Solving the Snake in the Box Problem with Heuristic Search: First Results

Alon Palombo, Roni Stern, Rami Puzis and **Ariel Felner**
Department of Information Systems Engineering
Ben-Gurion University of the Negev
Beer Sheva, Israel
palomboalon@gmail.com
sternron, puzis, felner at *post.bgu.ac.il*

Scott Kiesel and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
palomboalon@gmail.com
skiesel and ruml at *cs.unh.edu*

Abstract

Snake in the Box (SIB) is the problem of finding the longest simple path along the edges of an n-dimensional cube, subject to certain constraints. SIB has important applications in coding theory and communications. State of the art algorithms for solving SIB apply uninformed search with symmetry breaking techniques. We formalize this problem as a search problem and propose several admissible heuristics to solve it. Using the proposed heuristics is shown to have a huge impact on the number of nodes expanded and, in some configurations, on runtime. These results encourage further research in using heuristic search to solve SIB, and to solve maximization problems more generally.

Introduction

The longest simple path (LPP) problem is to find the longest path in a graph such that any vertex is visited at most once. This problem is known to be NP-Complete (Garey and Johnson 1990). The *Snake in the Box* (SIB) problem is a special case of LPP in which: (1) the searched graph, denoted Q_n, is an n-dimensional cube, and (2) the only paths allowed are *induced paths* in Q_n. An induced path in a graph G is a sequence of vertices such that every two vertices adjacent in the sequence are connected by an edge in G and every pair of vertices from the sequence that are not adjacent in it are not connected by an edge in G. An optimal solution to SIB is the longest induced path in Q_n. For example, the optimal solution to Q_3 is shown in Figure 1 by the green line (for now, ignore the difference between solid and dashed lines). The *Coil in the Box* (CIB) is a related problem where the task is to find the longest induced cycle in Q_n.

SIB and CIB have important applications in error correction codes and communications. In particular, a snake of length d in an n-dimensional cube represents a special type of *Gray code* that encodes each of the numbers $0, \ldots, d$ to n bits (Kautz 1958).

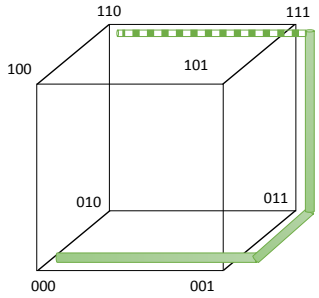

Figure 1: An example of a snake in a 3 dimensional cube.

The mapping between a snake of length d and a code is as follows. Every vertex in the n-dimensional cube is labeled by a binary bit vector of length n (see Figure 1 for an example of this labeling). A snake $s = (v_0, \ldots, v_d)$ represents the code that encodes each number $i \in (0, \ldots, d)$ by the label of v_i. Gray codes generated from snakes are particularly useful for error correction and detection because they are relatively robust to one-bit errors: flipping one bit in the label of v_i is either the label of v_{i-1}, v_{i+1}, or a label not in s. Thus, a one-bit error would result in the worst case in an error of one in the decoded number (mistaking i for $i-1$ or $i+1$).

Due to its importance, SIB and CIB have been studied for several decades (Kautz 1958; Douglas 1969; Abbott and Katchalski 1988; Potter et al. 1994; Kochut 1996; Hood et al. 2011; Kinny 2012). Existing suboptimal solvers use genetic algorithms (Potter et al. 1994) and Monte Carlo tree search algorithms (Kinny 2012). In this work we focus on finding optimal solutions to SIB and CIB. Existing approaches to optimally solving SIB and CIB employ exhaustive search, symmetry breaking, and mathematical bounds to prune the search space (Kochut 1996; Hood et al. 2011).

In this paper we formulate SIB and CIB as heuristic search problems and report initial results on using heuristic search techniques to solve them. SIB and CIB are both maximization problems (MAX problems), and thus search algo-

rithms may behave differently from their more traditional shortest-path (MIN) counterparts (Stern et al. 2014b). The most obvious difference is that admissible heuristic functions, which are key components in optimal search algorithms such as A* and IDA*, must upper bound solution cost instead of lower bound it (as in MIN problems).

We propose several admissible heuristics for SIB and CIB. Notably, one of the heuristics can be viewed as a maximization version of additive pattern databases (Felner, Korf, and Hanan 2004; Yang et al. 2008). Another family of admissible heuristics we propose are based on decomposing the searched graph into a tree of biconnected components.

We compared the previous state of the art – exhaustive DFS, with the MAX problem versions of A* and DFBnB that use the proposed heuristics. Results suggests that DFBnB is in general superior to A* and DFS in this problem. With the proposed heuristics, more than two orders of magnitude savings is achieved in terms of number of nodes expanded. In terms of runtimes, a huge speedup is observed in some cases, while in other cases the speedup is more modest due to overhead incurred by the proposed heuristics. We hope this work will bring this fascinating problem to the attention of the heuristic search community.

Problem Definition

The SIB is defined on a graph $Q_n = (V_n, E_n)$ that represents an n-dimension unit cube, i.e., Q_0 is a dot, Q_1 is a line, Q_2 is a square, Q_3 is a cube, etc. There are $|V_n| = 2^n$ vertices and each is labeled as a bit-vector $[b_0 b_1 ... b_{n-1}]$ of length n. Every two vertices $u, v \in V_n$ are connected by an edge in E_n iff their Hamming distance equals 1 (i.e. only one bit in the two vectors is different). A path s across an edge $(u, v) \in E_n$ where u and v differ in the i^{th} bit is said to be *traversing* the i^{th} dimension. The first vertex in the path is called the *tail* and denoted as $tail(s)$ while the last vertex is called the *head* and denoted as $head(s)$.

The original SIB problem is to find the longest path in Q_n such that any two vertices on the path that are adjacent in Q_n are also adjacent on the path (Kautz 1958). Such paths are called *snakes*.

Example 1 *Consider an example of a snake as given in Figure 1. The snake consists of the sequence of vertices* $s = (000, 001, 011, 111)$. *Vertex 000 is the* tail *of the snake and 111 is the* head*. Note that vertex 101 cannot be appended to the head of s, since it is adjacent to 001, which is a part of s. By contrast, vertex 110 can be appended to the head of s as it is not adjacent to any other vertex* $v \in s$.

A generalized form of the SIB problem considers the *spread* of the snake (Singleton 1966).

Definition 1 (Spread) *A sequence of vertices* $(v_0, \ldots, v_l)$ *is said to have a spread of* k *iff* $Hamming(v_i, v_j) \geq k$ *for* $|i-j| \geq k$ *and* $Hamming(v_i, v_j) = |i-j|$ *for* $|i-j| < k$.

Note that a snake as defined above has a spread of $k = 2$. Intuitively, the snake cannot pass through vertices that are fewer than k hops away from other vertices in it.

n \ k	2	3	4	5	6	7
3	4*	3*	3*	3*	3*	3*
4	7*	5*	4*	4*	4*	4*
5	13*	7*	6*	5*	5*	5*
6	26*	13*	8*	7*	6*	6*
7	50*	21*	11*	9*	8*	7*
8	98*	35*	19*	11*	10*	9*
9	190	63	28*	19*	12*	11*
10	370	103	47*	25*	15*	13*
11	707	157	68	39*	25*	15*
12	1302	286	104	56	33*	25*
13	2520	493	181	79	47	31*

Table 1: Longest known snakes for different values of n and k.

Example 2 *Consider again the snake shown in Figure 1,* $s = (000, 001, 011, 111, 110)$. *s has a spread of* $k = 2$ *as every pair of non adjacent vertices in it has a Hamming distance larger than one. s does not have a spread of* $k = 3$*, since* $Hamming(000, 110) = 2$ *but the indices (the order in the snake) of 000 and 110 are 0 and 5, respectively.*

Obviously, the length of the longest snake for a given n decreases as the spread grows. However, in some applications increasing the spread means better error detection capabilities, and thus can be desirable (Kautz 1958).

Definition 2 ((n,k)-SIB Problem) *(n,k)-SIB is the problem of finding the longest snake in Q_n with a spread of k.*

Mathmatical Bounds

In terms of computational complexity, the (n,k)-SIB problem is a special case of the longest induced path problem which has been proven to be NP-Complete (Garey and Johnson 1990). Due the importance of this problem in real-life applications, there have been numerous works on theoretical bounds of the longest snake for different values of n and k (Douglas 1969; Abbott and Katchalski 1988; Danzer and Klee 1967; Zémor 1997). Most theoretical results give an upper bound on the length of snakes of the form $\lambda 2^{n-1}$ where $0 < \lambda < 1$, but empirical results show that these bounds are not tight. The best currently known empirical lower bounds for snakes with different values of n and k are given in Table 1. These results were collected from different sources (Hood et al. 2011; Kinny 2012; Abbott and Katchalski 1991; Meyerson et al. 2014). Values marked with an asterisk are known to be optimal.

Existing Approaches

Several approaches have been proposed for solving the (n,k)-SIB problem. Some are geared towards finding long snakes but do not provide guarantees on the optimality of the solution. Such suboptimal approaches include Monte-Carlo Tree Search (Kinny 2012) and Evolutionary Algorithms (Potter et al. 1994; Casella and Potter 2005). Another approach is to exhaustively grow snakes from other snakes obtained by some other method, e.g., snakes obtained for lower values of n (Kinny 2012).

We focus in this paper on finding optimal solutions to this problem, i.e., finding the longest possible snakes. Due to the complexity of the problem, optimal solution have only been found for relatively small n and k values. Previous approaches include SAT (Chebiryak et al. 2008), building all possible snakes of a given length using isomorphic rejection (Östergård and Pettersson 2014), and exhaustive search using symmetry-breaking methods (Kochut 1996; Hood et al. 2011). We describe these methods later in the paper, and use some of them in the proposed heuristic search.

(n,k)-SIB as a Search Problem

(n,k)-SIB is fundamentally a path-finding problem in a graph. As such, heuristic search algorithms can be applied to solve it. Unlike much work in heuristic search, SIB is a maximization (MAX) problem – we want the *longest* path and not the shortest path. We formalize the state space for this MAX problem as follows. A state represents a snake $s = (v_0, v_1, ..., v_l)$. A snake is said to be *valid* if it has a spread of k (Definition 1). *Expanding* a state means generating all states that represent valid snakes obtained by appending a single vertex to the head of the snake. A *goal* state is a snake to which no vertex can be appended.[1] The cost of a state is the number of edges in the snake, and the task is to find the longest snake, i.e., the highest cost state.

Recently, Stern et al. (2014b) showed that uninformed search algorithms, such as Dijkstra's algorithm (1959) and bidirectional search, may not be effective for MAX problems. Indeed, state-of-the-art algorithms for solving the SIB problem optimally use an exhaustive Depth-First Search (Kochut 1996; Hood et al. 2011). Heuristic search algorithms, however, can be used to find optimal solutions faster than uninformed search algorithms even in MAX problems by using an *admissible* heuristic (Stern et al. 2014b; Puzis, Elovici, and Dolev 2007; Stern et al. 2014a). An important difference between MAX and MIN problems is the definition of an *admissible* heuristic: in MIN problems an admissible heuristic is a lower bound on future costs, while in MAX problems it is an upper bound. Heuristic search algorithms usually consider the cost of the traversed path (g) and the admissible heuristic value (h) for every state. For the (n,k)-SIB problem, every state s is a snake, $g(s)$ is the length of s, and $h(s)$ upper bounds the length of the snake that starts from $head(s)$ and, together with s form the longest valid snake.

The most widely used heuristic search algorithm is A* (Hart, Nilsson, and Raphael 1968) which is a Best-First Search algorithm that expands states according to $f(s) = g(s) + h(s)$ and halts when a goal state is found. For MAX problems, Stern et al. (2014b) showed that A* can only halt when the best f-value is lower than or equal to the g-value of the incumbent solution. Another popular search algorithm for MAX problems is depth-first branch and bound (DFBnB). DFBnB, with an admissible heuristic, runs a depth-first search, and prunes states with f-value that is worse (smaller, in MAX problems) than the g-value of the incumbent solution.

The state space of the (n,2)-SIB problem is very large, containing $O(n^{2^n})$ states (all paths in an n-dimensional cube). Prior work handled this combinatorial explosion by employing symmetry breaking methods. In particular, Kochut (1996) observed that the identity of the dimensions is not important, and thus every snake has $n!-1$ snakes that are equal to it except for the order of dimensions. Building on this observation, Kochut proposed the following symmetry breaking method. A snake is allowed to traverse a dimension d only if it has already traversed every dimension $d' < d$ at least once. In our experiments we used this symmetry breaking method.

Next, we propose several novel admissible heuristics to further handle the large size of this problem's state space.

Notation	Represents
Q_n	A graph representing an n-dimensional cube.
$Q_n[s]$	Subgraph of Q_n containing only vertices legal for s
$R_n[s]$	Subgraph of $Q_n[s]$ containing only vertices reachable from $head(s)$
$N_k(v, G)$	Subgraph of G containing only vertices that are at most $k-1$ steps from vertex v
$\overline{N}_k(v, G)$	Subgraph of G that does not contain the vertices in $N_k(v, G)$

Table 2: Notations used to describe the proposed heuristics

Heuristics

Key to the success of A* and DFBnB, especially in large problems, is having an effective admissible heuristic. As one of the main contributions of this work, we propose several admissible heuristic functions for the (n,k)-SIB problem.

Legal Vertices Heuristic

The first admissible heuristic function we propose for (n,k)-SIB counts the number of legal vertices in a state. A vertex v is *legal* for a snake s if there exists a path s' with a spread of k such that s' contains v and s is a prefix of s'.

We call this heuristic, which counts the number of legal vertices, the *Legal Vertices* heuristic and denote it by h_{legal}. h_{legal} is computed for a snake s by maintaining the set of illegal vertics for s. A vertex is *Illegal* if it is not legal and we denote by $IL[s]$ the set of illegal vertices for s. Given $IL[s]$, computing the legal vertices heuristic is trivial: $h_{legal} = 2^n - |IL[s]|$. Showing that h_{legal} is admissible is also trivial.

The exact manner in which $IL[s]$ is computed is somewhat technical, and is based on measuring the Hamming distance between every vertex $v \in Q_n$ and every vertex $u \in s$. [2] This can be done efficiently in an incremental manner in

[1] Actually, one can define any state as a goal, since all states are valid snakes. However, as we want the longest snake, if a snake can be extended it is always desirable.

[2] Hood et al. (2011) showed that a vertex v cannot be added to a snake s if its Hamming distance is less than k for at least k vertices in s. Building on Hood et al.'s observation, we can deduce that a vertex v is *illegal* and can be added to $IL[s]$ if the number of vertices in s that has a Hamming distance less than k from v is not

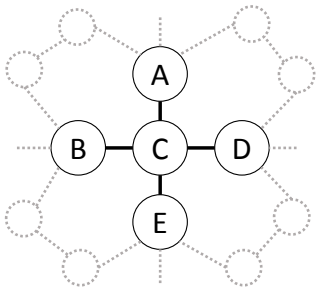

Figure 2: A small graph to demonstrate $h_{\overline{N}}$ and h_{IS}.

$O(n^{k-1})$. Since n and k are fairly small, maintaining $IL[s]$ requires reasonable computational overhead.

Reachable Heuristic

Removing the illegal vertices (and the edges adjacent to them) for a snake s from Q_n results in a subgraph of Q_n, denoted $Q_n[s]$. $Q_n[s]$ may not be connected. Thus not all legal vertices are reachable from $head(s)$. Let $R_n[s]$ be the connected component in $Q_n[s]$ that includes $head(s)$. The next admissible heuristic we propose, called *Reachable*, counts the number of vertices in $R_n[s]$. Computing $R_n[s]$ can be done by performing a simple exhaustive search on $Q_n[s]$ starting from $head(s)$.

Snake Head Pruning Heuristic

Consider the vertices in $R_n[s]$ adjacent to the head of the snake. Only one of these vertices can be part of a snake that s is a prefix of. Thus, the number of vertices in $R_n[s]$ minus the number of neighbors of $head(s)$ in $R_n[s]$ plus one is an admissible heuristic. Further pruning and thus a stronger heuristic can be obtained for $k > 2$, as follows.

Let $N_k(v, G)$ denote the subgraph of a graph G that contains all vertices that are $k-1$ or fewer hops away from a vertex v, and let $\overline{N}_k(v, G)$ be the complement subgraph that contains all vertices in G that are not in $N_k(v, G)$. Next, consider the vertices in $N_k(head(s), R_n[s])$ and $\overline{N}_k(head(s), R_n[s])$. A valid snake can pass through at most $k-1$ vertices in $N_k(head(s), R_n[s])$, and thus, the following heuristic, denoted $h_{\overline{N}}$, is an admissible heuristic:

$h_{\overline{N}}(s) = min(|R_n[s]|, |\overline{N}_k(head(s), R_n[s])| + k - 1)$

For convenience we list notations used so far in Table 2.

Example 3 *Consider the snake $s = (E, C)$, in the graph that consists of all the solid vertices and edges in Figure 2 and assume that $k = 2$. The head of s is at vertex C, and therefore the vertices A, B, and D are all reachable. Thus, $|R_n[s]| = 3$. However, only one of these vertices might be added to s in a valid snake with a spread of 2. Thus, $h_{\overline{N}}(s) = min(|R_n[s]|, |\overline{N}_2(head(s), R_n[s])| + 2 - 1) = min(3, 0 + 1) = 1$.*

Additive Disjoint Partition Heuristic

Let P=$\{P_1, ..., P_M\}$ be a disjoint partition of $\overline{N}_k(head(s), R_n[s])$, and let $U(P_i)$ be the maximal number of vertices that can be added to any snake that passes through P_i. For example, observe Figure 2. The solid vertices and edges represent a graph P_i, which is a subgraph of a graph G that is an item of a disjoint partition of G. Regardless of the topology of G, no valid snake can contain all the vertices in P_i for any $k > 1$. Specifically, for $k = 2$ it is easy to see that $U(P_i) = 4$, as a possible snake may contain the vertices A,B,D, and E. The dotted vertices and edges show a graph G where such a snake is possible. For $k \geq 3$, $U(P_i)$ will be 3.

Theorem 1 (Additive Disjoint Partition) *For any disjoint partition $P = \{P_1, .., P_M\}$ of any graph G it holds that $\sum_{i=1}^{m} U(P_i)$ is an admissible heuristic.*

Proof outline: Proof by contradiction. Assume that there is a snake s that is longer than $\sum_{i=1}^{m} U(P_i)$. Since P is a disjoint partition of G, it means that there exists P_i such that s contains more vertices from it than $U(P_i)$. This contradicts the definition of $U(P_i)$. □

Theorem 1 can be viewed as the MAX problem equivalent of the well-known additive disjoint pattern database heuristic which is known to be very effective in MIN problems (Felner, Korf, and Hanan 2004; Yang et al. 2008).

As in additive PDBs for MIN problems, the challenge of using Theorem 1 to construct an effective heuristic is to choose the best partition such that the resulting heuristic is tight and the computation of $U(P_i)$ is fast. We present below the partition we used in our experiments.

Note that heuristics derived from Theorem 1 are admissible for any graph G. Thus, it can augment all the heuristics proposed above, by setting G to be either $Q_n[s]$, $R_n[s]$, or $\overline{N}_k(head(s), R_n[s])$. As expected, the best results were obtained when using $\overline{N}_k(head(s), R_n[s])$ as the baseline graph to partition.

Independent Set Partition The first heuristic based on Theorem 1 that we propose generates a disjoint partition by considering the neighborhoods of a (not necessarily maximal) independent set of $N_2(v, \overline{N}_k(head(s), R_n[s]))$. Constructing this disjoint partition is done as follows.

First, a single vertex v is chosen from $\overline{N}_k(head(s), R_n[s])$ according to some vertex ordering.[3] The first partition P_1 is the vertex-induced subgraph of $\overline{N}_k(head(s), R_n[s])$ that contains v and all its neighbors, i.e., $P_1 = N_2(v, \overline{N}_k(head(s), R_n[s]))$.[4] The next partition is created similarly, choosing a vertex v' from $\overline{N}_k(head(s), R_n[s]) \setminus P_1$ and setting $P_2 = N_2(v', \overline{N}_k(head(s), R_n[s]) \setminus P_1)$. This process continues until all vertices in $\overline{N}_k(head(s), R_n[s])$ are covered.

To compute $U(P_i)$ we exploit the special structure of Q_n (recall that $\overline{N}_k(head(s), R_n[s])$ is a subgraph of Q_n). In

exactly equal to $k - min(k, Hamming(v, head(s)))$.

[3] In our experiments we used an arbitrary node ordering, and leave to future research the impact of different orderings on the heuristic's performance.

[4] A vertex-induced subgraph G' of a graph G is a subgraph of G whose edges are exactly all edges that exists in G between the vertices of G'.

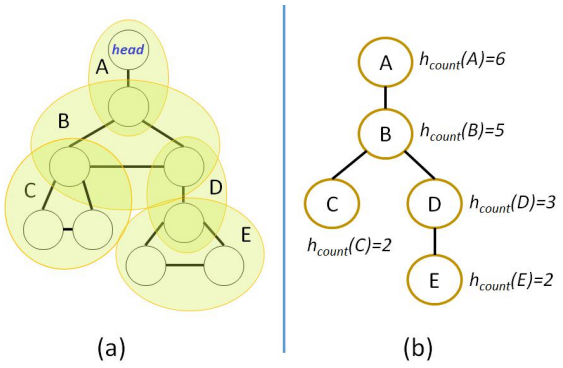

Figure 3: A small graph to demonstrate the BCT heuristic.

Q_n, there are no cliques of size 3 or more. Therefore, for every P_i we set $U(P_i)$ to be $max(|P_i|-1, 3)$. The heuristic resulting from the above partition and this computation of $U(P_i)$ is denoted by h_{IS}.

Bi-Connected Component Tree Heuristic

A biconnected graph is a graph that remains connected after removing any single vertex. An equivalent definition is that every two vertices have at least two vertex-disjoint paths connecting them. A *biconnected component* of a graph G is a maximal sub-graph of G that is biconnected. Every graph can be decomposed into a set of biconnected components (a.k.a blocks). Cut-points and blocks form a block-cut-point-tree that can be constructed in linear time (Harary and Prins 1966; Hopcroft and Tarjan 1973). Let $BCT[s]$ denote the block-cut-point-tree constructed from the subgraph of reachable vertices in $R_n[s]$. By definition, two biconnected components cannot have more than a single cut-point between them. Thus, once a snake traverses a cut-point, it can never return to the component it just left and all the vertices in that component become unreachable. Therefore, any valid snake can pass through a single branch in $BCT[s]$. An example of a graph and the biconnected components that compose it is shown in Figure 3(a).

A direct implication of the above observation is that for any biconnected component c, the maximal snake in the subtree rooted by c traverses through c and possibly through one of its children in $BCT[s]$. This observation can be formalized as follows. Let $H^*(c)$ be the length of the longest valid snake that visits only vertices in c, and let $H^*(c,c')$ be the vertices in c that are part of the longest valid path that visits both c and c'. The length of the longest valid snake in the subtree of $BCT[s]$ rooted by c, denoted h^*_{BCT}, is given by

$$h^*_{BCT}(c) = \max(H^*(c), \max_{c' \in child(c)} H^*(c,c') + h^*_{BCT}(c'))$$

Computing $h^*_{BCT}(c)$ is difficult because $H^*(c)$ and $H^*(c,c')$ are hard to compute. However, the above equation can be used to derive an admissible heuristic if $H^*(c)$ and $H^*(c,c')$ are replaced by *admissible estimates* of $H^*(c)$ and $H^*(c,c')$. We say that functions $H(c)$ and $H(c,c')$ are admissible estimates of $H^*(c)$ and $H^*(c,c')$ if $H(c) \geq H^*(c)$ and $H(c,c') \geq H^*(c,c')$, respectively. The resulting admissible heuristic h_{BCT} is computed as follows:

$$h_{BCT}(c) = \max(H(c), \max_{c' \in child(c)} H(c,c') + h_{BCT}(c'))$$

To instantiate this admissible heuristic, one needs to propose admissible $H(c)$ and $H(c,c')$ functions. Next, we propose several ways to compute such admissible functions.

Counting Heuristic The first admissible $H(c)$ and $H(c,c')$ functions ignores the difference between $H(c)$ and $H(c,c')$ and returns for both functions the number of reachable vertices in c minus one (to account for the head of the snake). We denote by h_{count} the implementation of h_{BCT} using the above $H(c)$ and $H(c,c')$.

h_{BCT} in general and h_{count} specifically are general heuristics in the sense that they apply not just to the (n,k)-SIB problems but to finding snakes (with spread k) in any graph. For simplicity, we demonstrate how h_{count} works on a simple graph depicted in Figure 3 (a). A through E denote the biconnected components of the graph and the BCT they compose is shown in Figure 3(b). E and C are leaves of this BCT, and thus $h_{count}(E) = 2$ and $h_{count}(C) = 2$. Following $h_{count}(D)$= 2-1+2=3. Now, $h_{count}(B) = max(2, 2+2, 2+3) = 5$, and finally we have $h_{count}(A) = max(1, 5+1) = 6$. By contrast, the reachable heuristic for the same setting is 8. Recall that since in MAX problems admissible heuristics are upper bounds on future costs, a heuristic that returns smaller values is stronger.

Intra-Component Heuristics The above implementation of $H(c)$ and $H(c,c')$ can be viewed as applying the reachable heuristic on the corresponding bicomponents. Next, we show how the same notions behind the more advanced heuristics, $h_{\overline{N}}$ and h_{IS}, can be used to implement more accurate admissible $H(c)$ and $H(c,c')$. We call the resulting BCT-based heuristics $h_{BCT+\overline{N}}$ and h_{BCT+IS}.

Snake Head Pruning. Every biconnected component c has an *entry point*, denoted $entry(c)$. This is the first vertex in c that will be added to a snake that will visit c. If c is the root of $BCT[s]$, its entry point is $head(s)$. Otherwise, the entry point is the vertex that connects c to its parent in $BCT[s]$.

Under the same reasoning explained for $h_{\overline{N}}$, of all the vertices in $N_k(entry(c), c)$ at most $k-1$ vertices will be added to any future valid snake. An admissible $H(c)$ based on this understanding is as follows:

$$H_{\overline{N}}(c) = min(|c|, \overline{N}_k(entry(c), c) + k - 1)$$

We extend the reasoning of $H_{\overline{N}}$ to obtain a better $H(c,c')$ heuristic, denoted similarly as $H_{\overline{N}}(c,c')$. The key idea in $H_{\overline{N}}(c,c')$ is that one can obtain a better estimate of how many vertices in c can be part of a snake if it is known that the snake will move from c to c'.

Every two biconnected components c and c' such that c' is a child of c in $BCT[s]$, have a single shared vertex. We call this vertex the *exit point* from c to c' and denote it by $exit(c,c')$. Every snake that visits c and c' must visit

$exit(c, c')$. Thus, from all the vertices in c that are $k-1$ steps from $exit(c, c')$, at most $k-1$ will be in the snake that passes through $exit(c, c')$. Thus, the same "pruning" done for the neighborhood of the entry point can be done for the neighborhood of the exit point. Formally, $H(c, c') = \min(|c|, |\overline{N}_k(exit(c), \overline{N}_k(entry(c), c))| + 2 \cdot (k\text{-}1))$

Independent Set Partitioning. Applying the same logic behind h_{IS} to further improve $H_{\overline{N}}(c)$ and $H_{\overline{N}}(c, c')$ is straightforward. The part of c that is not "pruned" is partitioned in the same manner described for $h_{\overline{N}}$, and the resulting heuristic is the sum over the maximal snake that can pass through each of these partitions.

From Snakes to Coils

n \ k	2	3	4	5	6	7
3	6*	6*	6*	6*	6*	6*
4	8*	8*	8*	8*	8*	8*
5	14*	10*	10*	10*	10*	10*
6	26*	16*	12*	12*	12*	12*
7	48*	24*	14*	14*	14*	14*
8	96*	36*	22*	16*	16*	16*
9	188	64	30*	24*	18*	18*
10	358	102	46*	28*	20*	20*
11	668	160	70	40*	30*	22*
12	1276	288	102	60	36*	32*
13	2468	494	182	80	50*	36*

Table 3: Longest known coils for different values of n and k.

A related problem to (n,k)-SIB is the problem of finding the longest cyclic path in Q_n having a spread of k. Such cyclic paths are called *coils* and have similar real-life application as the (n,k)-SIB problem. We denote by (n,k)-CIB the problem of finding the longest coil in Q_n with a spread of k. The complexity of (n,k)-CIB is similar to (n,k)-SIB, and as with (n,k)-SIB, the known bounds are not tight. Table 3 shows best currently known empirical lower bounds for coils with different values of n and k, as collected from (Hood et al. 2011; Kinny 2012; Abbott and Katchalski 1991; Meyerson et al. 2014). Values marked with an asterisk are known to be optimal.

(n,k)-SIB and (n,k)-CIB are very similar problems. We formulate (n,k)-CIB as a search problem in a very similar manner to how the described above formulation of (n,k)-SIB. A key difference is that the goal state is now more restricted, as the coil must end in the same vertex that it started. Next, we show how this difference manifests and can be exploited to solve (n,k)-CIB problem faster.

Improved Symmetry Breaking

The first main difference between (n,k)-SIB and (n,k)-CIB is that in (n,k)-CIB there is another form of symmetry-breaking we can use to avoid *rotational equivalence*. This method, called *bit count sequence*, was introduced by Hood et al. (2011). A given snake $s = (v_0, v_1, \ldots, v_l)$ is translated into a bit count sequence, $b = (b_0, b_1, \ldots, b_l)$ where the b_i is the number of bits equal to one in v_i. A snake s is pruned according to this symmetry-breaking method if the bit count sequence starting from a b_i where $i > 0$ is lexicographically larger than the bit count sequence starting from b_0. For details and an example, see Hood et al. (2011). Since this symmetry-breaking is orthogonal to our heuristics, we use it in all of our experiments on coils.

	Algorithm	Expanded	Runtime
n=6	A*	**1,883**	**0.067**
	DFBnB	2,642	0.167
n=7	A*	**272,185,127**	25,420
	DFBnB	275,613,574	**9,356**

Table 4: Performance of A* vs DFBnB.

Improved Snake Head Pruning

$h_{\overline{N}}$ was defined with respect to the head of the snake. However, in coils we also know where the coil will end up – at the initial vertex. Thus, we improve $h_{\overline{N}}$ by also "pruning" $N_k(tail(s), R_n[s])$. This is done exactly like the entry and exit point "pruning" described for $h_{BCT+\overline{N}}$.

Improved BCT

The BCT-based heuristics gain substantially from the knowledge that a coil will end up in the initial vertex. Instead of checking all branches of the BCT, BCT heuristics for (n,k)-CIB know exactly which branch of $BCT[s]$ the snake will traverse – the branch that contains $tail(s)$. This speeds up substantially all BCT-based heuristics.

Experimental Results

Next, we performed a set of experiments on solving (n,k)-SIB and (n,k)-CIB for different values of n and k with different search algorithms and heuristics.

Except when noted otherwise, all experiments were run on a Windows 8 computer with a 2.4GHz Intel Core i7-4700hq CPU and 8 GB of main memory.

Comparing Search Algorithms

First, we perform a set of experiments to choose which heuristic search algorithm to use to find provably optimal solutions to (n,k)-SIB and (n,k)-CIB. We compared A* with Depth First Branch and Bound (DFBnB). Since both (n,k)-SIB and (n,k)-CIB are MAX problems, A* was ran until a node is expanded with f-value lower than or equal to the length of the incumbent solution. DFBnB was executed until all states have been either visited or pruned. As a heuristic, we used the best heuristic we developed, which is h_{BCT+IS} (see below for a comparison of the different heuristics).

Table 4 shows the results for solving optimally the (6,2)-SIB and (7,2)-SIB problems. The "Expanded" and "Runtime" columns show the number of nodes expanded and total runtime in seconds, respectively, until the optimal solution has been verified. Due to memory limitations, the experiments of A* for (7,2)-SIB were conducted on a large computer with 1TB of main memory. Marked in bold are the best results in terms of nodes expanded and in terms of runtime.

n	k	Finding	Proving	Max Gain
11	5	182,166	8,402,424	1.02
7	2	6,405,745	286,365,771	1.02
10	5	254	2,693	1.09
12	6	36199	238,209	1.15
9	4	4045	22,965	1.18
8	3	339,040	731,753	1.46
11	6	152	143	2.06
7	3	293	71	5.13
6	2	2024	438	5.62

Table 5: Number of nodes expanded by DFBnB to find optimal solutions and to verify the optimality of that solution.

For both $n = 6$ and $n = 7$, A* expanded fewer nodes than DFBnB. This follows from the known "optimally efficient" property of A* (Dechter and Pearl 1985), which (roughly) states that any algorithm using the same heuristic would have to expand all the states expanded by A* to find solutions that are guaranteed to be optimal. In more detail, A* with a consistent heuristic expands all states with f value smaller than the optimal solution cost (denoted C^*) and some states with $f = C^*$. Other algorithms, including DFBnB, will also need to expand all states with $f < C^*$ and some of the states with $f = C^*$. However, DFBnB may also expand states with $f > C^*$, which A* would never expand. This explains the advantage A* has in terms of number of nodes expanded.

A*'s advantage in number of nodes expanded is translated to better runtime only for the easier problem of $n = 6$, where A* solves the problem in 0.067 seconds while DFBnB needed 0.167 seconds. However, for the harder problem of $n = 7$, DFBnB was faster than A* by more than a factor of 2. Indeed for $n = 7$ the difference in terms of number of nodes expanded is very small (DFBnB expanded 1.2% more nodes) and the computation time per state for DFBnB is much smaller, as there is no need to store states or use any sort of priority queue. Also, note that in this domain we did not implement any form of duplicate detection, as each state represents a path and not a single vertex, and preliminary experiments showed that performing duplicate detection in this problem is not worthwhile.

Measuring the Potential of A* and IDA* Using A* to solve hard (n,k)-SIB and (n,k)-CIB problems is not feasible due to A*'s high memory demands. An alternative is to use Iterative Deepening A* (Korf 1985) (IDA*). Like A*, IDA* expands all states with $f > C^*$, some of the states with $f = C^*$ and not a single state with $f < C^*$.[5] Thus, it has the potential to improve on DFBnB, which may expand states with $f < C^*$. In practice, however, we observed IDA* and A* to be inferior to DFBnB in practically all settings we experimented with. To understand why, consider the results shown in Table 5. Table 5 shows the number of states expanded by DFBnB with h_{BCT+IS} when solving (n,k)-CIB for different values of n and k. The "Finding" column shows the number of states expanded until a coil with the optimal solution length has been found. Some of these states may have $f < C^*$. The "Proving" column shows the number of states expanded afterwards until the optimal solution has been verified. Since DFBnB prunes states with f-values larger than the incumbent, all states counted in the "Proving" column have $f > C^*$ and thus will also be expanded by A* as well as IDA*.Thus, the potential gain, in terms of number of nodes expanded, of A* and IDA* over DFBnB is at most in the number of states expanded until an optimal coil has been found. The "Max Gain" column shows the ratio of such nodes from all the nodes expanded by DFBnB. As can be seen, the potential gain of using A* and IDA* over DFBnB in this domain is usually negligible, as most of the effort by DFBnB is exerted on proving optimality and not in finding the optimal solutions. This explains the superior runtime we observed for DFBnB in most settings: DFBnB has a much lower computational cost per node than A*, and does not traverse states multiple times (on the same branch) as IDA* (due to IDA*'s iterations).

[5]Note that for MIN problems these inequalities are reversed.

			Finding+Proving		Finding	
n	k	Heuristic	Expanded	Time	Expanded	Time
7	2	DFS	221,404,940,296	81,180	310,058,503	113
7	2	h_{legal}	31,603,173,285	36,138	139,479,007	168
7	2	Reachable	13,077,322,306	34,918	74,250,581	214
7	2	$h_{\overline{N}}$	5,146,255,022	15,084	40,117,094	112
7	2	h_{IS}	1,322,734,291	5,461	18,592,509	71
7	2	h_{count}	3,808,678,007	24,461	34,560,151	190
7	2	$h_{BCT+\overline{N}}$	1,163,736,005	7,769	14,855,619	92
7	2	h_{BCT+IS}	**292,771,516**	**2,407**	6,405,745	50
8	3	DFS	26,620,597	12	4,341,189	2
8	3	h_{legal}	16,848,839	35	3,337,321	7
8	3	Reachable	8,126,872	30	1,565,583	6
8	3	$h_{\overline{N}}$	3,437,740	14	837,311	3
8	3	h_{IS}	2,543,581	13	704,728	4
8	3	h_{count}	5,281,071	37	1,079,209	8
8	3	$h_{BCT+\overline{N}}$	1,549,806	14	427,836	4
8	3	h_{BCT+IS}	**1,070,793**	**11**	339,040	4
11	5	DFS	351,252,436	851	1,248,645	3
11	5	h_{legal}	270,183,042	4,103	1,186,531	21
11	5	Reachable	95,032,776	1,182	613,290	9
11	5	$h_{\overline{N}}$	20,274,615	317	283,038	6
11	5	h_{IS}	18,981,168	385	278,471	7
11	5	h_{count}	74,718,320	1,324	555,895	13
11	5	$h_{BCT+\overline{N}}$	9,563,469	**289**	187,320	7
11	5	h_{BCT+IS}	**8,584,590**	305	182,166	7

Table 6: (n,k)-CIB results with different heuristics

The results in Table 5 and 4 suggest that for (n,k)-SIB and (n,k)-CIB DFBnB is superior to A* in most cases, and in particular in the harder problems. All experiments reported below thus use DFBnB as the search method of choice.

Comparing Heuristics

Next, we compared the performance of the proposed heuristics against each other and against plain DFS, which is the current state of the art (Hood et al. 2011). We chose as representative the (7,2)-CIB, (8,3)-CIB, and (11,5)-CIB. These settings of n and k were the hardest problems solved in reasonable time. The number of nodes expanded and runtime in seconds are given in Table 6. The results under "Find-

			Finding+Proving		Finding	
n	k	Heuristic	Expanded	Time	Expanded	Time
8	3	DFS	556,120,186	117	37,943,854	8
8	3	h_{legal}	163,055,814	173	12,850,416	14
8	3	Reachable	89,405,692	112	7,242,748	9
8	3	$h_{\overline{N}}$	36,297,451	72	3,024,380	6
8	3	h_{IS}	20,214,140	**60**	1,790,446	5
8	3	h_{count}	37,531,696	233	3,141,984	18
8	3	$h_{BCT+\overline{N}}$	10,888,569	106	957,963	9
8	3	h_{BCT+IS}	**6,108,429**	84	639,229	8
11	5	DFS	9,219,466,466	25,092	5,836,031,188	15,578
11	5	h_{legal}	3,296,260,745	25,883	2,181,872,051	17,178
11	5	Reachable	1,789,435,857	7,054	1,193,196,676	4,605
11	5	$h_{\overline{N}}$	245,345,738	**1,633**	162,763,500	1,047
11	5	h_{IS}	196,828,023	2,147	132,791,420	1,432
11	5	h_{count}	739,969,859	7,076	499,355,508	4,689
11	5	$h_{BCT+\overline{N}}$	79,787,772	1,891	53,427,418	1,272
11	5	h_{BCT+IS}	**61,184,873**	3,428	41,686,854	2,301

Table 7: (n,k)-SIB results with different heuristics

ing" show the number of nodes expanded and runtime spent until the optimal coil has been found, while the results under "Finding+Proving" show the total number of nodes expanded and runtime, including finding an optimal solution and proving that it is optimal.

First, observe that as discussed in Table 5, most effort is spent in proving optimality. For example, for $n = 7$ and $k = 2$ when using h_{BCT+IS}, the number of nodes expanded until an optimal solution is found was 6,405,745 but the total number of nodes expanded until optimality was verified was 292,771,516. This again suggests that DFBnB is a suitable search algorithm for these tasks.

The second trend observed is that using more sophisticated heuristics reduces the number of nodes expanded, as expected. The best heuristic in terms of number of nodes expanded is h_{BCT+IS}, and in all configurations it resulted in a huge reduction in the number of nodes expanded compared to DFS without a heuristic. For example, h_{BCT+IS} expanded almost 40 times fewer states than DFS. These results clearly demonstrates the big potential that heuristic search algorithms have in solving MAX problems in general, and the (n,k)-CIB problem in particular.

Next, consider the runtime results when using the different heuristics. More sophisticated heuristics often require more computations. Thus, the huge savings in expanded nodes does not always translate to improved runtime. For example, consider the runtime results for $n = 8$ and $k = 3$ in Table 6. Although DFBnB with the best heuristic h_{BCT+IS} expanded more than 20 times fewer states than plain DFS, its gain in runtime was about 9%. This is because the computation time of h_{BCT+IS} is more than 20 times slower than the time needed just to expand a node (without any heuristic). By contrast, for $n = 7$ and $k = 2$, the reduction in the number of expanded nodes obtained by using DFBnB with h_{BCT+IS} was so large that a substantial reduction in runtime was also observed, solving $n = 7$ and $k = 2$ 30 times faster than DFS without a heuristic (which is the previous state of the art). Lastly, observe the runtime results of $n = 11$ and $k = 5$. Here too, h_{BCT+IS} is the strongest heuristic, as shown by the larger number of nodes DFBnB expands with the other heuristics. However, $h_{BCT+\overline{N}}$ expands only slightly more states and is easier to compute. Thus, the fastest configuration here is DFBnB with $h_{BCT+\overline{N}}$.

Results for (n,k)-SIB For the (n,k)-SIB we compared the proposed heuristics for solving the (8,3)-SIB and (11,5)-SIB problems. Table 7 shows the resulting number of nodes expanded and runtime in the same format of Table 6.

Notice that the number of nodes expanded and runtime for solving (n,k)-SIB problems is significantly larger than those reported for (n,k)-CIB in Table 6. This is because when searching for coils we also used the stronger "bit count sequence symmetry breaking method, which is not suitable for finding snakes.

Here too DFBnB using our proposed heuristics expands significantly fewer states than DFS. For (8,3)-SIB using h_{BCT+IS}, DFBnB expands 76 times fewer states than DFS, and for (11,5)-SIB as much as 137 times fewer expansions. In terms of runtime, however, the results of the more sophisticated BCT-based heuristics are less impressive and for both settings the best performing heuristic was $h_{\overline{N}}$ even though it expanded more states than h_{BCT+IS}. The runtime results for $h_{\overline{N}}$ surpass that of the current state of the art (DFS).

This difference in the performance of BCT-based heuristics between (n,k)-SIB and (n,k)-CIB problems highlights the different complexity of computing BCT-based heuristics for snakes and for coils. As explained earlier, BCT-based heuristics for snakes must consider all branches of the BCT while for coils only a single branch needs to be considered. Thus, the slower runtime for BCT-based heuristics in (n,k)-SIB is expected.

In conclusion, we observe the following behaviors:

- DFBnB is usually better than A* in this domain.
- Using the proposed heuristics results in a huge reduction in the number of states expanded.
- The more sophisticated heuristics are stronger but the cost of computing them may outweigh their benefit.

Conclusion and Future Work

This paper reports the first study of using heuristic search techniques to optimally solve the (n,k)-SIB and (n,k)-CIB problems. Both problems are MAX problems that are variants of the longest path problem. Beyond academic interest, solving these problems has important applications in coding theory and communications. Several domain-specific admissible heuristics were proposed, based on adapting additive pattern databases to MAX problems and based on decomposing the searched graph into a tree of biconnected components. Experimental results over a range of n and k values showed that DFBnB is the method of choice for finding optimal solutions to these problems and that with the proposed heuristics huge savings are achieved in terms of number of nodes expanded. In some cases, these savings translate into substantial reductions in runtime, while in other cases the cost of computing these heuristics outweighs their benefit.

Much research is left for future work. Faster implementations of the existing heuristics, e.g., by computing them incrementally, may result in substantial overall speedup. Metareasoning mechanisms can be developed to decide which heuristic is most cost effective to use in which states. Another promising direction is to apply memory-based heuristics to this problem. An orthogonal research direction is to develop and evaluate suboptimal and bounded suboptimal algorithms for the (n,k)-SIB and (n,k)-CIB problems. Ultimately, we aim to find optimal solutions to larger instances. We hope this work intices other search researchers to join us in this quest.

Acknowledgments

This research was supported by the Israel Science Foundation (ISF) under grant #417/13 to Ariel Felner and by the US NSF under grant #1150068 to Wheeler Ruml.

References

Abbott, H., and Katchalski, M. 1988. On the snake in the box problem. *Journal of Combinatorial Theory, Series B* 45(1):13–24.

Abbott, H. L., and Katchalski, M. 1991. On the construction of snake in the box codes. *Utilitas Mathematica* 40:97–116.

Casella, D. A., and Potter, W. D. 2005. Using evolutionary techniques to hunt for snakes and coils. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, 2499–2505. IEEE.

Chebiryak, Y.; Kröning, D.; Kröning, D.; and Kröning, D. 2008. *An efficient SAT encoding of circuit codes*. IEEE.

Danzer, L., and Klee, V. 1967. Lengths of snakes in boxes. *Journal of Combinatorial Theory* 2(3):258–265.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM (JACM)* 32(3):505–536.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

Douglas, R. J. 1969. Some results on the maximum length of circuits of spread k in the d-cube. *Journal of Combinatorial Theory* 6(4):323 – 339.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)* 22:279–318.

Garey, M. R., and Johnson, D. S. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Harary, F., and Prins, G. 1966. The block-cutpoint-tree of a graph. *Publ. Math. Debrecen* 13:103–107.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Hood, S.; Recoskie, D.; Sawada, J.; and Wong, D. 2011. Snakes, coils, and single-track circuit codes with spread k. *Journal of Combinatorial Optimization* 1–21.

Hopcroft, J., and Tarjan, R. 1973. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM* 16(6):372–378.

Kautz, W. H. 1958. Unit-distance error-checking codes. *Electronic Computers, IRE Transactions on* EC-7(2):179–180.

Kinny, D. 2012. A new approach to the snake-in-the-box problem. In *ECAI*, volume 242, 462–467.

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

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible treesearch. *Artificial Intelligence* 27(1):97–109.

Meyerson, S.; Whiteside, W.; Drapela, T.; and Potter, W. 2014. Finding longest paths in hypercubes, snakes and coils. In *Computational Intelligence for Engineering Solutions (CIES), 2014 IEEE Symposium on*, 103–109. IEEE.

Östergård, P. R., and Pettersson, V. H. 2014. Exhaustive search for snake-in-the-box codes. *Graphs and Combinatorics* 1–10.

Potter, W. D.; Robinson, R. W.; Miller, J. A.; Kochut, K.; and Redys, D. 1994. Using the genetic algorithm to find snake-in-the-box codes. In *IEA/AIE*, 421–426.

Puzis, R.; Elovici, Y.; and Dolev, S. 2007. Finding the most prominent group in complex networks. *AI Communications* 20(4):287–296.

Singleton, R. C. 1966. Generalized snake-in-the-box codes. *Electronic Computers, IEEE Transactions on* (4):596–602.

Stern, R.; Felner, A.; van den Berg, J.; Puzis, R.; Shah, R.; and Goldberg, K. 2014a. Potential-based bounded-cost search and anytime non-parametric A*. *Artificial Intelligence* 214:1–25.

Stern, R. T.; Kiesel, S.; Puzis, R.; Felner, A.; and Ruml, W. 2014b. Max is more than min: Solving maximization problems with heuristic search. In *Symposium on Combinatorial Search (SoCS)*.

Yang, F.; Culberson, J. C.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence research (JAIR)* 32:631–662.

Zémor, G. 1997. An upper bound on the size of the snake-in-the-box. *Combinatorica* 17(2):287–298.