Chinese Remainder Encoding for Hamiltonian Cycles

Marijn J. H. Heule

Computer Science Department
Carnegie Mellon University, Pittsburgh, PA, United States
marijn@cmu.edu

Abstract. The Hamiltonian Cycle Problem (HCP) consists of two constraints: i) each vertex contributes exactly two edges to the cycle; and ii) there is exactly one cycle. The former can be encoded naturally and compactly, while the encodings of the latter either lack arc consistency or require an exponential number of clauses. We present a new, small encoding for HCP based on the Chinese remainder theorem. We demonstrate the effectiveness of the encoding on challenging HCP instances.

1 Introduction

Satisfiability (SAT) solvers have become very powerful tools to solve many hard combinatorial problems in a broad range of applications. However, the quality of the encoding can have a significant impact on the effectiveness of a SAT solver, in particular for problems with complicated constraints.

One problem class for which the encoding plays a crucial role in solver performance is the Hamiltonian Cycle Problem (HCP) [14], an NP-complete problem that has been studied from theoretical and practical viewpoints [2,3,6,13]. Given a graph, HCP asks whether there exists a cycle that visits all vertices of the graph exactly once. One graph with and one without a Hamiltonian cycle are shown in Figure 1. On a high level, HCP requires two constraints: a *degree constraint* stating that each vertex contributes exactly two edges to the cycle and an *exactly-one-cycle constraint*. The degree constraint can be compactly encoded with arc consistency [4], a property that is important for efficient solving.

Effectively dealing with the exactly-one-cycle constraint is more challenging. Several encodings have been proposed use $\Theta(|V|^3)$ clauses. Determining the existence of a Hamiltonian cycle is easy for small graphs (< 100 vertices). For larger graphs, $\Theta(|V|^3)$-sized encodings result in a huge formula that are hard to solve [9–11]. The challenge is to come up with a compact encoding that can also be solved efficiently. Our encoding will be quasilinear in the number of edges.

One way to avoid the costliness of the exactly-one-cycle constraint is to use incremental SAT [1,12]. In this setting, the initial formula only consists of the degree constraint. If the SAT solver produces a solution that represents multiple cycles, then a clause is added that blocks the shortest cycle. This is repeated until either a Hamiltonian cycle is found or if the formula becomes unsatisfiable, showing that no such cycle exists.

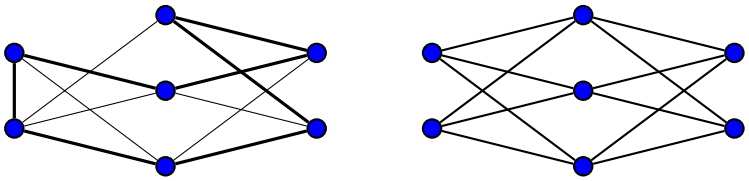


Fig. 1. The left graph has a Hamiltonian cycle (bold), while the right one does not.

Recently two new HCP encodings have been proposed. Both encodings assign a binary index to each vertex using $k = \lceil \log_2 |V| \rceil$ variables per vertex. The first one is based on linear-feedback shift registers (LFSR) [5,8]. LFSR loops through the numbers $\{1, \ldots, 2^k - 1\}$ by shifting a binary number by one position to the left and puts the parity of some bits in the vacated position. This facilitates a compact SAT encoding. The second encoding uses a binary adder that loops through the numbers $\{0, \ldots, 2^k - 1\}$ in ascending order and returns to 0 after $2^k - 1$ [14]. The binary adder encoding requires auxiliary variables, more clauses, and/or longer clauses compared to LFSR. Yet, the binary adder is more effective as it facilitates quick refutation of some subcycles, e.g., cycles of odd length.

In this paper, we present the Chinese remainder encoding that aims to combine the best of the incremental SAT, binary adder, and LFSR approaches. From the incremental approach, we borrow the observation that only some subcycles need to be blocked. From the binary adder approach we borrow techniques to easily refute some subcycles. Finally, from the LFSR approach we borrow the compact encoding with short clauses without auxiliary variables.

We implemented the binary adder, LSFR, and the Chinese remainder encodings (and corresponding decoding tools), and evaluated their effectiveness on graphs from the Flinders HCP challenge [7]. This is a suite of 1001 graphs with HCP instances of varying difficulty. The experimental results show that the Chinese remainder encoding beats the other two on most large graphs.

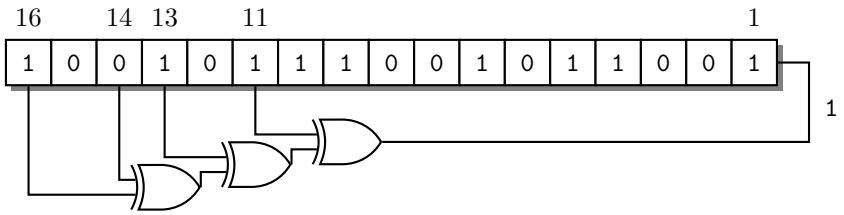
2 Preliminaries

Boolean Satisfiability: We consider formulas in *conjunctive normal form* (CNF), defined as follows. A *literal* is either a variable x or the negation $\overline{x}$ of a variable x. For a literal l, $var(l)$ denotes the variable of l. A *clause* is a disjunction of literals and a *formula* is a conjunction of clauses. An *assignment* is a function from a set of variables to the truth values 1 (*true*) and 0 (*false*). A formula is *satisfiable* if there exists an assignment that satisfies it and is *unsatisfiable* otherwise.

A *unit clause* is a clause that contains only one literal. The result of applying the unit-clause rule to a formula F is the formula F without all clauses containing the unit literal and without all occurrences of the negated unit literal. The iterated application of the unit-clause rule to a formula, until no unit clauses are left, is called unit propagation. If unit propagation on a formula F yields the empty clause, we say that it derived a conflict on F.

Linear-Feedback Shift Register: An LFSR [5] is a register that in each step shifts all bits by one position to the left and replaces the vacated position by the result of an XOR operation of some of the bits. Given the right XOR, an LFSR visits all bit-vectors of a given length except the all-zero bit-vector. The shift and the XOR operations can be compactly encoded using clauses.

Example 1. An example 16-bit LFSR fills the vacant bit by $x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$, resulting in $2^{16} - 1 = 65,535$ states. The figure below illustates this LFSR with state `10010111001011001`. The next state is `00101110010110011`.



3 Encodings

In this section, we focus on encodings that have been reasonably effective for HCP in the past. This excludes unary-based encodings [14]. We first discuss an encoding for the degree constraint and afterwards two encodings for the exactly-one-cycle constraint. To improve readability, we show the constraints using the logical connectives $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies), $\leftrightarrow$ (equivalence), and $\not\leftrightarrow$ (xor).

3.1 Degree Constraint

All of the encodings share the same variables and clauses to enforce that exactly two edges from each vertex are in the cycle, thereby ensuring that each vertex is in exactly one cycle. Given an undirected graph $G = (V, E)$, we introduce two variables $e_{i,j}$ and $e_{j,i}$ for each edge $(i, j) \in E$. In the case that G is a directed graph, only $e_{i,j}$ is used for arcs from i to j and only $e_{j,i}$ is used for arcs from j to i. The degree constraint is encoded by enforcing that for each vertex $v \in V$ exactly one of the literals $e_{i,v}$ is true (one incoming edge) and exactly one of the literals $e_{v,j}$ is true (one outgoing edge). Each ExactlyOne constraint is partitioned into an AtLeastOne constraint (i.e., a clause) and an AtMostOne constraint.

HCP is typically only hard for graphs with low degree. For graphs with high degree, we expect dedicated heuristics to outperform SAT solving. The constraint $\mathsf{AtMostOne}(x_1, \ldots, x_n)$ can therefore simply be encoded using the pairwise encoding $(\overline{x}_i \vee \overline{x}_j)$ for $1 \leq i < j \leq n$. However, some graphs in the Flinders HCP challenge set are dense. To avoid a blow-up in size due to the pairwise encoding, we only use the pairwise encoding for AtMostOne constraints of 4 or less inputs. Larger AtMostOne constraints are split recursively as follows:

$$\mathsf{AtMostOne}(\mathsf{x}_1, \ldots, \mathsf{x}_n) := \mathsf{AtMostOne}(\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3, \overline{\mathsf{y}}) \wedge \mathsf{AtMostOne}(\mathsf{y}, \mathsf{x}_4, \ldots, \mathsf{x}_n)$$

3.2 Binary Adder

Given a graph $G = (V, E)$, the binary adder encoding assigns a unique index from the range $\{0, 1, \ldots, |V| - 1\}$ to each vertex in the graph. Each vertex has two neighbors, of which one is the successor ($+1 \pmod{|V|}$), while the other is its predecessor ($-1 \pmod{|V|}$). If edge variable $e_{u,v}$ is true, then the successor property is enforced, i.e., u is assigned i and v is assigned $i + 1 \pmod{|V|}$. The clauses that enforce the successor property also enforce the predecessor property.

We will use parts of the binary encoding in the Chinese remainder encoding. For consistency we therefore use the naming of the bit-vectors. The i^{th} bit of the bit-vector of vertex v is denoted by the Boolean variable v_{2^i} with $i \in \{1, \ldots, k\}$.

Example 2. Consider a graph with 7 vertices, thus $k = \lceil \log_2 7 \rceil = 3$. For vertex v, the variables v_2, v_4, and v_8 denote the least, middle, and most significant bit, respectively. For an edge variable $e_{u,v}$, we add the clauses represented by:

$$e_{u,v} \rightarrow (u_2 \not\leftrightarrow v_2) \qquad \begin{array}{l} (e_{u,v} \wedge \overline{u}_2) \rightarrow (u_4 \leftrightarrow v_4) \\ (e_{u,v} \wedge u_2) \rightarrow (u_4 \not\leftrightarrow v_4) \end{array} \qquad \begin{array}{r} (e_{u,v} \wedge \overline{u}_2) \rightarrow (u_8 \leftrightarrow v_8) \\ (e_{u,v} \wedge \overline{u}_4) \rightarrow (u_8 \leftrightarrow v_8) \\ (e_{u,v} \wedge u_2 \wedge u_4) \rightarrow (u_8 \not\leftrightarrow v_8) \end{array}$$

No auxiliary variables are introduced in our implementation, resulting in $O(k^2)$ clauses per edge variable. Using auxiliary variables can reduce the number of clauses to $O(k)$ per edge variables, but this is only effective for large k.

It is important to observe that this encoding is able to quickly refute certain subcycles. For example, if an assignment to the edge variables forms a subcycle of odd length, then assigning v_2 (of any vertex v in that cycle) to true or false results in a conflict by unit propagation. Thus with two conflicts the cycle can be refuted. In a similar way, one can additionally refute all cycles of length 2 (mod 4) by the four assignments to the variables v_2 and v_4. This is the key property that will be used in the Chinese remainder encoding.

3.3 Linear-Feedback Shift Register

Haythorpe and Johnson propose to use LFSR to enforce the exactly-one-cycle constraint in HCP [8]. This encoding has several aspects that are similar to the binary adder encoding. It uses bit-vectors of length $k = \lceil \log_2(|V| + 1) \rceil$ for each vertex. One vertex is assigned the bit-vector with all-zeros except for the least significant bit. The bit-vectors of adjacent vertices are forced to be the next and previous state of a k-bit LFSR.

We use the following variable naming: Given a k-bit LFSR, let $n = 2^k - 1$. For each $i \in \{1, \ldots, k\}$, position i in the bit-vector of vertex v is denoted by $v_{n,i}$. We block the all-zero bit-vector, by adding $(\overline{v}_{n,1} \vee \cdots \vee \overline{v}_{n,k})$ for each $v \in V$.

Example 3. Consider a 7-vertex graph, thus $k = \lceil \log_2(7+1) \rceil = 3$. A 3-bit LFSR filling the vacant bit by $x_2 \oplus x_3$ has 7 states. The bit-vector variables of vertex

v are $v_{7,1}$, $v_{7,2}$, and $v_{7,3}$. For an edge variable $e_{u,v}$, we add the clauses:

$$e_{u,v} \rightarrow (v_{7,1} \leftrightarrow (u_{7,2} \not\leftrightarrow u_{7,3})$$
$$e_{u,v} \rightarrow (v_{7,2} \leftrightarrow u_{7,1})$$
$$e_{u,v} \rightarrow (v_{7,3} \leftrightarrow u_{7,2})$$

For most small k, there exists a k-bit LFSR of $2^k - 1$ states that uses just a single XOR of 2 bits. For such an LFSR, the encoding uses $2k-2$ ternary clauses and 4 clauses of length 4, as in the above example. An encoding based on LFSR is thus compact with short clauses and without auxiliary variables. We will use this property in the Chinese remainder encoding.

In contrast to the binary adder encoding, the LFSR encoding cannot quickly refute some subcycles. If an assignment to the edge variables forms a subcycle, then one needs $2^k - 1$ conflicts (i.e., all states of a k-bit LFSR) to refute it. This helps explain why the LFSR encoding is less effective in practice.

4 Chinese Remainder Encoding

The challenge in coming up with an effective SAT encoding for HCP lies in enabling the solver to quickly refute an assignment for which the edge variables represent a subcycle. The encodings described in the previous section may require many conflict clauses to a refute a subcycle. Our encoding tries to reduce that number, while keeping the encoding compact. We aim to get the best of three worlds (incremental SAT, binary adder, and LFSR).

From the incremental SAT approach, we borrow the partial encoding of the exactly-one-cycle constraint. One vertex is special in the encoding and indicates where the Hamiltonian cycle "starts". This vertex is denoted by s. The encoding picks the first vertex (based on its index) of smallest degree as s, because reasoning from a vertex with a small degree is considered effective to find a Hamiltonian cycle. Recall that the degree constraint ensures that every vertex is in exactly one cycle. We use a parameter m and additionally enforce that any cycle that does not include s must have length 0 (mod m). Moreover, the cycle that includes s must have length $|V|$ (mod m). The expectation is that one can frequently use $m < |V|$, while still managing to find a Hamiltonian cycle with high probability as it will be difficult to satisfy the constraints for multiple cycles if m is large. Using $m < |V|$ reduces the size of the encoding, which improves solver performance.

Next, we combine ideas of the binary adder and the LFSR encodings. Instead of enforcing cycle lengths to be 0 (mod m) or $|V|$ (mod m), we factorize m into $m = m_1 \times m_2 \times \cdots \times m_q$ such that $m_i = {p_i}^{k_i}$ with prime p_i and positive integer k_i. Furthermore the m_i are pairwise coprime, i.e., for each pair m_i and m_j holds that $p_i \neq p_j$. We enforce that each subcycle has length 0 (mod m_i). By the Chinese remainder theorem, we know that the cycles will be of length 0 (mod m) or $|V|$ (mod m), respectively. In case $k_i > 1$ for some m_i, the Chinese remainder encoding constructs a p_i-ary counter (but with LFSRs for $p_i > 2$). This is shown in Example 2 for $m = 8$, thus $p = 2$ and $k = 3$.

For the prime factors of 2 in m, we use the binary adder encoding, while for the primes of the form $2^k - 1$, such as 3 and 7, we use LFSR. This, of course, excludes several primes. For such a prime p, we use a scheme similar to the binary adder, but having 0 as the successor of $p - 1$. Furthermore, we block all assignments representing indices p to $2^k - 1$ with $k = \lceil \log_2 p \rceil$.

Example 4. For the prime 5, we can encode the cycle $0 \to 1 \to 2 \to 3 \to 4 \to 0$ using three variables for each vertex. For a specific vertex v, they are denoted by $v_{5,1}$, $v_{5,2}$, and $v_{5,3}$. We prevent that variables are assigned to values corresponding to 5, 6, or 7 by adding the binary clauses $(\overline{v}_{5,1} \vee \overline{v}_{5,3}) \wedge (\overline{v}_{5,2} \vee \overline{v}_{5,3})$ for $v \in V$. For an edge variable $e_{u,v}$, we add the ten clauses represented by:

$$e_{u,v} \to (v_{5,1} \leftrightarrow (\overline{u}_{5,1} \wedge \overline{u}_{5,3}))$$
$$e_{u,v} \to (v_{5,2} \leftrightarrow (u_{5,1} \not\leftrightarrow u_{5,2}))$$
$$e_{u,v} \to (v_{5,3} \leftrightarrow (u_{5,1} \wedge u_{5,2}))$$

The main constraint in this encoding enforces that if an edge variable $e_{u,v}$ is true, then v is the successor of u based on the binary adder or LFSR. This is enforced for all edge variables apart from the ones of the form $e_{u,s}$, i.e., the final edge that creates the cycle starting with s. Instead, if an edge variable $e_{u,s}$ is true, then all variables $u_{p_i,j}$ of u must be assigned in such a way that it enforces $|V| - 1 \pmod{p_i}$. Note that for the primes that are encoded using LFSR, the assignment would be the $|V|^{th} \pmod{p_i}$ state of the LFSR with the bit-vector 1 being the first state.

Finally, we apply the following symmetry-breaking clauses: $(\overline{e}_{s,u} \vee \overline{e}_{v,s})$ for $u > v$ for some total ordering of the vertices. We use the vertex number in the input file. This ensures that every Hamiltonian cycle is represented by a unique assignment to the variables.

5 Results

We implemented an encoding tool, called `HCP-encode`[1], that takes as input a graph and an integer m. Using the techniques described in the earlier sections, it enforces that all cycles, apart from the one that includes the initial vertex, must have length $0 \pmod{m}$. We solved all resulting formulas with the default settings of CaDiCaL SAT solver, version 1.4. We selected two subsets of the Flinders HCP challenge: 1) graphs that were used in the 2019 XCSP competition; and 2) some larger graphs that resulted in rather high runtimes during our experiments.

Table 1 shows statistics of the graphs in our benchmark suite and the runtimes for the binary adder encoding and the LSFR encoding. The graphs used in the 2019 XCSP competition are shown on top, the larger ones below them. The results confirm earlier work that the binary adder encoding is more effective

[1] The encoding and decoding tools together with the graphs are available on GitHub at https://github.com/marijnheule/ChineseRemainderEncoding.

Table 1. Statistics of the selected Flinders HCP challenge graphs and the CaDiCaL runtime in seconds on the binary adder encoding and the LFSR encoding.

graph #	$\|V\|$	$\|E\|$	adder (2^k)	LSFR ($2^k - 1$)
48	338	776	47.22	> 3600
162	909	206571	171.23	180.24
171	996	1495	10.12	20.05
197	1188	1783	10.89	84.75
223	1386	2268	272.46	80.01
237	1476	2215	13.65	22.45
249	1558	2338	19.69	150.70
252	1572	2359	13.71	86.07
254	1582	2374	36.31	127.35
255	1584	2799	48.98	40.60
424	2466	4240	> 3600	> 3600
446	2557	4368	> 3600	> 3600
470	2740	4509	2500.61	> 3600
491	2844	4267	173.46	245.92
506	2964	4447	78.29	244.48
522	3060	4591	84.51	611.46
526	3108	4663	160.73	544.97
529	3132	4699	69.69	275.13

Table 2. Runtime statistics in seconds of the selected Flinders HCP challenge graphs using CaDiCaL and various values for the cycle length. The symbols ✓ and ✗ denote whether the satisfying assignment represents a single or multiple cycles, respectively.

graph #	2	6	12	60	105	420
48	0.10 ✗	6.14 ✗	10.02 ✗	45.20 ✗	84.33 ✓	**73.63** ✓
162	53.62 ✗	34.96 ✓	**32.12** ✓	39.43 ✓	35.75 ✓	40.11 ✓
171	0.01 ✗	2.37 ✗	**2.92** ✓	12.76 ✓	4.89 ✓	5.86 ✓
197	0.02 ✗	0.83 ✗	7.16 ✓	**6.77** ✓	8.87 ✓	14.62 ✓
223	0.02 ✗	1.77 ✗	11.65 ✗	22.14 ✓	**15.93** ✓	70.67 ✓
237	0.04 ✗	2.19 ✗	**7.81** ✓	12.64 ✓	6.90 ✗	16.80 ✓
249	0.19 ✗	**0.81** ✓	4.52 ✓	4.36 ✓	3.01 ✓	7.29 ✓
252	0.02 ✗	1.76 ✗	25.33 ✓	14.62 ✓	**9.66** ✓	32.73 ✓
254	0.33 ✗	2.95 ✓	**0.76** ✓	3.11 ✓	2.42 ✓	4.09 ✓
255	1.27 ✗	2.56 ✗	5.17 ✗	14.31 ✗	8.36 ✗	**9.03** ✓
424	9.81 ✗	665.18 ✗	340.11 ✗	307.71 ✗	494.11 ✓	**488.70** ✓
446	13.24 ✗	334.62 ✗	169.52 ✗	380.47 ✗	**573.38** ✓	722.23 ✓
470	17.08 ✗	166.16 ✗	152.31 ✗	933.36 ✗	501.91 ✗	**840.89** ✓
491	0.06 ✗	22.04 ✗	**7.47** ✓	34.45 ✓	123.36 ✓	135.22 ✓
506	0.11 ✗	31.75 ✗	**19.24** ✓	33.48 ✓	28.73 ✓	63.20 ✓
522	0.63 ✗	5.66 ✗	32.95 ✓	133.40 ✓	**30.40** ✓	67.03 ✓
526	0.05 ✗	24.16 ✗	71.67 ✓	**34.37** ✓	34.69 ✗	158.69 ✓
529	0.40 ✗	17.90 ✗	60.19 ✓	48.09 ✓	**42.33** ✓	365.58 ✓

compared to LSFR on these challenge graphs [14]: only once did LSFR outperform the binary adder. However, we observed shorter runtime in our experiments with the binary adder encoding and LSFR compared to recent work [14]. These experiments differ in two ways. First, we use a more recent and possibly a stronger SAT solver for these instances. Second, our implementation does not include preprocessing techniques. It is not clear whether preprocessing helps or hurts performance on these instances. Note that the binary adder and LFSR encodings timed out after an hour for some of the larger graphs.

Table 2 shows the results of our Chinese remainder encoding for various values of the cycle length. We selected cycle lengths that have only small primes in their factorization. The largest cycle length used in the experiments is 420, a number that is divisible by 4, 5, 6, and 7. The table shows the runtime and whether the solution produced by the solver represented a single cycle. The shortest runtime resulting in a single cycle are shown in bold. In general, the larger the cycle, the longer the runtime and the more likely that the result is a single cycle. In particular, for all graphs a cycle length of $0 \pmod 2$ resulted in multiple cycles, while a cycle length of $0 \pmod{420}$ resulted in a single cycle, i.e., a Hamiltonian cycle. In some cases, a smaller cycle length resulted in a single cycle, while a larger cycle length resulted in multiple cycles (see graphs #237 and #526).

The Chinese remainder encoding with cycle length 420 outperformed the binary adder encoding for most graphs. Also, none of the experiments with cycle length 420 timed out. Using a smaller cycle length can frequently reduce the runtime, although that increases the probability of multiple cycles. In practice one could run the Chinese remainder encoding for various cycle lengths in parallel.

6 Conclusions

We presented the Chinese remainder encoding for HCP, which combines elements from the incremental SAT, binary adder, and LFSR approaches. Experimental results on graphs from the Flinders HCP challenge show that the Chinese remainder encoding generally outperforms the alternatives when using a cycle length that can be factored into multiple small primes. In the experiments we used a cycle length of $420 = 2^2 \times 3 \times 5 \times 7$. The Chinese remainder encoding is equivalent to the binary adder encoding when the cycle length is the smallest power of 2 that is larger than the number of vertices, so it can be seen as a generalization of the binary adder.

We only experimented with a single SAT call for each encoding. The effectiveness of a small cycle length, in particular $m = 12$, indicates that an incremental SAT approach using that encoding could be effective. Recent work showed that the pure incremental SAT approach is not effective for large graphs [14], but blocking some cycles with our encoding could be helpful. Also, we plan to explore the best combination of cycle lengths in a parallel solving approach.

Acknowledgements. Supported by the NFS under grant CCF-2006363.
We thank Emre Yolcu and Neng-Fa Zhou for comments on an earlier draft.

References

1. Bomanson, J., Gebser, M., Janhunen, T., Kaufmann, B., Schaub, T.: Answer set programming modulo acyclicity. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) Logic Programming and Nonmonotonic Reasoning. pp. 143–150. Springer International Publishing (2015)
2. Buratti, M., Del Fra, A.: Cyclic Hamiltonian cycle systems of the complete graph. Discrete Mathematics **279**(1), 107–119 (2004), in Honour of Zhu Lie
3. Chiba, N., Nishizeki, T.: The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. Journal of Algorithms **10**(2), 187–211 (1989)
4. Gent, I.P.: Arc consistency in SAT. In: Proceedings of the 15th European Conference on Artificial Intelligence. p. 121–125. ECAI'02, IOS Press, NLD (2002)
5. Golomb, S.W.: Shift Register Sequences. Aegean Park Press (1982)
6. Grebinski, V., Kucherov, G.: Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping. Discrete Applied Mathematics **88**(1), 147–165 (1998), computational Molecular Biology DAM - CMB Series
7. Haythorpe, M.: FHCP challenge set: The first set of structurally difficult instances of the Hamiltonian cycle problem (2019), https://arxiv.org/abs/1902.10352v1
8. Haythorpe, M., Johnson, A.: Change ringing and Hamiltonian cycles: The search for Erin and Stedman triples. EJGTA **7**, 61–75 (2019)
9. Hertel, A., Hertel, P., Urquhart, A.: Formalizing dangerous SAT encodings. In: Marques-Silva, J., Sakallah, K.A. (eds.) Theory and Applications of Satisfiability Testing – SAT 2007. pp. 159–172. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
10. Lin, F., Zhao, J.: On tight logic programs and yet another translation from normal logic programs to propositional logic. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence. p. 853–858. IJCAI'03, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2003)
11. Prestwich, S.: SAT problems with chains of dependent variables. Discrete Appl. Math. **130**(2), 329–350 (Aug 2003)
12. Soh, T., Le Berre, D., Roussel, S., Banbara, M., Tamura, N.: Incremental SAT-based method with native boolean cardinality handling for the Hamiltonian cycle problem. In: Fermé, E., Leite, J. (eds.) Logics in Artificial Intelligence. pp. 684–693. Springer International Publishing (2014)
13. Velev, M.N., Gao, P.: Efficient SAT techniques for absolute encoding of permutation problems: Application to hamiltonian cycles. In: Bulitko, V., Beck, J.C. (eds.) Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009, Lake Arrowhead, California, USA, 8-10 August 2009. AAAI (2009)
14. Zhou, N.F.: In pursuit of an efficient SAT encoding for the Hamiltonian cycle problem. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming. pp. 585–602. Springer International Publishing, Cham (2020)