symmetry

MDPI

Article

Counting Complexity of Period-3 Orbits in Sequential Dynamical Systems and Their Relationship to Error-Correcting Codes over Ternary Fields

Tuğçe Ulutaş [1] and Mehmet Emin Köroğlu [2,*]

[1] Department of Mathematics, Istanbul Bilgi University, Eyüpsultan, 34060 Istanbul, Türkiye; tugce.ulutas@bilgi.edu.tr
[2] Department of Mathematics, Yildiz Technical University, Esenler, 34220 Istanbul, Türkiye
* Correspondence: mkoroglu@yildiz.edu.tr

Abstract

Sequential dynamical systems (SDSs) are discrete dynamical systems defined by a graph, a collection of local update functions, and an update order. A central problem in their theory is to determine the maximum number of periodic orbits of a given period that can appear in the phase space of a system on a fixed graph. Extending Defant's correspondence between period-2 orbits and binary codes to the ternary setting, we recently showed that the maximum number of period-3 orbits of an SDS on the complete graph equals the maximum size of a ternary code with a minimum Hamming distance of at least four. Building on that structural correspondence, the present paper provides a stage-by-stage computational analysis of the associated graph construction, exact search, SDS realization, consistency checking, and candidate-verification procedures. The admissibility graph realizing the extremal quantity as a clique number can be constructed in $\mathcal{O}(3^{2m-2}m)$ time, and a supplied clique of size k can be processed through the construction, consistency-checking, and orbit-verification stages in $\mathcal{O}(k^2m^2)$ time under the stated assumptions. This is polynomial in the explicit candidate parameters. Substituting $k \leq 3^{m-1}$ yields the m-only upper bound $\mathcal{O}(3^{2m-2}m^2)$; this reparameterization does not imply non-polynomial verification in the explicit input size. Determining the extremal value from scratch, however, requires solving a Maximum Clique problem. By the Moon–Moser theorem, an N-vertex graph can have as many as $3^{N/3}$ maximal cliques, with $N = 3^{m-1}$ in the present setting; thus an enumeration-based exact strategy may face a doubly exponential family of maximal cliques in m. This general bound is not a prediction of the search tree observed for the tested instances. Since the SDS graph and the corresponding coding graph have vertex sets of equal size, equality of vertex counts alone gives no immediate asymptotic advantage at the level of the presented worst-case search-space bound; it does not imply identical practical difficulty. The SDS formulation is therefore best understood as a structured framework for constructing and checking candidate solutions rather than as a faster exact algorithm. We do not claim a new complexity-class characterization, lower bound for the admissibility-graph family, or general-purpose Maximum Clique algorithm.

Keywords: sequential dynamical systems; ternary codes; optimal codes; clique numbers; computational complexity

MSC: 37B15; 05C69; 94B25; 68Q25

Check for updates

Academic Editor: Calogero Vetro

Received: 31 August 2026
Revised: 15 September 2026
Accepted: 24 September 2026
Published: 3 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

A sequential dynamical system (SDS) is a mathematical model for discrete-time computation over a network; given a finite graph Y, a state set A, a local update function at each vertex, and a permutation σ specifying the order in which vertices are updated, the system evolves by applying the local functions sequentially according to σ. This formalism provides a rigorous combinatorial framework for studying systems whose global dynamics are obtained by sequentially applying local functions; see [1] for a comprehensive treatment.

The structural theory of SDSs has been developed along several lines. Invertibility, fixed points, and the enumeration of distinct systems arising from different update orderings were studied in [2,3]. Sequentially updated cellular automata on general graphs were examined in [4], and extensions to permutation-based update schedules, together with a collection of open problems, appear in [5]. The behavior of linear SDSs on directed graphs, which is considerably richer than the undirected case, is analyzed in [6]. Cycle equivalence and orbit structure of finite graph dynamical systems are treated in [7–10], and the existence of graphs realizing prescribed orbit structures is established in [11]. Phase-space representations of graph states are studied in [12], while applications to Boolean networks appear in [13,14].

A separate thread of inquiry concerns the counting and enumeration of periodic orbits. Enumerating simple cycles of prescribed length in directed and undirected graphs is the subject of [15]. The question of which orbit structures are computationally hard to realize is addressed in [16], where NP-completeness is established for certain classes of SDSs. Period-length distributions and equilibrium counts for computational sequential systems are examined in [17].

From a dynamical viewpoint, periodic orbits describe recurrent long-term behaviors of a finite-state system. Fixed points represent stationary configurations, whereas cycles of length greater than one represent persistent non-equilibrium recurrence. Such recurrent behavior is important in finite network models where the system may repeatedly return to the same configurations instead of converging to a single equilibrium; limit-cycle structure has, for example, been studied explicitly for dynamic bi-threshold systems [14]. Period-3 is therefore the first case treated here beyond the period-2 orbit–coding correspondence of Defant [18], and it provides a concrete setting in which the computational cost of detecting, constructing, and verifying nontrivial recurrent behavior can be analyzed exactly. We do not claim that a period-3 orbit has one universal physical interpretation across all applications of SDSs. Rather, its significance in the present work is as a specific recurrent dynamical pattern whose maximum possible number has a precise connection with ternary error-correcting codes. Understanding the computational cost of this correspondence also helps distinguish the difficulty of finding recurrent structures from that of verifying a proposed dynamical realization.

The connection between SDS periodic orbits and algebraic coding theory was opened by Defant [18], who proved that the maximum number of 2-cycles attainable in the phase space of an SDS is precisely the maximum size of a binary code of length $n-1$ with minimum Hamming distance at least 3. This result transforms a dynamical question into an extremal coding problem and motivates the study of longer cycles. Standard references for the coding-theoretic background are [19,20]; the extremal problem of maximizing $A_q(n, m)$, the largest code of length n and minimum distance m over $\mathbb{F}_q$, is a central open problem in algebraic coding theory, with special cases discussed in [21]. The relationship between reversible cellular automata and error-correcting codes over finite fields provides a direct structural precedent for the SDS-coding correspondence analyzed in [22–25].

The graph-theoretic tools required for the present analysis include the relationship between clique numbers, chromatic numbers, and degree sequences [26], the structure of

maximal cliques in graphs of simplex codes [27], the covering radius of cycle codes [28], and results on acyclic orientations and their connections to clique enumeration [29–32]. Combinatorial background on k-colored graphs can be found in [33].

The companion paper Ulutaş and Köroğlu [34] extends Defant's results to period-3 orbits over the ternary field. Denoting by θ_{n+1} the maximum number of period-3 orbits achievable in the phase space of any SDS on K_{n+1} with a common local function, the main result of [34] is the identity

$$\theta_{n+1} = A_3(n, 4), \tag{1}$$

where $A_3(n, 4)$ is the largest ternary code of length n with minimum Hamming distance at least 4. The argument rests on two successive graph-theoretic reductions, given by Theorems 4 and 5 of [34], respectively,

$$\theta_{n+1} \xrightarrow{\text{Theorem 4}} \psi(\widehat{T}_{n+1}) \xrightarrow{\text{Theorem 5}} A_3(n, 4),$$

where $\widehat{T}_{n+1}$ is an explicitly constructed graph whose clique number captures the extremal orbit count.

The structural correspondence underlying this study was established in our previous work [34]. In particular, that work introduced the admissibility graph, proved the clique characterization $\theta_m = \psi(\widehat{T}_m)$, established the relation $\theta_{n+1} = A_3(n, 4)$, and provided a construction of an SDS realization from an admissible clique. These structural results are therefore not claimed as new contributions of the present paper. The present work provides a systematic computational analysis of the procedures induced by this previously established correspondence. Its independent contribution is not a new structural correspondence or a new general-purpose Maximum Clique algorithm, but a stage-by-stage analysis of the computational procedures that the correspondence induces. In particular, we quantify the costs of the main computational stages, including admissibility-graph construction, exact extremal search, construction and consistency checking of an SDS realization, and certificate verification. We further complement these bounds with reproducible computational experiments on the instances $m = 4, \ldots, 7$. We do not claim a new complexity-class characterization, a lower bound for the admissibility-graph family, or a new general-purpose Maximum Clique algorithm.

Although the correspondence between period-3 orbits of SDSs and ternary error-correcting codes was established in [34], its computational implications have not been analyzed in detail. In particular, the previous structural results do not quantify the cost of constructing the associated admissibility graph, testing its adjacency relation, constructing an SDS realization from a given clique, or verifying the resulting period-3 orbits. Moreover, the computational relationship between the SDS-based formulation and the corresponding coding-theoretic formulation has not been clarified. This leads to an important algorithmic question: does the SDS formulation provide a computational advantage for determining the extremal quantity, or is it mainly useful for constructing and verifying candidate solutions?

In this paper, we study the computational complexity of the period-3 orbit–coding correspondence. Rather than only giving complexity estimates for the previous structural results, we divide the correspondence into separate computational steps: construction of the admissibility graph, exact search, construction of the SDS vertex function from a candidate clique, consistency checking, and verification of the resulting period-3 orbits. This allows us to separate the difficult problem of finding an extremal solution from the simpler task of verifying a given candidate solution.

In particular, the contributions of this paper are as follows:

- We formulate the structural period-3 orbit–coding correspondence as an explicit computational pipeline and separate its search, construction, consistency, and verification

stages. This allows us to distinguish the cost of finding a solution from the cost of verifying a given solution.

- We determine the computational cost of constructing the admissibility graph $\widehat{T}_m$ and of testing its adjacency condition.
- We analyze the cost of constructing the SDS-compatible vertex function from a given clique and of verifying the corresponding period-3 orbit conditions.
- We separate candidate verification from exact extremal search and identify Maximum Clique search as the principal computational bottleneck when the extremal value is to be determined from scratch.
- We compare the SDS-based graph formulation with the corresponding coding-theoretic graph formulation. Since the two graphs relevant to the same extremal quantity have vertex sets of the same cardinality, there is no immediate asymptotic advantage at the level of the presented worst-case search-space bound. Equal vertex counts alone do not imply identical computational difficulty, which may also depend on graph structure and on the exact solver. The main computational role of the SDS formulation is instead to provide a structured framework for constructing and verifying candidate solutions.

The contribution of this paper is therefore complementary to [34]: that paper establishes the mathematical correspondence between period-3 SDS orbits, clique numbers, and ternary codes, whereas the present paper measures the algorithmic cost of using it. The SDS formulation should not be viewed as an asymptotically faster method for exact extremal computation; its verification cost is polynomial in k and m, but not necessarily polynomial in m alone, since k may grow with the instance.

Table 1 summarizes the distinction between the structural results of [34] and the contributions of the present paper.

Table 1. Comparison between our previous work [34] and the contributions of the present study.

Aspect	Previous Work [34]	Present Work
Period-3 orbit characterization	Structural characterization of period-3 orbits	Computational cost of the associated procedures
Admissibility graph	Definition and structural use of $\widehat{T}_m$	Explicit graph-construction algorithm and complexity analysis
Clique formulation	$\theta_m = \psi(\widehat{T}_m)$	Complexity analysis of the resulting exact Maximum Clique problem
Coding correspondence	$\theta_m = A_3(m-1,4)$ (equivalently $\theta_{n+1} = A_3(n,4)$ in the notation of [34])	Computational consequences of the correspondence
SDS realization	Construction of a vertex function from an admissible clique	Complexity analysis of construction and consistency checking
Verification	Correctness established theoretically	Explicit certificate-verification procedure and complexity bounds
Complexity bounds	Not the focus	Bounds for graph construction, search, construction, consistency checking, and verification
Computational experiments	Illustrative small examples	Systematic experiments for $m = 4, 5, 6, 7$
Solver evaluation	Not considered	Implementation-level comparison with a Bron–Kerbosch enumeration baseline

The remainder of the paper is organized as follows. In Section 2 we recall the necessary background on undirected graphs, cliques, and SDSs, and we introduce the graphs $\widehat{T}_m$, T_{m-1}, and L_{m-1} together with the subsequence-detection routine used to decide adjacency. Section 3 contains the main complexity analysis: we bound the costs of constructing $\widehat{T}_m$, of

verifying period-3 orbits, and of each lemma and theorem entering the proof of (1), and we present the corresponding procedures as formal algorithms. In Section 4 we compare the SDS-based approach with the direct computation of $A_3(m-1,4)$ through the coding graph L_{m-1}. Finally, Section 5 summarizes the resulting complexity picture and concludes the paper.

2. Preliminaries

In this section, we summarize the necessary graph-theoretic and dynamical concepts required throughout this paper. For further foundational details regarding graph theory and finite dynamical systems, we refer the reader to [1,35,36].

Throughout this work, an undirected graph is expressed as an ordered pair $Y = (V, E)$, where $V \neq \varnothing$ represents a finite collection of vertices and $E \subseteq \{\{u, v\} : u, v \in V,\ u \neq v\}$ denotes the set of edges. The graphs under consideration are simple, meaning they contain no loops or multiple edges, and their edges are unoriented. A subset $C \subseteq V$ forms a clique in Y if every pair of distinct vertices in C is linked by an edge, which means that the induced subgraph $Y[C]$ is complete. The maximum cardinality among all cliques in Y is defined as the clique number of Y, denoted by

$$\psi(Y) = \max\{\, |C| : C \subseteq V,\ C \text{ is a clique in } Y \,\}. \tag{2}$$

The parameter (2) serves as the central combinatorial quantity analyzed across the subsequent sections.

A sequential dynamical system (SDS) is specified by four constituent components:

- A underlying finite simple graph Y with vertex set $V = \{v_1, v_2, \ldots, v_n\}$.
- A finite alphabet of states A. In our setup, this state space is identified with the finite field $A = \mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}$.
- A family of vertex functions $\{f_{v_i}\}_{i=1}^{n}$. Each node v_i is equipped with a local mapping $f_{v_i} : A^{d(v_i)+1} \to A$, where $d(v_i)$ is the degree of v_i in Y.
- An update schedule defined by a permutation $\sigma \in S_n$, which establishes the sequential order in which the local functions are executed.

Let $s(v) \in A$ represent the local state at vertex v. For a given node v_i with neighbors $v_{j_1}, \ldots, v_{j_{d(v_i)}}$, the local state configuration centered at v_i is recorded as

$$X(v_i) = (s(v_{j_1}), s(v_{j_2}), \ldots, s(v_{j_t}), s(v_i), s(v_{j_{t+1}}), \ldots, s(v_{j_{d(v_i)}})).$$

The global system state corresponds to the joint vector $(s(v_1), s(v_2), \ldots, s(v_n)) \in A^n$. When Y is the complete graph K_n, $X(v_i)$ incorporates the states of every vertex in the network. Corresponding to each vertex function f_{v_i}, we define a local update operator $L_{v_i} : A^n \to A^n$ that alters only the i-th coordinate:

$$L_{v_i}(x_1, x_2, \ldots, x_n) = (x_1, x_2, \ldots, x_{i-1}, f_{v_i}(X(v_i)), x_{i+1}, \ldots, x_n). \tag{3}$$

Composing these local maps in the sequence dictated by σ yields the global SDS evolution operator $T : A^n \to A^n$, defined as

$$T = L_{v_{\sigma(n)}} \circ L_{v_{\sigma(n-1)}} \circ \ldots \circ L_{v_{\sigma(1)}}. \tag{4}$$

Because the update rule (4) is executed sequentially, the output of a given local map (3) immediately updates the state vector passed to subsequent functions in the evaluation chain. Example 1 demonstrates this sequential dependency explicitly.

Example 1. *Let $Y = K_3$, let the state alphabet be $\mathbb{F}_3 = \{0,1,2\}$, and take $\sigma = (1,2,3)$. Since K_3 is complete, each vertex function may depend on the entire current state. Suppose that all three vertices use the same function*

$$g(x_1, x_2, x_3) = x_1 + x_2 + x_3 \pmod 3.$$

Starting from $\mathbf{x} = (1,0,2)$, we first update vertex 1. Since $g(1,0,2) = 0$,

$$(1,0,2) \xrightarrow{L_{v_1}} (0,0,2).$$

The second update is evaluated at this new intermediate state. Thus $g(0,0,2) = 2$, and

$$(0,0,2) \xrightarrow{L_{v_2}} (0,2,2).$$

Finally, $g(0,2,2) = 1$, so

$$(0,2,2) \xrightarrow{L_{v_3}} (0,2,1).$$

Consequently,

$$T(1,0,2) = L_{v_3} \circ L_{v_2} \circ L_{v_1}(1,0,2) = (0,2,1).$$

This example shows explicitly that one application of the global SDS map is a sequence of ordered local updates and that an earlier update may change the input used in a later one; see Figure 1.

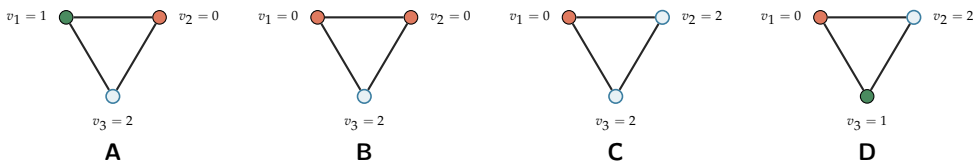


Figure 1. The system update of the SDS given in Example 1 from the initial state $(v_1, v_2, v_3) = (1,0,2)$ on K_3 with state alphabet $\mathbb{F}_3 = \{0,1,2\}$. (**A**) displays the initial state. (**B**,**C**) illustrate the intermediate states after local updates L_{v_1} and L_{v_2}, respectively. (**D**) shows the final system state after the update is completed via L_{v_3}. Vertex colors indicate the current state of each vertex: red for 0, green for 1, and blue for 2.

To track the state progression after step i, we introduce the partial update map $G_i : A^n \to A^n$ for each $i \in \{1,2,\ldots,n\}$ as

$$G_i = L_{v_{\sigma(i)}} \circ L_{v_{\sigma(i-1)}} \circ \ldots \circ L_{v_{\sigma(1)}}. \tag{5}$$

An SDS instance over a graph Y equipped with local functions $\{f_{v_i}\}_{i=1}^n$ and permutation σ is denoted by $[Y, \{f_{v_i}\}_{i=1}^n, \sigma]$.

We now review key definitions introduced in [34] that are central to this paper.

Definition 1. *Let $\alpha = \alpha_1 \cdots \alpha_k$ be a string of elements over $\mathbb{F}_3$. We denote by $S_n(\alpha)$ the set comprising all vectors in $\mathbb{F}_3^n$ that contain α as a (not necessarily contiguous) subword.*

Definition 2. *For $n \geq 3$, let $\widehat{\mathbb{F}}_3^n = \{(u_1, \ldots, u_n) \in \mathbb{F}_3^n : u_n = 0\}$. The graph $\widehat{T}_n$ is defined as a simple undirected graph with*

$$V(\widehat{T}_n) = \widehat{\mathbb{F}}_3^n, \qquad E(\widehat{T}_n) = \left\{\{\vec{\mathbf{u}}, \vec{\mathbf{v}}\} \subseteq \widehat{\mathbb{F}}_3^n : \ \vec{\mathbf{u}} + \vec{\mathbf{v}} \in S_n(102)\right\}, \tag{6}$$

where addition between vectors is evaluated component-wise over $\mathbb{F}_3$.

The condition involving the string 102 in (6) arises naturally from the minimum-distance constraints needed in the orbit construction. The ordered appearance of 102 spans three distinct coordinate locations, enforcing necessary vector separation across at least three positions while encoding the required compatibility conditions for 3-cycles. The elements $1, 0, 2$ form an ordered subsequence, meaning these entries need not occupy adjacent positions.

Definition 3. *For an integer $m \geq 3$, let T_m denote the simple undirected graph with vertex set $V(T_m) = \mathbb{F}_3^m$ and edge set $E(T_m) = \{\{\vec{u}, \vec{v}\} \subseteq \mathbb{F}_3^m : \vec{u} + \vec{v} \in S_m(102)\}$.*

The projection mapping $\rho_n : \widehat{T}_n \longrightarrow T_{n-1}$ given by $\rho_n(x_1, x_2, \ldots, x_n) = (x_1, x_2, \ldots, x_{n-1})$ defines a graph isomorphism between $\widehat{T}_n$ and T_{n-1}. Consequently,

$$\theta_n = \psi(\widehat{T}_n) = \psi(T_{n-1}). \tag{7}$$

Remark 1. *Theorem 4 of [34] establishes that $\theta_n = \psi(\widehat{T}_n)$, where the vertex set of $\widehat{T}_n$ consists of vectors in $\mathbb{F}_3^n$ with a zero in the final coordinate. The isomorphism $\widehat{T}_n \cong T_{n-1}$ from [34] leads directly to $\theta_n = \psi(T_{n-1})$ as shown in (7), and Theorem 5 proves that $\psi(T_{n-1}) = A_3(n-1, 4)$.*

Definition 4. *Let L_n be the simple undirected graph defined on $V(L_n) = \mathbb{F}_3^n$ with edge set $E(L_n) := \{\{\vec{u}, \vec{v}\} \subseteq \mathbb{F}_3^n : d_H(\vec{u}, \vec{v}) \geq 4\}$. By construction,*

$$A_3(n, 4) = \psi(L_n). \tag{8}$$

Throughout our computational analysis, m represents the number of nodes in the underlying complete graph K_m, and

$$N = 3^{m-1}$$

denotes the number of vertices in the corresponding admissibility graph. The associated ternary code has length $m - 1$. In the computational sections, this coding relationship is expressed as

$$\theta_m = A_3(m-1, 4),$$

with L_{m-1} representing the corresponding coding graph. The parameter n is reserved for standard coding-theoretic notation $A_3(n, 4)$ or when directly citing results from [34] in their original form (where $n = m - 1$).

All algorithms presented specify explicit input and output parameters. Vectors in $\mathbb{F}_3$ are represented via coordinate arrays, and $V(\widehat{T}_m)$ is maintained as a set of m-tuples. The graph edge set constructed during execution can be stored using either adjacency lists or a Boolean adjacency matrix.

Unless explicitly noted, graph construction complexities count the cost of evaluating all unordered pairs of vertices and are independent of the choice of graph storage. In contrast, subsequent clique-verification step bounds rely on constant-time adjacency lookups via a precomputed Boolean adjacency matrix.

Algorithm 1 takes an array representation of $\vec{s} \in \mathbb{F}_3^n$ alongside a target word α, returning a Boolean indicator. Utilizing only a single loop index j, its auxiliary space footprint is $\mathcal{O}(1)$. For the fixed pattern $\alpha = 102$, the algorithm executes in $\mathcal{O}(n)$ time.

Algorithm 1 CONTAINS SUBSEQUENCE($\vec{s}$, α)

Require: Vector $\vec{s} = (s_1, \ldots, s_n) \in \mathbb{F}_3^n$; word $\alpha = \alpha_1 \cdots \alpha_k$ over $\mathbb{F}_3$.
Ensure: **true** if and only if $\vec{s}$ contains α as a subsequence.
1: $j \leftarrow 1$
2: **for** $i \leftarrow 1$ **to** n **do**
3: **if** $j \leq k$ **and** $s_i = \alpha_j$ **then**
4: $j \leftarrow j + 1$
5: **end if**
6: **end for**
7: **return** $(j > k)$

3. The Complexity Analysis of a Graph Based Approach to $A_3(m-1, 4)$

The aim of this section is to assign an explicit computational cost to the graph-theoretic constructions used in the SDS-coding correspondence. We restrict our attention to the graph-theoretic and combinatorial verification steps. In particular, the phase space is $\mathbb{F}_3^m$, the distinguished graph $\widehat{T}_m$ is constructed on a subset of $\mathbb{F}_3^m$, and the extremal quantity is obtained through clique search and verification.

Three computational tasks are distinguished throughout, and the terminology is used consistently in the statements, proofs and algorithms below. Construction denotes the production of the vertex function $g : \mathbb{F}_3^m \to \mathbb{F}_3$ from a given clique C of $\widehat{T}_m$. Consistency checking denotes the verification that the values prescribed during that construction are mutually compatible, so that the piecewise definition of g is single-valued. Verification denotes the checking of a supplied object, and is separated into clique verification, the checking that a supplied candidate C is indeed a clique of $\widehat{T}_m$, and orbit verification, the checking that the states produced by g lie on period-3 orbits. Finally, the task of determining an extremal clique is referred to as exact search, or equivalently as Maximum Clique search; it is never called verification. Complexity statements that cover several of these tasks name the individual stages explicitly rather than reporting a single "verification complexity".

We distinguish three input models throughout this section. First, when the admissibility graph is generated from the structural parameter m, its vertex count is $N = 3^{m-1}$, so a bound that is polynomial in N may still be exponential in m. Second, once an explicitly constructed graph is supplied as the input, graph algorithms are measured in terms of N (and, where relevant, $|E|$). Third, when a candidate clique C is supplied, its size k and the length-m representation of its vertices are treated as part of the input. Accordingly, every use of the terms polynomial or exponential below is tied to the corresponding input representation rather than to m alone.

3.1. Setting and Elementary Counts

The main objects are $\widehat{T}_m$, $C = \{\vec{u}_1, \ldots, \vec{u}_k\}$, $g : \mathbb{F}_3^m \to \mathbb{F}_3$, where C is a clique of $\widehat{T}_m$ and g is the SDS-compatible map constructed from C. In the usual last-coordinate-zero model,

$$N := |V(\widehat{T}_m)| = 3^{m-1}. \tag{9}$$

Remark 2. *Let $C_f(m)$ denote the cost of evaluating the local rule $f : \mathbb{F}_3^m \to \mathbb{F}_3$ at one state. Since an SDS over K_m applies m local updates sequentially, the cost of one global SDS update satisfies*

$$C_T(m) = \mathcal{O}\Big(mC_f(m)\Big). \tag{10}$$

If evaluating f requires scanning all coordinates, then $C_f(m) = \mathcal{O}(m)$ and hence

$$C_T(m) = \mathcal{O}(m^2). \tag{11}$$

The complete graph K_m is not used here as a model for arbitrary SDS network topologies. It is part of the specific period-3 extremal problem considered in [34], where the systems under study have the form $[K_m, g, \mathrm{id}]$. Consequently, the complexity bounds derived in this section apply to this complete-dependency setting.

More general sequential dynamical systems may be defined on sparse dependency graphs. In that case, a local update depends only on the state variables in the corresponding neighborhood, and the cost of evaluating the global map may depend on structural parameters such as the vertex degrees and the number of edges. For example, if the evaluation cost of a local function is proportional to the size of its dependency neighborhood, then the total amount of neighborhood data inspected over one sequential sweep is proportional to

$$\sum_{v \in V(G)} (\deg(v) + 1) = m + 2|E(G)|.$$

This observation is intended only to illustrate how sparsity can affect the evaluation cost; a full complexity analysis for arbitrary sparse SDSs would require a separate specification of the local-function representation and evaluation model. We therefore do not interpret the $\mathcal{O}(m^2)$ bound for K_m as a universal bound for all SDS topologies.

An SDS is defined by an undirected simple graph, vertex functions, and an update permutation. In the construction used in the paper, however, these choices are restricted.

Before analyzing $\widehat{T}_m$, we record the sizes of the spaces from which the SDS data are drawn. These counting results quantify the unrestricted search space; the constructive proof in [34] avoids exhaustive enumeration. Each of the $\binom{m}{2}$ unordered pairs of distinct vertices is independently declared an edge or a non-edge. Hence, the number of labeled undirected simple graphs on m vertices is $2^{\binom{m}{2}}$. Constructing the adjacency relation of the fixed graph K_m takes $\mathcal{O}(m^2)$ time.

The number of functions $f : \mathbb{F}_3^m \to \mathbb{F}_3$ is 3^{3^m}, since the domain $\mathbb{F}_3^m$ has 3^m elements, each mappable to one of 3 values. If m independent vertex functions are allowed, the total is 3^{m3^m}.

An update order is a permutation of $\{1, \ldots, m\}$, giving $m!$ possibilities. Storing the identity order $\mathrm{id} = (1, 2, \ldots, m)$ takes $\mathcal{O}(m)$ time; no search is required. The update order may be chosen freely in a general SDS; in the present paper it is fixed as the identity.

3.2. Construction of the Admissibility Graph

The graph $\widehat{T}_m$ is the main combinatorial object. Its construction requires generating vertices and deciding adjacency by the 102-subsequence condition. Since $\widehat{T}_m$ is a simple undirected graph, its number of edges satisfies

$$|E(\widehat{T}_m)| \leq \binom{N}{2} = \frac{N(N-1)}{2}. \tag{12}$$

Consequently, by (12),

$$|E(\widehat{T}_m)| = \mathcal{O}(N^2) = \mathcal{O}(3^{2m-2}). \tag{13}$$

Thus, Equation (13) gives the worst-case edge-count bound used below.

The following result gives the cost of construction of the edge set of $\widehat{T}_m$.

Theorem 1. *The edge set of $\widehat{T}_m$ can be constructed in*

$$\mathcal{O}(N^2 m) = \mathcal{O}(3^{2m-2} m) \tag{14}$$

time, where $N = 3^{m-1}$. Equation (14) is the principal graph-construction bound.

Proof. There are

$$\binom{N}{2} = \frac{N(N-1)}{2} = \mathcal{O}(N^2)$$

unordered pairs of distinct vertices. For each pair $\{\vec{u}, \vec{v}\}$, adjacency is determined by testing whether

$$\vec{s} = \vec{u} + \vec{v} \pmod 3$$

contains 102 as a subsequence. Computing $\vec{s}$ coordinate-wise requires $\mathcal{O}(m)$ operations. Once $\vec{s}$ has been obtained, checking whether 102 occurs as a subsequence can be performed by a single left-to-right scan and therefore also requires $\mathcal{O}(m)$ time. Hence one adjacency test requires $\mathcal{O}(m)$ time. Since this test is performed for every unordered pair of vertices, the total construction time is

$$\mathcal{O}(N^2) \cdot \mathcal{O}(m) = \mathcal{O}(N^2 m).$$

Using $N = 3^{m-1}$ gives

$$\mathcal{O}(N^2 m) = \mathcal{O}(3^{2m-2} m).$$

Thus, the dominant operation in the construction is the adjacency testing over all unordered pairs of vertices. □

Remark 3. *The bound above assumes that adjacency is tested for every unordered pair of distinct vertices of $\widehat{T}_m$. No structural pruning or prior information about nonadjacent pairs is assumed. Therefore, the bound represents the direct worst-case construction cost of the admissibility graph. The $\mathcal{O}(m)$ cost of one adjacency test is likewise a worst-case bound for the direct scan. In an implementation, early detection of the fixed subsequence* 102 *may terminate the scan before all coordinates are examined, so average observed costs can be smaller; no distributional assumption is made here and no average-case bound is claimed. Table 2 collects the resulting counts and costs.*

Table 2. Summary of the main construction costs for the admissibility graph $\widehat{T}_m$, where $N = 3^{m-1}$.

Operation	Number/Cost	Complexity
Vertices	$N = 3^{m-1}$	$\mathcal{O}(Nm)$ storage
Possible vertex pairs	$\binom{N}{2}$	$\mathcal{O}(N^2)$
Single adjacency test	one vertex pair	$\mathcal{O}(m)$
Construction of all edges	$\binom{N}{2}$ tests	$\mathcal{O}(N^2 m)$
Adjacency matrix	$N \times N$	$\mathcal{O}(N^2)$ memory
Adjacency list	$N + \lvert E \rvert$	$\mathcal{O}(N + \lvert E \rvert)$ memory
Clique verification	$\binom{k}{2}$ lookups	$\mathcal{O}(k^2)$ with matrix

3.3. Memory Requirements

The memory requirement depends on the representation used for $\widehat{T}_m$. Explicitly storing all $N = 3^{m-1}$ vertices as m-tuples requires

$$\mathcal{O}(Nm)$$

storage. If the graph is represented by a Boolean adjacency matrix, the matrix contains N^2 Boolean entries and therefore requires

$$\mathcal{O}(N^2) = \mathcal{O}(3^{2m-2})$$

memory. This representation permits constant-time adjacency queries, which is useful for repeated clique verification. Alternatively, an adjacency-list representation requires

$$\mathcal{O}(N + |E|)$$

memory. Since $|E| \leq \binom{N}{2}$, its worst-case memory requirement is also $\mathcal{O}(N^2)$, although it may require substantially less storage when the admissibility graph is sparse. Thus, the adjacency matrix favors fast repeated adjacency queries, whereas the adjacency-list representation may be preferable when memory consumption is the primary concern.

Algorithm 2 is representation-independent at the level of adjacency testing: every unordered pair of vertices is examined once. The pseudocode stores the detected edges in the set E. In an implementation, this edge relation may subsequently be represented either by adjacency lists or by a Boolean adjacency matrix. If a Boolean adjacency matrix is used, inserting or querying an edge takes $\mathcal{O}(1)$ time and the matrix requires $\mathcal{O}(N^2)$ memory. With adjacency lists, the storage requirement is $\mathcal{O}(N + |E|)$, while an arbitrary adjacency query is not necessarily constant time. Therefore, the $\mathcal{O}(N^2 m)$ graph-construction bound is unchanged by the final representation, whereas the later $\mathcal{O}(k^2)$ clique-verification bound assumes the Boolean adjacency-matrix representation.

Algorithm 2 BUILDING THE GRAPH $\widehat{T}_m$

Require: Integer $m \geq 3$.
Ensure: The vertex set $V = V(\widehat{T}_m)$ and the edge set $E = E(\widehat{T}_m)$.

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
for each $\vec{u} \in \mathbb{F}_3^m$ **do**
  if $u_m = 0$ **then**
    $V \leftarrow V \cup \{\vec{u}\}$
  end if
end for ▷ $|V| = 3^{m-1}$
for each ordered pair $(\vec{u}, \vec{v}) \in V \times V$ with $\vec{u} < \vec{v}$ (lexicographic) **do**
  $s \leftarrow \vec{u} + \vec{v} \pmod 3$ ▷ Coordinate-wise addition in $\mathbb{F}_3$
  if CONTAINS SUBSEQUENCE$(s, (1, 0, 2))$ **then**
    $E \leftarrow E \cup \{\{\vec{u}, \vec{v}\}\}$
  end if
end for
return (V, E)

Lemma 1 of [34] concerns structural properties of period-3 orbits in the SDS phase space. Computationally, this requires checking whether a given state returns to itself after three applications of the global SDS map.

To verify that a given state $\vec{x} \in \mathbb{F}_3^m$ lies on a period-3 orbit, one computes $T(\vec{x})$, $T^2(\vec{x})$ and $T^3(\vec{x})$; a constant number of global SDS evaluations is required, so the cost is $\mathcal{O}(C_T(m))$, which equals $\mathcal{O}(m^2)$ under the assumptions of Remark 2.

Remark 4. *If the goal is to count all period-3 orbits in the full phase space, one must apply this verification over all 3^m states. This yields $\mathcal{O}(3^m C_T(m))$.*

Since each period-3 orbit has exactly three points, the number of distinct period-3 orbits is obtained by dividing the number of detected period-3 points by 3.

3.4. From a Candidate Clique to an SDS Vertex Function

Lemma 2 of [34] constructs an SDS-compatible map g from a clique $C = \{\vec{u}_1, \ldots, \vec{u}_k\}$ of $\widehat{T}_m$. The computational cost comes from verifying the clique, constructing the local rule g, checking that g is single-valued, and verifying the resulting 3-cycles.

Definition 5. *Adjacency preprocessing means that, after the graph $\widehat{T}_m = (V, E)$ has been constructed, its adjacency relation is stored in a Boolean matrix*

$$A \in \{0,1\}^{N\times N}, \qquad A_{ij} = 1 \Longleftrightarrow v_i \sim v_j.$$

This representation requires $\mathcal{O}(N^2)$ Boolean storage. Once it is available, each individual adjacency query takes $\mathcal{O}(1)$ time. This does not make verification of a k-vertex candidate clique constant-time: all $\binom{k}{2}$ unordered pairs must still be checked, so clique verification requires $\mathcal{O}(k^2)$ time with preprocessing and $\mathcal{O}(k^2 m)$ time without preprocessing.

Theorem 2. *Let $C = \{\vec{u}_1, \ldots, \vec{u}_k\}$ be a clique of $\widehat{T}_m$, where $k \geq 1$. The construction of the SDS-compatible map $g : \mathbb{F}_3^m \to \mathbb{F}_3$ from C, together with the verification that the piecewise definition of g is single-valued, requires*

$$\mathcal{O}(km^2 + k^2m^2).$$

Since $k \geq 1$, we have $km^2 \leq k^2m^2$, and therefore the bound may be simplified to

$$\mathcal{O}(k^2m^2).$$

No simplification to $\mathcal{O}(m^2)$ is made unless k is assumed to be bounded by an absolute constant independent of m. In the unrestricted worst case, $k \leq 3^{m-1}$, and therefore the same bound expressed only as a function of m is

$$\mathcal{O}(3^{2m-2}m^2).$$

Proof. The construction has two parts. First, for each clique vertex $\vec{u}_i$, the proof prescribes a local orbit pattern that realizes $\vec{u}_i$ as a point on a 3-cycle. Under the identity update order $\mathrm{id} = (1, 2, \ldots, m)$, one must compute or record the intermediate states (5),

$$G_1(\vec{u}_i), G_2(\vec{u}_i), \ldots, G_m(\vec{u}_i).$$

Since the dependency graph is K_m, each local update may depend on all m coordinates. Thus one local update costs at most $\mathcal{O}(m)$, and one full sequence of m updates costs $\mathcal{O}(m^2)$. Repeating this for k clique vertices gives $\mathcal{O}(km^2)$.

Second, the map g is defined piecewise. Therefore one must verify that no state receives two different prescribed values. Such conflicts can occur only through overlaps between the local constructions generated by two clique vertices. Hence one checks all unordered pairs $(\vec{u}_i, \vec{u}_j)$, $i \neq j$. There are $\mathcal{O}(k^2)$ such pairs. For each pair, the forbidden equalities appearing in the construction are checked coordinate-wise, together with the associated 102-pattern condition. We use the upper bound $\mathcal{O}(m^2)$ for each pairwise consistency check. Therefore the total consistency-checking cost is $\mathcal{O}(k^2m^2)$. Combining both stages gives

$$\mathcal{O}(km^2) + \mathcal{O}(k^2m^2) = \mathcal{O}(km^2 + k^2m^2).$$

Because $k \geq 1$, $km^2 \leq k^2m^2$, so this becomes

$$\mathcal{O}(k^2m^2).$$

Here k is retained as an input parameter; the bound reduces to $\mathcal{O}(m^2)$ only under an additional constant-k assumption. □

Remark 5. *The bound of Theorem 2 is stated for the direct implementation and does not depend on how the partial assignment table $\mathcal{G}$ of Algorithm 3 is realized. In practice $\mathcal{G}$ may be stored in a hash map keyed by the coordinate vector, or, since $|\mathbb{F}_3^m| = 3^m$ is known in advance, in a directly indexed*

array obtained by reading a state as a ternary integer; the latter gives worst-case $\mathcal{O}(1)$ lookup at the cost of $\mathcal{O}(3^m)$ memory, whereas hashing gives expected $\mathcal{O}(1)$ lookup with smaller memory when few states are assigned. Such choices affect the constant factor of an individual lookup and the memory profile, but not the stated worst-case bound, because the $\mathcal{O}(k^2)$ pairwise consistency checks and the $\mathcal{O}(km^2)$ construction work are performed in either case. For the same reason, we do not use implementation-dependent improvements to reduce $\mathcal{O}(k^2m^2)$.

Algorithm 3 CONSTRUCT SDS VERTEX FUNCTION (C, m)

Require: Integer $m \geq 3$; clique $C = \{\vec{\mathbf{u}}_1, \ldots, \vec{\mathbf{u}}_k\} \subseteq V(\widehat{T}_m)$, stored as arrays of coordinates.
Ensure: A single-valued map $g : \mathbb{F}_3^m \to \mathbb{F}_3$ realizing the prescribed construction, or CONSISTENCY FAILURE.

Initialize an empty partial map $\mathcal{G}$
for $i \leftarrow 1$ **to** k **do**
  Write $\vec{\mathbf{u}}_i = (c_{i_1}, c_{i_2}, \ldots, c_{i_m})$ ▷ $c_{i_m} = 0$ by Definition 2
  $\overline{c_{i_j}} \leftarrow c_{i_j} + 2 \pmod 3$ and $\widehat{c_{i_j}} \leftarrow c_{i_j} + 1 \pmod 3$ for all $j \in \{1, \ldots, m\}$
  for $l \leftarrow 0$ **to** $m - 1$ **do**
    $z \leftarrow (\overline{c_{i_1}}, \ldots, \overline{c_{i_l}}, c_{i_{l+1}}, \ldots, c_{i_m})$
    if ASSIGN$(\mathcal{G}, z, \overline{c_{i_{l+1}}}) =$ CONSISTENCY FAILURE **then return** CONSISTENCY FAILURE ▷ Case *(i)* of Lemma 2 of [34]
    $z \leftarrow (\widehat{c_{i_1}}, \ldots, \widehat{c_{i_l}}, \overline{c_{i_{l+1}}}, \ldots, \overline{c_{i_m}})$
    if ASSIGN$(\mathcal{G}, z, \widehat{c_{i_{l+1}}}) =$ CONSISTENCY FAILURE **then return** CONSISTENCY FAILURE ▷ Case *(ii)* of Lemma 2 of [34]
    for $v \leftarrow l + 2$ **to** $m - 1$ **do**
      $z \leftarrow (c_{i_1}, \ldots, c_{i_l}, \overline{c_{i_{l+1}}}, \ldots, \overline{c_{i_v}}, c_{i_{v+1}}, \ldots, c_{i_m})$
      if ASSIGN$(\mathcal{G}, z, c_{i_{v+1}}) =$ CONSISTENCY FAILURE **then return** CONSISTENCY FAILURE ▷ Case *(iii)* of Lemma 2 of [34]
    end for
  end for
end for
for each $\vec{\mathbf{a}} \in \mathbb{F}_3^m$ with $\vec{\mathbf{a}} \notin \mathrm{dom}(\mathcal{G})$ **do**
  $\mathcal{G}[\vec{\mathbf{a}}] \leftarrow 0$ ▷ Case *(iv)* of Lemma 2 of [34]
end for
Define $g(\vec{\mathbf{a}}) \leftarrow \mathcal{G}[\vec{\mathbf{a}}]$ for every $\vec{\mathbf{a}} \in \mathbb{F}_3^m$
return g

procedure ASSIGN$(\mathcal{G}, z, a)$
  if $z \notin \mathrm{dom}(\mathcal{G})$ **then**
    $\mathcal{G}[z] \leftarrow a$
  else if $\mathcal{G}[z] \neq a$ **then**
    return CONSISTENCY FAILURE
  end if
  return OK ▷ An existing identical assignment is kept unchanged
end procedure

The bound $\mathcal{O}(k^2m^2)$ is an input-sensitive bound in the two parameters k and m; it does not assume that k is constant. If the clique size grows with m, this stage need not be polynomial in m alone. Since $k \leq |V(\widehat{T}_m)| = 3^{m-1}$, the largest possible substitution gives

$$\mathcal{O}(k^2m^2) \subseteq \mathcal{O}(3^{2m-2}m^2).$$

A certificate consisting of k vertices of $\mathbb{F}_3^m$ has encoding length $\mathcal{O}(km)$ under the coordinate representation used here. With a precomputed Boolean adjacency matrix, checking that the certificate is a clique requires $\mathcal{O}(k^2)$ adjacency lookups; if adjacency is tested directly

from the coordinate vectors, the corresponding cost is $\mathcal{O}(k^2m)$. Thus, for an explicitly supplied clique certificate, these checking costs are polynomial in the explicit candidate representation. The substitution $k \leq 3^{m-1}$ merely rewrites the same estimate as a function of the structural parameter m alone; it does not imply that verification is non-polynomial in its explicit input size. This is distinct from exact extremal search. The Moon–Moser bound discussed below is a general upper bound on the number of maximal cliques and is not a prediction of the search tree of the exact solver used in our experiments.

Throughout the remaining analysis, factors involving the clique size k are retained unless one term is dominated by another term that still contains k. For example,

$$\mathcal{O}(km^2 + k^2m^2) = \mathcal{O}(k^2m^2) \quad (k \geq 1),$$

because $km^2 \leq k^2m^2$; this is not a replacement of k by a constant.

Theorem 3. *The total running time associated with Lemma 2 of [34] is*

$$\mathcal{O}\Big(k^2m + km^2 + k^2m^2 + kC_T(m)\Big).$$

For $k, m \geq 1$, the terms k^2m and km^2 are dominated by k^2m^2, and hence the bound simplifies to

$$\mathcal{O}\Big(k^2m^2 + kC_T(m)\Big).$$

If the adjacency matrix of $\widehat{T}_m$ is already constructed and $C_T(m) = \mathcal{O}(m^2)$, then

$$\mathcal{O}\Big(k^2m^2 + km^2\Big) = \mathcal{O}(k^2m^2)$$

for $k \geq 1$. Again, k is an explicit parameter and is not absorbed into the big-$\mathcal{O}$ constant unless a separate constant-k assumption is stated. Thus, under $C_T(m) = \mathcal{O}(m^2)$ and without a constant-k assumption, the m-only worst-case estimate obtained from $k \leq 3^{m-1}$ is

$$\mathcal{O}(3^{2m-2}m^2).$$

Proof. Lemma 2 of [34] consists of the following sequential stages:

clique verification $\rightarrow$ construction of g with consistency checking $\rightarrow$ orbit verification.

Without adjacency preprocessing, clique verification costs $\mathcal{O}(k^2m)$. The construction of g together with consistency checking of the prescribed assignments costs

$$\mathcal{O}(km^2 + k^2m^2) = \mathcal{O}(k^2m^2).$$

Finally, verifying that each of the k clique vertices lies on a period-3 orbit costs

$$\mathcal{O}(kC_T(m)).$$

Since these stages are sequential, their costs are added:

$$\mathcal{O}(k^2m) + \mathcal{O}(km^2 + k^2m^2) + \mathcal{O}(kC_T(m)).$$

This gives

$$\mathcal{O}(k^2m + km^2 + k^2m^2 + kC_T(m)) = \mathcal{O}(k^2m^2 + kC_T(m)).$$

If the adjacency matrix has already been constructed, clique verification becomes $\mathcal{O}(k^2)$. If moreover $C_T(m) = \mathcal{O}(m^2)$, then the total cost is $\mathcal{O}(k^2 m^2)$. □

Algorithm 3 rewrites the construction used in Lemma 2 of [34] as an explicit assignment procedure. Instead of initializing every state to zero and then overwriting selected entries, we first maintain a partial assignment table $\mathcal{G}$. Each value prescribed by Cases *(i)–(iii)* is inserted through a consistency check. If a state has not yet been assigned, the required value is stored; if it has already been assigned the same value, no action is needed; and if a different value has already been assigned, the procedure ASSIGN returns CONSISTENCYFAILURE and the construction is aborted. Only after all prescribed values have been processed are the remaining states assigned the default value 0, corresponding to Case *(iv)*. This ordering makes the single-valuedness of g explicit and makes every assignment traceable to the corresponding case of Lemma 2.

After Algorithm 3 returns a map g, the resulting SDS transitions are checked by the period-3 verification stage described in the proof of Theorem 3. Hence construction, single-valuedness checking, and orbit verification remain separate and individually traceable computational steps.

3.5. Auxiliary Lemmas and Phase-Space Enumeration

Lemma 3 of [34] concerns three distinct nonadjacent vertices. Its complexity depends on whether the three vertices are already given or whether all triples must be searched.

Theorem 4. *If three distinct vertices $\vec{u}, \vec{v}, \vec{w} \in V(\widehat{T}_m)$ are given, then verifying the nonadjacency relations and the coordinate comparisons used in Lemma 3 of [34] costs $\mathcal{O}(m)$, both with and without adjacency preprocessing. Including a period-3 verification, the total running time is*

$$\mathcal{O}(m + C_T(m)).$$

Proof. For a fixed triple, there are only three pairwise nonadjacency checks

$$(\vec{u}, \vec{v}), \quad (\vec{u}, \vec{w}), \quad (\vec{v}, \vec{w}).$$

Without preprocessing, each adjacency or nonadjacency test costs $\mathcal{O}(m)$, so the total is $\mathcal{O}(m)$. With a Boolean adjacency matrix, the three checks cost $\mathcal{O}(1)$.

The coordinate comparisons appearing in the proof inspect at most m coordinates, so they cost $\mathcal{O}(m)$ in both cases. If one also verifies a period-3 condition, this adds $\mathcal{O}(C_T(m))$. Therefore the total is $\mathcal{O}(m + C_T(m))$. □

Theorem 5. *If all triples of distinct vertices of $\widehat{T}_m$ must be searched, then the triple-selection cost is $\mathcal{O}(N^3)$, and the total cost with coordinate verification is*

$$\mathcal{O}(N^3 m) = \mathcal{O}(3^{3m-3} m).$$

Proof. The number of unordered triples of distinct vertices is $\binom{N}{3} = \mathcal{O}(N^3)$. For each triple, checking the required nonadjacency and coordinate conditions costs at most $\mathcal{O}(m)$ without assuming additional preprocessing. Hence the total cost is $\mathcal{O}(N^3 m)$. Substituting $N = 3^{m-1}$ gives $\mathcal{O}(3^{3m-3} m)$. □

Theorem 4 of [34] establishes an extremal equality of the form $\theta_m = \psi(\widehat{T}_m)$. Its proof has two directions: a lower-bound direction using Lemma 2 of [34] and an upper-bound direction extracting a clique from period-3 orbit representatives.

Theorem 6. *With adjacency preprocessing and $C_T(m) = \mathcal{O}(m^2)$, the total running time associated with Theorem 4 of [34] is*

$$\mathcal{O}\Big(k^2m^2 + 3^m m^2 + \theta_m m + \theta_m^2\Big).$$

If, specifically, the lower-bound construction is evaluated at $k = \theta_m$, then the same simplification is valid for the admissible values $m \geq 3$. Indeed, θ_m is a nonnegative integer; if $\theta_m = 0$ the relevant terms vanish, while for $\theta_m \geq 1$ we have $\theta_m m \leq \theta_m^2 m^2$ and $\theta_m^2 \leq \theta_m^2 m^2$. Hence the bound may be written as

$$\mathcal{O}\Big(3^m m^2 + \theta_m^2 m^2\Big).$$

Proof. The lower-bound direction applies Lemma 2 of [34]. Therefore its complexity is the complexity of Lemma 2 of [34]. Under adjacency preprocessing and $C_T(m) = \mathcal{O}(m^2)$, this contribution is $\mathcal{O}(k^2m^2)$.

The upper-bound direction requires enumerating the SDS phase space to detect period-3 orbits. There are 3^m states, and each state requires $\mathcal{O}(C_T(m))$ time to process, with $C_T(m)$ as in (10). This gives

$$\mathcal{O}(3^m C_T(m)).$$

Under the stated assumption (11), this contribution is therefore

$$\mathcal{O}(3^m m^2),$$

and the factor m^2 must remain in the bound. After detecting the period-3 orbits, one selects representatives. If θ_m representatives are selected and each representative is an m-tuple, this costs $\mathcal{O}(\theta_m m)$. Finally, the representatives must be checked to form a clique in $\widehat{T}_m$. With a Boolean adjacency matrix, this costs $\mathcal{O}(\theta_m^2)$. Without preprocessing, each adjacency check costs $\mathcal{O}(m)$, giving $\mathcal{O}(\theta_m^2 m)$. Thus, the stated bounds follow. Theorem 4 is proved by applying the lower-bound and upper-bound sequentially. Hence their complexities are added

$$\mathcal{O}(k^2m^2) + \mathcal{O}(3^m C_T(m) + \theta_m m + \theta_m^2).$$

If $C_T(m) = \mathcal{O}(m^2)$, this becomes

$$\mathcal{O}(k^2m^2 + 3^m m^2 + \theta_m m + \theta_m^2).$$

If $k = \theta_m$, the lower-bound term becomes $\mathcal{O}(\theta_m^2 m^2)$. Since θ_m is a nonnegative integer, the case $\theta_m = 0$ is trivial. If $\theta_m \geq 1$, then for the admissible values $m \geq 3$,

$$\theta_m m \leq \theta_m^2 m^2, \qquad \theta_m^2 \leq \theta_m^2 m^2.$$

Thus both terms are absorbed into $\mathcal{O}(\theta_m^2 m^2)$, and the simplified bound is $\mathcal{O}(3^m m^2 + \theta_m^2 m^2)$. □

In the notation of [34], Theorem 5 establishes $\theta_{n+1} = A_3(n, 4)$. Since m is the principal SDS parameter in the present paper, we set $n = m - 1$ and henceforth write the same correspondence as $\theta_m = A_3(m - 1, 4)$. Accordingly, the associated coding graph is denoted L_{m-1} when it is compared with $\widehat{T}_m$.

Theorem 7. *Assume that the correspondence between L_{m-1} and T_{m-1} used in Theorem 5 of [34] is given by a coordinate-wise linear map and that adjacency preservation is checked pairwise. Then the construction of L_{m-1} together with checking the correspondence requires*

$$\mathcal{O}\Big((m-1)^2 3^{2m-2}\Big) = \mathcal{O}\Big(m^2 3^{2m-2}\Big)$$

time in the worst case.

Proof. The graph L_{m-1} has 3^{m-1} vertices, hence $\mathcal{O}(3^{2m-2})$ unordered pairs. For each pair, checking the Hamming-distance condition requires scanning $m-1$ coordinates, so the construction of L_{m-1} costs $\mathcal{O}((m-1)3^{2m-2})$. For the same pairs, evaluating the linear map and checking adjacency preservation may require up to $\mathcal{O}((m-1)^2)$ arithmetic and coordinate operations in a conservative implementation, which gives $\mathcal{O}((m-1)^2 3^{2m-2})$. The two stages are sequential, yielding the stated bound. □

Theorem 8. *Let $M(N)$ denote the maximum possible number of maximal cliques in an N-vertex graph. By the Moon–Moser theorem,*

$$M(N) \leq 3^{N/3}. \tag{15}$$

For $\widehat{T}_m$, where $N = 3^{m-1}$, this yields

$$M(3^{m-1}) \leq 3^{3^{m-2}}. \tag{16}$$

Proof. The Moon–Moser theorem states that an N-vertex graph has at most $3^{N/3}$ maximal cliques. Substituting $N = 3^{m-1}$ gives

$$3^{N/3} = 3^{3^{m-1}/3} = 3^{3^{m-2}},$$

which proves the stated bound. □

Consequently, by (16), an exact strategy based on the enumeration of maximal cliques may face a search space that is doubly exponential in m. Here the admissibility graph is used as a finite graph representation of the extremal problem: our exact procedure invokes a Maximum Clique computation on this graph. We do not claim a new polynomial-time reduction proving NP-hardness, nor do we establish NP-hardness for the specific family $\widehat{T}_m$. The consequences of the search-space bound for the comparison with the coding-theoretic formulation are discussed in Section 4.

4. Comparison with Direct Computation of $A_3(m-1, 4)$

4.1. Worst-Case Complexity

For verification tasks, the graph-theoretic SDS formulation is efficient once both a candidate clique and the adjacency relation are available. With a Boolean adjacency matrix, a candidate clique of size k can be verified in $\mathcal{O}(k^2)$ time, while the construction of the associated SDS-compatible map together with consistency checking require $\mathcal{O}(k^2 m^2)$ time under the assumptions described in Section 3.

The situation is different when the extremal value must be determined from scratch. In that case, one must solve a Maximum Clique problem in the associated graph. By (9), the admissibility graph $\widehat{T}_m$ has 3^{m-1} vertices. By (8), the coding graph L_{m-1} relevant to $A_3(m-1, 4)$ also has 3^{m-1} vertices. Thus, passing from the coding-theoretic formulation to the SDS formulation does not reduce the number of vertices in the general exact-search problem.

Moreover, by (15), an N-vertex graph may contain as many as $3^{N/3}$ maximal cliques, which is doubly exponential in the SDS parameter m because $N = 3^{m-1}$. This quantifies the worst-case size of the family of maximal cliques that an enumeration-based approach may have to handle; it is not a running-time bound for every Maximum Clique algorithm, whose actual performance depends on the structure of the particular graph and on the pruning rules employed. The experiments reported below are restricted to the SDS admissibility graphs $\widehat{T}_m$ for $m = 4, \ldots, 7$ and are not intended to validate the Moon–Moser worst-case growth. We do not perform an experimental solver comparison with the coding graphs L_{m-1}. Consequently, the equality of the two vertex-set cardinalities is used only at the level of the vertex-count-based worst-case bound and is not evidence of identical practical complexity. Density, degree distribution, degeneracy, symmetry, adjacency-test cost, preprocessing, and solver behavior may differ between the two graph families.

4.2. Preprocessing Before Exact Clique Search

Before the exact search, our implementation applies two standard reduction rules to the admissibility graph. Both are stated here so that their position in the pipeline and their cost are explicit; their measured effect is reported in Section 4.3.

Algorithm 2 gives the baseline exhaustive construction of $\widehat{T}_m$: all unordered pairs are tested before any clique-search preprocessing is applied. The first reduction rule, degree filtering, is applied after graph construction and before the exact Maximum Clique search. If ℓ is a known lower bound on the clique number, obtained here from a greedy clique, then a vertex belonging to a clique of size at least ℓ must have degree at least $\ell - 1$. Hence vertices whose current degree is smaller than $\ell - 1$ can be removed without deleting any clique of size at least ℓ. Since deleting a vertex can decrease the degrees of its neighbors, the rule is applied iteratively until no further deletion is possible.

With adjacency lists, the iterative filter can be implemented by maintaining current degrees and a queue of vertices that fall below the threshold. Each vertex is deleted at most once and each incident edge is processed only when one of its endpoints is deleted, giving $\mathcal{O}(|V| + |E|)$ time after the graph has been constructed. This preprocessing therefore does not alter the $\mathcal{O}(3^{2m-2}m)$ worst-case bound for the exhaustive pairwise graph construction in Algorithm 2. Its benefit is instead instance-dependent: it can reduce the graph supplied to the exact clique solver, but in the worst case it removes no vertices.

The second reduction rule is decomposition into connected components; it is likewise applied after graph construction and before the exact Maximum Clique search.

If adjacency lists are used, breadth-first search visits each vertex once and examines each edge at most twice. Hence the decomposition requires $\mathcal{O}(|V| + |E|)$ time and $\mathcal{O}(|V|)$ auxiliary space. Since $|V(\widehat{T}_m)| = 3^{m-1}$, its running time can equivalently be written as $\mathcal{O}(3^{m-1} + |E|)$. If the precomputed Boolean adjacency matrix of Definition 5 is used directly and each row is scanned to obtain the neighbors of a vertex, the corresponding implementation takes $\mathcal{O}(|V|^2) = \mathcal{O}(3^{2m-2})$ time. In either representation, this decomposition is performed only after the graph has been constructed and therefore does not reduce the cost of the exhaustive pairwise construction itself.

Let $C_1, \ldots, C_r$ be the connected components of $\widehat{T}_m$. Every clique is contained entirely in one connected component, and consequently

$$\psi(\widehat{T}_m) = \max_{1 \le i \le r} \psi\Big(\widehat{T}_m[C_i]\Big).$$

Thus, the exact Maximum Clique solver may be run independently on the induced subgraph of each component. This can reduce the practical search space when the graph has

several nontrivial components, but it gives no clique-search reduction when the graph is connected and does not improve the general worst-case bound.

Symmetry reduction provides another possible direction: if nontrivial automorphisms of $\widehat{T}_m$ can be identified, equivalent vertices or search branches may be treated collectively. However, the automorphism structure of $\widehat{T}_m$ is not characterized in the present work, so symmetry reduction is mentioned here only as a possible optimization and is not used in the stated complexity bounds. None of these techniques changes the general worst-case conclusions above.

4.3. Computational Verification on Small Instances

To complement the worst-case complexity analysis, we implemented the admissibility-graph construction and an exact Maximum Clique search in Java and performed a small-instance computational verification. The implementation uses arrays for vectors over $\mathbb{F}_3$ and a `BitSet`-based adjacency representation. The Maximum Clique routine is a branch-and-bound procedure with a greedy coloring upper bound; its pseudocode is given in Appendix A. Before exact search, a greedy clique supplies the lower bound ℓ used by Algorithm 4, and the connected-component structure of each graph is recorded through Algorithm 5.

Algorithm 4 ITERATIVE DEGREE FILTERING BEFORE EXACT MAXIMUM CLIQUE SEARCH

Require: A graph $G = (V, E)$ and a clique lower bound ℓ
Ensure: A reduced graph containing every clique of size at least ℓ
$H \leftarrow G$
repeat
  $R \leftarrow \{v \in V(H) : \deg_H(v) < \ell - 1\}$
  $H \leftarrow H - R$
until $R = \varnothing$
return H

Algorithm 5 CONNECTED-COMPONENT DECOMPOSITION OF $\widehat{T}_m$

Require: The adjacency representation of $\widehat{T}_m = (V, E)$
Ensure: The connected components $C_1, \ldots, C_r$
mark every vertex in V as unvisited
$\mathcal{C} \leftarrow \varnothing$
for each $v \in V$ **do**
  if v is unvisited **then**
    $C \leftarrow \varnothing$
    initialize a queue Q with v and mark v as visited
    while $Q \neq \varnothing$ **do**
      remove a vertex u from Q and add u to C
      for each $w \in N(u)$ **do**
        if w is unvisited **then**
          mark w as visited and insert w into Q
        end if
      end for
    end while
    add C to $\mathcal{C}$
  end if
end for
return $\mathcal{C}$

The experiments were performed under Microsoft Windows 10 Home Single Language (version 10.0.19045, build 19045) on an Intel(R) Core(TM) i7-6500U CPU at 2.50 GHz with 8 GB RAM. Both exact solvers were implemented in Java using JavaSE-21/OpenJDK HotSpot 21.0.8 and only the Java standard library. The same `BitSet`-based adjacency representation was used throughout. No time or iteration limit was imposed. The coloring-based branch-and-bound search terminates by candidate exhaustion together with the greedy-coloring upper-bound test; its search-node counter is incremented once for each call to the recursive expansion routine. The returned clique is verified independently by checking every unordered pair of its vertices.

Graph construction, connected-component decomposition, iterative degree filtering, exact Maximum Clique search, and independent clique verification were timed separately using `System.nanoTime`. Thus, the clique-search times reported below do not include graph construction or preprocessing. The greedy lower-bound construction was used as an algorithmic initialization step but was not timed as a separate phase.

Because short Java runs may be affected by class loading and just-in-time compilation, each instance was first executed for five warm-up runs. We then performed twenty measured runs and report the median running time. The timings should therefore be interpreted as implementation-level evidence for the relative behavior of the computational stages, rather than as machine-independent complexity bounds.

Table 3 summarizes the results for $m = 4, 5, 6, 7$. For each m, the admissibility graph has $N = 3^{m-1}$ vertices. The column ρ denotes graph density, ψ is the Maximum Clique size returned by the exact solver, and "nodes" is the number of branch-and-bound search nodes. The phase-specific construction, component-decomposition, filtering, clique-search, and verification times are medians over the twenty measured runs. The "Filtered" column gives the number of vertices remaining after iterative degree filtering, and "Comp." gives the number of connected components.

Table 3. Small-instance structural and computational statistics. Times are medians in milliseconds over twenty measured runs after five warm-up runs and are rounded to four decimal places.

m	N	$\lvert E\rvert$	ρ	Comp.	Filtered	ψ	Nodes	Construct.	Components	Filter	Clique	Verify
4	27	13	0.0370	14	26	2	1	0.0936	0.0559	0.0382	0.0299	0.0019
5	81	360	0.1111	1	81	4	13	0.4329	0.0636	0.0481	0.0969	0.0029
6	243	6171	0.2099	1	243	11	46	1.4958	0.0811	0.0554	0.3072	0.0148
7	729	84,812	0.3196	1	729	33	296	8.9698	0.2702	0.2049	5.4069	0.1196

The experiments confirm several features of the theoretical analysis. First, the observed graph sizes agree with $N = 3^{m-1}$, while the number of edges and the graph density increase rapidly with m. Second, the exact clique search remains inexpensive for these small instances, but the number of search nodes grows from 1 at $m = 4$ to 296 at $m = 7$, illustrating the increasing search effort even in this restricted range. Third, the simple degree filter provides only a 3.70% reduction for $m = 4$ and no reduction for $m = 5, 6, 7$. Likewise, connected-component decomposition is useful only at $m = 4$ in this experiment: the graphs for $m = 5, 6, 7$ are connected. Thus, the preprocessing techniques discussed above should not be assumed to provide a substantial reduction on every instance.

For every tested instance, the clique returned by the exact search was checked independently by testing all pairs of returned vertices against the stored adjacency relation. All such verifications succeeded. The median verification times were 0.0019, 0.0029, 0.0148, and 0.1196 ms for $m = 4, 5, 6, 7$, respectively. This supports the distinction emphasized in the theoretical analysis: verification of a proposed clique is inexpensive relative to the task of obtaining an extremal solution.

These experiments are intentionally limited to small dimensions and are not used to infer asymptotic running times. Their role is to verify the algorithmic pipeline, illustrate the practical growth of the graph and clique-search stages, and test the preprocessing suggestions on concrete instances.

4.4. Comparison with an Enumeration-Based Baseline

Since the Maximum Clique computation is the dominant exact-search stage of the framework, we also report an implementation-level comparison with a classical enumeration-based baseline. The baseline is the Bron–Kerbosch maximal-clique enumeration algorithm with pivoting [37], stated as Algorithm A2. The second solver, Algorithm A1, is a coloring-based branch-and-bound method in which a greedy coloring of the current candidate subgraph supplies an upper bound for pruning. This is the same general algorithmic principle used in several established exact Maximum Clique methods; see, for example, the computational study of Prosser [38] and the coloring-based branch-and-bound literature cited above.

The two procedures do not have identical computational objectives: the coloring-based branch-and-bound routine directly optimizes the clique size, whereas Bron–Kerbosch enumerates maximal cliques and obtains the Maximum Clique number as a by-product. Their recursive states therefore do not define a common "search-node" unit, and cross-algorithm node counts are not used as a performance metric here.

To make the timing observations as controlled as possible, both procedures were implemented in Java, used the same `BitSet`-based adjacency representation, and were run on exactly the same admissibility graphs. For each value of m, five warm-up runs were followed by twenty measured runs, and the median running time was recorded. Both algorithms returned the same clique number on every tested instance, and the returned clique was independently verified against the stored adjacency relation.

Both procedures returned the same Maximum Clique number on every tested instance. Under the stated implementation and measurement protocol, the coloring-based branch-and-bound routine had lower median wall-clock times on these four admissibility graphs. These measurements are implementation-specific observations only. Because Bron–Kerbosch enumerates maximal cliques whereas the coloring-based procedure directly solves the Maximum Clique optimization problem, the timing results are not interpreted as a controlled general performance ranking of Maximum Clique algorithms.

The timing experiment in Table 4 was performed as an independent measurement campaign from Table 3; therefore differences in the measured running time of the coloring-based solver between the two tables are expected. The structural outputs, including graph sizes and Maximum Clique numbers, are unchanged.

Table 4. Implementation-level timing comparison between Bron–Kerbosch with pivoting and the coloring-based branch-and-bound routine. Times are medians in milliseconds over twenty measured runs after five warm-up runs, rounded to four decimal places. Because the procedures have different objectives, the timings are reported as observations for these implementations and instances, not as a general algorithmic ranking.

m	N	ψ	Bron–Kerbosch (ms)	Color B&B (ms)
4	27	2	0.1025	0.0513
5	81	4	0.2190	0.0731
6	243	11	3.7633	0.3222
7	729	33	526.3831	3.5132

4.5. Practical Computational Considerations

The worst-case search-space bounds above do not by themselves predict the practical running time of modern Maximum Clique algorithms. Exact solvers typically avoid exhaustive examination of the entire search space by using branch-and-bound techniques, vertex ordering, coloring bounds, and other pruning rules. For example, coloring-based branch-and-bound methods can obtain upper bounds on the size of a clique that can be generated from a current search node; if such a bound cannot improve the best clique already found, the corresponding branch can be discarded [39].

The experimental solver used here belongs to this coloring-based branch-and-bound family. To place its behavior in a standard algorithmic context, Section 4.4 compares it with the classical Bron–Kerbosch method with pivoting [37]. We also refer to the computational study of Prosser [38], which compares several established exact Maximum Clique algorithms and emphasizes the importance of implementation details, ordering, and coloring-based pruning.

Another useful structural parameter is graph degeneracy. Degeneracy-based variants of maximal-clique enumeration can be substantially more effective on sparse graphs, because the search can be organized around a degeneracy ordering rather than the full vertex set [40]. Thus, the practical difficulty of the admissibility graphs considered here should be evaluated not only through their number of vertices, but also through graph parameters such as density, degree distribution, degeneracy, and connected-component structure.

As Table 3 shows, the effectiveness of the two reduction rules of Section 4.2 is strongly instance-dependent, so stronger preprocessing—improved vertex orderings, tighter coloring bounds, or degeneracy-based techniques—may be needed for larger instances.

4.6. Structural Advantages of the SDS Formulation

Although the SDS formulation does not reduce the worst-case size of the general exact-search problem, it provides structural advantages for verification and realization. First, a candidate clique has a direct dynamical interpretation: its vertices can be used to construct candidate period-3 orbit representatives. Thus, the graph-theoretic object is not merely an abstract encoding of the extremal value.

Second, the SDS formulation separates the problem into explicit computational stages:

$$\text{candidate clique} \longrightarrow \text{construction of } g \longrightarrow \text{period-3 verification.}$$

Each stage can be checked independently, providing a transparent certificate for a proposed lower bound. Third, once the adjacency relation has been preprocessed, a proposed clique can be verified efficiently without solving the extremal problem again. Therefore, the SDS formulation is particularly useful when a candidate construction is already known and the objective is to verify that it produces the required dynamical behavior.

Hence, the principal computational advantage of the SDS formulation is not a better worst-case exact-search bound, but rather a structured framework for constructing, interpreting, and verifying candidate solutions.

4.7. Schematic Summary of the Correspondence

The computational relationships discussed above can be summarized schematically as follows. The period-3 orbit problem for the SDS is represented through the associated admissibility graph, where the extremal problem becomes a Maximum Clique problem. The resulting extremal quantity is then related to the corresponding ternary coding-theoretic formulation. Figure 2 summarizes these connections.

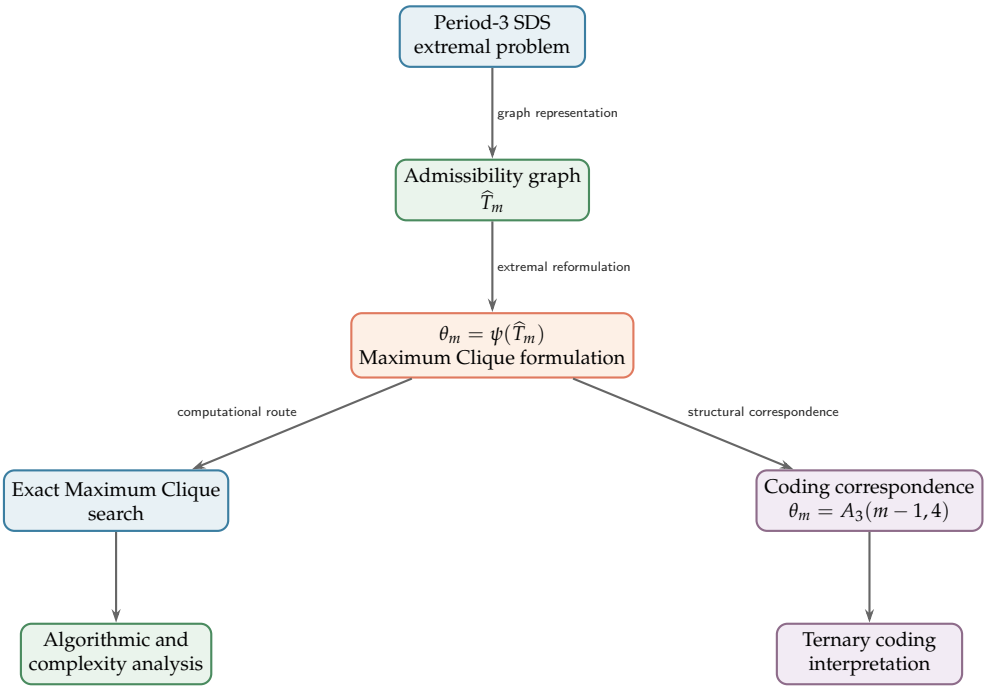


Figure 2. Schematic representation of the structural and computational roles of the period-3 SDS correspondence. The central clique formulation leads both to an exact-search problem, whose algorithmic cost is analyzed in the present paper, and to the ternary coding interpretation established in our previous work. In the notation used throughout the computational sections, the associated ternary code has length $m-1$.

5. Conclusions and Future Remarks

The structural period-3 orbit–coding correspondence, written in the notation of the present computational analysis as $\theta_m = A_3(m-1,4)$, was established in our previous work [34] and is not claimed as a new result here. In this paper we have provided a computational analysis of the algorithmic stages induced by that correspondence. The analysis distinguishes graph construction, exact extremal search, construction and consistency checking of the SDS vertex function, and direct verification of candidate properties. In particular, the admissibility graph on $N = 3^{m-1}$ vertices can be constructed in $\mathcal{O}(3^{2m-2}m)$ time by testing all unordered vertex pairs, while processing a supplied clique through the SDS construction and consistency checking requires $\mathcal{O}(k^2m^2)$ time under the assumptions stated in Section 3. These bounds describe the computational stages considered in this paper; they do not constitute a new complexity-class characterization or lower-bound result, and actual running time may depend on the implementation and the exact clique-search procedure used.

The theoretical analysis identifies exact Maximum Clique as the principal bottleneck when the extremal quantity is determined from scratch. The small-instance experiments complement this conclusion without being used to infer asymptotic running times. For $m = 4, 5, 6, 7$, the implemented admissibility graphs have $27, 81, 243$, and 729 vertices, respectively, and all returned cliques were independently verified. For these tested values, iterative degree filtering removes one vertex at $m = 4$ and no vertices at $m = 5, 6, 7$; likewise, $\widehat{T}_4$ has 14 connected components, whereas the tested graphs for $m = 5, 6, 7$ are connected. These observations are empirical findings for the reported instances and are not extrapolated to larger values of m.

The implementation-level comparison in Section 4.4 reports that both procedures return the same clique number on every tested instance. Under the stated experimental setup, the coloring-based branch-and-bound routine has lower median wall-clock times than the Bron–Kerbosch enumeration baseline. Because the two procedures have different computational objectives and different recursive states, their search-node counts are not compared, and the timing observation is not presented as a general performance ranking of Maximum Clique algorithms.

The theoretical and experimental results show that the SDS formulation does not remove the worst-case difficulty of exact extremal search and does not provide a faster general Maximum Clique algorithm. Its main computational role is to separate the exact-search stage from construction, consistency checking, and candidate verification, so that once a candidate clique is known the corresponding SDS can be processed without solving the extremal problem again.

The experiments in this paper are limited to small values of m and to the implementations described above. Larger instances, comparisons with optimized Maximum Clique software, a controlled empirical comparison with the corresponding coding graphs, and a more detailed study of the structural properties of the admissibility graphs remain for future work. Possible directions include studying whether the algebraic or combinatorial structure of $\widehat{T}_m$ can be used to improve clique search, extending the analysis to longer orbit periods and larger alphabets, and testing the methods on larger instances.

Author Contributions: Conceptualization, T.U. and M.E.K.; methodology, T.U. and M.E.K.; formal analysis, T.U. and M.E.K.; investigation, T.U. and M.E.K.; writing–original draft preparation, T.U. and M.E.K.; writing–review and editing, T.U. and M.E.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Yildiz Technical University Scientific Research Projects Coordination Department (Project Number: FDK-2026-7620).

Data Availability Statement: All computational results reported in this study are contained within the article. No external dataset was used; the Java code implementing Algorithms 1–5 and the routines of Appendix A is available from the first author on request.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Exact Maximum Clique Routines Used in the Experiments

This appendix states the two exact routines compared in Section 4.4. Both operate on the adjacency representation produced by Algorithm 2 and, in our implementation, on the reduced graph returned by Algorithm 4.

Algorithm A1 MAX CLIQUE(G): coloring-based branch and bound

Require: Undirected simple graph $G = (V, E)$ with $V = \{v_1, \ldots, v_t\}$; $N_G(v)$ denotes the neighborhood of v.
Ensure: The clique number $\psi(G)$ and a Maximum Clique C^*.

$C^* \leftarrow \emptyset$; $best \leftarrow 0$
Order V so that $d(v_1) \geq d(v_2) \geq \cdots \geq d(v_t)$
for $i \leftarrow 1$ **to** t **do**
    $P \leftarrow N_G(v_i) \cap \{v_{i+1}, \ldots, v_t\}$ ▷ Cliques whose first vertex is v_i
    $(C, s) \leftarrow$ EXPAND$(\{v_i\}, P, best)$
    if $s > best$ **then**
        $best \leftarrow s$; $C^* \leftarrow C$
    end if
end for
return $(best, C^*)$ ▷ $best = \psi(G)$

procedure EXPAND($C, P, best$)
    if $P = \emptyset$ **then**
        return $(C, |C|)$
    end if
    Greedily colour $G[P]$ and let χ be the number of colours used
    if $|C| + \chi \leq best$ **then** ▷ $|C| + \chi$ bounds any extension of C within P
        return $(\emptyset, 0)$
    end if
    Choose a branching vertex $v \in P$ of largest colour class
    $(C_1, s_1) \leftarrow$ EXPAND$(C \cup \{v\},\ P \cap N_G(v),\ best)$
    $best \leftarrow \max(best, s_1)$ ▷ Tighten the bound before the second branch
    $(C_2, s_2) \leftarrow$ EXPAND$(C,\ P \setminus \{v\},\ best)$
    return the pair with the larger second component
end procedure

Algorithm A2 BRON–KERBOSCH with pivoting [37], used as the exact baseline

Require: Undirected simple graph $G = (V, E)$
Ensure: The clique number $\psi(G)$

$best \leftarrow 0$
BK$(\emptyset, V, \emptyset)$
return $best$

procedure BK(R, P, X)
    if $P = \emptyset$ **and** $X = \emptyset$ **then**
        $best \leftarrow \max(best, |R|)$ ▷ R is a maximal clique
        return
    end if
    Choose a pivot $u \in P \cup X$ maximizing $|P \cap N_G(u)|$
    for each $v \in P \setminus N_G(u)$ **do**
        BK$(R \cup \{v\},\ P \cap N_G(v),\ X \cap N_G(v))$
        $P \leftarrow P \setminus \{v\}$; $X \leftarrow X \cup \{v\}$
    end for
end procedure

References

1. Mortveit, H.; Reidys, C. *An Introduction to Sequential Dynamical Systems*; Springer Science and Business Media: New York, NY, USA , 2007.
2. Barrett, C.L.; Mortveit, H.S.; Reidys, C.M. Elements of a theory of computer simulation II: Sequential dynamical systems. *Appl. Math. Comput.* **2000**, *107*, 121–136. [CrossRef]
3. Barrett, C.L.; Mortveit, H.S.; Reidys, C.M. Elements of a theory of computer simulation III: Equivalence of SDS. *Appl. Math. Comput.* **2001**, *122*, 325–340. [CrossRef]

4. Barrett, C.L.; Reidys, C.M. Elements of a theory of computer simulation I: Sequential CA over random graphs. *Appl. Math. Comput.* **1999**, *98*, 241–259. [CrossRef]
5. Garcia, L.D.; Jarrah, A.S.; Laubenbacher, R. Sequential dynamical systems over words. *Appl. Math. Comput.* **2006**, *174*, 500–510. [CrossRef]
6. Chen, W.Y.; Li, X.; Zheng, J. Matrix method for linear sequential dynamical systems on digraphs. *Appl. Math. Comput.* **2005**, *160*, 197–212. [CrossRef]
7. Macauley, M.; Mortveit, H.S. Cycle equivalence of graph dynamical systems. *Nonlinearity* **2009**, *22*, 421. [CrossRef]
8. Laubenbacher, R.; Pareigis, B. Equivalence relations on finite dynamical systems. *Adv. Appl. Math.* **2001**, *26*, 237–251. [CrossRef]
9. Barrett, C.L.; Mortveit, H.S.; Reidys, C.M. ETS IV: Sequential dynamical systems: Fixed points, invertibility and equivalence. *Appl. Math. Comput.* **2003**, *134*, 153–171. [CrossRef]
10. Mallik, S.; Yildiz, B. Graph theoretic aspects of minimum distance and equivalence of binary linear codes. *Australas. J. Comb.* **2021**, *79*, 515–526.
11. Mowshowitz, A.; Dehmer, M.; Emmert-Streib, F. A note on graphs with prescribed orbit structure. *Entropy* **2019**, *21*, 1118. [CrossRef]
12. Klimov, A.B.; Munoz, C.; Sánchez-Soto, L.L. Graph states in phase space. *J. Phys. Math. Theor.* **2012**, *45*, 215303. [CrossRef]
13. Bridoux, F.; Durbec, A.; Perrot, K.; Richard, A. Complexity of fixed point counting problems in Boolean networks. *J. Comput. Syst. Sci.* **2022**, *126*, 138–164. [CrossRef]
14. Wu, S.; Adiga, A.; Mortveit, H.S. Limit cycle structure for dynamic bi-threshold systems. *Theor. Comput. Sci.* **2014**, *559*, 34–41. [CrossRef]
15. Alon, N.; Yuster, R.; Zwick, U. Finding and counting given length cycles. *Algorithmica* **1997**, *17*, 209–223. [CrossRef]
16. Barrett, C.L.; Hunt, H.B., III; Marathe, M.V.; Ravi, S.S.; Rosenkrantz, D.J.; Stearns, R.E.; Tosic, P.T. Gardens of Eden and fixed points in sequential dynamical systems. *Discret. Math. Theor. Comput. Sci.* **2001**, *AA*, 95–110 . [CrossRef]
17. Aledo, J.A.; Diaz, L.G.; Martinez, S.; Valverde, J.C. On periods and equilibria of computational sequential systems. *Inf. Sci.* **2017**, *409*, 27–34. [CrossRef]
18. Defant, C. Binary codes and period-2 orbits of sequential dynamical systems. *Discret. Math. Theor. Comput. Sci.* **2017**, *19*, 10 . [CrossRef]
19. Van Lint, J.H.; Wilson, R.M. *A Course in Combinatorics*; Cambridge University Press: New York, NY, USA , 2001.
20. Ling, S.; Xing, C. *Coding Theory: A First Course*; Cambridge University Press: New York, NY, USA , 2004.
21. Milshtein, M. A new binary code of length 16 and minimum distance 3. *Inf. Process. Lett.* **2015**, *115*, 975–976. [CrossRef]
22. Köroğlu, M.E.; Şiap, I.; Akın, H. The reversibility problem for a family of two-dimensional cellular automata. *Turk. J. Math.* **2016**, *40*, 665–678. [CrossRef]
23. Köroğlu, M.E.; Şiap, I.; Akın, H. Error correcting codes via reversible cellular automata over finite fields. *Arab. J. Sci. Eng.* **2014**, *39*, 1881–1887. [CrossRef]
24. Şiap, I.; Akın, H.; Köroğlu, M.E. Reversible cellular automata with penta-cyclic rule and ECCs. *Int. J. Mod. Phys.* **2012**, *23*, 1250066. [CrossRef]
25. Şiap, I.; Akın, H.; Köroğlu, M.E. The reversibility of (2r + 1)-cyclic rule cellular automata. *TWMS J. Appl. Eng. Math.* **2013**, *4*.
26. Berlov, S. Relationships between the clique number, chromatic number, and the degree for some graphs. *Autom. Control. Comput. Sci.* **2010**, *44*, 407–414. [CrossRef]
27. Kwiatkowski, M.; Pankov, M. On maximal cliques in the graph of simplex codes. *J. Geom.* **2024**, *115*, 10. [CrossRef]
28. Solé, P.; Zaslavsky, T. The covering radius of the cycle code of a graph. *Discret. Appl. Math.* **1993**, *45*, 63–70. [CrossRef]
29. Reidys, C.M. Acyclic orientations of random graphs. *Adv. Appl. Math.* **1998**, *21*, 181–192. [CrossRef]
30. West, D.B. Acyclic orientations of complete bipartite graphs. *Discret. Math.* **1995**, *138*, 393–396. [CrossRef]
31. Savage, C.D.; Zhang, C.Q. The connectivity of acyclic orientation graphs. *Discret. Math.* **1998**, *184*, 281–287. [CrossRef]
32. Reidys, C.M. On acyclic orientations and sequential dynamical systems. *Adv. Appl. Math.* **2001**, *27*, 790–804. [CrossRef]
33. Read, R. The number of *k*-coloured graphs on labelled nodes. *Can. J. Math.* **1960**, *12*, 410–414. [CrossRef]
34. Ulutaş, T.; Köroğlu, M.E. Period-3 Orbits of Sequential Dynamical Systems and Their Relationship to Error-Correcting Codes over Finite Fields. *Graphs Comb.* **2026**, *42*, 14. [CrossRef]
35. Diestel, R. *Graph Theory*; Springer Nature: Berlin/Heidelberg, Germany, 2025; Volume 173.
36. West, D.B. *Introduction to Graph Theory*; Prentice Hall: Upper Saddle River, NJ, USA , 2001; Volume 2.
37. Bron, C.; Kerbosch, J. Algorithm 457: Finding all cliques of an undirected graph. *Commun. Acm* **1973**, *16*, 575–577. [CrossRef]
38. Prosser, P. Exact algorithms for maximum clique: A computational study. *Algorithms* **2012**, *5*, 545–587. [CrossRef]

39. Tomita, E.; Sutani, Y.; Higashi, T.; Wakatsuki, M. A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments. *IEICE Trans. Inf. Syst.* **2013**, *E96-D*, 1286–1298. [CrossRef]
40. Eppstein, D.; Löffler, M.; Strash, D. Listing all maximal cliques in large sparse real-world graphs. *ACM J. Exp. Algorithmics* **2013**, *18*, 3.1–3.21. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.