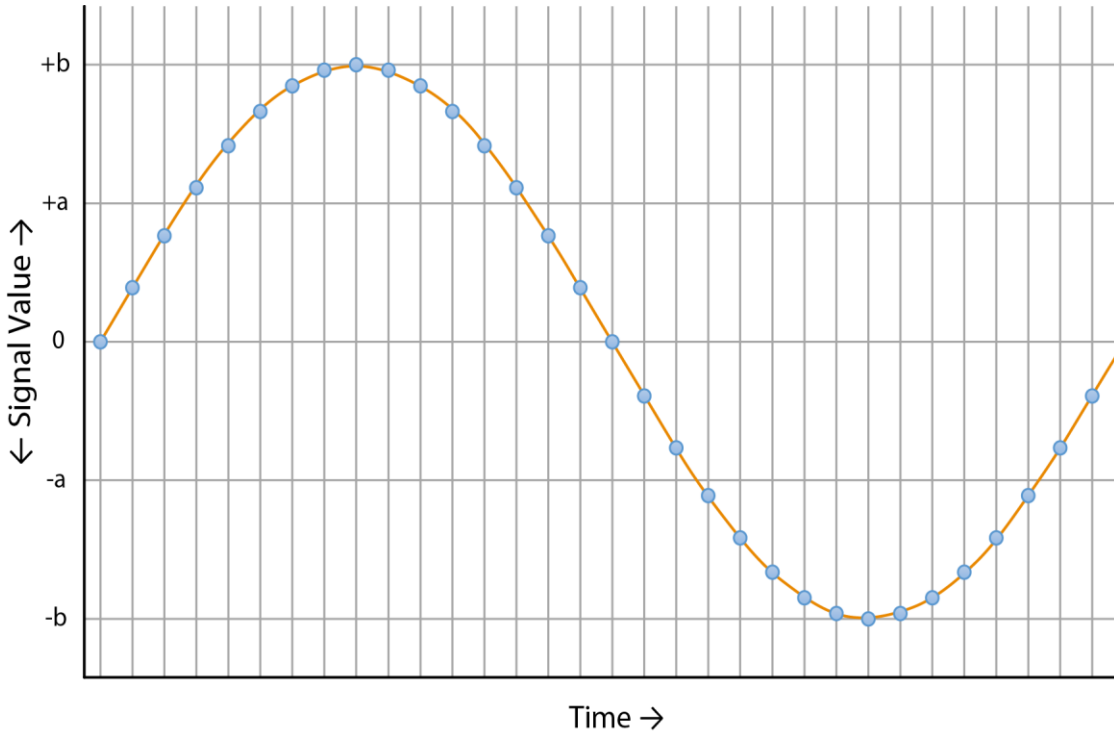

Figure 9: Digitization of an analog signal. The smooth line represents an analog signal, while the dots indicate the digital sampling of the signal at a fixed sampling rate.

multiple sources for transmission along a single shared wire. There are a number of methods for sampling data streams. One such method is *pulse code modulation*.

Pulse code modulation (PCM) entails recording the values of a continuous signal a number of times per second [Reeves 1942] [Pierce 1948] [Oliver & Shannon 1957]; reducing the continuous signal to a series of discrete snapshots—similar to how moving objects appear when illuminated by a strobe light. How often these values are recorded or *sampled* is called a sample's **sampling rate**, and the total number of "words" (commonly: bits) available for recording each sampled value is a sample's **bit depth**. Figure 9 illustrates.

A **bit** (binary digit) is a basic unit of information capable of expressing one of two values: 0 or 1. By stringing together a number of bits, a range of numeric values beyond

Table 3: Numerical encodings compared.

Decimal	Binary	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

the scope of a single bit may be expressed using binary encoding. For example, a string of three bits can express numeric values of 0 to 7. Table 3 illustrates.

The act of changing a bit from 0 to 1 (or 1 to 0) is commonly called a *bit flip*, due to a bit being readily likened to a simple (two position) switch—like a light switch. One flips switches, hence one also flips bits.

Exactly how a bit holds information varies by medium. Anything which can express two measureable states (on/off, up/down, present/absent, high/low, etc.) may serve as a bit. The earliest devices used various mechanical means; moving parts which with repeated use became increasingly error prone over time.

2.1.2 Opportunities for Error

For devices that use mechanical means to flip bits, the conventional binary numbering method is not an optimal encoding—as fully half of all numeric operations performed using the method require (at least) twice as many bit flips as the other half, which only require one. Specifically, every even-to-odd increment and odd-to-even decrement requires two or more bit flips [Gray 1953]. With half of all bit operations being more costly, and increasingly so the larger the numeric values being calculated, the

opportunity for error is not only increased but skewed toward half of all numeric operations. Additionally, and more significantly, when multiple bit flips are required, it is unlikely that the flips will all occur in perfect synchrony. Within the interval it takes for all flips to complete, it is possible—more so with slower media—for bit strings to register spurious values. For instance, to flip 000 to 111 requires three bit flips, an operation during which it is conceivable for the bit string to spuriously register any of the six other 3-bit values.

Out of these issues arose research into reliable error-detecting and error-correcting methods with applications in both computing hardware and software. Two initial breakthroughs came in the form of the Gray code and Hamming codes.

The reflected binary code or *Gray code*—named for Frank Gray who first described the encoding method in his 1947 patent (issued 1953) [Gray 1953]—is an alternate binary numbering method which requires only a single bit flip for any individual binary increment or decrement operation. The Gray code both equalized the opportunity for error across all bit flip operations, and alleviated the concern over spurious readings during bit flips. Table 3 illustrates.

Hamming codes—introduced by [Hamming 1950]—are a foundation of modern error-checking and error-correcting theory, and implement codes with spreads greater than 1 to reliably detect (and in some cases correct) errors in transmitted digital data.

This essentially was the state of things when SIB was introduced.

2.2 Genesis

SIB was first described by [Kautz 1958], and originated with the application of Hamming's error-detecting codes to measurement and analog-to-digital conversion

systems which used Gray codes [Adelson et al. 1973a] [Paterson & Tuliani 1998]. Kautz observed that as (1) a set of binary code elements with the (Hamming) single-error-detecting property was represented by a set of disjoint nodes in the unit n-cube, and (2) a set of binary code elements with the unit-distance property (Gray code) was represented by an ordered set of adjacent nodes in the unit n-cube; then a set of binary code elements possessing both properties was represented by *an ordered set of adjacent nodes of which all non-successive nodes were disjoint* [Kautz 1958].

Knowing the maximum number of elements which may be packed into a code type is useful in code design [Kautz 1958]. Longer codes mean greater accuracy in the systems employing them. Such maximums were already known for both Hamming's error-checking codes [Hamming 1950] and Gray's unit-distant codes [Gray 1947]. So the next step was to determine similar maximums for Kautz's new unit-distance error-checking codes—an ongoing effort which continues to this day and is known as SIB.

2.2.1 LEXICAL EVOLUTION

Kautz, while parenthetically coining the term "snake" based on the interpretation of these paths as forming unit-radius tubes in the hypercube, mainly referred to paths as *chains* and differentiated open and closed paths as *segments* and *cycles* respectively. However, these latter terms did not completely take hold, and for some time thereafter each new publication effectively proffered its own SIB lexicon. As attention oscillated between closed and open paths, the lack of established terminology proved cumbersome—particularly across disciplines, where discussions of the problem could be undertaken using wholly unfamiliar or conflicting vocabularies.

Thirty years passed before some manner of consensus was reached in the SIB research community. Much of the modern SIB lexicon was codified (so to speak) by [Harary et al. 1988], in their comprehensive survey of hypercube graph theory. Most notably, Harary defined the closed path as a *coil* and fixed the open path as a *snake*.

2.2.2 SIB_k

The subdomain of spread $k > 2$ path search originated with [Singleton 1966] who opened the problem to higher spreads as SIB_k. Higher spread SIB_k codes offer increased error-detection precision [Klee 1967] [Douglas 1969a] as well as possibilities for error correction [Paterson & Tuliani 1998] [Klee 1970b]—but at additional computational expense. SIB_k codes remain a viable subdomain of SIB, but are beyond the scope of this primer.

2.3 Parallel Genesis

Interest in induced paths in n-cubes did not originate solely out of error-checking theory. About the same time in the then Soviet Union, math theorists—having recognized the apparent unavoidability of *exhaustive search* (*perebor* in Russian) under certain circumstances—were actively pursuing a mathematical proof of this inevitability [Trakhtenbrot 1984]. As part of this effort, Zhuravlev and Vasil'ev [Vasil’ev 1963] were effectively using SIB codes in their research to explain the difficulties in minimizing *disjunctive normal forms* of Boolean functions using local algorithms [Evdokimov 1969] [Emelyanov & Lukito 2000].

2.3.1 SIB Codes and the Minimization of Disjunctive Normal Form

Disjunctive normal form (DNF) is a normalized form (*canonical form*) for representing propositional formulae in Boolean logic. Here a normalized (normal) form is any formula which may not be further reduced. Essentially, a function or formula is any finite combination of variables and constants joined by a set of operators. A propositional formula is a formula which uses propositional variables and connectives. Propositional variables denote both variables and constants, while propositional connectives denote logic operations—most commonly: conjunction, disjunction, implication, negation, and equivalence. In DNF, only conjunction, disjunction, and negation are used. [Sobolev 2002]

Structurally, DNF is a disjunction of conjunctions. That is, (1) every clause in DNF is connected by a disjunction, and (2) every disjunct (clause) is itself comprised solely of conjunctions of literals (single terms)—each of which includes every variable in the original formula. Any propositional formula may be converted to a DNF, and there are a number of methods to do so. [Reeves 1972:37–40]

As with all disjunctions, for a formula in DNF to be *true*, only one of its disjuncts need be true. Furthermore, given that the disjuncts in DNF each contain all of its formula's variables, they may be likened to rows in a truth table that all result in true.

Conversion to DNF typically entails a reduction in the number of operators—down to the three noted above—available to express the formula. As a result, a formula's DNF will often be longer than the original formula. Some cases even result in combinatorial explosions of length. In the case of an arbitrary Boolean formula with n variables and a DNF possessing 2^n disjuncts, identifying a true-disjunct is a relatively

inexpensive exhaustive search provided that the true-disjunct occurs early in the DNF, but less so the later it occurs. However, if no true-disjunct exists, then all 2^n disjuncts must be evaluated in order to prove the formula false—an exponentially expensive prospect as the number of formula variables increase. Determining if there exists an arrangement of variable values for a propositional formula that return true is also known as the propositional (or Boolean) satisfiability problem (SAT). [Trakhtenbrot 1984]

The primary focus in the theory of normal forms of Boolean formulas is the minimization of Boolean functions (like DNF)—the construction of normal forms of minimal complexity [Zhuravlev 2002]. It is here that the connection, nay equivalence, to SIB becomes evident—with n analogous to the number of variables in a propositional formula, and the n-cube analogous to the complex search space within which optimal minimizations may be found. Hence, that which is useful and of benefit to SIB is also useful and of benefit to Boolean function minimization, and vice versa.

Lastly, DNF renders a propositional formula to its canonical form. A canonical form is the form which every formula within a class of formulae will return when the canonical operation—in this case, conversion to DNF—is performed on them. Canonical form in the context of SIB is discussed later in section 3.8.

SIB coils, known in this context as cyclic Boolean functions, have application in circuit design and cryptology.

2.4 The Searchers

Research into SIB may be divided into five general approaches: mathematical proofs, exhaustive search augmented by pruning methods, construction methods, heuristic search, and propositional satisfiability (SAT) solvers. While SAT solvers are

technically a subdomain of mathematical proofs, they are differentiated here due to their compelling successes in SIB. Each of these approaches remains viable and actively explored today, and none exclude the others as sources of inspiration and insight. In fact, hybridization of these approaches is quite common.

2.4.1 BOUNDS

Given the complexity of SIB, definitive solutions (snake, coil, symmetrical coil) for n are rare. So, results typically report improvements to a solution's *bounds*—valid ranges for solutions for one or more n. **Lower bounds** represent the minimum predicted solution value to the problem for n—and in maximizing problems (of which SIB is one) represent the best known potential solutions. Conversely, **upper bounds** represent the maximum predicted solution values to the problem for n. Taken together, upper and lower bounds predict a range wherein solutions for n should occur. Once a potential solution is proven to be a definitive solution, it is known as an **absolute bound** for n—noted herein as $S(n)$ for snakes, $C(n)$ for coils, and $D(n)$ for symmetrical coils.

Presently absolute bounds for SIB have been determined for $n \leq 8$ snakes and coils, and $n \leq 7$ symmetrical coils. For all greater n, only upper and lower bounds are known.

2.4.2 BASELINE

Exhaustive search, also known as *brute force search*, is the search method which considers every possible solution to a problem in order to determine the best solution. The most thorough of all searches, it is also the most computationally expensive (least efficient), especially as a problem's complexity increases. For problems with exponentially explosive complexities, obtaining solutions via exhaustive search quickly

become infeasible. Given present computing power, exhaustive search is infeasible for dimensions $n \geq 7$ in SIB. For reference, [Östergård & Pettersson 2014b] recently estimated that approximately 45 core-years of CPU-time are needed to exhaustively search $n = 8$.

To find solutions beyond the feasible reach of exhaustive search, alternative search techniques must be employed. To succeed where exhaustive search falters, these techniques selectively sacrifice search completeness for various methods which if successful will more quickly (or intelligently) steer exploration toward the better potential solutions in the solution space—banking that one or more of these will turn out to be the best solution(s).

2.4.3 The Mathematicians

Mathematicians draw on discrete mathematics and coding theory to devise and refine proofs which identify solution bounds for some or all dimensions of n. Their resulting equations do not describe specific SIB solutions, but rather curves—over a range of n—of either upper or lower bounds of the problem. Mathematicians work primarily in the higher otherwise intractable (unexplorable) dimensions, and typically focus on coils. Fortunately, coil bounds are applicable to snakes too, as any coil bound length less 2 equates to a snake bound.

For *upper bounds*, the current best estimates were described by [Zémor 1997] and [Lukito 2001]—the latter being best for $7 \leq n \leq 19079$. Previous upper bounds superseded by Zémor and Lukito were reported by [Chien et al. 1964], [Danzer & Klee 1967], [Douglas 1969a], [Evdomikov 1969], Evdokimov in 1971 [Korshunov 2009], [Deimer 1985], [Solov'eva 1987] for $n \geq 7$, [Glagolev 1990], [Snevily 1994] for $n \geq 12$,

Emelyanov in 1995 [Korshunov 2009], Emelyanov in 1997 [Emelyanov & Lukito 2000], Lukito in 1998 [Korshunov 2009], [Emelyanov & Lukito 2000], and others.

For *lower bounds*, present best estimates have been provided by [Kautz 1958], [Vasil'ev 1963], [Evdomikov 1969], and Evdomikov in 1971 [Korshunov 2009]. Previous lower bounds were reported by [Ramanujacharyulu & Menon 1964], Abbott in 1965 [Wojciechowski 1989] [Korshunov 2009], [Singleton 1966], Brown via [Danzer & Klee 1967], [Wojciechowski 1989], Röpling-Lenhart in 1991 [Korshunov 2009], and others. The current best coil lower bounds reported via mathematical proof hold for $15 \leq n \leq 20$ [Abbott & Katchalski 1991] and $21 \leq n \leq 30$ [Klee 1967].

2.4.4 The Pruners

Pruners focus on developing pruning methods—**search space reduction techniques**—for making exhaustive searches of presently *intractable* lower dimensions of n feasible. Populations of potential solutions generally start with a single minimal subsequence or a small selection of exhaustively generated subsequences and grow as the search progresses. Pruning methods include implementations of canonical ordering [Davies 1965] [Adelson et al. 1973a] [Kochut 1996] [Wong & Sawada 2008] [Östergård & Pettersson 2014a] [Östergård & Pettersson 2014b], parallel virtual machines [Rickabaugh & Shende 1998], evolved pruners [Tuohy et al. 2007], and bit-count sequences [Hood et al. 2010] [Hood et al. 2013].

For coils, the last dimension solved using pruning methods was $n = 8$ [Östergård & Pettersson 2014a], which proved the lower bound given by [Paterson & Tuliani 1998]. [Kochut 1996] proved $n = 7$ given by Eastman [Even 1963] and [Adelson et al. 1973a]. [Adelson et al. 1973b] proved the lower bounds for $2 \leq n \leq 5$ given by [Kautz 1958]

and $n = 6$ by [Even 1963]. The current best coil lower bounds obtained using pruners hold for $n = 14$ [Hood et al. 2010]. Previous lower bounds were reported by [Adelson et al. 1973a].

For snakes, the last dimension *solved* in this manner was $n = 8$ by [Östergård & Pettersson 2014b], which proved the lower bound given by [Carlson & Hougen 2010]. [Kochut 1996] proved $n = 7$ given by [Potter et al. 1994]. Solutions for $2 \leq n \leq 6$ were given by [Harary et al. 1988]. The current best lower bounds obtained using pruners hold for $9 \leq n \leq 10$ [Tuohy et al. 2007] and $n = 14$ [Hood et al. 2010]. Previous lower bounds were obtained by [Abbott & Katchalski 1991], [Rajan & Shende 1999], [Bitterman 2004], [Casella & Potter 2005a], and [Meyerson et al. 2014].

2.4.5 The Constructors

Constructors devise techniques for constructing long snake and coil sequences, typically from shorter SIB sequences or similar seeds. Construction methods are used by searchers of the other groups to varying degrees in their efforts to improve SIB bounds.

[Vasil'ev 1963] constructed SIB codes using coils of varying spread. Distance preserving codes are coil path sequences in which [Chien et al. 1964] constructed SIB codes by combining two smaller codes into one larger code, while [Evdomikov 1969] expanded the spread of Hamiltonian circuits (spread 1 coil paths). [Danzer & Klee 1967] and [Klee 1967] constructed SIB codes by combining lower dimensional sequences of differing spreads. [Preparata & Nievergelt 1974] constructed SIB codes for use in comparing feature vectors. Feature vectors are n-dimensional vectors of values which individually describe specific features of an object and together describe the object as a whole—relative to other objects within the domain. Feature vectors have application in

machine learning and pattern recognition. [Abbott & Katchalski 1991] constructed SIB codes using the symmetrical properties of the hypercube to extend snakes and coils from lower dimensions. [Paterson & Tuliani 1998] constructed SIB codes using *equivalence classes* of coils. [Haryanto & van Zanten 2004] constructed SIB codes using a technique based on Reed-Muller codes—a class of linear error-checking codes with application in computational complexity theory. [Haryanto 2007] further constructed SIB codes non-recursively using a linear algebraic code. Later, [van Zanten 2008] presented a non-recursive method based on binary linear algebraic codes for calculating covers of the n-cube. Covers are classes of coils that together use every node in the hypercube as path nodes. Of note, [Carlson & Hougen 2009] implemented construction rules within a genetic algorithm, which established the best snake lower bound for $n = 8$. This lower bound would ultimately prove to be the absolute bound.

2.4.6 The Heuristicians

Heuristicians apply heuristic search methods to unsolved SIB dimensions in order to increase known lower bounds of n. Populations of potential solutions generally start with multiple solutions—either randomly generated or seeded with valid snake subsequences—and improve over time as the search progresses. To date, genetic algorithms (GA) [Dontas & De Jong 1990] [Juric et al. 1994] [Potter et al. 1994] [Bitterman 2004] [Diaz-Gomez & Hougen 2006a] [Diaz-Gomez & Hougen 2006b] [Carlson & Hougen 2010] [Griffin & Potter 2010], population-based stochastic hill-climbers (PBSHC) [Casella & Potter 2005c] [Casella & Potter 2005a] [Tuohy et al. 2007], artificial neural networks (ANN) [Bishop 2006], nested Monte-Carlo search

[Kinny 2012], sequence permutation [Wynn 2012], and stochastic beam search [Meyerson et al. 2014] [Meyerson et al. 2015] have been applied to SIB.

For coils, the current best lower bounds reported by heuristic search hold for $n = 9$ [Casella & Potter 2005c], $10 \leq n \leq 13$ [Meyerson et al. 2015], and $n = 14$ [Hood et al. 2010]. Previous lower bounds were obtained by [Klee 1967], [Adelson et al. 1973a], [Abbott & Katchalski 1991], [Bitterman 2004], [Casella & Potter 2005c], and [Meyerson et al. 2014].

For snakes, lower bounds reported by heuristic search which would later prove absolute were reported for $n = 7$ [Potter et al. 1994] and $n = 8$ [Carlson & Hougen 2009]. Additionally, current best lower bounds hold for $n = 9$ [Wynn 2012], $n = 10$ [Kinny 2012], and $11 \leq n \leq 13$ [Meyerson et al. 2015]. Previous lower bounds obtained through heuristic search reported by [Bitterman 2004] and [Casella & Potter 2005c].

2.4.7 The SAT Solvers

A subgroup of the mathematicians, SAT solvers apply satisfiability solvers to SIB. A SAT solver tests a propositional formula, given in *conjunctive normal form*, for satisfiability. The application of SAT solvers to SIB have been reported by [Chebiryak & Kroening 2008], [Chebiryak et al. 2009], and [Zinovik et al. 2008].

Presently, SAT solvers have only reported record lower bounds for coils of spreads $k > 2$.

2.4.7.1 Conjunctive Normal Form

Like DNF, **Conjunctive normal form** (CNF) is a normalized form (*canonical form*) for representing propositional formulae in Boolean logic. It also is the dual of DNF.

The duality principle, also known as de Morgan's laws, is a pair inference rules in propositional logic which allow conjunctions and disjunctions to be expressed solely in terms of each other via negation. That is, $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$ and $\neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$, where $\wedge, \vee, \neg,$ and $\equiv$ denote conjunction, disjunction, negation, and equivalence respectively. As such, CNF, like DNF, renders a propositional formula to a canonical form—in this case, its CNF-based canonical form. Canonical form in the context of SIB is discussed later in section 3.8.

Structurally, CNF is a conjunction of disjunctions. That is, (1) every clause in CNF is connected by a conjunction, and (2) every conjunct (clause) is itself comprised solely of disjunctions of literals (single terms)—each of which includes every variable in the original formula. Any propositional formula may be converted to a CNF, and there are a number of methods to do so.

Like DNF, conversion to CNF typically entails a reduction in the number of operators—down to the three noted above—available to express the formula. As a result, a formula's CNF will also often be longer than the original formula. And again, some cases even result in combinatorial explosions of length.

As with all conjunctions, for a formula in CNF to be *false*, only one of its conjuncts need be false. Furthermore, given that the conjuncts in CNF each contain all of its formula's variables, they may be likened to rows in a truth table that all result in false.

In the case of an arbitrary Boolean formula with n variables and a CNF possessing 2^n conjuncts, identifying a false-conjunct is a relatively inexpensive exhaustive search provided that the false-conjunct occurs early in the CNF, but less so the later it occurs. However, if no false-conjunct exists, then all 2^n conjuncts must be

evaluated in order to prove the formula true—an (equally) exponentially expensive prospect as the number of formula variables increase.

2.4.7.2 CNF, DNF, AND POLYNOMIAL TIME

SAT solvers are equally capable of handling propositional formulae given in DNF or CNF form. However, CNF is the prevalent form employed. Given that DNF is generally easier to read than is CNF—compare: (DNF) "If A or B then C," versus (CNF) "if not A or not B then not C"—why *choose* to work with the less intuitive CNF? It is due to the simple fact that algorithms capable of *quickly* transforming arbitrary Boolean formulae to CNF are known, while similar algorithms for converting to DNF are not.

"Quickly" here means "in polynomial time." Polynomial time (**P**) is a class of time complexity which quantifies the amount of time it takes for a function to run based on its input. For functions which run in **P**, this amount of time may be determined ahead of time. For functions which do not run in polynomial time, or non-polynomial time (**NP**), there are no ways to find answers quickly. These definitions just scratch the surface of **P** and **NP**, but suffice for our purposes. Suffice it to say, known algorithms for transforming arbitrary Boolean formulae to DNF take an unpredictable amount of time to run. Hence, transforming to CNF is the better option.

On a side note, finding longest induced paths *in general* is a non-polynomial operation. For more on the **NP** status of SIB, see [Rajan & Shende 1999], [Bitterman 2004] [Diaz-Gomez & Hougen 2006a], [Wong & Sawada 2008], and [Korshunov 2009].

2.5 APPLICATIONS OF SIB CODES

SIB codes have applications in a number of diverse areas, including: encoding schemes for analogue-to-digital converters and quantization of signal noise [Kautz 1958]

[Klee 1970b] [Hiltgen & Paterson 2000] [Kim & Neuhoff 2000] [Lukito & van Zanten 2002], DNF simplification of Boolean functions in local searches [Vasil'ev 1963] [Evdokimov 1969] [Lukito & van Zanten 2002], worst-case search bounds [Potter et al. 1994], electronic combination locking schemes [Black 1964] [Chien et al. 1964] [Davies 1965] [Paterson & Tuliani 1998] and telemetry [Davies 1965], pattern recognition and classification problems [Preparata & Nievergelt 1974] [van Zanten & Lukito 1999], fault diagnosis in multiprocessor networks [Kautz 1958], massively parallel computing [Harary et al. 1988] [Blass et al. 2001], hypercube computer network topologies [Casella & Potter 2005b], charge modulation schemes in multi-level flash memories [Yehezkeally & Schwartz 2011], and systems biology and gene regulatory networks [Glass 1977] [De Jong 2002] [Chebiryak & Kroening 2008] [Chebiryak et al. 2009] [Zinovik et al. 2008].

CHAPTER 3

CHARACTERISTICS OF n-CUBES, SNAKES AND COILS

To understand how to search for snakes and coils in the hypercube requires additional insight into the characteristics of both the hypercube (search space) and the snake and coil paths (potential solutions) which may traverse it.

3.1 THE MANY FACETS OF THE n-CUBE

Recall that the n-dimensional hypercube is composed of 2^n vertices, each of which connects to n other vertices via n orthogonal (mutually perpendicular) edges. This makes the hypercube highly symmetrical—in fact, the hypercube is **hypersymmetrical**, meaning that it exhibits symmetry across more than two dimensions. As such, it may be easily divided into numerous equally symmetrical subgraphs. The n-cube contains

$$2^{n-m}\binom{n}{m}, \qquad 0 \leq m \leq n \tag{1}$$

m-dimensional hypercube substructures, or m-cubes (also m-faces), where m is the dimensionality of the desired element to count. Table 4 illustrates for $n = 4$.

Table 4: Sub-elements of the 4-cube.

n	m	$2^{n-m}\binom{n}{m}$	m-Terms	Common Terms
4	0	16	0-cube	nodes, points, vertices
4	1	32	1-cube	edges, lines
4	2	24	2-cube, 2-face	faces, squares
4	3	8	3-cube, 3-face	cubes
4	4	1	4-cube, 4-face	identity[1]

[1] Reserved for the n-cube itself. For $n \geq 5$, only m-terms are used to refer to elements for $3 < m < n$.

Additionally, edges and 2-faces may be divided into n parallel subgroups respectively.

Whereas the number of m elements increases as n increases, the symmetrical relationship among the elements remains constant. Counting and characterizing nodes and edges is the basis of many of the results reported by SIB researchers. A few have also made gains examining 2-faces [Solov'eva 1987] [Snevily 1994] [Emelyanov 200] and 3-cubes [Danzer & Klee 1967] [Douglas 1969b].

3.2 Node Types

As previously noted, [Kautz 1958] coined the term "snake" based on his interpretation of a spread 2 path as forming a unit-radius tube in the hypercube. That is, he imagined the unusable nodes at the ends of edges radiating from the path nodes as forming the skin of a biological snake. We call these **skin nodes**. Additionally, the nodes that make up the path are called **path nodes**, and the nodes which are neither path nor skin are called **available nodes**. Collectively, path and skin nodes are referred to as **unavailable nodes**.

Later, some additional node types will be introduced as part of a few new heuristics described in section 3.9.

3.2.1 Edge Types

Less frequently considered are edges types. Edges that form the path are **path edges**. Edges which connect path nodes to skin nodes, skin nodes to skin nodes, or skin nodes to available nodes, are called **skin edges**.

3.3 Distances between Identical Transition Values

The *minimum distance* between two identical transition values in a valid path of spread k is $k + 1$. Conversely, the *maximum distance* is $S(n - 1)$. Similarly, the longest possible subsequence using only m transitions, for $1 \leq m \leq n$, is $S(m)$.

3.4 Growing Paths and Collapsing Search Space

With the exception of the head and tail of a snake, every node added to a path designates $n - 2$ adjacent nodes as skin nodes. These nodes may be drawn from available nodes and existing skin nodes. A single skin node may serve up to n path nodes. (Head and tail nodes designate $n - 1$ adjacent nodes as skin nodes.) Hence, the number of

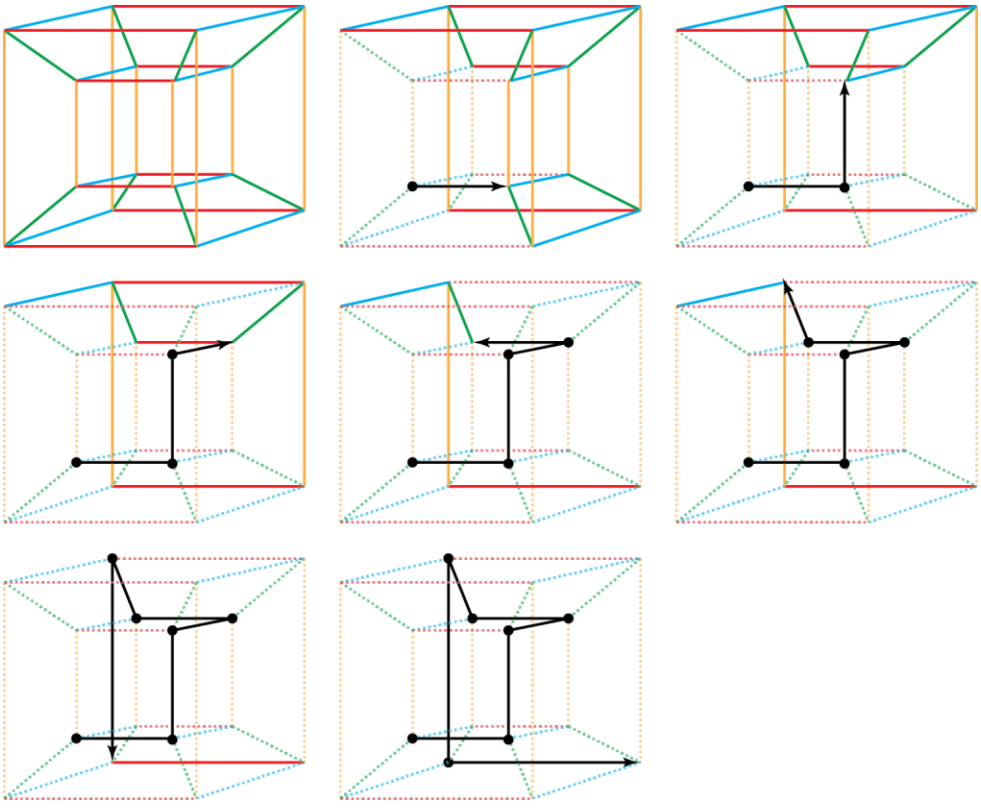

Figure 10: The number of available nodes shrinks rapidly as the path builds. 4-cube example. Parallel edges are colored alike. Solid edges are available for transition. Dotted edges are unavailable.

Table 5: Length of $S(n)$ paths when half of n-cube nodes are consumed.

n	Length of $S(n)$	Path length when half of n-cube nodes consumed
3	4	1
4	7	2
5	13	4
6	26	8
7	50	16
8	98	24

available nodes in the hypercube—nodes yet to become path or skin—decreases by up to n nodes with every node added to a path. Furthermore, with the exception of edges in a path, for every node that becomes unavailable, up to n edges are eliminated.

Consider the 4-cube in Figure 10 with its 16 nodes and 32 edges. After the first transition in the path is applied, 5 nodes (2 path, 3 skin) and 13 edges (1 path, 12 skin) became unavailable—leaving 11 nodes and 19 edges. Note that the edges radiating from node at the head of the path do not become unavailable until after the path extends from it, or the path becomes maximal. With the second transition applied, an additional 3 nodes (1 path, 2 skin) and 6 edges (1 path, 5 skin) become unavailable—leaving 8 nodes and 13 edges. At this point only half of the 4-cube's nodes remain available for growing the path. Table 5 lists the transitions at which this halfway point occurs for $3 \leq n \leq 8$.

3.5 The Changing Faces of the n-Cube

Just as nodes (and edges) are consumed at knowable rates, so too are 2-faces. For $k \geq 2$, [Solov'eva 1987] named the six arrangements of spread 2 path segments on a 2-face (2-cube), and presented formulae enumerating each for $n \geq 7$ coils paths. Table 6 lists the six 2-face types—including alternative (more intuitive) labels. Based on these

Table 6: The six arrangements of spread 2 path segments on n-cube 2-faces.

Author's Label		Description	Solov'eva's Label
F_0		A 2-face with no nodes in the snake/coil path.	A_5
F_1		A 2-face with one node in the snake/coil path.	A_4
F_{2a}		A 2-face with two nodes and one edge in the path.	A_2
F_{2d}		A 2-face with two nodes and no edges in the path.	A_3
F_3		A 2-face with three nodes and edges in the path.	A_1
F_4		$C(2)$	A_6

formulae, the rates at which face types change as a path grows may be determined. The rates cited below are for snakes.

An n-cube devoid of a path begins with

$$2^{n-2}\binom{n}{2}F_0. \tag{2}$$

Initializing Node 0, converts

$$\binom{n}{2}F_0 \to F_1. \tag{3}$$

Each subsequent node (after the first) added to the path converts

$$\begin{gathered}\left(\binom{n}{2}-(n-1)\right)F_0 \to F_1, \text{ and} \\ \frac{\binom{n}{2}}{2}F_1 \to F_{2a}.\end{gathered} \tag{4}$$

Each subsequent node (after the second) added to the path converts

$$\begin{gathered}\left(\binom{n}{2}-(n-1)\right)F_0 \to F_1, \text{ and} \\ (n-2)F_1 \to F_{2a}, \text{ and} \\ 1F_{2a} \to F_3.\end{gathered} \tag{5}$$

Each node (after the $(k+3)$rd, inclusive) added to the path *occasionally* converts

$$\left(\frac{\binom{n}{2}}{2}-2\right)F_1 \rightarrow F_{2d}. \tag{6}$$

The maximum rate at which F_1 faces convert to F_{2d} faces is not known; however, their occurrence is easily detected in the collision matrix of the growing path. When generating a collision matrix for a path, after the first $k+1$ bit flips in each bit string, every case where a bit string's cardinality equals k indicates the formation of a new F_{2d} face. Occurrences of F_{2d} faces enable a path to grow longer than it would without their presence. Face F_{2d} is spread 2 specific.

3.6 Extracting Paths from the Representations

In a node sequence (NS) each node is indicated by its unique node label. Reading from left to right, the nodes in the sequence form a path. NS path node values range from 0 to 2^{n-1}. NS may start from any node, though Node 0 is preferred.

In a transition sequence (TS), where individual path values range only from 0 to $n-1$, each transition indicates the bit position of difference between the binary labels of two adjacent nodes. As such, each transition is implicitly anchored between two adjacent nodes—the identities of which wholly depend on all of the transitions preceding it. When extracting a path from a TS, it is customary to start from Node 0. To illustrate, the transition sequence 0 1 2 0 describes the node sequence 0 1 3 7 6. (Note that there is one less element in the transition sequence than in the node sequence.) For $n=3$ the node sequence reads as 000 001 011 111 110—n bits each. The first transition flips the 0-bit in node 000 resulting in node 001. The second transition flips the 1-bit in node 001

resulting in node 011. Next, the third transition flips the 2-bit in node 011 giving node 111. Finally, the fourth transition flips the 0-bith resulting in node 110.

In a binary sequence (BS), each bit position correlates to a node in the hypercube with "1" bits indicating nodes in the path and "0" bits indicating nodes not in the path. To extract a path from a BS requires constructing a path from all of the "1" bits in the BS, starting from the 0-bit position (Node 0). Note that potentially more than one path may be encoded into a single BS if more "1" bits are set than are required to encode a single path [Diaz-Gomez & Hougen 2006a] [Diaz-Gomez & Hougen 2006b]. In these cases, additional effort is required to extract the multiple paths from the BS.

3.6.1 Converting between Representations

As may already be evident, converting between representations is straightforward. To convert from NS to TS, map the bit differences between each node pair. To convert from NS to BS, set the bits corresponding to each node to "1". To convert from TS to NS or BS, map the transition bits from Node 0, generating a new node from every transition—and setting the corresponding bits to "1" for BS. To convert from BS to NS or TS, first requires extracting the path(s) as previously noted, and then convert using the appropriate NS or TS method above.

3.7 Path Validation

In path construction, the objective is to construct (at least) a valid path—preferably a snake or coil. Snake and coil construction goes hand in hand with validation. As previously noted, snakes and coils are induced paths—that is, paths with no adjacencies between non-adjacent nodes. Paths which violate this constraint are neither snakes nor coils, and are **invalid**.

Table 7: Adjacency matrix for $n = 4$.

n-Cube Node	Adjacent Nodes			
0	1	2	4	8
1	0	3	5	9
2	0	3	6	10
3	1	2	7	11
4	0	5	6	12
5	1	4	7	13
6	2	4	7	14
7	3	5	6	15
8	0	9	10	12
9	1	8	11	13
10	2	8	11	14
11	3	9	10	15
12	4	8	13	14
13	5	9	12	15
14	6	10	12	15
15	7	11	13	14

3.7.1 The Adjacency Matrix

The most common method for determining whether or not a path is valid is to check it against an *adjacency matrix* of the hypercube. An **adjacency matrix** maps node adjacencies of a graph. For the hypercube, it is a $2^{n-1} \times n$ integer matrix which lists the n adjacent nodes for each of the 2^{n-1} nodes in the hypercube graph. Table 7 shows the adjacency matrix for $n = 4$. Alternately, a $2^{n-1} \times 2^{n-1}$ binary matrix may be used—with "1" bits set at the intersections of adjacent nodes in the matrix.

Traversing the nodes in the path, one first verifies that the next node in the sequence is indeed adjacent to the previous node in the sequence, and then verifies that none of the other nodes adjacent to the previous node appear in the path. If any do, the path is invalid.

Table 8: Example collision matrix for a spread 2 path. The bold box at the start of the transition sequence is a moving window in which evaluations occur. It advances one position with each step through the sequence. Changing bits in each column are bolded (pink). The shaded cells (extending from the left) mark the first $k + 1$ steps of each bit string. The blue cell (far right) indicates a spread-2 collision between the 3rd and 7th transitions, meaning nodes 3 and 11. The sequence is valid to length 6.)

Collision matrix bit strings	**Transition Sequence (evaluated left to right)**						
	0	1	2	0	3	2	0
1st	000**1**	00**1**1	0**1**11	011**0**	**1**110	1**0**10	101**1**
2nd		00**1**0	0**1**10	011**1**	**1**111	1**0**11	101**0**
3rd			0**1**00	010**1**	**1**101	1**0**01	100**0**
4th				000**1**	**1**001	1**1**01	110**0**
5th					**1**000	1**1**00	110**1**
6th						0**1**00	010**1**
7th							000**1**

Adjacency matrices are typically generated at the beginning of an experiment and referenced during runtime; however, individual node adjacencies may also be generated as needed during experiment runtime.

3.7.2 The Collision Matrix

The **collision matrix**[1] (alternately, a spread Δ matrix) is an alternative method which validates a path by measuring the minimum distances (spreads) among every node described by a transition sequence. In the worst case—where the entire transition sequence is valid—the collision matrix is an $n \times |T|$ binary matrix, where $|T|$ is the length of the transition sequence. For cases with collisions, it only forms an $n \times |T_0 \ldots T_{c-1}|$, where c is the position of the transition at which the collision occurs.

[1] This collision matrix method traces its origins to the heyday of the Institute for Artificial Intelligence's "Snake Pit"—an active open collaboration among a handful of SIB exploring graduate students overseen by Dr. Potter—from fall 2010 through spring 2012. It is difficult to credit any individual group member with the first application of transition bits as a validation method. However, the point is moot, given that all of the method's building blocks were first reported by [Singleton 1966].

At each transition in the sequence, a new n-bit string is instantiated with one bit flipped at the position of the current transition value. Additionally, all previously instantiated bit strings also have the same bit flipped. For the first $k + 1$ bit flips in each bit string, its cardinality (number of "1" bits) must increase each step. Thereafter, if at any time its cardinality drops below k, a collision has occurred and *the remainder of the sequence*—including the transition resulting in the collision—is invalid. Table 8 illustrates.

Without modification, collision matrices easily detect collisions for any valid k. Additionally, the specific nodes involved in a collision are readily identifiable. They are: (1) the node *preceding* the transition at which a collision reporting bit string originated, and (2) the node *following* the transition at which the collision was detected. Note that where multiple collisions occur within the same step, each colliding pair of nodes is described by one bit string.

Collision matrices may be generated as needed during experiment runtime, or carried with each transition sequence throughout generation. Optionally, a $2^{n-1} \times 2^{n-1}$ integer spread Δ matrix—with the distance between adjacent nodes set at their intersections in the matrix—may be generated for the hypercube and referenced during experiment runtime [Horton 2015]. The "Binary Fibonacci Snake Representation" validation method described in [Khan 2015] is a collision matrix method that uses n-integer sets in lieu of the n-bit strings described above.

3.8 Equivalence Classes and Canonical Ordering

For every TS comprised of m transition values, there exist $m!$ symmetrical paths—comprised of all permutations of ordered transition values. Maximal paths use all

available n transitional values. Thus, for maximal paths, $m = n$. Likewise, every maximal path is part of a *class* of $n!$ symmetrical maximal paths. [Kochut 1996]

An **equivalence class** (EC) is any group of symmetrical objects. Any test of one member of an EC applies equally to all members of the class. Thus it is enough to validate a single path within an EC to validate the entire class. In order to compare different EC, a method of selection is needed, which will return the *same* representative—the **canonical representative** of the EC—when performed on *any* member of the class.

Canonical ordering is one such method [Kochut 1996]. In a canonically ordered sequence, transitions are introduced into the sequence in lexicographic (alphanumeric) order. That is, the first transition introduced must be 0 followed by 1, then 2, 3, etc. Values may freely recur within the limits of the path's spread constraint. That is, they need not be introduced consecutively to remain canonical. For example, the sequences 0 1 2 3 and 0 1 2 0 3 1 0 4 are both canonically ordered. It is only when a new dimension is traversed for the first time that the next transition value in lexicographic order is introduced. A canonically ordered sequence is known as a **canonical sequence** or is said to be in **canonical form** (CF).

Any non-canonical sequence may be easily converted to CF by swapping the

Table 9: Example converting to canonical form. Underscored transitions in sequences reveal transition order.

Non-canonical Sequence	Transition Order Map	Canonical Sequence
3 1 0 3 2 1 3	3 1 0 2 ↓ ↓ ↓ ↓ 0 1 2 3	0 1 2 0 3 1 0

order in which its unique transition values are introduced for the canonical ordering. Table 9 illustrates. Likewise, any transition sequence may be translated to the transition order of any other transition sequence.

Snake CF always occur in pairs, with each pair being mutual canonical reversals. A **canonical reversal** is the CF of a reversed canonical sequence. That is, take a snake CF, reverse it, and then convert the reversed sequence to CF. The occurrence of CF pairs may not be immediately apparent considering that the snake CF for dimensions $3 \leq n \leq 8$ number 1, 1, 8, 1, 12, and 2 respectively. How then can there be only one CF each for

Table 10: Equivalence classes for $S(n), 3 \leq n \leq 8, k = 2$.

n	id	Equivalency Classes (canonically ordered)
3	A	0120 ← canonical palindrome (i.e., canonical reversal of itself)
4	A	0120310 ← canonical palindrome
5	A	0123014021032
	B	0123014312301
	C	0123024012031
	D	0123024012301 ← canonical reversal of B
	E	0123024102301
	F	0123024321032 ← canonical reversal of A
	G	0123024321302 ← canonical reversal of C
	H	0123104312301 ← canonical reversal of E
6	A	012310430540134024104315 34 ← canonical palindrome
7	A	0120310421035012406504203401240350420340120610421 0
	B	0120310421035012406504203401240350420340120624012 4
	C	0120310421035102406504203401240350420340120610421 0
	D	0123014021035023065032103502306403201650123015021 0 ← c. r. of C
	E	0123014021035023065032103502306403210650123015021 0 ← c. r. of A
	F	0123014312305310365301230531036430123653210325312 3 ← c. r. of B
	G	0123142053024012304205326102401350140310241035014 2
	H	0123142153124012314215326012401351041301240135104 2
	I	0123142153214012314215316512351401531451235140153 2
	J	0123453210325312345231032605430135023103543012502 3 ← c. r. of G
	K	0123453210325312345321032605430135032103543013503 2 ← c. r. of H
	L	0123453210325312345321032631230531034530123053103 4 ← c. r. of I
8	A	0123104154314634013467431042740147514016310415431423401342743104674 0 14756134014274104316410476401 3
	B	0123453213410321356312301475312354321035630123063103713210412317531 2 35632103543012304310371321061 2 ← c. r. of A

dimensions 3, 4, and 6? The CF for these dimensions are **canonical palindromes**—meaning, their canonical reversals are identical to their original CF. Table 10 lists the equivalence classes for $S(n), 3 \leq n \leq 8$.

Enumerating coil CF is trickier. Whereas every snake has but one head node and one tail node, any pair of adjacent nodes in a coil may serve as tail and head nodes respectively. Even with canonical ordering enforced, a single coil may be represented by up to its length in CF. As with the snake CF which are canonical reversals of each other, all of said coil CF would be canonical shifts of each other.

3.9 Fitness Measures

What makes one snake or coil path better than another? What is the measure of a great snake or coil? Fitness measures are heuristics meant to distinguish between seemingly indistinguishable potential solutions. In a nutshell, **heuristics** are informed guesses—a means of enabling an otherwise blind search operation to gather information about its surroundings (search domain) to inform its decision making. Search domains include both a solution space (here, the hypercube) and potential solutions (here, snake and coil paths). The more effective an heuristic is, the more successes a search operation can achieve.

Keep in mind, however, that success always comes at a price. Typically, the more effective a heuristic is, the more specialized it is to its specific domain. Additionally, the tradeoff for improving the resolution of any heuristic—the clarity with which it perceives its domain—is often paid in the form of additional computational resources.

3.9.1 PATH HEURISTICS

Path heuristics focus on path sequences—looking for telltale indicators that one potential solution is either better than, or has the potential to be better than, another potential solution.

Length is the simplest and most important measure of a snake or coil path. Regardless of any other qualities, a length r snake or coil is always better than an $r-1$ snake or coil. The length of a snake or coil is equal to its number of edges. For snakes, length equals the number of transitions in a TS, and one less than the number of nodes in an NS or "1" bits in a BS. For coils, length equals the number of nodes in an NS or "1" bits in a BS, and one more than the number of transitions in a TS.

Additional measures attempt to gauge the quality of paths through the presence or absence of special patterns in a path or its skin nodes. However, these have met with limited success.

3.9.2 SEARCH SPACE HEURISTICS

Search space heuristics have shown to be more effective than their path heuristic counterparts. This may be due in part to the fact that when a path is short, more data points may be found in the search space than in the path; and later, when the path is longer, successful analysis of the remaining available space is the key to guiding the path to its maximum potential. At any point, the available search space is comprised of all remaining available nodes in the hypercube.

Keeping track of the remaining **available nodes** in the n-cube is a good method for gauging the future growth potential of a snake or coil sequence. A simple metric is to sum the current length of a path with the available nodes to get a rough idea of the

maximum length the path will be able to achieve. Additionally, if the sum is less than some desired length, the path may be confidently discarded, because it will never achieve the goal.

Recently a number of improvements to the standard available nodes measure were implemented—as part of a collaboration involving the author—that resulted in some impressive gains across the spectrum of spread 2 lower bounds; specifically eleven new lower bound improvements for snakes, coils, and symmetrical coils [Meyerson et al. 2014] [Meyerson et al. 2015].

Determining **Reachable available nodes** [Meyerson et al. 2014] [Meyerson et al. 2015] offers greater fidelity in gauging the future growth potential of paths. Not all available nodes will be useable in all instances. So determining which nodes are actually reachable by the current path gives a better sense of the length that path might achieve.

Dead end pruning [Meyerson et al. 2015] is a further refinement of reachable available nodes, and looks to remove additional nodes that will be of no benefit to the future potential growth of a snake or coil sequence. Dead end pruning removes nodes that are connected to only one other reachable available node.

Blind alley pruning [Meyerson et al. 2015] is an extension of dead end pruning more applicable to coil search than snake search. In it, chains of available nodes which end in dead ends—but would otherwise be overlooked by dead end pruning—are removed. This enhancement is less useful in snake search as a longest path may require traveling down one of these blind alleys.

Articulation point pruning [Meyerson et al. 2015] is an additional extension of dead end pruning. An **articulation point** (also cut vertices) is any node in a graph which

if removed splits the graph into two disjoint (separate) subgraphs [Hopcroft & Tarjan 1973]. In the subgraph of reachable available nodes, an articulation point is a one-way pass from one cluster of nodes to another—in essence a super blind alley entrance detector. For coils, articulation points may be used to quickly prune cul-de-sacs of any size from the reachable available nodes, further refining growth potential prediction and speeding up potential solution evaluation. For snakes, articulation points may be used in strategizing path construction by helping to target the largest clusters of reachable available nodes in which to grow a snake.

Further reduction of the search space may be achievable through the application of additional graph theory manipulations.

3.9.3 Population Reduction Heuristics

Some search methods—of which many are heuristic techniques—utilize populations of potential solutions in their searches. That is, multiple snake paths are explored concurrently so as to improve the chances of success. In these methods, population sizes expand and contract as new potential solutions are added, and then lower potential solutions are removed.

One common method for determining which potential solutions are removed from a population is *tournament selection*. In **tournament selection**, two or more potential solutions are randomly selected from a population, their respective growth potentials (fitness) are compared, with the single best fit of these remaining in the population and the others being removed.

Recently, a variant of tournament selection called *reverse tournament selection* was described by [Meyerson, et al. 2014] [Meyerson, et al. 2015]. Identical to standard

tournament selection with one exception, **reverse tournament selection** removes only the single worst fit potential solution from the population, and retains the remaining potential solutions in the population. While this variant method requires more tournaments to be conducted in order to reduce a population to a desired level, its less aggressive approach appears to longer preserve variation in the population—allowing for more exploration of the search space.

CHAPTER 4

CONCLUSION

This primer introduced many of the essential concepts of SIB, its history and continued relevance, as well as methods for representing, validating, and evaluating snake and coil solutions. With explanations of prerequisite concepts necessary for understanding SIB included throughout, it has been structured to serve as a convenient reference for those exploring SIB.

Details of the various search methods applied to SIB and listed herein have been omitted, and are left for the reader to explore. Particularly, the reader is invited to examine papers—herein and elsewhere—related to their avenue of inquiry, and to become practiced in these approaches within the SIB domain by exploring the lower dimensions where absolute bounds are known. Recreating experiments from previous work may also prove useful in gaining a sense of the SIB domain. Then, once ready, set out to extend the known bounds of the problem themselves.

As to this primer in particular, future improvements could include discussions of each of the individual techniques briefly touched on in 2.4, plus the addition of future applied methods. Additional discussions—with illustrated walkthroughs—further detailing the methods discussed in 3.7 and 3.9 could better help less experienced readers to more quickly grasp these concepts.

4.1 The Appendices

Additional information regarding SIB is present as appendices. Appendices A–E are timelines of the discoveries of upper and lower bounds. Appendix F gives an equation for calculating the total number of canonically ordered paths to be found in dimension n, of length r, and spread k. Appendix G presents some generalized patterns found to occur in $S(n)$, $3 \leq n \leq 8$, for spread 2.

The reader is invited to judge for themselves the usefulness of these appendices.

REFERENCES

ALPHABETICAL BY AUTHOR

Abbott, H. L., and Katchalski, M., "On the Construction of Snake in the Box Codes," *Utilitas Mathematica*, Vol. 40, pp. 97–116, 1991.

Adelson, L. E., Alter, R., and Curtz, T. B., "Long snakes and a characterization of maximal snakes on the d-cube," in *Proceedings of 4th SouthEastern Conference on Combinatorics, Graph Theory and Computing*, Congr. Numer. 8, pp. 111–124, 1973.

Adelson, L. E., Alter, R., and Curtz, T. B., "Computation of d-Dimensional Snakes," in *Proceedings of 4th SouthEastern Conference on Combinatorics, Graph Theory and Computing*, Congr. Numer. 8, pp. 135–139, 1973.

Bishop, J. (2006). "Investigating the Snake-in-the-box problem with Neuroevolution," Department of Comp. Science, University of Texas, Austin, Texas, USA.

Bitterman, S., New Lower Bounds for the Snake-In-The-Box Problem: a Prolog Genetic Algorithm and Heuristic Search Approach. Master Thesis. University of Georgia, Georgia, USA, 2004.

Black W. L., "Electronic combination locks," *Quart. Progress Report of the Research Laboratory of Electronics*, No. 73, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, pp. 232–233, April, 1964.

Blass, U., Honkala, I., Karpovsky, M., and Litsyn, S., "Short dominating paths and cycles in the binary hypercube," *Ann. Combin*, Vol. 5, pp. 51–59, 2001.

Blaum, M., and Etzion, T. 2002. Use of snake-in-the-box codes for reliable identification of tracks in servo fields of a disk drive. U.S. patent 6,496,312 B2.

Carlson, B. P., and Hougen D., "Phenotype Feedback Genetic Algorithm Operators for Heuristic Encoding of Snakes and Hypercubes," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10*, pp. 791–798, Portland, Oregon, USA, July 07–11, 2010.

Casella, D. A., and Potter, W. D., "New Lower Bounds for the Snake-In-The-Box Problem: Using Evolutionary Techniques to Hunt for Snakes," in *Proceedings of the 18th International FLAIRS Conference*, pp. 264–269, Clearwater Beach, Florida, USA, May, 2005.

Casella, D. A., and Potter, W. D., "Using Evolutionary Techniques to Hunt for Snakes and Coils," in *Proceedings of 2005 IEEE Congress on Evolutionary Computing, CEC'05*, pp. 2499–2505, Edinburgh, Scotland, UK, September 2–5, 2005.

Casella, D. A., and Potter, W. D., "New Lower Bounds for the Coil-In-The-Box Problem: Using Evolutionary Techniques to Hunt for Coils," in *Proceedings of the International Conference on Computational Intelligence, Man-Machine Systems, and Cybernetics, CIMMACS '05*, CD Proceedings Paper No. 501–111, Miami, Florida, USA, November 17–19, 2005.

Chebiryak, Y., and Kroening, D., "An efficient SAT encoding of circuit snakes," in *Proceedings of IEEE International Symposium on Information Theory and its Applications*, Auckland, New Zealand, pp. 1235–1238, December 7–10, 2008.

Chebiryak, Y., Wahl, T., Kroening, D., and Haller, L., "Finding Lean Induced Cycles in Binary Hypercubes," in *Proceedings of SAT Conference, Lecture Notes in Computer Science*, No. 5584, Springer Verlag, pp. 18–31, June, 2009.

Chien, R.T., Freiman, C.V., and Tang, D.T., "Error connection and circuits on the n-cube," in *Proceedings of the 2nd Allerton Conference on Circuit and System Theory*, University of Illinois, Monitcello, Illinois, USA, pp. 899–912, September 28–30, 1964.

Cook, S. A., "The complexity of theorem proving procedures," in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, Shaker Heights, Ohio, USA, pp. 151- 158, May 3–5, 1971.

Danzer, L., and Klee, V., "Lengths of Snakes in Boxes," *Journal of Combinatorial Theory*, Vol. 2, pp. 258–265, 1967.

Davies, D. W., "Longest "Separated" Paths and Loops in an N Cube," *IEEE Transactions on Electronic Computers*, Vol. 14, p. 261, 1965.

De Jong, H. D. "Modeling and Simulation of Genetic Regulatory Systems: A Literature Review," *Journal of Computational Biology*, Vol. 9, No. 1, 2002, pp. 67–103.

Diaz-Gomez, P., and Hougen, D., "Genetic algorithms for hunting snakes in hypercubes: fitness function analysis and open questions," in *Proceedings of the 7th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributive Computing*, Las Vegas, Nevada, USA, June 19–20, 2006. SNPD '06. IEEE Computer Society, Los Alamitos, California, USA, pp. 389–394.

Diaz-Gomez, P., and Hougen, D., "The snake in the box problem: Mathematical Conjecture and a Genetic Algorithm Approach," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, Seattle, Washington, USA, July 08–12, 2006. *GECCO '06*. ACM, New York, New York, USA, pp. 1409–1410.

Dontas, K, and de Jong, K., "Discovery of maximal distance codes using Genetic Algorithm," in *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pp. 805–811, 1990.

Douglas, R. J., "Some results on the maximum length of circuits of spread *k* in the d-cube," *Journal of Combinatorial Theory*, Vol. 6, pp. 323–339, 1969.

Douglas, R. J., "Upper Bounds on the length of circuits of even spread in the *d*-Cube," *Journal of Combinatorial Theory*, Vol. 7, pp. 206–214, 1969.

Emelyanov, P. G., and Lukito, A., "On the maximal length of a snake in hypercubes of small dimension," *Discrete Mathematics*, Vol. 218, pp. 51–59, 2000.

Evdokimov, A. A., "Maximal length of a chain in a unit *n*-dimensional cube," *Matematicheskie Zametki*, Vol. 6, pp. 306–319, 1969.

Even, S., "Snake in the Box Codes," *Correspondence in IRE Transactions on Electronic Computers*, Vol. EC-12, p. 18, 1963.

Glass, L., "Combinatorial aspects of dynamics in biological systems," *Statistical mechanics and statistical methods in theory and applications*, U. Landman, Ed., Plenum, 1977, pp. 585–611.

Gray, F. 1953. Pulse code communication. U.S. patent 2,632,058.

Griffin, J. D., and Potter, W. D. "Pruning the Search Space for the Snake-in-the-Box Problem," *IEA/AIE 2010, Part III, LNAI 6098*, Cordoba, Spain, pp. 528–537, 2010.

Hamming, R. W. "Error Detecting and Error Correcting Codes," *Bell Systems Technical Journal*, Vol. 29, No. 2, April 1950.

Harary, F., Hayes, J. P., and Wu, H. J., "A survey of the theory of hypercube graphs," *Computers & Mathematics with Applications*, Vol. 15, pp. 277–289, 1988.

Haryanto, L., and van Zanten, A. J. (2004). "Snake-in-the-box Codes and Euclidean Geometries," Delft University of Technology, Delft, Netherlands.

Haryanto, L., Constructing Snake-In-The-Box Codes and Families of such Codes Covering the Hypercube. Ph. D. Dissertation (A. J. van Zanten, advisor), Delft University of Technology, Delft, Netherlands, 2007.

Hiltgen, A. P., and Paterson K. G., "Single Track Circuit Codes," *IEEE Transactions on Information Theory*, Vol. 47, pp. 2587–2595, 2000.

Hood, S., Sawada, J., and Wong, C. (2010) "Generalized Snakes and Coils in the Box," School of Computer Science, University of Guelph, Ontario, Canada.

Hood, S., Recoskie, D., Sawada, J., and Wong, D., "Snakes, coils, and single-track circuit codes with spread *k*," Journal of Combinatorial Optimization, 2013, DOI 10.1007/s10878-013-9630-z

Hopcroft, J., and Tarjan, R., "Efficient algorithms for graph manipulation," *Communications of the ACM*, Vol. 16, No. 6, pp. 372–378, 1973, DOI 10.1145/362248.362272

Horton, T., "Personal communications," March 30, 2015.

Juric, M., Potter, W. D., and Pleasing, M., "Using PVM for Hunting Snake In The Box Codes," in *Proceedings of the 1994 Transputer Research and Applications Conference (NATUG-7)*, pp. 97–102, Athens, Georgia, USA, October, 1994.

Kautz, W. H., "Unit-Distance Error-Checking Codes," *IRE Trans. Electronic Computers*, Vol. 7, pp. 179–180, 1958.

Khan, MD. S., Building snakes from DNA — A step towards generalizing the snake in the box problem. Master Thesis. University of Georgia, Georgia, USA, 2015.

Kim, S., and Neuhoff, D. L., "Snake-in-the-box codes as robust quantizer index assignments," *in Proceedings of the IEEE International Symposium on Information Theory*, Sorento, Italy, p. 402, June 25–30, 2000.

Kinny, D., "Monte-Carlo Search for Snakes and Coils," in *Proceedings of the Sixth International Workshop, MIWAI'2012, LNCS-7694: Multi-Disciplinary Trends in Artificial Intelligence*, Ho Chi Minh City, Vietnam, pp. 271–283, 2012.

Klee, V., "A method for constructing circuit codes," *Journal of the ACM*, Vol. 14, pp. 520–538, 1967.

Klee, V., “The Use of Circuit Codes in Analog-to-Digital Conversion,” *Graph Theory and its Applications*, B. Harris, ed., Academic Press, New York, pp. 121–131, 1970.

Kochut, K. J., “Snake-in-the-box codes for dimension 7,” *Journal of Combinatorial Mathematics and Combinatorial Computing*, Vol. 20, pp. 175–185, 1996.

Korshunov, A. D., “Some unsolved problems in discrete mathematics and mathematical cybernetics,” *Russian Mathematical Surveys*, Vol. 64:5, pp. 787–803, 2009.

Lukito, A., and van Zanten, A. J., “Vertex Partitions of Hypercubes into Symmetric Snakes,” *Electronic Notes in Discrete Mathematics*, Vol. 11, pp. 459–467, 2002.

Meyerson, S., Whiteside, W., Drapela, T., and Potter, W., “Finding Longest Paths in Hypercubes: Snakes and Coils,” in *Proceedings of the IEEE Symposium on Computational Intelligence for Engineering Solutions*, *CIES ‘14*, Orlando, Florida, USA, December, 2014.

Meyerson, S. J., Drapela, T. E., Whiteside, W. E., and Potter, W. D., “Finding Longest Paths in Hypercubes, 11 New Lower Bounds: Snake, Coils, and Symmetric Coils,” in *Proceedings of the 28th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, IEA/AIE 2015*, Seoul, Korea, June 10–12, 2015. (To appear)

Oliver, B. M., and Shannon, C. E. 1957. Communication system employing pulse code modulation. U.S. patent 2,801,281.

Östergård, P. R. J., and Pettersson, V. H., “On the Maximum Length of Coil-in-the-Box Codes in dimension 8,” *Discrete Applied Mathematics*, 2014, DOI 10.1016/j.dam.2014.07.010

Östergård, P. R. J., and Pettersson, V. H., "Exhaustive Search for Snake-in-the-Box Codes," *Graphs and Combinatorics,* 2014, DOI 10.1007/s00373-014-1423-3

Paterson, K. G. and Tuliani, J., "Some New Circuit Codes," *IEEE Transactions on Information Theory*, Vol. 44, No. 3, pp. 1305–1309, 1998.

Pierce, J. R. 1948. Communicating system employing pulse code modulation. U.S. patent 2,437,707.

Potter, W. D., Robinson R. W., Miller J. A., and Kochut, K. J., "Using the Genetic Algorithm to Find Snake-In-The-Box Codes," in *Proceedings of the 7th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, Austin, Texas, pp. 421–426, 1994.

Potter, W. D., "Latest Records for the Snake-in-the-Box Problem (UGA)," http://ai1.ai.uga.edu/sib/sibwiki/doku.php/records, Accessed Feb. 25, 2015.

Preparata, F., and Nievergelt, J., "Difference-preserving codes," *IEEE Transactions on Information Theory*, Vol. 20, pp. 643–649, 1974.

Rajan D. S., and Shende A. M., "Maximal and Reversible Snakes in the Hypercube," in *Proceedings of the Annual Australian Conference on Combinatorial Mathematics and Combinatorial Computing*, Darwin, Australia, 5-9 July 1999.

Ramanujacharyulu, C., and Menon, V. V., "A Note on the Snake-in-the-Box Problem," Publications de l'Institut de statistique de l'Université de Paris, Vol. 13, pp. 131–135, 1964.

Reeves, A. H. 1942. Electrical Signaling System. U.S. patent 2,272,070.

Reeves, C. M. (1972) *An Introduction to Logical Design of Digital Circuits*. New York, New York: Cambridge University Press.

Rickabaugh, B. P., and Shende, A. M., "Using PVM to Hunt Maximal Snakes in Hypercubes," *Journal of Computing in Small Colleges*, Vol. 14, No. 2, pp. 76–84, 1998.

Singleton, R. C., "Generalized Snake-in-the-Box Codes," *IEEE Trans. Electronic Computers*, Vol. 15, pp. 596–602, 1966.

Snevily, H. S., "The snake-in-the-box problem: A new upper bound," *Discrete Mathematics*, Vol. 133, pp. 307–314, 1994.

Sobolev, S. K. (2002) Propositional formula. In Hazewinkel, M., ed., *Encyclopedia of Mathematics.* http://www.encyclopediaofmath.org/index.php?title=Propositional_formula&oldid=32250. Accessed April 10, 2015.

Solov'eva, F. I., "An Upper Bound for the Length of a Cycle in an n-Dimensional Unit Cube," *Dickretnyj Analiz,* Vol. 45, pp. 71–76, 1987. English translation 2009.

Tuohy, D. R., Potter, W. D., and Casella, D. A., "Searching for snake-in-the-box codes with evolved pruning models," in *Proceedings of the 2007 International Conference on Genetic and Evolutionary Methods (GEM'2007)*, CSREA Press, Las Vegas, Nevada, USA, pp. 3–9, 2007.

Trakhtenbrot, B. A. "A Survey of Russian Approaches to Perebor (Brute-Force Search) Algorithms," *Annals of the History of Computing*, Vol. 6, No. 4, pp. 384–400, 1984.

van Zanten, A. J., and Lukito, A., "Construction of Certain Cyclic Distance-Preserving Codes Having Linear-Algebraic Characteristics," *Designs, Codes, and Cryptography*, Vol. 16, No. 2, pp. 185–199, 1999.

Vasil'ev, Ju. I., "On the length of a cycle in an n-dimensional unit cube," *Soviet Mathematics Doklady*, Vol. 4, pp. 160–163, 1963.

Wojciechowski, J., "A New Lower Bound for Snake-in-the-Box Codes," *Combinatorica*, Vol. 9, No. 1, pp. 91–99, 1989.

Wojciechowski, J., "Long snakes in powers of the complete graph with an odd number of vertices," *Journal of the London Mathematical Society*, Vol. 50, No. 3, pp. 465–476, 1994, DOI 10.1112/jlms/50.3.465

Wong, C., and Sawada, J. (2008). "Exhaustive Search for Maximal Length Coil-in-the-Box Codes," Technical Report TR-CIS-UG-2008-001, Department of Computing and Information Science, University of Guelph, Guelph, Ontario, Canada.

Wyner, A. D., "Note on Circuits and Chains of Spread *k* in the n-Cube," IEEE Transactions on Computers, Vol. 20, No. 4, pp. 474, 1971.

Wynn, E., "Constructing Circuit Codes by Permuting Initial Sequences," arXiv: 1201.1647v1, 2012.

Yehezkeally, Y., and Schwartz, M., "Snake-in-the-Box Codes for Rank Modulation," arXiv: 1107.3372v1, 2011.

Zémor, G., "An Upper Bound on the Size of the Snake-In-The-Box," *Combinatorica*, Vol. 17, No. 2, pp. 287–298, 1997.

Zhuravlev, Yu. I. (2002) Boolean functions, normal forms of. In Hazewinkel, M., ed., *Encyclopedia of Mathematics.* http://www.encyclopediaofmath.org/index.php?title=Boolean_functions,_normal_forms_of&oldid=16364. Accessed April 10, 2015.

Zinovik, I., Kroening, D., and Chebiryak, Y. “Computing Binary Combinatorial Gray Codes Via Exhaustive Search With SAT Solvers,” *IEEE Trans. Informational Theory*, Vol. 54, No. 4, April 2008.

APPENDIX A

TIMELINE OF COIL LOWER BOUND FORMULATIONS

Table 11: Timeline of coil lower bound formulations. Values for $n, k = 2$ are included where given by authors; otherwise, only formulae were given.

Reported by	Values of n	Notes
[Kautz 1958]		
Zhuravlev 1962		Cited in [Vasil'ev 1963].
[Vasil'ev 1963]		
[Ramanujacharyulu 1964]		
Abbott 1965 (Ph.D. thesis)		Cited in [Klee 1970], [Wojciechowski 1989], and [Korshunov 2009].
T. A. Brown 1965?, unpublished		Via [Danzer & Klee 1967]
[Singleton 1966]		
[Danzer & Klee 1967]		
[Klee 1967]	$4 \leq n \leq 30$	
[Evdomikov 1969]		
Evdomikov 1971		Cited in [Korshunov 2009].
[Wojciechowski 1989]		
Evdokimov 1990		Cited in [Emelyanov & Lukito 2000].
Röpling-Lenhart 1991		Cited in [Korshunov 2009].
[Abbott & Katchalski 1991]	$8 \leq n \leq 20$	

APPENDIX B

TIMELINE OF COIL UPPER BOUND FORMULATIONS

Table 12: Timeline of coil upper bound formulations. Values for n, $k = 2$ are included where given by authors; otherwise, only formulae were given.

Reported by	Values of n	Notes
[Chien et al. 1964]		
Glagolev 1966		Cited in [Korshunov 2009].
[Singleton 1966]		
Klee 1967 (book chapter)		Cited in [Douglas 1969a].
[Danzer & Klee 1967]		
[Douglas 1969b]	Even n	
D. G. Larman 1969?, unpublished		Cited in [Douglas 1969b] as Larmen [sic].
Evdomikov 1971		Cited in [Korshunov 2009].
[Wyner 1971]		
[Deimer 1985]	$n \geq 7$	
[Solov'eva 1987]	$n \geq 7$	
[Abbott 1988b]		
Glagolev & Evdokimov 1990		Cited in [Wojciechowski 1989].
Kochut et al. 1994, unpublished	$C(7) \leq 50$	Cited in [Kochut 1996].
[Snevily 1994]	$n \geq 12$	
Emelyanov 1995		Cited in [Korshunov 2009].
Emelyanov 1997		Cited in [Emelyanov & Lukito 2000] and [Lukito 2001].
[Zémor 1997]		
Lukito 1998		Cited in [Emelyanov & Lukito 2000].
[Emelyanov & Lukito 2000]		
[Lukito 2001]	$7 \leq n \leq 19079$	

APPENDIX C

TIMELINE OF COIL LOWER BOUNDS

Table 13: Timeline of coil lower bounds. Bounds are for $C(n), k = 2$. Shaded rows indicate solved dimensions.

Reported by	Bounds	Notes
[Kautz 1958]	$\boldsymbol{C(2) = 4}$	See $D(2)$ in Appendix D.
[Kautz 1958]	$\boldsymbol{C(3) = 6}$	See $D(3)$ in Appendix D.
[Kautz 1958]	$\boldsymbol{C(4) = 8}$	See $D(4)$ in Appendix D.
[Kautz 1958]	$\boldsymbol{C(5) = 14}$	See $D(5)$ in Appendix D.
[Even 1963]	$\boldsymbol{C(6) = 26}$	See $D(6)$ in Appendix D.
Eastman (via [Even 1963]) [Adelson et al. 1973a] [Kochut 1996]	$C(7) \geq 48$ $C(7) \geq 48$ $\boldsymbol{C(7) = 48}$	- Different sequence. -
[Klee 1967] [Adelson et al. 1973a] [Abbott & Katchalski 1991] [Paterson & Tuliani 1998] [Östergård & Pettersson 2014a]	$C(8) \geq 64$ $C(8) \geq 86$ $C(8) \geq 88$ $C(8) \geq 96$ $\boldsymbol{C(8) = 96}$	Estimated via equation. - - - -
[Klee 1967] [Adelson et al. 1973a] [Abbott & Katchalski 1991] [Bitterman 2004] [Casella & Potter 2005a/b]	$C(9) \geq 112$ $C(9) \geq 128$ $C(9) \geq 170$ $C(9) \geq 174$ $C(9) \geq 180$	Estimated via equation. - - - -
[Klee 1967] [Abbott & Katchalski 1991] [Paterson & Tuliani 1998] [Casella & Potter 2005a/b] [Meyerson et al. 2014] [Meyerson et al. 2015]	$C(10) \geq 224$ $C(10) \geq 324$ $C(10) \geq 340$ $C(10) \geq 344$ $C(10) \geq 358$ $C(10) \geq 362$	Estimated via equation. Estimated via equation. - - - See $D(10)$ in Appendix D.
[Klee 1967] [Abbott & Katchalski 1991] [Abbott & Katchalski 1991] [Casella & Potter 2005a/b] [Meyerson et al. 2014] [Meyerson et al. 2015]	$C(11) \geq 410$ $C(11) \geq 416$ $C(11) \geq 640$ $C(11) \geq 630$ $C(11) \geq 666$ $C(11) \geq 668$	Estimated via equation. - Estimated via equation. - - -
[Klee 1967] [Abbott & Katchalski 1991] [Meyerson et al. 2014] [Meyerson et al. 2015]	$C(12) \geq 832$ $C(12) \geq 1238$ $C(12) \geq 1268$ $C(12) \geq 1276$	Estimated via equation. Estimated via equation. - -
[Klee 1967] [Abbott & Katchalski 1991] [Hood et al. 2010] [Meyerson et al. 2015]	$C(13) \geq 1536$ $C(13) \geq 2468$* $C(13) \geq 1908$ $C(13) \geq 2354$	Estimated via equation. Estimated via equation. Does not show seq. See $D(13)$ in Appendix D.

Reported by	Bounds	Notes
[Klee 1967] [Abbott & Katchalski 1991] [Hood et al. 2010]	$C(14) \geq 3072$ $C(14) \geq 4934$* $C(14) \geq 2976$	Estimated via equation. Estimated via equation. Does not show sequence
[Abbott & Katchalski 1991]	$C(15\text{–}20) \geq \cdots$	Estimated via equation.
[Klee 1967]	$C(21\text{–}30) \geq \cdots$	Estimated via equation.

APPENDIX D

TIMELINE OF SYMMETRICAL COIL LOWER BOUNDS

Table 14: Timeline of symmetrical coil lower bounds. Bounds are for $D(n), k = 2$. Shaded rows indicate solved dimensions.

Reported by	Bounds	Notes
[Adelson et al. 1973a]	$\boldsymbol{D(2) = 4}$	See $C(2)$ in Appendix C.
[Adelson et al. 1973a]	$\boldsymbol{D(3) = 6}$	See $C(3)$ in Appendix C.
[Adelson et al. 1973a]	$\boldsymbol{D(4) = 8}$	See $C(4)$ in Appendix C.
[Adelson et al. 1973a]	$\boldsymbol{D(5) = 14}$	See $C(5)$ in Appendix C.
[Adelson et al. 1973a]	$\boldsymbol{D(6) = 26}$	See $C(6)$ in Appendix C.
[Adelson et al. 1973b]	$\boldsymbol{D(7) = 46}$	-
[Adelson et al. 1973a] [Wynn 2012]	$D(8) \geq 86$ $D(8) \geq 94$	- Cited in [Potter 2015]
[Meyerson et al. 2015]	$D(9) \geq 186$	-
[Meyerson et al. 2015]	$D(10) \geq 362$	See $C(10)$ in Appendix C.
[Meyerson et al. 2015]	$D(11) \geq 662$	-
[Meyerson et al. 2015]	$D(12) \geq 1222$	-
[Meyerson et al. 2015]	$D(13) \geq 2354$	See $C(13)$ in Appendix C.

APPENDIX E

TIMELINE OF SNAKE LOWER BOUNDS

Table 15: Timeline of snake lower bounds. Bounds are for $S(n), k = 2$. Shaded rows indicate solved dimensions.

Reported by	Bounds	Notes
[Harary et al. 1988]	$\boldsymbol{S(2) = 2}$	-
[Harary et al. 1988]	$\boldsymbol{S(3) = 4}$	-
[Harary et al. 1988]	$\boldsymbol{S(4) = 7}$	-
[Harary et al. 1988]	$\boldsymbol{S(5) = 13}$	-
[Harary et al. 1988]	$\boldsymbol{S(6) = 26}$	-
[Potter et al. 1994] [Kochut 1996]	$S(7) \geq 50$ $\boldsymbol{S(7) = 50}$	- -
[Abbott & Katchalski 1991] [Rajan & Shende 1999] [Carlson 2009] [Östergård & Pettersson 2014b]	$S(8) \geq 89$ $S(8) \geq 97$ $S(8) \geq 98$ $\boldsymbol{S(8) = 98}$	- - - -
[Bitterman 2004] [Bitterman 2004] [Casella & Potter 2005b & 2005c] [Tuohy et al. 2007]	$S(9) \geq 168$ $S(9) \geq 182$ $S(9) \geq 186$ $S(9) \geq 188$	Derived from $C(9) \geq 170$. - - -
[Casella & Potter 2005b & 2005c] [Tuohy et al. 2007]	$S(10) \geq 358$ $S(10) \geq 363$	- -
[Casella & Potter 2005b & 2005c] [Meyerson et al. 2014] [Meyerson et al. 2015]	$S(11) \geq 680$ $S(11) \geq 705$ $S(11) \geq 707$	- - -
[Casella & Potter 2005b & 2005c] [Meyerson et al. 2015]	$S(12) \geq 1260$ $S(12) \geq 1302$	- -
[Hood et al. 2010] [Meyerson et al. 2015]	$S(13) \geq 1916$ $S(13) \geq 2520$	- -
[Hood et al. 2010]	$S(14) \geq 3067$	-

APPENDIX F

CANONICAL SELECTION WITH REPLACEMENT AND SPREAD

Herein is given an equation for calculating the total number of canonically ordered paths to be found in dimension n, of length r, and spread k. The results returned by the equation are for *all valid paths*—of which snakes and coils are subsets. For example, for $n = 3, r = 9, k = 2$, the sequence 0 1 2 0 1 2 0 1 2 is valid path, but is neither a valid snake or coil. This is due to the equation not being constrained by any specific search space, including the n-cube. As such, it is incomplete for determining SIB lower bounds. Regardless, it does provide some insight into how ordering, canonical ordering and spread affect selection.

F.1 SELECTION WITH REPLACEMENT

The number of possible ordered arrangements of n elements selected r times with *replacement* is n^r. This is because there are n elements to choose from each time a selection is made. Hence, $n \times n \cdots (r \text{ times}) = n^r$. **Replacement** means that elements are permitted to occur more than once in an arrangement.

TS are ordered arrangements of elements with replacement—where the number of available transitions is analogous to n, and the length of the transition sequence is analogous to r.

Let $P_o(n,r),\ n \geq 0,\ r \geq 0$ define this function which calculates the number of possible *ordered* arrangements of n elements selected r times with replacement. That is,

$$P_o(n,r) = n^r. \tag{7}$$

Lacking a spread constraint n^r produces paths of minimum spread 0.

F.2 Accounting for Spread

Incorporating k into $P_o(n,r)$ reduces the number of elements to choose from for the first $k+1$ transitions by 1 each time, and by $n-k$ elements thereafter for the remaining $r-(k+1)$, or $r-k-1$, choices. That is, the first $k+1$ choices build a permutation, while the subsequent choices remain a selection string (with a reduced number of elements to choose from).

Let $P_o(n,r,k),\ n \geq 0,\ r \geq 0,\ 0 \leq k \leq n$ define this function which calculates the number of possible *ordered* arrangements of n elements selected r times with replacement and constrained by spread k. That is,

$$P_o(n,r,k) = \frac{n!\,(n-k)^{r-k-1}}{(n-k-1)!}. \tag{8}$$

Verification: For $k > 0$ the aforementioned permutation (in blue) becomes evident.

$$\begin{aligned}
P_o(n,r,1) &= [(n-0)(n-1)] \times [(n-1)(n-1)\cdots(r-1-1 \text{ times})] \\
&= [n(n-1)] \times [(n-1)(n-1)\cdots(r-2 \text{ times})] \\
P_o(n,r,2) &= [(n-0)(n-1)(n-2)] \times [(n-2)(n-2)\cdots(r-2-1 \text{ times})] \\
&= [n(n-1)(n-2)] \times [(n-2)(n-2)\cdots(r-3 \text{ times})] \\
P_o(n,r,3) &= [(n-0)(n-1)(n-2)(n-3)] \times [(n-3)(n-3)\cdots(r-3-1 \text{ times})] \\
&= [n(n-1)(n-2)(n-3)] \times [(n-3)(n-3)\cdots(r-4 \text{ times})]
\end{aligned} \tag{9}$$

This permutation generalizes to $\frac{n!}{(n-k-1)!}$ and the remaining selection string generalizes to $(n-k)^{r-k-1}$.

For $k = 0$, $P_o(n, r, k)$ safely reduces to $P_o(n, r)$.

$$\begin{aligned} P_o(n, r, 0) &= [(n - 0)] \times [(n - 0)(n - 0) \cdots (r - 0 - 1 \text{ times})] \\ &= [n] \times [n \times n \cdots (r - 1 \text{ times})] \\ &= n \times n \cdots (r \text{ times}) \\ &= n^r \\ &= P_o(n, r) \end{aligned} \tag{10}$$

F.3 Considering Only Canonical Arrangements

Given that there are $n!$ symmetrical paths for every TS comprised of n transition values [Kochut 1996], the number of possible *canonical* arrangements of n elements selected r times with replacement and constrained by spread k is $\frac{P_o(n,r,k)}{n!}$.

Let $P_c(n, r, k)$ define this function which calculates this number. That is,

$$P_c(n, r, k) = \frac{(n - k)^{r-k-1}}{(n - k - 1)!}. \tag{11}$$

Note that the number of *valid* snakes and coils will be subsets of the value returned for a given n, r, k.

APPENDIX G

GENERALIZED PATTERNS IN SNAKE TRANSITION SEQUENCES

Herein is presented a number of generalized patterns found to occur in the EC of $S(n)$, $3 \leq n \leq 8$, for spread 2. A *Chutes & Ladders*[2] analogy is adopted for discussing these patterns. The omission of $S(1), S(2)$ do not alter the findings.

To separate these patterns from the familiar transition values of 0 to $n-1$, the symbols $(A, B, C, D, ...)$ are used to represent the transition values in each pattern. If any of these symbols were previously assigned special meaning or value, these assignments are temporarily suspended for the duration of this pattern discussion.

G.1 CHUTES

Chutes are TS subsequences without duplicate transition values—for example, $A\ B\ C\ D\ E$. They occur between pairs of adjacent ladders, as well as between a ladder and the head or tail of a snake. Chutes occur in lengths from $-1 \ldots n-2$; in whole or in part outside of ladder rungs. Note that the -1-chute is a special case and is discussed in G.3.

In addition to connecting ladders, chutes frequently occur in mirrored pairs bracing either side of a ladder like a pair of bookends—occasionally with part of one or both chutes extending into neighboring ladders. As such, there are two chute patterns for each length—that is, $A\ B\ C\ D\ E$ and its mirror $E\ D\ C\ B\ A$.

[2] *Chutes and Ladders* is a board game by the Milton Bradley Company based on the ancient Indian board game known as *Snakes and Ladders*. Its use herein is as a simple mnemonic for distinguishing between two pattern groups.

G.2 LADDERS

Ladders are TS subsequences in which a single transition value repeats every $k+1$ positions from the first to the last transition inclusively; and can be likened to the rungs of a ladder. For example, the subsequence $A\ B\ C\ A\ D\ E\ A$ contains three A rungs. The length of a spread k ladder pattern generalizes to $ku + (u - k)$, where u is the number of rungs. The ladders presented here are specific to spread 2. Ladders in paths of spread $k > 2$ will exhibit similar patterns, but with greater variability between rungs.

G.2.1 L_2 PATTERNS

There is only one length 2 pattern (L_2). Pattern length is $k+2$. The L_2 pattern string forms a simple 2-rung ladder with the (boxed) A-transitions as the rungs like so:

$$L_{2a}: \boxed{A}\quad B\quad C\quad \boxed{A} \tag{12}$$

L_{2a} represents the shortest valid subsequence for spread k wherein transition repetition may occur. The number of possible arrangements of L_{2a} with A fixed is $\binom{n-1}{2}$. L_{2a} is valid for $n \geq k+1$. Pattern L_{2a} occurred in all examined $S(n)$. For $k = 2$, $S(3) \equiv L_{2a}$. In the induced path in the 3-cube in Figure 8: the first A-transition occurs between nodes 000 and 001; the B-transition is between nodes 001 and 011; etc.

G.2.2 L_3 PATTERNS

There are two length 3 patterns (L_3). Pattern length is $2k+3$. The L_3 pattern strings form 3-rung ladders with the A-transitions as the rungs. (Additional repeated transitions are underlined.)

$$\begin{aligned} L_{3a}&: \boxed{A}\quad \underline{B}\quad C\quad \boxed{A}\quad D\quad \underline{B}\quad \boxed{A} \\ L_{3b}&: \boxed{A}\quad B\quad C\quad \boxed{A}\quad D\quad E\quad \boxed{A} \end{aligned} \tag{13}$$

L_{3a} is comprised of two instances of pattern L_{2a}, the latter offsetting the former by $k+1$ positions. The number of possible arrangements of L_{3a} with A fixed is $\binom{n-1}{3}$. L_{3a} is valid for $n \geq 2k$. For $k = 2$, $S(4) \equiv L_{3a}$.

L_{3b} differs from L_{3a} in that it contains no repeated transition values between its rungs, meaning it produces the lesser compact ladder of the two L_3 patterns. The number of possible arrangements of L_{3b} with A fixed is $\binom{n-1}{4}$. L_{3b} is valid for $n \geq 2k+1$. Pattern L_{3b} occurred less frequently than L_{3a}, and was only found in $S(8)$.

G.2.3 L_4 PATTERNS

There are seven length 4 patterns (L_4). Pattern length is $3k+4$. The L_4 pattern strings form 4-rung ladders with the A-transitions as the rungs. (Additional repeated transitions appear alternately underlined or overlined.)

$$
\begin{array}{lcccccccccc}
L_{4a}: & \boxed{A} & \underline{B} & C & \boxed{A} & \overline{D} & \underline{B} & \boxed{A} & E & \overline{D} & \boxed{A} \\
L_{4b}: & \boxed{A} & \underline{B} & \overline{C} & \boxed{A} & D & E & \boxed{A} & \overline{C} & \underline{B} & \boxed{A} \\
L_{4c}: & \boxed{A} & \underline{B} & C & \boxed{A} & D & \underline{B} & \boxed{A} & E & F & \boxed{A} \\
L_{4d}: & \boxed{A} & B & C & \boxed{A} & \underline{D} & E & \boxed{A} & F & \underline{D} & \boxed{A} \\
L_{4e}: & \boxed{A} & \underline{B} & C & \boxed{A} & \overline{D} & E & \boxed{A} & \underline{B} & \overline{D} & \boxed{A} \\
L_{4f}: & \boxed{A} & \underline{B} & \overline{C} & \boxed{A} & D & \underline{B} & \boxed{A} & E & \overline{C} & \boxed{A} \\
L_{4g}^{D}: & \boxed{A} & B & C & \boxed{A} & D & E & \boxed{A} & F & G & \boxed{A}
\end{array}
\tag{14}
$$

L_{4a} is comprised of two instances of L_{3a}, the latter offsetting the former by $k+1$ positions. The number of possible arrangements of L_{4a} with A fixed is $\binom{n-1}{4}$. L_{4a} is valid for $n \geq 2k+1$ dimensions. Pattern L_{4a} was only found in $S(8)$, wherein it only occurred as part of a hook with L_{4b}.

L_{4b} could actually be an instance of L_{2b} bounded on braced by mirrored chutes of *at least* length 3. As such, it is unclear whether or not this string should be classified as a

distinct pattern. The number of possible arrangements of L_{4b} with A fixed is $\binom{n-1}{4}$. L_{4b} is valid for $n \geq 2k+1$ dimensions. Pattern L_{4b} was only found in $S(8)$, wherein it occurred twice: once as part of a hook with L_{4a}, and once alone.

L_{4c} could actually be an instance of L_{3a} joined on its right side to a chute of *at least* length 3. As such, it is also unclear whether or not this string qualifies as a distinct pattern. The number of possible arrangements of L_{4c} with A fixed is $\binom{n-1}{5}$. L_{4c} is valid for $n \geq 2k+2$. Pattern L_{4c} was only found in $S(7)$, and occurred exclusive of pattern L_{4d}; meaning that they were only found in respective EC reversals.

L_{4d} is a reversal of L_{4c}, and thus possesses similar characteristics.

L_{4e} is taken from the *beginning* of pattern L_{6a}. Standalone, L_{4e} is valid for $n \geq 2k+1$, and the number of possible arrangements of L_{4e} with A fixed is $\binom{n-1}{4}$. Pattern L_{6a} only occurred in $S(7)$.

L_{4f} is taken from the *middle* of pattern L_{6a}. Standalone, L_{4f} is valid for $n \geq 2k+1$, and the number of possible arrangements of L_{4f} with A fixed is $\binom{n-1}{4}$. Pattern L_{6a} only occurred in $S(7)$.

L_{4g}^{D}, like patterns L_{3b} and L_{2b}, contains no repeated transition values between its rungs, meaning it produces the least compact ladder of the L_4 patterns. The number of possible arrangements of L_{4g}^{D} with A fixed is $\binom{n-1}{6}$. L_{4g}^{D} is valid for $n \geq 3k+1$. Pattern L_{4g}^{D} was not found in the examined sequences, but is nonetheless a valid pattern. Denoted with a "D" superscript, this *deduced* pattern is presented here for completeness.

G.2.4 L_5 PATTERNS

No length 5 patterns (L_5) were found in the examined $S(n)$. Such patterns, if found, will be of length $4k + 5$—the length of $S(5)$—and form 5-rung ladders.

G.2.5 L_6 PATTERNS

Only one length 6 pattern (L_6) was found. Pattern length is $5k + 6$. The L_6 pattern string forms a 6-rung ladder with the A-transitions as the rungs. (Additional repeated transitions appear alternately underlined, overlined, etc.)

$$L_{6a}: \boxed{A}\ \underline{B}\ C\ \boxed{A}\ \overline{D}\ \underline{E}\ \boxed{A}\ \underline{B}\ \overline{D}\ \boxed{A}\ \overline{\overline{F}}\ \underline{E}\ \boxed{A}\ G\ \overline{\overline{F}}\ \boxed{A} \qquad (15)$$

L_{6a} may be viewed in a number of ways: (1) as an overlap of patterns L_{4e} and L_{4d}, with the latter being offset by $2(k + 1)$ positions; (2) as overlaps of patterns L_{4e}, L_{4f}, and L_{4d}, with each offsetting its predecessor by $k + 1$ positions; (3) as pattern L_{3a} appended to pattern L_{4e}; (4) as pattern L_{3b} prefixed to pattern L_{4d}; or (5) as pattern L_{4f} bounded on either side by non-identical length 3 chutes. Pattern L_{6a} was found in $S(7)$, and occurred bounded on either side by mirrored length 4 chutes, of which the last transition of one of the chutes doubled as the starting rung of a separate 4-rung ladder, while the other was a terminus (head or tail). Patterns L_{4e} and L_{4f} originated from this pattern.

G.3 THE HOOK

At first glance, the hook pattern may appear to be a special ladder pattern with two rung values. However, it may also be viewed as two ladders joined by a length -1 chute. The term "hook" stems from how ladders joined by this pattern appear hooked together.

G.3.1 H_6 PATTERNS

Only one hook pattern was found. Keeping with the naming convention established while discussing ladders, only one length 6 hook (H_6) was found. Pattern length is $2k + 2$. The H_6 pattern string denotes a hook between two ladder patterns. A-transitions form the rungs of the first ladder; and C-transitions form the rungs of the second ladder.

$$H_{6a}: \cdots \quad \boxed{A} \quad B \quad \boxed{C} \quad \boxed{A} \quad D \quad \boxed{C} \quad \cdots \tag{16}$$

H_{6a} joins two ladders L_X, L_Y together, overlapping the end of the first ladder, and the start of the second ladder, by one position. Pattern H_{6a} only appeared in $S(8)$, wherein it twice linked patterns L_{4a} and L_{4b}, and in a third instance linked patterns L_{3b} and L_{2b}.

A closer examination of the presence of an H_{6a} reveals the path that is orbiting a central skin node. To demonstrate, we begin with an instance of the minimum length ladder pattern, L_{4a}. The focus of the orbit is established as the skin node most used by the L_{4a} pattern. In the 3-cube in Figure 8, this is node 010, which is shared by nodes 000, 011, and 110. At this point, the path is 0 1 2 0. Hooking a second L_{4a} to the first is enough to begin to see the orbit. The path is now 0 1 2 0 3 2, and forms an H_{6a} hook.

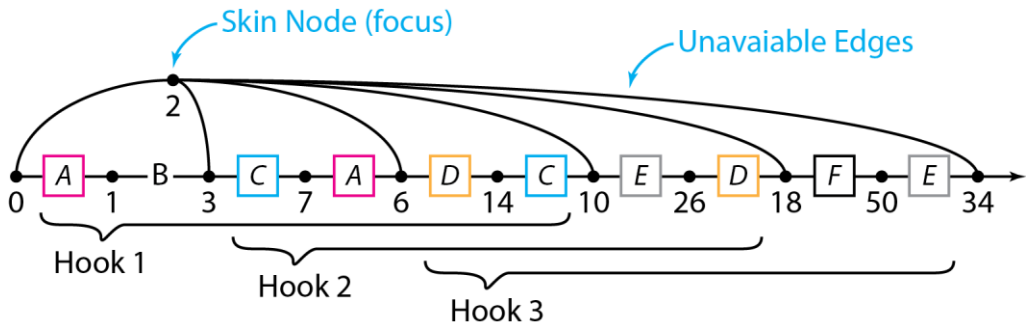

Figure 11: Snake of hooks orbiting a single skin node. Here nodes of an $(n \geq 6)$-cube are laid out in a line. Only nodes and edges specific to the example are pictured. Solid dots are nodes and are correctly numbered. Solid lines are edges. Letter symbols indicate a chain of hooked ladder patterns. The TS correlating to the numbered nodes is 0 1 2 0 3 2 4 3 5 4 …

If taken to an extreme—with multiple nested L_{4a} patterns hooked one after the other—the orbit is even more visible. Figure 11 illustrates this extreme example.

The hook is not an efficient use of n-cube space. This is likely why it fails to appear in $S(n)$ for $4 \leq n \leq 7$. There is simply too little space within these lower dimensional n-cubes for longest maximal paths featuring hooks to occur.