Finding Lean Induced Cycles in Binary Hypercubes*

Yury Chebiryak[1], Thomas Wahl[1,2], Daniel Kroening[1,2], and Leopold Haller[2]

[1] Computer Systems Institute, ETH Zurich, Switzerland
[2] Computing Laboratory, Oxford University, United Kingdom

Abstract. Induced (chord-free) cycles in binary hypercubes have many applications in computer science. The state of the art for computing such cycles relies on genetic algorithms, which are, however, unable to perform a complete search. In this paper, we propose an approach to finding a special class of induced cycles we call *lean*, based on an efficient propositional SAT encoding. Lean induced cycles dominate a minimum number of hypercube nodes. Such cycles have been identified in Systems Biology as candidates for stable trajectories of gene regulatory networks. The encoding enabled us to compute lean induced cycles for hypercubes up to dimension 7. We also classify the induced cycles by the number of nodes they fail to dominate, using a custom-built All-SAT solver. We demonstrate how *clause filtering* can reduce the number of blocking clauses by two orders of magnitude.

1 Introduction

Cycles through binary hypercubes have applications in numerous fields in computing. The design of algorithms that reason about them is an active area of research. This paper is concerned with obtaining a subclass of these cycles with applications in Systems Biology.

Biochemical reactions in gene networks are frequently modeled using a system of piece-wise linear ordinary differential equations (PLDE), whose number corresponds to the number of genes in the network [4]. It is of critical importance to obtain *stable* solutions, because only stable orbits describe biologically relevant dynamics of the genes. We focus on Glass PLDE, a specific type of PLDE that simulates neural and gene regulatory networks [7].

The phase flow of Glass networks spans a sequence of coordinate orthants, which can be represented by the nodes of a binary hypercube. The orientation of the edges of the hypercube is determined by the choice of focal points of the PLDE. The orientation of the edge shows the direction of the phase flow

* A part of this work was presented at the 7th Australia – New Zealand Mathematics Convention, Christchurch, New Zealand, December 11, 2008. The work was supported by ETH Research Grant TH-19 06-3.

O. Kullmann (Ed.): SAT 2009, LNCS 5584, pp. 18–31, 2009.

at the coordinate plane separating the orthants. Thus, the paths in oriented binary hypercubes serve as a discrete representation of the continuous dynamics of Glass gene regulatory networks. A special kind of such paths, *coil-in-the-box* codes, is used for the identification of stable periodic orbits in the Glass PLDE. Coil-in-the-box codes with maximum length represent the networks with longest sequence of gene states for a given number of genes [10].

If a cycle in the hypercube is defined by a coil-in-the-box code, the orientation of all edges adjacent to the cycle can be chosen to direct the flow towards it (the cycle is then called a *cyclic attractor*). Such orientation ensures the convergence of the flow to a periodic attractor that lies in the orthants included in the path. If a node of the hypercube is not adjacent to the cycle, the node does not have edges adjacent to the cycle, and the orientation of the edges at this node does not affect the stability of the flow along the orthants that are defined by the coil-in-the-box code. The choice of edge orientation in turn is linked to the specification of focal points of the PLDE. Therefore, the presence of nodes that are not dominated indicates that the phase flow along the attractor is robust to any variations of the coefficients that define the equations in the orthant corresponding to that node [20]. We say that a node that is not dominated by the cycle is *shunned* by the cycle.

The computation of (preferably long) induced (i.e., chord-free) cycles that dominate as few nodes as possible is therefore highly desirable in this context. We call such cycles *lean induced cycles*.

The state-of-the art in computing longest induced cycles and paths relies on genetic algorithms [5]. However, while this technique is able to identify individual cycles with desired properties, it cannot guarantee completeness, i.e., it may miss specific cycles. Many applications, including those in Systems Biology, rely on a classification of *all* solutions, which precludes the use of any incomplete random search technique.

Recent research suggests that SAT-based algorithms can solve many combinatorial problems efficiently: applications include oriented matroids [18], the coverability problem for unbounded Petri nets [1], bounds on van der Waerden numbers [12,6], and many more. Solving a propositional formula that encodes a desired combinatorial object with a state-of-the-art SAT solver can be more efficient than the alternatives.

Contribution. We encode the problem of identifying lean induced cycles in binary hypercubes as a propositional SAT formula and compute solutions using a state-of-the-art solver. As we aim at the complete set of cycles, we modify the solver to solve the All-SAT problem, and present three orthogonal optimizations that reduce the number of required blocking clauses by two orders of magnitude.

Our implementation enabled us to obtain a broad range of new results on cycles of this kind. L. Glass presented a coil-in-the-box code with one shunned node in the 4-cube [10]. We show that this is the maximum number of shunned nodes that any *lean* induced cycle may have for that dimension. Then, we show

that the longest induced cycles in the next two dimensions are *cube-dominating*: these cycles dominate every node of the cube. In dimension 7, where an induced cycle can be almost twice as long as the shortest cube-dominating cycles, there are lean induced cycles shunning at least three nodes.

2 Preliminaries

We define basic concepts used frequently throughout the paper. The *Hamming distance* between two bit-strings $u = u_1 \dots u_n$, $v = v_1 \dots v_n \in \{0,1\}^n$ of length n is the number of bit positions in which u and v differ:

$$d_H^n(u,v) = |\{\, i \in \{1,\dots,n\} \,:\, u_i \neq v_i \,\}|\,.$$

The *n-dimensional Hypercube*, or *n-cube* for short, is the graph (V,E) with $V = \{0,1\}^n$ and $(u,v) \in E$ exactly if $d_H^n(u,v) = 1$ (see also [14]). The n-cube has $n \cdot 2^{n-1}$ edges. We use the standard definitions of *path* and *cycle* through the hypercube graph. The length of a path is the number of its vertices. A Hamiltonian path (cycle) through the n-cube is called a *(cyclic) Gray code*. The *cyclic distance* of two nodes W_j and W_k along a cycle of length L in the n-cube is

$$d_C^n(W_j, W_k) = \min\{|k-j|,\ L - |k-j|\}\,.$$

In this paper, we are concerned with particular cycles through the n-cube.

Definition 1. *An **induced cycle** $I_0 \dots I_{L-1}$ in the n-cube is a cycle such that any two nodes on the cycle that are neighbors in the n-cube are also neighbors in the cycle:*

$$\forall j,k \in \{0,\dots,L-1\} \quad (d_H^n(I_j,I_k) = 1 \ \Rightarrow\ d_C^n(I_j,I_k) = 1)\,. \tag{1}$$

Fig. 1 shows an induced cycle (bold edges) in the 4-cube. In this paper, we are also interested in the immediate neighborhood of the cycle:

Definition 2. *The cycle $I_0 \dots I_{L-1}$ **dominates** node W of the n-cube if W is adjacent to some node of the cycle:*

$$\exists j \in \{0,\dots,L-1\} \quad d_H^n(I_j, W) = 1\,. \tag{2}$$

We say the cycle *shuns* the nodes it does not dominate. A cycle is called *cube-dominating* if it dominates every node of the n-cube; such cycles can be thought of as "fat". In contrast, in this paper we are interested in "lean" induced cycles, which dominate as few nodes as possible:

Definition 3. *A **lean** induced cycle is an induced cycle through the n-cube that dominates a minimum number of cube nodes, among all induced n-cube cycles **of the same length**.*

Especially significant are induced cycles of maximum length. The induced cycle in Fig. 1 is longest (length 8) in dimension 4. It is also lean, as it dominates 15 of the 16 cube nodes, and there is no induced cycle of length 8 dominating less than 15 nodes.

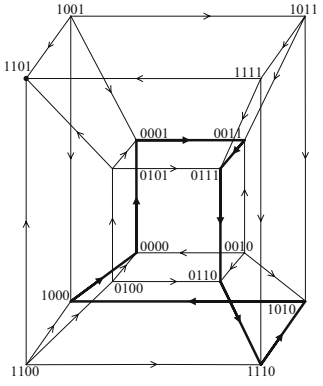

Fig. 1. A lean induced cycle in the 4-cube. The cycle shuns node 1101.

Lean induced cycles in cell biology. Hypercubes with lean induced cycles can aid the synthesis of Glass Boolean networks with stable periodic orbits and stable equilibrium states. For example, *C. elegans* vulval development is known to exhibit a series of cell divisions with 22 nuclei formed in the end of the development. The cell division represents a complex reactive system and includes at least four different molecular signaling pathways [15]. If the state of every signaling pathway is represented by a valuation of a Boolean variable, the 4-cube in Fig. 1 is useful for synthesizing a Glass Boolean network with a stable periodic orbit describing the cell division and an equilibrium depicting the finale state (at node 1101) of the gene regulatory system.

Co-existence of an induced cycle of maximum length and a shunned node in a hypercube indicates that during cell division, the gene network may traverse the maximum possible number of the different states before switching to the final equilibrium.

3 Computing Lean Induced Cycles

In this section, we describe an encoding of induced cycles of a given length into a propositional-logic formula. We then strengthen the encoding to assert the existence of a certain number of shunned nodes. We finally illustrate how we used the MiniSat solver to determine lean induced cycles where this number of shunned nodes is maximized.

3.1 A SAT-Encoding of Induced Cycles with Shunned Nodes

Our encoding relies heavily on comparing the Hamming distance between two hypercube nodes against some constant. We implement such comparisons efficiently using *once-twice* chains, as described in [3]. In brief, a once-twice chain

identifies differences between two strings up to some position j based on (i) comparing them at position j, and (ii) recursively comparing their prefixes up to position $j-1$.

Induced Cycles. We use $n \cdot L$ Boolean variables $I_j[k]$, where $0 \le j < L$ and $0 \le k < n$, to encode the coordinates of an induced cycle of length L in the n-cube. The variable $I_j[k]$ denotes the k-th coordinate of the j-th node. In order to form a cycle in an n-cube, consecutive nodes of the sequence must have Hamming distance 1, including the last and the first:

$$\varphi_{cycle} \;:=\; \left(\bigwedge\nolimits_{i=0}^{L-2} d_H^n(I_i, I_{i+1}) = 1\right) \;\wedge\; d_H^n(I_{L-1}, I_0) = 1\,.$$

To make the cycle induced, we eliminate chords as follows:

$$\varphi_{chord\text{-}free} \;:=\; \bigwedge_{\substack{0 \le i < j < L,\\ d_C^n(I_i, I_j) \ge 2}} d_H^n(I_i, I_j) \ge 2\,.$$

This also ensures that the nodes along the cycle are pairwise distinct. In practice, the formula $\varphi_{chord\text{-}free}$ can be optimized by eliminating half of its clauses, using an argument presented in [2].

The conjunction of these constraints is an encoding of induced cycles:

$$\varphi_{IC} \;:=\; \varphi_{cycle} \;\wedge\; \varphi_{chord\text{-}free}\,.$$

Shunned Nodes. We encode the property that a cycle $I_0 \dots I_{L-1}$ shuns nodes $u_0, \dots, u_{S-1}$, by requiring the distance of the nodes to the cycle to be at least 2:

$$\varphi_{shunned} \;:=\; \bigwedge\nolimits_{i=0}^{S-1} \bigwedge\nolimits_{j=0}^{L-1} d_H^n(u_i, I_j) \ge 2\,.$$

We combine this with the condition that the nodes are distinct,

$$\varphi_{distinct} \;:=\; \bigwedge\nolimits_{0 \le i < j < S} d_H^n(u_i, u_j) \ge 1\,,$$

to obtain an encoding of induced cycles with at least S shunned nodes:

$$\varphi_{ICS} \;:=\; \varphi_{IC} \;\wedge\; \varphi_{shunned} \;\wedge\; \varphi_{distinct}\,. \tag{3}$$

We point out some basic monotonicity properties of formula φ_{ICS}. Let $IC(n, L, S^+)$ be the number of induced cycles of length L in the n-cube with at least S shunned hypercube nodes. It is easy to see that

$$\begin{array}{lcl} n_1 \le n_2 \;\Rightarrow\; IC(n_1, L, S^+) \;\le\; IC(n_2, L, S^+)\,, & \text{and} \\ S_1 \le S_2 \;\Rightarrow\; IC(n, L, S_1^+) \;\ge\; IC(n, L, S_2^+)\,. & \end{array}$$

There is no analogous monotonicity law for the length parameter L of an induced cycle. Intuitively, a medium value for L provides the greatest degree of freedom for a cycle.

Table 1. Length of longest induced cycles, and number of shunned nodes

dim. n	length L	max. # shunned nodes
3	6	0
4	8	1
5	14	0
6	26	0
7	48	≥ 3

3.2 Computing Lean Induced Cycles Using a SAT Solver

Every solution to equation (3) corresponds to an induced cycle of length L in the n-cube with at least S shunned nodes. In order to make the cycle *lean*, we need to maximize S. We achieve this by starting with *cube-dominating* induced cycles, i.e., with $S = 0$, and increasing S in equation (3) until the SAT solver reports unsatisfiability.[1]

Table 1 shows our findings for hypercubes up to dimension 7. For the classical cube of dimension 3, the longest induced cycles have length 6. All of those are cube-dominating. In dimension 4, the longest induced cycles have length 8; an example is shown in Fig. 1. Some of these cycles shun 1 of the 16 cube nodes; the others are cube-dominating. Interestingly, in dimensions 5 and 6, all longest induced cycles are again cube-dominating.

In dimension 7, we found longest (length 48) induced cycles shunning 3 nodes. For larger values of S, our search timed out after 24h. In our experiments, we used the MiniSat solver by Eén and Sörensson [9]. MiniSat provides interfaces for incremental solving and All-SAT; the current version uses preprocessing techniques [8] that simplify the original formula. All experiments were carried out on an Intel Xeon 3.0 GHz, 4-GB RAM PC running Linux.

4 Classification of Induced Cycles

The goal of this section is to determine how many distinct induced cycles of length L and with S shunned nodes exist in the n-cube, for a given triple (n, L, S). By *distinct*, we mean that the cycles cannot be transformed into each other by applying a symmetry permutation of the n-cube. That is, for each tuple (n, L, S), we *classify* the induced cycles into equivalence classes.

The classification of induced cycles with respect to symmetries of a hypercube is of interest in Glass models for neural and gene regulatory networks, because the number of the equivalence classes of the codes indicates how many different types of cells can be regulated by a set of genes [21,10].

[1] Since the range of values for S for which (3) is satisfiable is contiguous, a binary search strategy is also possible, using a heuristically determined initial value for S.

The enumeration of the equivalence classes is achieved using a custom-made All-SAT solver derived from MiniSat. We introduce blocking clauses that suppress solutions symmetric to one encountered before. We observe that cycles identical up to cube symmetries belong to the same class (n, L, S). This ensures that the symmetry breaking does not eliminate solutions with a different set of parameters. In the rest of this section, we describe the classification and the symmetry breaking in more detail.

4.1 Identifying Equivalence Classes Using Coordinate Sequences

In order to identify symmetry equivalence classes of cycles, it proved efficient to encode cycles in a slightly different way.

Definition 4 ([10]). *The* ***coordinate sequence*** *of a cycle $I_0 \dots I_{L-1}$ in the n-cube is the sequence $(c_0, \dots, c_{L-1}) \in \{0, \dots, n-1\}^L$ such that c_i is the unique coordinate that distinguishes I_i and $I_{i+1 \bmod L}$.*

For example, the coordinate sequence of the cycle in Fig. 1 is the sequence $cs := (0,\ 1,\ 2,\ 0,\ 3,\ 2,\ 1,\ 3)$, assuming $I_0 = 0000$ and $I_1 = 0001$. The dimensions are listed in the order 3210 in the figure.

Given coordinate sequences, we can define cube symmetries.

Definition 5. *Two cycles C_1 and C_2 in the n-cube are* ***equivalent****, $C_1 \sim C_2$, if their coordinate sequences are identical up to axis permutations, reflections about the center position, and rotations by an arbitrary number of coordinates.*

Given n and L, let CS denote the set of coordinate sequences of cycles of length L in the n-cube. A reflection or rotation on CS is a permutation π on the set $\{0, \dots, L-1\}$ that maps a coordinate sequence $(c_i)_{i=0}^{L-1}$ to the sequence $(c_{\pi(i)})_{i=0}^{L-1}$, that is, by acting on the position indices of the sequence. In contrast, an axis permutation on CS is a permutation π on the set $\{0, \dots, n-1\}$ that maps a coordinate sequence $(c_i)_{i=0}^{L-1}$ to the sequence $(\pi(c_i))_{i=0}^{L-1}$, that is, by acting on the coordinate values of the sequence. For example, the coordinate sequence $cs' := (1,\ 0,\ 2,\ 3,\ 0,\ 1,\ 3,\ 2)$ is equivalent to sequence cs above, since cs' can be obtained from cs by a left-rotation by one position, followed by a reflection and an axis permutation (1 2 3 0), mapping 1 to 2, 2 to 3, etc.

Our goal is to classify induced cycles based on cube symmetries, for a given parameter tuple (n, L, S). In order for this classification to be sound, the symmetry permutations must not alter the (n, L, S) parameters of a cycle.

Lemma 1. *Let C_1 and C_2 be two equivalent cycles. Then C_1 and C_2 have the same length and shun the same number of cube nodes.*

Proof (sketch). Since C_1 and C_2 are equivalent, there is a sequence Π of permutations, of the type mentioned in definition 5, such that $\Pi(C_1) = C_2$. Reflections and rotations of the coordinate sequence of C_1 translate to reversals of C_1's orientation, and to rotations of C_1, respectively. These operations change neither the length of the cycle, nor the distance of cube nodes to it.

For an axis permutation π, we have to show that definition 2, *dominates*, is invariant under π. We omit the technical derivation of this property. □

As an example, the unique cycle of the 4-cube corresponding to the above-mentioned coordinate sequence cs', after fixing $I_0 := 0000$ and $I_1 := 0010$, is lean and induced, as is the cycle in Fig. 1. Both cycles shun one cube node. Conversely, cycles with the same parameters (n, L, S) may not be equivalent: Table 2 (see Appendix) lists two distinct – in the above sense – cycles with $(n, L, S) = (4, 8, 0)$.

We determine the number $IC(n, L, S)$ of $\sim$ equivalence classes of induced cycles of length L with *exactly* S shunned nodes as the difference between the number of classes of cycles shunning at least S and $S + 1$ nodes, respectively:

$$IC(n, L, S) \;=\; IC(n, L, S^+) \;-\; IC(n, L, (S+1)^+)\,. \tag{4}$$

The quantities on the right are computed, separately for S and $S + 1$, by enumerating satisfying assignments to Eq. (3), using an All-solutions SAT solver, implemented on top of MiniSat (see Algorithm 1 on the next page).

As proposed in [3], we encode coordinate sequences using XOR gates on Boolean variables denoting coordinates of a cycle. We write $xor^k[m]$ to refer to the m-th bit in bitwise xor-operation over coordinates of nodes I_k and $I_{k+1 \bmod L}$. For example, if $xor^3[2]$ evaluates to true, dimension 2 is traversed while going from I_3 to I_4. We call the variables $xor^k[m]$ the "xor-variables".

To ensure a single representative for each $\sim$ equivalence class, we add blocking clauses for each solution found that prevent permutations of axes, rotations and reflections of the coordinate sequence of the solution. The number of blocking clauses to add per solution is $(2L \cdot n!)$. This is clearly a computational burden for the SAT solver, especially when the solution space is nearly exhausted, and the All-SAT procedure is about to find the formula to be unsatisfiable. In the rest of this section, we present techniques that reduce both the number and the length of the blocking clauses.

4.2 Optimizations

Compressing blocking clauses. A blocking clause for a given induced cycle, barring permutations of axes and rotations/reflections of a coordinate sequence, is expressed in terms of the variables encoding the sequence. For instance, to block permutations of the cycle in Fig. 1, we add the following clause:

$$\begin{aligned}
&(\neg xor^0[0] \vee xor^0[1] \vee xor^0[2] \vee xor^0[3] \\
&\vee xor^1[0] \vee \neg xor^1[1] \vee xor^1[2] \vee\ xor^1[3] \\
&\vdots \\
&\vee xor^7[0] \vee xor^7[1] \vee xor^7[2] \vee \neg xor^7[3]\ )\,.
\end{aligned}$$

Algorithm 1: COMPUTE-EQUIVALENCE-CLASSES

Input: the SAT instance I with fixed n, L, S;
the equivalence relation $\sim$
Output: The set of equivalence classes EC

```
1: EC := {}
2: SAT_solver.solve(I)
3: while SAT
4:   do IC = SAT_solver.decode()
5:      EC ← EC ∪ {IC}
6:      ∀IC_j ∼ IC.  I.add_blocking_clause(IC_j)
7:      SAT_solver.solve(I)
```

The length of this blocking clause is $(n \cdot L)$. Our first, and simplest, optimization is to omit literals that evaluate to *false*, since we know that these variables encode unit Hamming distance:

$$(\neg xor^0[0] \vee \neg xor^1[1] \vee \neg xor^2[2] \vee \neg xor^3[0] \vee \dots) .$$

This reduces the length of a clause to L.

Symmetric Cycles. The following optimization applies to specific cycles, called *symmetric induced cycles*. A Gray code is *symmetric*[2] if elements of its coordinate sequence that are $L/2$ apart are identical [19]. For a symmetric induced cycle, the number of blocking clauses to be added can be reduced by one-half: rotations by more than $L/2$ positions result in cycles that were already blocked.

Prefix Filtering. Without loss of generality, we fix the first two elements of the coordinate sequence to $(0, 1)$. For the next coordinate, dimension 0 cannot be traversed because this would form a chord. Neither can dimension 1, since the cycle must be simple. Out of higher dimensions, we can restrict the search to the *canonical class*[3] with prefix $(0, 1, 2)$. We enforce this prefix by fixing the values of the corresponding xor-variables using the following three clauses:

$$xor^0[0] \wedge xor^1[1] \wedge xor^2[2] . \tag{5}$$

This drastically reduces the number of solutions in each equivalence class, and eliminates a large number of blocking clauses. For example, it becomes unnecessary to add a blocking clause for the coordinate sequence cs' on page 24, as cs' is blocked by Eq. (5).

[2] This definition is not to be confused with the definition in [13], where this term refers to a code for which the number of bit changes is uniformly distributed among the bit positions, hence called a *balanced* Gray code in [17, p. 7].

[3] A canonical coordinate sequence is the one in which each coordinate k appears before the first appearance of $k + 1$ [11].

Phase Saving. In an attempt to speed up the enumeration of solutions, we added phase-saving [16] to MiniSat. By default, MiniSat assigns *false* to all decision variables. With phase saving, they are assigned their most recent values in the search. Phase saving combines well with aggressive restarting schemes, since it retains more information between restarts. Our intuition was that after finding a solution, the solver might be able to quickly identify neighboring solutions. Phase-saving alone, however, did not result in any speedups.

Ordering decision variables. Upon closer inspection of All-SAT runs, we found that the activity-based variable selection heuristic mainly chooses from a small set of branching variables. These variables correspond closely to the encoding of solutions in the input CNF. In order to make use of this insight, we extended the solver to allow for prioritization of important variables in the decision heuristic: In this modification, unprioritized variables are only considered for branching after all prioritized variables are assigned a value. We tested a number of possible restrictions, and found that prioritizing the variables that encode the induced-cycle nodes $I_0, \ldots, I_{L-1}$ works well for some instances, but yields bad results in general.

Combined Restart Policy. We found that the enumeration of solutions could be sped up by disabling the geometric restart scheme, but this led to bad performance on the final hard instances. By combining an initial high restart limit (100000 conflicts) with a subsequent switch to MiniSat's original geometric policy, starting again from a very low limit (100 conflicts), we were able to gain a 20% overall speed-up. Easier SAT instances can then be solved before the first restart, while hard instances still profit from aggressive restarts.

Further experiments with different combinations of the discussed strategies revealed that a combination of a high-restart limit, variable prioritization, and phase saving also led to a performance increase of about 20%.

4.3 Evaluation

Using prefix filtering and the optimizations for symmetric cycles, we are able to reduce the number of clauses drastically. As an example, consider an instance encoding induced cycles of length 26 in a 6-dimensional hypercube. In order to block a solution, we need to add only 312 blocking clauses in the non-symmetric case and 156 clauses for a symmetric cycle, instead of originally 37440. Our findings are presented in Fig. 2 and extend the classification presented in [22].

For some circuit length values L, the time required by the All-SAT solver increases with the number of shunned nodes. For such values of L, it is faster to perform the classification for a small value of S and then check how many nodes the cycles dominate.

In general, the time required to find the first induced cycle is a few orders of magnitude less than that to perform the classification, even in the case of one class only, as the run-time is dominated by the final unsatisfiable instance.

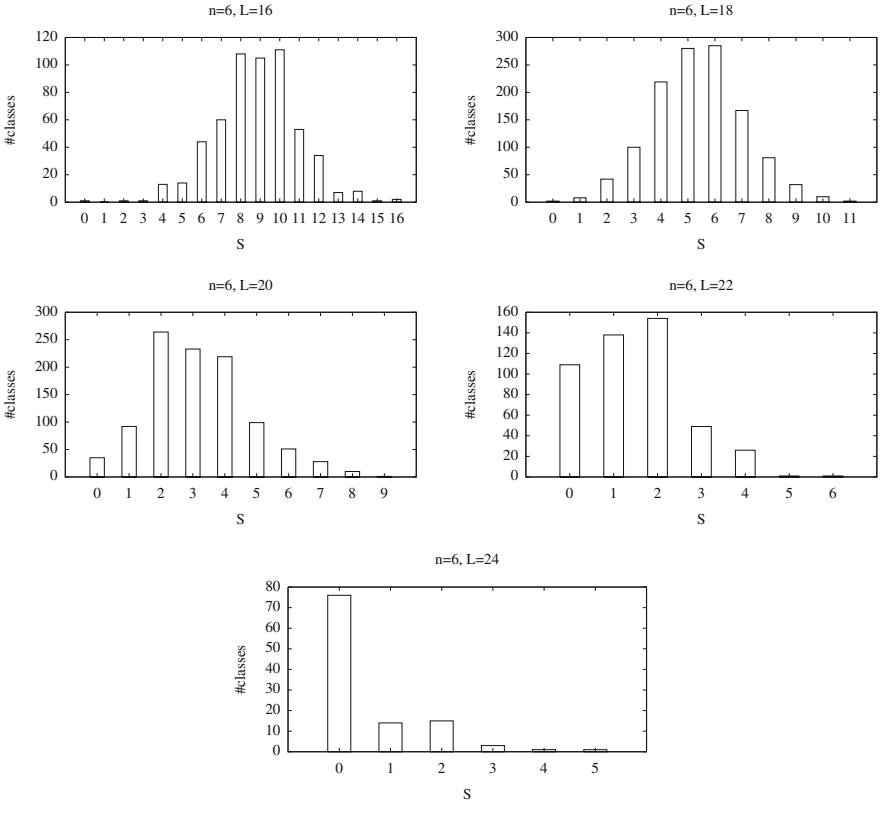

Fig. 2. Classification of induced cycles by cube symmetries, for select triples (n, L, S)

5 Conclusion

In this paper we have formalized a combinatorial problem relevant in Systems Biology: finding lean induced cycles in a hypercube, i.e., induced cycles that dominate a minimum number of hypercube nodes. We have presented a solution to this problem based on an efficient SAT encoding, and used this encoding to find lean induced cycles using a SAT solver. When compared to genetic algorithms, our method can provide guarantees for finding solutions, or prove the absence thereof.

Our method is suitable for classifying large sets of solutions into symmetry equivalence classes. As suggested by Fig. 2, this allows insights into the distribution of distinct solutions across the parameters n, L, and S. The SAT solver's performance is improved by filtering blocking clauses based on combinatorial properties of induced cycles, and by applying All-SAT specific internal tunings.

Acknowledgments

The authors would like to thank Dr. Igor Zinovik for bringing their attention to the problem of lean induced cycles and helping with preparing this script. They also thank the anonymous reviewers for suggestions on how to improve the draft.

References

1. Abdulla, P.A., Iyer, S.P., Nylén, A.: SAT-solving the coverability problem for Petri nets. Formal Methods in System Design 24(1), 25–43 (2004)
2. Chebiryak, Y., Kroening, D.: An efficient SAT encoding of circuit codes. In: Procs. IEEE International Symposium on Information Theory and its Applications, Auckland, New Zealand, December 2008, pp. 1235–1238 (2008)
3. Chebiryak, Y., Kroening, D.: Towards a classification of Hamiltonian cycles in the 6-cube. Journal on Satisfiability, Boolean Modeling and Computation (JSAT) 4, 57–74 (2008)
4. de Jong, H., Page, M.: Search for steady states of piecewise-linear differential equation models of genetic regulatory networks. IEEE/ACM Trans. Comput. Biology Bioinform. 5(2), 208–222 (2008)
5. Diaz-Gomez, P.A., Hougen, D.F.: Genetic algorithms for hunting snakes in hypercubes: Fitness function analysis and open questions. In: SNPD-SAWN 2006: Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Washington, DC, USA, pp. 389–394. IEEE Computer Society, Los Alamitos (2006)
6. Dransfield, M.R., Marek, V.W., Truszczynski, M.: Satisfiability and computing van der Waerden numbers. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 1–13. Springer, Heidelberg (2004)
7. Edwards, R.: Symbolic dynamics and computation in model gene networks. Chaos 11(1), 160–169 (2001)
8. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
9. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
10. Glass, L.: Combinatorial aspects of dynamics in biological systems. In: Landman, U. (ed.) Statistical mechanics and statistical methods in theory and applications, pp. 585–611. Plenum Press (1977)
11. Knuth, D.E.: The Art of Computer Programming. fascicle 2: Generating All Tuples and Permutations, vol. 4. Addison-Wesley Professional, Reading (2005)
12. Kouril, M., Franco, J.V.: Resolution tunnels for improved SAT solver performance. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 143–157. Springer, Heidelberg (2005)
13. Liu, X., Schrack, G.F.: A heuristic approach for constructing symmetric Gray codes. Appl. Math. Comput. 155(1), 55–63 (2004)
14. Livingston, M., Stout, Q.: Perfect dominating sets. Congressus Numerantium 79, 187–203 (1990)

15. Na'aman Kam, D., Kugler, H., Rami Marelly, A., Hubbard, J., Stern, M.: Formal modelling of C. elegans development. A scenario-based approach. Modelling in Molecular Biology, 151–174 (2004)
16. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 294–299. Springer, Heidelberg (2007)
17. Savage, C.: A survey of combinatorial Gray codes. SIAM Review 39(4), 605–629 (1997)
18. Schewe, L.: Generation of oriented matroids using satisfiability solvers. In: Iglesias, A., Takayama, N. (eds.) ICMS 2006. LNCS, vol. 4151, pp. 216–218. Springer, Heidelberg (2006)
19. Singleton, R.C.: Generalized snake-in-the-box codes. IEEE Transactions on Electronic Computers EC-15(4), 596–602 (1966)
20. Zinovik, I., Chebiryak, Y., Kroening, D.: Cyclic attractors in Glass models for gene regulatory networks. IEEE Trans. Inf. Theory: Special Issue on Molecular Biology and Neuroscience (December 2009) (accepted)
21. Zinovik, I., Kroening, D., Chebiryak, Y.: An algebraic algorithm for the identification of Glass networks with periodic orbits along cyclic attractors. In: Anai, H., Horimoto, K., Kutsia, T. (eds.) Ab 2007. LNCS, vol. 4545, pp. 140–154. Springer, Heidelberg (2007)
22. Zinovik, I., Kroening, D., Chebiryak, Y.: Computing binary combinatorial Gray codes via exhaustive search with SAT-solvers. IEEE Transactions on Information Theory 54(4), 1819–1823 (2008)

6 Appendix

Table 2 shows runtimes, and number of equivalence classes of induced cycles found, for various values of (n, L, S).

Table 2. Classification of induced cycles, with runtimes

n	L	S	Time (sec)		#classes	
			first cycle	All-SAT	$IC(n, L, S^+)$	$IC(n, L, S)$
4	6	0	0.003	0.010	1	0
		1	0.006	0.017	1	0
		2	0.010	0.028	1	1
		3		0.031		0
	8	0	0.007	0.305	3	2
		1	0.009	0.048	1	1
		2		0.015		0
5	10	0	0.016	400.378	10	0
		1	0.017	419.881	10	0
		2	0.027	392.274	10	3
		3	0.031	370.277	7	3
		4	0.047	356.335	4	3
		5	0.043	210.137	1	0
		6	0.095	183.403	1	1
		7		167.397		0
	14	0	0.033	535.027	3	3
		1		3.012		0
6	16	0	0.02	486.37	563	1
		1	0.01	534.55	562	0
		2	0.03	481.12	562	1
		3	0.02	514.36	561	1
		4	0.04	501.77	560	13
		5	0.04	768.08	547	14
		6	0.04	3252.77	533	44
6	24	0	0.08	1183.50	110	76
		1	0.07	695.37	34	14
		2	0.30	689.22	20	15
		3	1.76	592.51	5	3
		4	5.65	1062.92	2	1
		5	26.56	1364.86	1	1
		6	-	1014.34		0
6	26	0	0.42	583.43	4	4
		1	-	750.39	0	0